Editorial
UNIMAGDALENA

ERNESTO GALVIS LISTA
ALEXANDER BUSTAMANTE MARTÍNEZ

BASES DE DATOS RELACIONALES

UN ENFOQUE APLICADO Y ORIENTADO A RESULTADOS DE APRENDIZAJE

Bases de Datos Relacionales

Un enfoque aplicado y orientado a resultados de aprendizaje

Ernesto Galvis Lista
Alexander Bustamante Martínez

Colección Ingeniería y Tecnología
Serie: Ingeniería Sistemas y Computación

Catalogación en la publicación – Biblioteca Germán Bula Meyer

Galvis Lista, Ernesto; Bustamante Martínez, Alexander
Base de datos relacionales : Un enfoque aplicado y orientado a resultados de aprendizaje / Ernesto Galvis Lista; Alexander Martínez Bustamante -- Primera edición -- Santa Marta: Editorial Unimagdalena, 2023
1 recurso en línea : archivo de texto: PDF. – (Ingeniería de Sistemas y Computación)

ISBN 978-958-746-613-3 (pdf) -- 978-958-746-614-0 (epub)

1. Bases de datos relacionales 2. Administración de bases de datos 3. Programas para computador

CDD: 005.756

Primera edición, abril de 2023

Editorial Unimagdalena
Carrera 32 n.º 22-08
Edificio de Innovación y Emprendimiento
(57 - 5) 4381000 Ext. 1888
Santa Marta D.T.C.H. - Colombia
editorial@unimagdalena.edu.co
https://editorial.unimagdalena.edu.co/

Colección Ingeniería y Tecnología, serie: Ingeniería Sistemas y Computación

Rector: Pablo Vera Salazar
Vicerrector de Investigación: Jorge Enrique Elías-Caro

Diseño editorial: Luis Felipe Márquez Lora
Diagramación: Jeynner Kevin Páez Vélez
Diseño de portada: Orlando Javier Contreras Cantillo
Corrección de estilo: Juan Diego Mican González

Santa Marta, Colombia, 2023

ISBN: 978-958-746-613-3 (pdf)
ISBN: 978-958-746-614-0 (epub)

DOI: https://doi.org/10.21676/9789587466133

Contenido

Presentación

El libro que tiene entre sus manos surgió de la intención de los autores, docentes del programa de Ingeniería de Sistemas de la Universidad del Magdalena, de ofrecer a sus estudiantes un material que sirviese de apoyo en el desarrollo de su primer curso de bases de datos (el cual aborda las bases de datos relacionales transaccionales). Luego de leer la anterior afirmación, un par de preguntas afloran de forma natural en cualquier persona con un mínimo de experiencia en el campo de las bases de datos: ¿por qué se necesitaría un nuevo libro de bases de datos?, ¿qué aporta este libro con respecto a la multiplicidad ya existente en la literatura académica?.

Es cierto que en la literatura académica existe un elevado número de libros de texto y de documentos que abordan la problemática de la enseñanza de las bases de datos relacionales, pero la mayoría de estos lo hacen centrándose en el contenido y en la sintaxis del lenguaje estructurado de consultas (SQL). Estos enfoques, aunque son válidos e importantes en el ecosistema que conforma el campo de las bases de datos, pueden ser complementados con el tomado por los autores: una perspectiva diferente que va en sintonía con las tendencias sobre el aprendizaje y el diseño curricular, basado en el aprendizaje y, para ser más precisos, en los resultados de aprendizaje.

En este orden de ideas, el libro se concentra en abordar tres resultados de aprendizaje que van de la mano con tres escenarios a los que un profesional en computación se puede ver enfrentado en su ámbito profesional: (1) el uso y la explotación de las bases de datos relacionales transaccionales, para lo cual se construyen *scripts* en SQL que obtienen información almacenada; (2) la estructuración, a nivel conceptual y lógico, de una base de datos relacional que brinde el almacenamiento estructurado de datos transaccionales en un contexto organizacional específico, y (3) la optimización de las bases de datos existentes o la creación de rutinas que mejoren la forma como se procesan las solicitudes, para lo cual se implementan los objetos especificados en el diseño lógico y otros asociados a la operación de una base de datos relacional, en un sistema de gestión de bases de datos de amplio uso y demanda en la industria. Así pues, los capítulos del uno al siete contribuyen con el primer resultado de aprendizaje; del ocho al diez hacen lo propio con el segundo resultado de aprendizaje, mientras que los capítulos once y doce corresponden al tercer resultado de aprendizaje.

Si bien para el desarrollo del contenido se utiliza el sistema gestor de bases de datos relacionales (SGBD) PostgreSQL, los conceptos abordados se ajustan a la mayor parte de

otros SGBD relacionales existentes en el mercado como Oracle, MSSQL Server y MySQL, sin perjuicio de las implementaciones específicas que cada uno de estos haga de los conceptos. Aclarado esto, para el correcto funcionamiento de lo presentado en el libro se debe tener instalado el SGBD PostgreSQL, mínimo en su versión 9.6. La descarga puede realizarse desde el sitio web oficial: https://www.postgresql.org/download/

Para el diseño a nivel conceptual, en el libro se utiliza la herramienta *online* ERDPlus, pero puede emplearse cualquier otra que soporte la notación de Chen para el modelado conceptual de bases de datos. De igual manera, para el modelo lógico es posible recurrir a cualquiera de las herramientas disponibles en el mercado, entre las que están DBDesigner y SqlDBM. Todas estas herramientas se encuentran directamente en los enlaces siguientes:

- Diseño conceptual: https://erdplus.com/
- Diseño lógico: https://www.dbdesigner.net/
- Diseño lógico: https://sqldbm.com/Home/

Asimismo, para facilitar el seguimiento de las instrucciones indicadas en cada capítulo del documento, se proporciona una serie de recursos que incluyen *scripts* para la creación de bases de datos, *scripts* para poblar las estructuras creadas con datos de prueba que permitan su utilización y *scripts* de las consultas mostradas en el documento. También se comparten otro tipo de recursos que van orientados, más que a permitir el seguimiento del material, a facilitar su comprensión. Con el fin de posibilitar el uso y la descarga de estos materiales, se encuentra un repositorio en GitHub abierto al público en el que los materiales están organizados capítulo a capítulo:

Repositorio: https://github.com/aprendiendobasesdedatos/relacionales.git

Finalmente, es clave destacar que el documento no tiene la pretensión de convertirse en los términos de referencia de ningún SGBD ni ser una réplica del estándar ANSI SQL. Lo que busca es mostrar los conceptos estructurales y formativos que constituyen las bases de datos relacionales. Por lo tanto, en algunos acápites los autores se toman libertades como utilizar funciones que, si bien no hacen parte del estándar, tienen un equivalente en todas los SGBM y producen el mismo efecto. Este es el caso, por ejemplo, de algunas funciones escalares, cuyo propósito no es explicar una función escalar en específico, sino cómo operan y cómo se deben usar este tipo de funciones.

Al final de este documento también podrá encontrar un glosario de términos y sus respectivas definiciones, con la intención de brindar una visión más clara de algunas palabras utilizadas dentro del texto. Se recomienda su consulta previa y ante cualquier inquietud específica de un concepto en particular.

Capítulo 1

Datos, tablas, columnas y filas

Resultados de aprendizaje

– *Identifica los componentes básicos de las tablas de una base de datos relacional para almacenar datos de forma estructurada.*
– *Especifica los elementos de análisis para construir consultas que den respuesta a necesidades del contexto de uso de la base de datos relacional.*

Las bases de datos están presentes en muchas actividades de la vida cotidiana. Comprar un producto en un supermercado, solicitar una cita médica en una entidad de servicios de salud, consultar la disponibilidad de un libro en una biblioteca o revisar los resultados de eventos deportivos son actividades en las que las bases de datos cumplen un rol esencial. Sin embargo, en muchas ocasiones pasan desapercibidas para la mayoría de las personas.

En este capítulo se proponen un conjunto de actividades que le permitirán aprender a identificar los elementos que componen las estructuras básicas de almacenamiento que se utilizan en las bases de datos relacionales, es decir, las tablas. También tendrá un primer acercamiento a una forma de emplear los datos almacenados en una base de datos, definiendo algunos elementos de análisis que permiten construir consultas sencillas sobre una tabla.

El hecho de que diariamente muchas personas en el mundo se beneficien de la existencia de las bases de datos no implica que tengan siquiera una pequeña noción de cómo están construidas, cómo fueron diseñadas o cómo se utilizan directamente. Esto último se plantea porque, en la gran mayoría de los casos, el uso de las bases de datos sucede cuando alguien interactúa con algún software y aprovecha alguna de sus funcionalidades; por ejemplo, cuando se emplea una aplicación para un dispositivo móvil que permite pedir un

domicilio a un restaurante o una aplicación web para reservar una habitación en un hotel o en otro tipo de alojamiento.

De forma general, como una primera aproximación, puede entenderse que una base de datos es un depósito digital en el que se almacenan datos, los cuales se ponen a disposición para ser consultados, actualizados, eliminados, administrados y controlados. También debe mencionarse que existen softwares construidos para administrar y operar las bases de datos, conocidos de forma genérica como DBMS por su denominación en inglés *(Database Management System)*. Estas definiciones serán suficientes por el momento para avanzar en el aprendizaje de las bases de datos y, específicamente, las relacionales.

1.1. Elementos básicos de una base de datos relacional

Para iniciar el aprendizaje de lo que son las bases de datos y cuál es su utilidad se tomará un caso muy sencillo. La música es algo que mucha gente en el mundo disfruta día a día. Incluso hay personas que no pueden vivir sin ella. Por esa razón han surgido varias plataformas que ofrecen colecciones de canciones para que el público las reproduzca gratuitamente o pagando una tarifa de suscripción mensual.

Los datos esenciales para administrar la colección de una plataforma de este tipo son los correspondientes a las canciones y a quienes las interpretan, es decir, los artistas. Hay que tener presente que estos últimos pueden ser solistas o grupos. También, una canción puede hacer parte de un álbum en el que se publican un número específico de canciones o puede ser un sencillo, es decir, una canción publicada o lanzada de forma individual.

Para almacenar los datos de una colección de canciones se ha diseñado e implementado una base de datos relacional compuesta por dos tablas: la tabla Artistas (Tabla 1.1) y la tabla Canciones (Tabla 1.2). Estas tablas tienen los datos iniciales de la colección de canciones, compuesta por doce canciones interpretadas por cinco artistas.

Cada fila de la tabla Artistas representa a un solista o a un grupo del cual se tienen canciones en la colección. Para cada artista se registra el nombre, el tipo —si es solista o es un grupo—, su género musical principal y el año de lanzamiento en el mercado. También hay un dato numérico que sirve de identificador individual para cada artista dentro de la tabla, el cual no tiene un significado específico; no es un dato que tenga sentido por fuera de la base de datos. Con los datos de la tabla Artistas puede determinarse que el artista *Carlos Vives* es un *Solista* cuyo género musical principal es el *Vallenato* y está activo en el mundo de la música desde el año *1986*. De la misma forma puede observarse que el artista llamado *Niche* es un *Grupo* que está activo en el mundo de la música desde el año *1979* y su género musical principal es la *Salsa*. En ambos casos, los datos registrados en la primera columna, es decir, los números *50001* y *50002*, no tienen ningún significado específico con el artista correspondiente; solamente son útiles para identificarlos dentro de la tabla, para llegar a la fila que le corresponde a cada uno, o para determinar filas de otras tablas con las que tengan relación.

En las tablas se registran únicamente los datos que se requiere almacenar, procesar y administrar, los cuales son apenas algunos de los que podrían identificarse. Por ejemplo, en la

Tabla 1.1 no están todos los datos de los artistas. Por lo tanto, lo que se almacena en cada fila es una representación parcial, limitada o simplificada de la realidad. En otras palabras, se está creando una abstracción de la realidad. En la Figura 1.1 se presentan dos ejemplos de los datos que deberían registrarse en la tabla Artistas para un grupo y un solista, los cuales son una abstracción de la realidad.

Tabla 1.1. Tabla Artistas

Artistas				
identificador	**nombre**	**año de lanzamiento**	**tipo**	**género principal**
50001	Carlos Vives	1986	Solista	Vallenato
50002	Niche	1979	Grupo	Salsa
50003	Shakira	1990	Solista	Pop
50004	Binomio de Oro de América	1976	Grupo	Vallenato
50005	J Balvin	2006	Solista	Urbano Latino

Tabla 1.2. Tabla Canciones

Canciones					
identificador	**título**	**duración**	**género**	**artista**	**álbum**
10001	La tierra del olvido	4:25	Vallenato	50001	La tierra del olvido
10002	Ojos así	3:57	Pop	50003	¿Dónde están los ladrones?
10003	Mi gente	3:05	Urbano Latino	50005	Sencillo
10004	Ambiente	4:08	Urbano Latino	50005	Vibras
10005	Cali pachanguero	4:51	Salsa	50002	No hay quinto malo
10006	La creciente	3:04	Vallenato	50004	El binomio de oro
10007	Sueños de conquista	4:02	Vallenato	50004	Por lo alto
10009	Carito	3:39	Pop	50001	Déjame entrar
10011	Una aventura	5:16	Salsa	50002	Cielo de tambores
10012	Ginza	4:39	Urbano Latino	50005	Sencillo
10013	Octavo día	4:32	Pop	50003	¿Dónde están los ladrones?
10014	Quiero verte sonreír	3:18	Pop	50001	Déjame entrar

Por otra parte, las columnas que conforman la tabla Canciones permiten almacenar los datos más importantes que se requieren para que los usuarios puedan utilizar la colección. En estas columnas, también llamadas campos o atributos, se registran el título de la canción, la duración, el género, el identificador del artista que la interpreta y el nombre del álbum en el cual está contenida. Cuando una canción no está contenida en un álbum se registra la palabra «*Sencillo*» como nombre del álbum.

Figura 1.1. Datos por registrar en la tabla Artistas para un grupo y un solista

Una banda de folk pop latino

nombre: Timbalina
lanzamiento: 2015
tipo: Grupo
género: Folk pop latino|

Un mariachi de la vieja escuela

nombre: El mariachi solitario
lanzamiento: 1965
tipo: Solista
género: Ranchera

Cada fila de la tabla Canciones corresponde a una y solamente una canción que hace parte de la colección. No tendría sentido tener dos filas con exactamente los mismos datos porque se estaría duplicando una canción, lo cual constituye una pérdida de integridad que puede generar errores en el procesamiento.

A partir de los datos de esta tabla puede decirse que la canción *La tierra del olvido* tiene una duración de *4:25*, hace parte de un álbum llamado *La tierra del olvido* y es de género *Vallenato*. El artista que interpreta esta canción es el que está registrado con el identificador *50001*. Así pues, si no existiera la tabla Artistas, no podría determinarse a quién corresponde ese identificador, pero en este caso puede observarse que el artista con identificador *50001* es el *Solista* llamado *Carlos Vives*.

Tomando como base lo que se ha presentado hasta el momento, puede argumentarse que las bases de datos son importantes porque permiten almacenar datos de forma ordenada, administrada y controlada. Sin embargo, tener un depósito de datos no genera valor a menos de que puedan utilizarse para resolver necesidades en el contexto específico en el que se diseñó e implementó esa solución tecnológica. En tal sentido, el siguiente paso introductorio en este mundo de las bases de datos es, precisamente, empezar a utilizarlas.

1.2. Uso básico de los datos almacenados en las tablas

Para utilizar los datos almacenados en una base de datos relacional es preciso especificar la necesidad que desea suplirse. Este requerimiento puede expresarse como una pregunta que espera ser respondida con los datos disponibles en las tablas. Algunos ejemplos de preguntas que especifican necesidades de datos son las siguientes:

¿Cuál es el nombre y el género principal de todos los artistas registrados en la colección?

¿Cuál es el título y la duración de todas las canciones de la colección?

¿Cuál es el título, el género y el álbum de las canciones que duran más de cuatro minutos?

Para responder a preguntas relacionadas con los datos almacenados en las tablas de una base de datos como las tres anteriores, es necesario realizar un análisis que permita identificar los cuatro elementos descritos en la Tabla 1.3.

Tabla 1.3. Elementos de análisis para definir cómo obtener los datos requeridos desde una base de datos

1. Ubicación de los datos	¿Están todos en una sola tabla? ¿Están distribuidos en varias tablas?
2. Filas necesarias	¿Todas las filas? ¿Solamente algunas filas? ¿Cuál condición deben cumplir las filas?
3. Columnas para mostrar	¿Todas las columnas? ¿Solamente algunas columnas? ¿Alguna columna generada para dar la respuesta?
4. Operaciones sobre los datos	¿Operaciones cuantitativas como contar, sumar, multiplicar o promediar? ¿Operaciones sobre textos como concatenar, separar o recortar? ¿Procesamiento de fechas? ¿Operaciones estadísticas? ¿Otras operaciones?

Siguiendo este esquema, puede definirse la forma en que se obtendrá la respuesta a las preguntas planteadas según las tablas de la base de datos. Para mostrar la forma de abordar las necesidades de datos expresadas en las tres preguntas enunciadas antes, en las siguientes páginas se presentan los análisis realizados a partir de los cuatro elementos definidos en la Tabla 1.3.

¿Cuál es el nombre y el género principal de todos los artistas registrados en la colección?

1. Ubicación de los datos	Todos los datos requeridos están en la tabla Artistas.
2. Filas necesarias	Todas las filas. No hay condiciones o filtros.
3. Columnas para mostrar	Las columnas nombre y género principal de la tabla Artistas.
4. Operaciones sobre los datos	No se requieren operaciones sobre los datos. Deben mostrarse tal y como están almacenados.

El análisis realizado para dar respuesta a la primera pregunta indica que se requiere seleccionar únicamente dos columnas de la tabla Artistas y que no es necesario aplicar filtros o condiciones porque se piden los datos de todos los artistas registrados en la colección. En la Tabla 1.4 se señala específicamente el conjunto de datos de la tabla Artistas que debería generarse como respuesta a la pregunta.

Tabla 1.4. Conjunto de datos por incluir en la respuesta

Artistas				
identificador	**nombre**	**año de lanzamiento**	**tipo**	**género principal**
50001	Carlos Vives	1986	Solista	Vallenato
50002	Niche	1979	Grupo	Salsa
50003	Shakira	1990	Solista	Pop
50004	Binomio de Oro de América	1976	Grupo	Vallenato
50005	J Balvin	2006	Solista	Urbano Latino

Las respuestas a las necesidades de datos normalmente se presentan en forma de una nueva tabla, generada a partir de datos almacenados en las tablas de la base de datos que fueron consultadas. Para este caso, la tabla resultante que da respuesta a la pregunta tiene dos columnas y cinco filas, tal y como se muestra en la Tabla 1.5.

Tabla 1.5. Conjunto de datos resultantes

nombre	**género principal**
Carlos Vives	Vallenato
Niche	Salsa
Shakira	Pop
Binomio de Oro de América	Vallenato
J Balvin	Urbano Latino

¿Cuál es el título y la duración de todas las canciones de la colección?

1. Ubicación de los datos	Todos los datos requeridos están en la tabla Canciones.
2. Filas necesarias	Todas las filas. No hay condiciones o filtros.
3. Columnas para mostrar	Las columnas título y duración de la tabla Canciones.
4. Operaciones sobre los datos	No se requieren operaciones sobre los datos. Deben mostrarse tal y como están almacenados.

Tabla 1.6. Conjunto de datos por incluir en la respuesta

Canciones					
identificador	**título**	**duración**	**género**	**artista**	**álbum**
10001	La tierra del olvido	4:25	Vallenato	50001	La tierra del olvido
10002	Ojos así	3:57	Pop	50003	¿Dónde están los ladrones?
10003	Mi gente	3:05	Urbano Latino	50005	Sencillo
10004	Ambiente	4:08	Urbano Latino	50005	Vibras
10005	Cali pachanguero	4:51	Salsa	50002	No hay quinto malo
10006	La creciente	3:04	Vallenato	50004	El binomio de oro
10007	Sueños de conquista	4:02	Vallenato	50004	Por lo alto
10009	Carito	3:39	Pop	50001	Déjame entrar
10011	Una aventura	5:16	Salsa	50002	Cielo de tambores
10012	Ginza	4:39	Urbano Latino	50005	Sencillo
10013	Octavo día	4:32	Pop	50003	¿Dónde están los ladrones?
10014	Quiero verte sonreír	3:18	Pop	50001	Déjame entrar

Tabla 1.7. Conjunto de datos resultantes

título	**duración**
La tierra del olvido	4:25
Ojos así	3:57
Mi gente	3:05
Ambiente	4:08
Cali pachanguero	4:51
La creciente	3:04
Sueños de conquista	4:02
Carito	3:39
Una aventura	5:16
Ginza	4:39
Octavo día	4:32
Quiero verte sonreír	3:18

El análisis realizado para dar respuesta a la segunda pregunta también indica que se requiere seleccionar únicamente dos columnas, pero en este caso de la tabla Canciones. Al igual que en el ejemplo anterior, tampoco es necesario aplicar filtros o condiciones porque se necesitan los datos de todas las canciones, es decir, se utilizan todas las filas de la tabla. En la Tabla 1.6 se señala específicamente el conjunto de datos de la tabla Canciones que debería generarse como respuesta.

En este caso, la tabla resultante que da respuesta a la pregunta tiene dos columnas y doce filas, tal y como se muestra en la Tabla 1.7.

¿Cuál es el título, el género y el álbum de las canciones que duran más de cuatro minutos?

1. Ubicación de los datos	Todos los datos requeridos están en la tabla Canciones.
2. Filas necesarias	El subconjunto de las filas en donde el valor almacenado en la columna duración sea mayor a cuatro minutos.
3. Columnas para mostrar	Las columnas título, género y álbum de la tabla Canciones.
4. Operaciones sobre los datos	No se requieren operaciones sobre los datos. Deben mostrarse tal y como están almacenados.

El análisis realizado para dar respuesta a la tercera pregunta plantea que, al igual que en el caso anterior, se requieren datos ubicados únicamente en la tabla Canciones. En esta oportunidad deben mostrarse los datos registrados en tres columnas de la tabla, pero con la diferencia de que es preciso utilizar los datos de la columna duración para determinar las filas que se mostrarán. Con esta acción de filtrado se excluyen las filas para las cuales la condición no se cumple.

En la Tabla 1.8 se señala el conjunto de datos de la tabla Canciones que debería generarse como respuesta a la pregunta. También se resaltan los valores almacenados en la columna duración que cumplen la condición de la pregunta.

Tabla 1.8. Conjunto de datos por incluir en la respuesta

Canciones					
identificador	**título**	**duración**	**género**	**artista**	**álbum**
10001	La tierra del olvido	4:25	Vallenato	50001	La tierra del olvido
10002	Ojos así	3:57	Pop	50003	¿Dónde están los ladrones?
10003	Mi gente	3:05	Urbano Latino	50005	Sencillo
10004	Ambiente	4:08	Urbano Latino	50005	Vibras
10005	Cali pachanguero	4:51	Salsa	50002	No hay quinto malo
10006	La creciente	3:04	Vallenato	50004	El binomio de oro
10007	Sueños de conquista	4:02	Vallenato	50004	Por lo alto
10009	Carito	3:39	Pop	50001	Déjame entrar
10011	Una aventura	5:16	Salsa	50002	Cielo de tambores
10012	Ginza	4:39	Urbano Latino	50005	Sencillo
10013	Octavo día	4:32	Pop	50003	¿Dónde están los ladrones?
10014	Quiero verte sonreír	3:18	Pop	50001	Déjame entrar

En este caso, la tabla resultante que da respuesta a la tercera pregunta tiene tres columnas y siete filas, tal y como se muestra en la Tabla 1.9 . La columna duración se utiliza para responder la pregunta, pero no se incluye en la tabla resultante. En otras palabras, no todas las columnas requeridas para realizar la consulta terminarán incluyéndose en la tabla resultante. Algunas, como en este caso, podrían utilizarse para filtrar las filas que harán parte del resultado.

Tabla 1.9. Conjunto de datos resultantes

título	género	álbum
La tierra del olvido	Vallenato	La tierra del olvido
Ambiente	Urbano Latino	Vibras
Cali pachanguero	Salsa	No hay quinto malo
Sueños de conquista	Vallenato	Por lo alto
Una aventura	Salsa	Cielo de tambores
Ginza	Urbano Latino	Sencillo
Octavo día	Pop	¿Dónde están los ladrones?

Para satisfacer las necesidades especificadas en las tres preguntas anteriores, fue suficiente con presentar los datos tal y como están almacenados en las tablas. Este uso es muy frecuente, pero muy básico. Lo que ocurre normalmente es que se necesita realizar alguna operación para mostrar datos derivados o calculados a partir de los que están almacenados en las tablas. Para ilustrar se abordará la siguiente pregunta:

¿Cuál es el nombre, el tipo, el género principal, el año de lanzamiento y los años de actividad que cumplen en el 2021 todos los artistas pop que fueron lanzados antes del año 2000?

1. Ubicación de los datos	Todos los datos están en la tabla Artistas.
2. Filas necesarias	Las filas en donde la columna año de lanzamiento sea menor a 2000 y la columna género principal tenga la palabra «Pop».
3. Columnas para mostrar	Las columnas nombre, tipo, género principal y año de lanzamiento de la tabla Artistas. En el resultado también debe mostrarse una columna derivada de la columna año de lanzamiento para mostrar los años de actividad al 2021.
4. Operaciones sobre los datos	Debe restarse a 2021 el valor almacenado en la columna año de lanzamiento y presentarse en la tabla resultante como una columna derivada llamada años de actividad al 2021.

En el análisis para responder la pregunta se incorporan algunos elementos que empiezan a mostrar nuevas posibilidades en el uso de los datos almacenados en una base de datos.

En esta ocasión también se necesitan únicamente los datos de una sola tabla, la tabla Artistas. Sin embargo, a diferencia del caso anterior, aquí se tiene una condición compuesta que deben cumplir las filas que se incluirán en la respuesta y que está asociada a los valores almacenados en las columnas año de lanzamiento y género principal. Para que una fila sea incluida en la respuesta, debe cumplir con las dos condiciones.

En la Tabla 1.10 se puede observar el conjunto de datos de la tabla Artistas que hará parte de la respuesta. También se resaltan en color verde los valores almacenados en las columnas año de lanzamiento y género principal que cumplen la condición requerida. Por otra parte, en amarillo se destacan los valores que solo cumplen parcialmente la condición y, por ende, no hacen parte de la respuesta.

La condición compuesta no es lo único diferente. Además, se destaca el hecho de que se requiere generar datos a partir de los que están almacenados en la tabla. Específicamente, en el resultado de esta consulta deberá mostrarse una columna que no existe en las tablas de la base de datos. Ahora bien, dicho requerimiento no quiere decir que se modifica la estructura de las tablas o que los datos almacenados en estas se cambian por nuevos valores; simplemente, en la tabla resultante, que es independiente de la tabla de origen, se agrega la columna necesaria para mostrar el dato calculado o derivado.

Tabla 1.10. Conjunto de datos por incluir en la respuesta

Artistas				
identificador	**nombre**	**año de lanzamiento**	**tipo**	**género principal**
50001	Carlos Vives	1986	Solista	Vallenato
50002	Niche	1979	Grupo	Salsa
50003	Shakira	1990	Solista	Pop
50004	Binomio de Oro de América	1976	Grupo	Vallenato
50005	J Balvin	2006	Solista	Urbano Latino

Además, puede observarse que el orden de aparición de las columnas del resultado es diferente al que tiene la tabla de origen, lo cual se hace para cumplir con precisión milimétrica lo especificado en la pregunta. Como resultado se tiene la Tabla 1.11, la cual está compuesta de cinco columnas y una fila.

Tabla 1.11. Conjunto de datos resultantes

nombre	**tipo**	**género principal**	**año de lanzamiento**	**años de actividad al 2021**
Shakira	Solista	Pop	1990	31

El uso de las bases de datos puede ir mucho más allá de la simple resolución de preguntas utilizando una única tabla. En el capítulo siguiente se muestra que las posibilidades para

usar y aprovechar los datos se expanden significativamente al trabajar con el poderoso lenguaje estructurado de consulta o SQL (de su nombre en inglés: *Structured Query Language*).

1.3. Aprendizajes más importantes del capítulo 1

Lo tratado en este capítulo debería haberle permitido aprender lo expresado en la siguiente lista de ideas:

- Las tablas son las estructuras básicas de almacenamiento que se utilizan en las bases de datos relacionales.
- Con las tablas se representan abstracciones de entidades tangibles, con existencia física en el mundo real, o intangibles, en forma de columnas y filas.
- En las columnas de una tabla se almacenan los datos que representan las características más importantes de una entidad de un contexto.
- Cada fila de una tabla representa una ocurrencia o instancia de la entidad.
- Es posible que varias tablas tengan columnas similares, con el mismo significado, lo cual permite relacionar los datos de dichas tablas.
- Los datos almacenados en las tablas pueden y deben ser utilizados para satisfacer necesidades de datos en un momento específico.
- Para obtener los datos requeridos debe identificarse en cuál tabla están almacenados, determinar si se requieren todas las columnas o solo algunas, definir si se necesitan todas las filas o un subconjunto de estas y, finalmente, establecer si es preciso realizar operaciones para derivar nuevos datos que permitan satisfacer plenamente la solicitud.

1.4. Actividades de aplicación para evidenciar lo aprendido

1. Proponga dos tablas que puedan utilizarse para almacenar los datos más importantes en las situaciones que se enuncian a continuación:

    - Una cadena de hoteles especializada en turismo de aventura.
    - Un club de aficionados a los deportes electrónicos o *e-sports*.
    - Una granja que produce hortalizas orgánicas.
    - Un restaurante de comidas saludables que solo vende a domicilio.
    - Una biblioteca comunitaria.
    - Un torneo profesional de un deporte de equipo.

2. Ejemplifique la forma en que se almacenan los datos en las tablas propuestas para dos de los casos de la actividad 1, mostrando al menos cinco filas de datos por cada tabla.
3. Especifique en forma de pregunta cuatro necesidades que puedan ser satisfechas con las tablas obtenidas en la actividad 2, dos por cada caso.

4. Realice el análisis para dar respuesta a las preguntas planteadas en la actividad 3 y muestre la tabla resultante. Debe aplicar el esquema de cuatro elementos utilizado en el capítulo (1. Ubicación de los datos, 2. Filas necesarias, 3. Columnas para mostrar y 4. Operaciones sobre los datos).
5. Utilizando la tabla Artistas, determine y argumente si el conjunto de datos presentado en la tabla siguiente es la respuesta a la pregunta «¿Cuál es el nombre y el género de los solistas que al año 2000 tenían más de diez años de vida artística?»:

nombre	género principal
Carlos Vives	Vallenato
Shakira	Pop

6. Realice el análisis para dar respuesta a la pregunta «¿Cuál es el nombre y la duración de las canciones cuya duración está en el rango de tres a cinco minutos?».
7. Utilizando la tabla Canciones, determine y argumente si el conjunto de datos presentado es la respuesta a la pregunta «¿Cuál es el nombre de las canciones que, siendo de género diferente al vallenato, tienen una duración superior o igual a la canción con mayor duración del género vallenato?»:

título	álbum
La tierra del olvido	La tierra del olvido
Cali pachanguero	No hay quinto malo
Una aventura	Cielo de tambores
Octavo día	¿Dónde están los ladrones?

8. Realice el análisis para dar respuesta a la pregunta «¿Cuál es el nombre y el tipo de los artistas que fueron lanzados en algún año posterior a la caída del muro de Berlín y anterior al ataque de las torres gemelas?».
9. Tomando como base las tablas Artistas y Canciones presentadas en este capítulo, escriba dos preguntas diferentes que permitan obtener el conjunto de datos resultante mostrado en la siguiente tabla:

identificador	título	duración	género
10002	Ojos así	3:57	Pop
10003	Mi gente	3:05	Urbano Latino
10004	Ambiente	4:08	Urbano Latino
10012	Ginza	4:39	Urbano Latino
10013	Octavo día	4:32	Pop

10. ¿Cuál es su opinión sobre las implicaciones éticas del trabajo con bases de datos?

Capítulo 2

Consultas básicas y filtrado de filas

Resultados de aprendizaje

- *Identifica los elementos básicos de una consulta escrita en el lenguaje SQL.*
- *Construye consultas desde una única tabla, en las que se filtran filas de la tabla de origen, se excluyen filas duplicadas y se ordenan las filas resultantes.*

Las bases de datos relacionales se construyen para que los usuarios, personas o aplicaciones software puedan consultar lo que está almacenado en ellas de forma sencilla, rápida y precisa. Para esto existe el lenguaje estructurado de consulta o SQL (sigla de la expresión en inglés *Structured Query Language*): un lenguaje de dominio específico que fue creado con base en lo planteado en el modelo relacional de bases de datos que propuso Edgar Frank Codd en 1970, específicamente con la aplicación del álgebra y el cálculo relacional. Luego, el SQL se convirtió en un estándar internacional, y en la actualidad está presente en la mayoría de los DBMS comerciales.

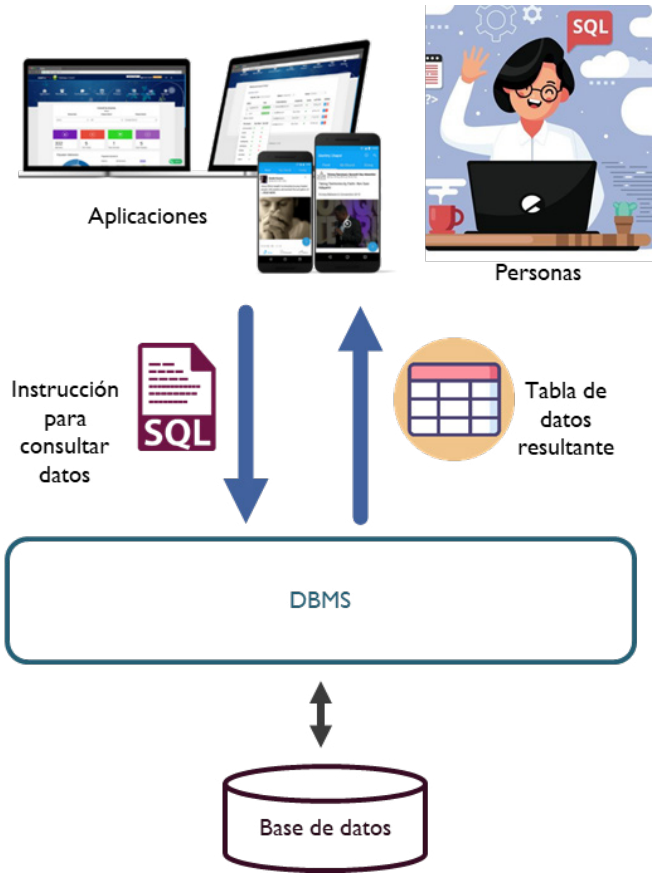
En este segundo capítulo se abordan los elementos para dar los primeros pasos en el uso del lenguaje y resolver necesidades de datos con consultas básicas realizadas únicamente sobre una tabla. En tal sentido, los resultados de aprendizaje están centrados en trabajar con la sentencia `SELECT`, la cual permite, de forma muy sencilla, solicitarle al DBMS la selección de un conjunto de datos requeridos, lo que dará como respuesta una tabla.

A diferencia de los lenguajes de programación de propósito general como C, C++, Java, C# o Python, los cuales tienen elementos que permiten generar aplicaciones para resolver problemas de diferente naturaleza, el SQL tiene un pequeño conjunto de poderosas operaciones que son muy sencillas de entender y utilizar. Todas las operaciones de SQL se enfocan exclusivamente en el ámbito de las bases de datos relacionales, y su carácter declarativo

a alto nivel, contrario a un lenguaje procedimental, permite que la programación sea muy cercana a una especificación de lo que debe hacerse sin que sea necesario entrar en detalles sobre cómo lograrlo. Esto es posible porque el DBMS es el encargado de ejecutar las instrucciones escritas en SQL, lo cual incluye la responsabilidad de definir la mejor forma de llevarlas a cabo.

Por ejemplo, una instrucción en SQL para realizar una consulta podría solamente especificar los datos, es decir, las columnas requeridas, la tabla en donde están dichas columnas y, si es el caso, la condición que deben cumplir las filas para ser incluidas en la tabla resultante. El DBMS tomará esa instrucción y determinará, entonces, si es necesario recorrer una a una las filas de la tabla para identificar aquellas en las que se almacenan ciertos datos o si puede utilizar alguna estructura de datos que permita la ejecución de procedimientos para hacer más eficiente el proceso. Esta forma de operación se representa de forma general en la Figura 2.1.

Figura 2.1. Esquema general de operación de un sistema de bases de datos



Las operaciones del lenguaje SQL se agrupan en tres categorías o en tres sublenguajes de acuerdo con la función general que cumplen. El primero reúne las operaciones de creación de los objetos de la base de datos y se conoce como lenguaje de definición de datos

o DDL (sigla de la expresión en inglés *Data Definition Language*). El segundo, denominado lenguaje de manipulación de datos o DML (sigla de la expresión en inglés *Data Manipulation Language*), está conformado por las operaciones con las que se procesan los datos de las tablas para, por ejemplo, insertar nuevas filas, consultarlos, modificarlos y eliminarlos. El tercero, conocido como lenguaje de control de datos o DCL (sigla de la expresión en inglés *Data Control Language*), contiene las operaciones para establecer y controlar quién puede acceder a la base de datos, a los objetos que la componen y a los datos almacenados.

2.1. Consultas básicas sobre una única tabla

Para iniciar el aprendizaje del SQL se continuará trabajando con la base de datos de la colección de canciones de una plataforma que le ofrece al público la posibilidad de escucharlas gratuitamente o pagando una tarifa de suscripción mensual. Sin embargo, no se utilizarán las mismas tablas porque, al igual que sucede en situaciones reales, el diseño original fue evaluado y mejorado, con lo cual se generó una nueva versión de la base de datos que contempla los elementos descritos en los siguientes párrafos.

Los datos esenciales para administrar la colección siguen siendo los correspondientes a las canciones y a los artistas que las interpretan. Por consiguiente, las tablas `Artistas` y `Canciones` se mantienen, pero adicionándoles algunas columnas o modificando el contenido de otras para reducir la redundancia. Además, se introdujeron dos nuevas tablas para almacenar los datos de los `Álbumes` en los cuales estuvieron incluidas las canciones en el momento del lanzamiento y los `Géneros` musicales, que permiten clasificar tanto a las `Canciones` como a los `Artistas`.

En la tabla `Artistas` se agregó una columna para registrar el año de retiro, es decir, cuando el solista deja la actividad musical profesional o cuando el grupo se desintegra. Esta nueva columna permite valores nulos (`NULL`) para los artistas que no se han retirado, es decir, que están activos actualmente. También se cambió el contenido de la columna `género principal` por un número que corresponde al identificador del género según lo registrado en la nueva tabla `Géneros`.

Tabla 2.1. Nueva versión de la tabla `Artistas`

Artistas					
identificador	**nombre**	**año de lanzamiento**	**año de retiro**	**tipo**	**género principal**
50001	Carlos Vives	1986	*NULL*	Solista	1
50002	Niche	1979	*NULL*	Grupo	2
50003	Shakira	1990	*NULL*	Solista	3
50004	Binomio de Oro de América	1976	*NULL*	Grupo	1
50005	J Balvin	2006	*NULL*	Solista	4

Tabla 2.2. Nueva versión de la tabla `Canciones`

Canciones						
identificador	**título**	**duración**	**género**	**idioma**	**artista principal**	**álbum original**
10001	La tierra del olvido	00:04:25	1	Español	50001	900001
10002	Ojos así	00:03:57	3	Español	50003	900002
10003	Mi gente	00:03:05	4	Español	50005	*NULL*
10004	Ambiente	00:04:08	4	Español	50005	900004
10005	Cali pachanguero	00:04:51	2	Español	50002	900005
10006	La creciente	00:03:04	1	Español	50004	900006
10007	Sueños de conquista	00:04:02	1	Español	50004	900008
10009	Carito	00:03:39	3	Español	50001	900009
10011	Una aventura	00:05:16	2	Español	50002	900011
10012	Ginza	00:04:39	4	Español	50005	*NULL*
10013	Octavo día	00:04:32	3	Español	50003	900002
10014	Quiero verte sonreír	00:03:18	3	Español	50001	900009

Tabla 2.3. Nueva tabla `Géneros`

Géneros	
identificador	**nombre**
1	Vallenato
2	Salsa
3	Pop
4	Urbano Latino

Tabla 2.4. Nueva tabla `Álbumes`

Álbumes			
identificador	**título**	**fecha de lanzamiento**	**sello discográfico**
900001	La tierra del olvido	25/07/1995	EMI Latin
900002	¿Dónde están los ladrones?	29/09/1998	Sony Music
900004	Vibras	25/05/2018	Universal
900005	No hay quinto malo	*NULL*	Internacional Records
900006	El binomio de oro	03/11/1976	Codiscos
900008	Por lo alto	02/11/1976	Codiscos
900009	Déjame entrar	06/11/2001	EMI Latin
900011	Cielo de tambores	20/12/1990	Codiscos

En la tabla `Canciones` se agregó una columna para registrar el idioma principal de la canción. También se cambiaron los nombres de las columnas `artista` y `álbum` por `artista principal` y `álbum original` respectivamente. En esta última columna, en particular, se modificó el contenido para registrar el identificador del álbum en el que fue incluida la

canción en el momento de su lanzamiento, de acuerdo con lo registrado en la nueva tabla `Álbumes`. Esta columna permite valores nulos (`NULL`) para que puedan registrarse las canciones que fueron lanzadas sin hacer parte de algún álbum. En la Tabla 2.1 se presenta la nueva versión de la tabla `Artistas`, y en la Tabla 2.2, la nueva versión de la tabla `Canciones`. Por su parte, la Tabla 2.3 corresponde a la nueva tabla `Géneros`, y la Tabla 2.4 es la nueva tabla `Álbumes`.

Teniendo la claridad sobre los componentes de la nueva versión de la base de datos, se iniciará el aprendizaje del SQL utilizando los cuatro elementos de análisis planteados en el capítulo I y su equivalencia en el lenguaje. Para esto se propone la siguiente pregunta que especifica una necesidad de datos que debe ser satisfecha:

¿Cuál es el identificador, el nombre, el año de lanzamiento, el año de retiro, el tipo y el género principal de todos los artistas?

1. Ubicación de los datos	Todos los datos requeridos están en la tabla `Artistas`.
2. Filas necesarias	Todas las filas. No hay condiciones o filtros.
3. Columnas para mostrar	Las columnas `identificador`, `nombre`, `año de lanzamiento`, `año de retiro`, `tipo` y `género principal` de la tabla `Artistas`. Son todas las columnas de la tabla.
4. Operaciones sobre los datos	No se requieren operaciones sobre los datos. Deben mostrarse tal y como están almacenados.

Este análisis presenta el escenario más sencillo de consulta de los datos de una tabla con SQL. Aquí deben obtenerse todas las columnas y todas las filas; no se establecen filtros ni operaciones sobre los datos. En tal sentido, lo siguiente será escribir la consulta que deberá ejecutar el DBMS para retornar los datos requeridos. Es preciso mencionar que este esquema de análisis de cuatro elementos se va interiorizando con la práctica y su elaboración escrita se vuelve opcional.

Vale recordar que en SQL se expresa lo que debe realizar el DBMS sin necesidad de definir y programar la forma para lograrlo. Por consiguiente, para este caso en concreto se le indica al DBMS que entregue una tabla conformada con los datos almacenados en todas las columnas y todas las filas de la tabla `Artistas`. El código SQL para lograr este resultado se presenta en el *Script* 2.1, y el resultado de esta consulta es la Tabla 2.5.

***Script* 2.1**

```
SELECT identificador,
       nombre,
       "año de lanzamiento",
       "año de retiro",
       tipo,
       "género principal"
FROM Artistas
```

Tabla 2.5. Resultado de la consulta del *Script* 2.1

identificador	nombre	año de lanzamiento	año de retiro	tipo	género principal
50001	Carlos Vives	1986	*NULL*	Solista	1
50002	Niche	1979	*NULL*	Grupo	2
50003	Shakira	1990	*NULL*	Solista	3
50004	Binomio de Oro de América	1976	*NULL*	Grupo	1
50005	J Balvin	2006	*NULL*	Solista	4

En el *Script* 2.1 puede verse que una consulta SQL tiene una estructura y una sintaxis sencillas. En este caso se utilizan las palabras `SELECT` y `FROM` para comunicarle al DBMS dos puntos necesarios para obtener la respuesta deseada. El primero especifica las columnas que deben incluirse en la tabla resultante, y el segundo define la tabla que contiene esas columnas. Al hacer un paralelismo con los cuatro elementos de análisis que se han venido utilizando desde el capítulo 1, puede verse que el elemento 3, *columnas para mostrar*, equivale en SQL a la palabra `SELECT`, y el elemento 1, *ubicación de los datos*, corresponde a la palabra `FROM` del SQL.

Los nombres de las columnas que tienen espacios en blanco están entre comillas, cumpliendo las reglas de sintaxis del SQL implementado en el DBMS, esto es, PostgreSQL. En otros DBMS se utilizan otros símbolos para indicar que el nombre de una columna incluye espacios en blanco; por ejemplo, SQL Server utiliza corchetes: [ ].

El lenguaje SQL no es sensible a las mayúsculas y tampoco interpreta los saltos de línea como el fin de una línea de código o de una instrucción. En tal sentido, para el DBMS el código del *Script* 2.2 es exactamente el mismo que el código del *Script* 2.1, mientras que para el observador humano la diferencia de estas dos versiones está dada por aspectos estéticos y de presentación que redundan en la facilidad de lectura y comprensión.

***Script* 2.2**

```
select identiFICADOR, nombre,
       "año de lanzamiento", "año de retiro",
       tipo, "género principal"
from  ARTISTAS
```

En el SQL puede usarse el carácter * para hacer referencia a todas las columnas de una tabla. Por lo tanto, el *Script* 2.3 genera el mismo resultado que el *Script* 2.1. Al respecto, cabe anotar que entre las diversas ventajas que implica usar dicho carácter se encuentra que permite reducir significativamente el tamaño código, y también que hace que la misma consulta funcione independientemente de que se agreguen o eliminen columnas de la tabla. Sin embargo, este recurso tiene a su vez limitantes, como que no puede alterarse el orden de las columnas en la tabla resultante.

***Script* 2.3**

```
SELECT* FROM Artistas
```

En las consultas en las que se declaran de forma explícita todas las columnas de la tabla consultada, se tiene la opción de cambiar el orden en que estas aparecerán en la tabla resultante. Para esto, deben ubicarse en el orden deseado dentro de la lista separada por comas después de la palabra `SELECT`, tal como se ejemplifica en el *Script* 2.4.

***Script* 2.4**

```
SELECT identificador,
       nombre,
       tipo,
       "género principal",
       "año de lanzamiento",
       "año de retiro"
FROM Artistas
```

El resultado de ejecutar el *Script* 2.4 contendrá los mismos datos obtenidos con el *Script* 2.1, pero con una estructura distinta, tal y como se puede ver en la Tabla 2.6.

Tabla 2.6. Resultado de la consulta del *Script* 2.4

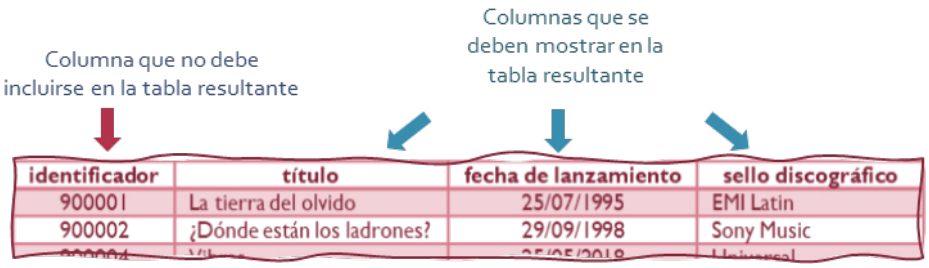
identificador	nombre	tipo	género principal	año de lanzamiento	año de retiro
50001	Carlos Vives	Solista	1	1986	*NULL*
50002	Niche	Grupo	2	1979	*NULL*
50003	Shakira	Solista	3	1990	*NULL*
50004	Binomio de Oro de América	Grupo	1	1976	*NULL*
50005	J Balvin	Solista	4	2006	*NULL*

Para resolver las necesidades de datos, es muy común que las consultas se limiten a un subconjunto de columnas. La declaración explícita, después de la palabra `SELECT`, de las columnas por incluir también permite que se especifiquen solamente aquellas columnas requeridas, evitando generar resultados con datos innecesarios. Para ilustrar esto se propone la necesidad expresada en la siguiente pregunta:

¿Cuál es el título, la fecha de lanzamiento y el sello discográfico de todos los álbumes?

Al analizar la pregunta elaborando mentalmente el esquema de cuatro elementos, se identifica que los datos requeridos están almacenados en la tabla `Álbumes` y que se deben mostrar tres columnas, tal y como se indica en la Figura 2.2.

Figura 2.2. Columnas requeridas para responder la pregunta



Específicamente, se requieren las columnas `título`, `fecha de lanzamiento` y `sello discográfico` de la tabla `Álbumes`. No se necesitan filtros ni operaciones sobre los datos para conformar la tabla resultante. De esta manera, entonces, puede proponerse la consulta SQL del *Script* 2.5, en la que se incluyen solamente los nombres de las tres columnas solicitadas luego de la palabra `SELECT` y separados por coma. La Tabla 2.7 presenta el resultado de ejecutar la consulta.

***Script* 2.5**

```
SELECT título,
       "fecha de lanzamiento",
       "sello discográfico"
FROM Álbumes
```

Tabla 2.7. Resultado de la consulta del *Script* 2.5

título	fecha de lanzamiento	sello discográfico
La tierra del olvido	25/07/1995	EMI Latin
¿Dónde están los ladrones?	29/09/1998	Sony Music
Vibras	25/05/2018	Universal
No hay quinto malo	*NULL*	Internacional Records
El binomio de oro	03/11/1976	Codiscos
Por lo alto	02/11/1976	Codiscos
Déjame entrar	06/11/2001	EMI Latin
Cielo de tambores	20/12/1990	Codiscos

Una posibilidad que ofrece el SQL para mejorar la presentación de los resultados, precisar el contenido que se muestra o eliminar ambigüedades es asignarles un nombre alternativo

o un alias a las columnas que se van a mostrar. Por ejemplo, si se quiere mejorar la presentación del resultado expuesto en la Tabla 2.7, en lugar de obtener una columna llamada `título`, se podría utilizar la palabra `álbum`; asimismo, en lugar de `fecha de lanzamiento`, el encabezado sería `lanzamiento`, y en lugar de `sello discográfico` se encontraría la palabra `disquera`. Para cumplir con este requisito de presentación, la consulta SQL debe modificarse tal como se muestra en el *Script* 2.6.

Script 2.6

```
SELECT título AS álbum,
       "fecha de lanzamiento" AS lanzamiento,
       "sello discográfico" disquera
FROM Álbumes
```

Puede observarse que hay dos formas de asignar el alias. La primera es explícita, en la que se utiliza la palabra AS seguida del nombre que va a tener la columna en la tabla resultante, como es el caso de los alias `álbum` y `lanzamiento`. En la segunda, implícita, simplemente se escribe el alias luego del nombre de la columna, como es el caso de la columna `sello discográfico` y el alias `disquera`.

Al asignar un alias no se modifica la tabla de la base de datos; lo que se modifica es la tabla resultante que genera el DBMS al ejecutar la consulta. En la Tabla 2.8 se presenta el resultado de la consulta con los alias asignados.

Tabla 2.8. Resultado de la consulta del *Script* 2.6

álbum	lanzamiento	disquera
La tierra del olvido	25/07/1995	EMI Latin
¿Dónde están los ladrones?	29/09/1998	Sony Music
Vibras	25/05/2018	Universal
No hay quinto malo	*NULL*	Internacional Records
El binomio de oro	03/11/1976	Codiscos
Por lo alto	02/11/1976	Codiscos
Déjame entrar	06/11/2001	EMI Latin
Cielo de tambores	20/12/1990	Codiscos

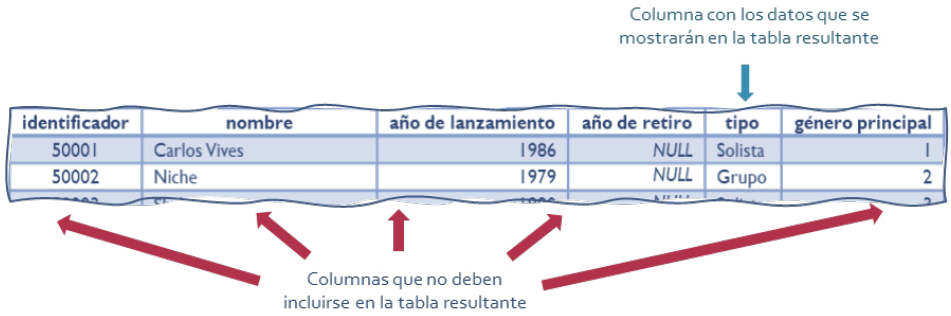
2.2. Exclusión de filas duplicadas y ordenamiento de resultados

En algunas ocasiones las consultas parecen realmente sencillas, pero tienen algunos detalles que deben cuidarse para evitar errores. Uno de estos casos puede darse al resolver la siguiente pregunta:

¿Cuáles son los tipos de artistas que están registrados en la base de datos?

Para responder la pregunta se identifica que los datos requeridos están almacenados en la tabla `Artistas` y solamente se requieren datos almacenados en una de las columnas de esta tabla, tal y como se presenta en la Figura 2.3.

Figura 2.3. Columna requerida para responder la pregunta



En este sentido, lo primero que puede pensarse es escribir una consulta para obtener la columna `tipo` de la tabla `Artistas`, como se muestra en el *Script* 2.7. Al ejecutarla, el DBMS genera como resultado la Tabla 2.9. Esta última tiene la columna requerida con los datos almacenados en todas las filas de la tabla.

Script 2.7

```
SELECT tipo
FROM Artistas
```

Tabla 2.9. Resultado de la consulta del *Script* 2.7

tipo
Solista
Grupo
Solista
Grupo
Solista

No obstante, este resultado no puede aceptarse como respuesta a la pregunta porque hay filas duplicadas. Del análisis directo sobre los datos, puede identificarse que la respuesta correcta a la pregunta debería tener únicamente dos filas: una para el tipo de artista denominado *«Grupo»* y otra para el tipo *«Solista»*. Para estos casos, el SQL proporciona una forma de eliminar las filas duplicadas y obtener el resultado requerido. Específicamente, debe

utilizarse la palabra `DISTINCT`, tal y como se presenta en el *Script* 2.8, lo que genera como resultado la Tabla 2.10.

***Script* 2.8**

```
SELECT DISTINCT
       tipo
FROM Artistas
```

Tabla 2.10. Resultado de la consulta del *Script* 2.8

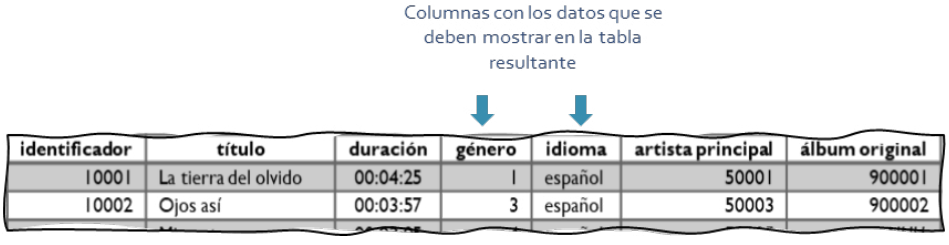
tipo
Grupo
Solista

La palabra `DISTINCT` también puede utilizarse en consultas que muestren más de una columna. En este caso, el DBMS dará como resultado únicamente las filas con combinaciones distintas. Para ilustrar esto se abordará la solución a la siguiente pregunta:

¿Cuáles son los idiomas de las canciones de la colección en cada género?

La respuesta esperada es una tabla con las columnas `género` e `idioma` de la tabla `Canciones`, tal y como se presenta en la Figura 2.4. Las filas esperadas son combinaciones distintas de géneros e idiomas. Por ejemplo, si del género 3 hay cinco canciones, tres en español y dos en inglés, la tabla resultante debería incluir dos filas con las combinaciones (3, español) y (3, inglés).

Figura 2.4. Columnas requeridas para responder la pregunta



La consulta SQL para responder la pregunta se presenta en el *Script* 2.9, con la cual se obtiene el resultado mostrado en la Tabla 2.11.

***Script* 2.9**

```
SELECT DISTINCT
       género,
       idioma
FROM Canciones
```

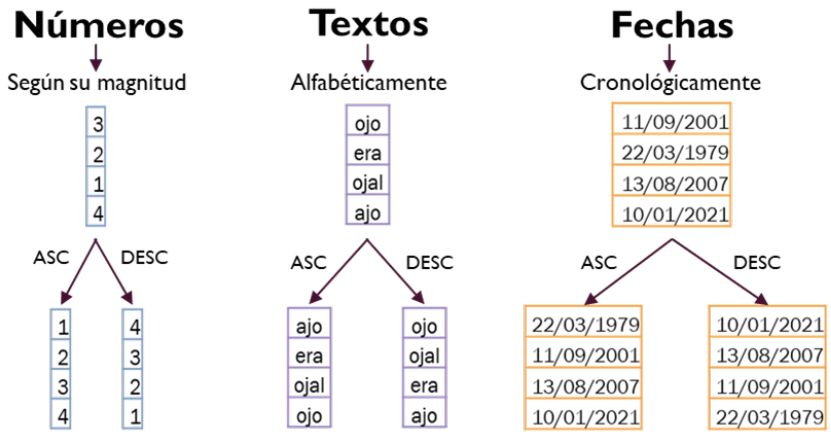
Tabla 2.11. Resultado de la consulta del *Script* 2.9

género	idioma
1	español
2	español
3	español
4	español

Frecuentemente deben mostrarse los resultados de una consulta en un orden particular definido por algún criterio basado en los datos almacenados en una o en varias columnas. Esto es algo para tener en cuenta en toda consulta y así facilitar el uso de los resultados que se obtengan de la ejecución.

El SQL incluye una instrucción que permite indicarle al DBMS el criterio de ordenamiento por utilizar, es decir, la columna o las columnas que debe tomar como base para esta operación, y el tipo de ordenamiento: si es ascendente o descendente. Esta operación tiene un comportamiento que depende del tipo de dato de las columnas utilizadas como criterio de ordenamiento. Los números se ordenan por su magnitud; los textos, alfabéticamente, y las fechas, cronológicamente. En la Figura 2.5 se describe el ordenamiento en estos casos.

Figura 2.5. Comportamiento del ordenamiento en algunas familias de tipos de datos



Para mostrar esta opción de ordenamiento se tomará como base la consulta del *Script* 2.6, con la cual se responde la pregunta sobre el título, la fecha de lanzamiento y el sello

discográfico de los álbumes registrados en la colección. En esa consulta se utilizaron alias de los nombres de las columnas para mejorar la presentación y facilitar la comprensión de los datos contenidos en la tabla resultante. Sin embargo, los datos se presentan sin un orden definido; resultaría mucho mejor si los datos aparecen ordenados, por ejemplo, por el título del álbum de forma alfabética. La consulta SQL que cumpliría este requisito se presenta en el *Script* 2.10.

***Script* 2.10**

```
SELECT título AS álbum,
       "fecha de lanzamiento" AS lanzamiento,
       "sello discográfico" disquera
FROM Álbumes
ORDER BY álbum ASC
```

Tabla 2.12. Resultado de la ejecución del *Script* 2.10

álbum	lanzamiento	disquera
¿Dónde están los ladrones?	29/09/1998	Sony Music
Cielo de tambores	20/12/1990	Codiscos
Déjame entrar	06/11/2001	EMI Latin
El binomio de oro	03/11/1976	Codiscos
La tierra del olvido	25/07/1995	EMI Latin
No hay quinto malo	*NULL*	Internacional Records
Por lo alto	02/11/1976	Codiscos
Vibras	25/05/2018	Universal

La expresión `ORDER BY` le indica al DBMS que el resultado de la consulta debe presentarse siguiendo un criterio de ordenamiento. La sintaxis del lenguaje especifica que, luego de la expresión `ORDER BY`, deben definirse las columnas que se utilizarán como criterios de ordenamiento. A su vez, por cada columna debe especificarse si el orden es ascendente (`ASC`) o descendente (`DESC`). Si esto no se define, el DBMS ordenará de forma ascendente. El resultado de la nueva consulta se presenta en la Tabla 2.12.

Teniendo en cuenta que el comportamiento predeterminado es ordenar de manera ascendente por aquellas columnas que aparecen luego de la expresión `ORDER BY`, la consulta del *Script* 2.11 es equivalente a la del *Script* 2.10. En esta oportunidad se utiliza el nombre de la columna `título` tal y como está en la tabla `Álbumes` en lugar del alias `álbum`.

En el criterio de ordenamiento pueden aparecer varias columnas. También es posible ordenar por los valores de columnas derivadas o calculadas a partir de los datos almacenados en la tabla. Para mostrar esto se propone la siguiente pregunta:

¿Cuál es el nombre, el tipo, el año de lanzamiento y los años de actividad al 2021?

El resultado debe ordenarse por tipo y luego, descendentemente, por los años de actividad.

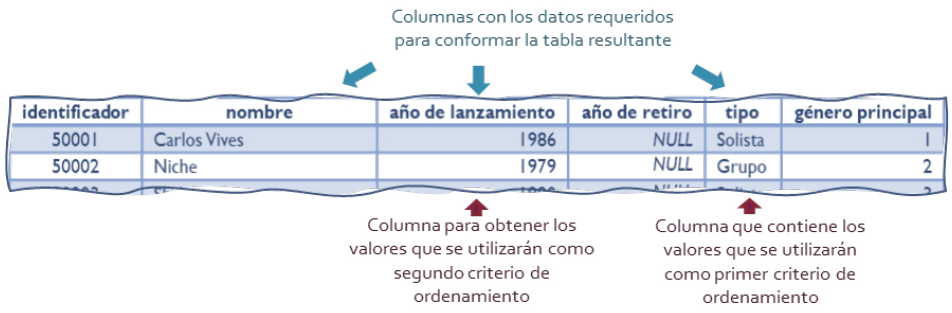
***Script* 2.11**

```
SELECT título álbum,
       "fecha de lanzamiento" lanzamiento,
       "sello discográfico" disquera
FROM Álbumes
ORDER BY título
```

En esta pregunta se observan tres hechos diferentes con respecto a las necesidades de datos abordadas hasta ahora. En primer lugar, se plantea que debe obtenerse una columna que no está en la tabla original, es decir, una columna derivada. También se especifica que debe ordenarse el resultado tomando como criterio los datos de dos columnas y que cada columna tiene un tipo de ordenamiento distinto.

Una forma de resolver la pregunta planteada sería con la consulta presentada en el *Script* 2.12. En este código se observa una expresión que permite obtener la cantidad de años de actividad de cada artista utilizando como base la columna `año de lanzamiento`. Con la operación aritmética *2021* – `"año de lanzamiento"` se genera una nueva columna en la tabla resultante cuyo nombre es `años de actividad al 2021`.

Figura 2.6. Columnas requeridas para responder la pregunta



La consulta permite obtener el conjunto de datos resultante que se presenta en la Tabla 2.13, los cuales están ordenados alfabéticamente de manera ascendente según la columna `tipo`, es decir, primero los de tipo *Grupo* y luego los de tipo *Solista*. Posteriormente, se encuentran ordenados de manera descendente en función de los años de actividad al 2021. Es

así como aparece primero el grupo *Binomio de Oro de América*, pues tiene *45* años, y luego el grupo *Niche*, que tiene *42* años.

***Script* 2.12**

```
SELECT nombre,
       tipo,
       "año de lanzamiento",
       2021 - "año de lanzamiento" AS "años de actividad al 2021"
FROM Artistas
ORDER BY tipo,
       "años de actividad al 2021" DESC
```

Tabla 2.13. Resultado de la ejecución del *Script* 2.12

nombre	tipo	año de lanzamiento	años de actividad al 2021
Binomio de Oro de América	Grupo	1976	45
Niche	Grupo	1979	42
Carlos Vives	Solista	1986	35
Shakira	Solista	1990	31
J Balvin	Solista	2006	15

2.3. Consultas con filtrado de filas

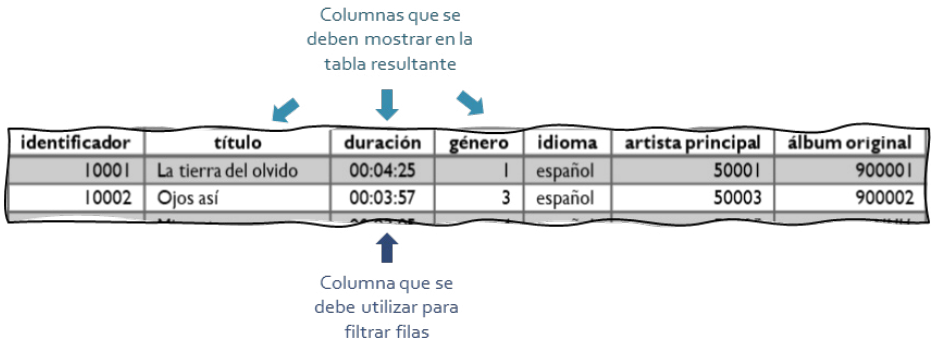
Usando únicamente las palabras `SELECT` y `FROM`, la tabla resultante creada por el DBMS incluirá todas las filas de la tabla especificada en la cláusula `FROM`. Sin embargo, es muy común que el comportamiento deseado requiera la obtención de un subconjunto de las filas, incluyendo en el resultado las que cumplan uno o varios criterios de filtrado. Para abordar este escenario se propone la siguiente pregunta:

¿Cuál es el título, el género y la duración de las canciones que duran de cuatro a cinco minutos?

Para resolver esta pregunta se requieren datos de la tabla `Canciones` (`FROM Canciones`); específicamente, el `título`, el `género` y la `duración` (`SELECT título, género, duración`), pero únicamente de las canciones (filas) con duración de entre cuatro y cinco minutos, tal y como se presenta en la Figura 2.7. La consulta SQL para responder la pregunta se presenta en el *Script* 2.13.

Los criterios o condiciones de filtrado de filas en la tabla resultante se expresan con la cláusula `WHERE`. Para esto puede utilizarse cualquier expresión que al evaluarse genere un valor booleano, es decir, «*Verdadero*» (TRUE) o «*Falso*» (FALSE). Esta expresión puede construirse a partir de valores, columnas y operadores.

Figura 2.7. Columnas requeridas para responder la pregunta



***Script* 2.13**

```
SELECT título,
       género,
       duración
FROM Canciones
WHERE duración >= '00:04:00' AND
       duración <= '00:05:00'
```

La expresión se evalúa por cada fila de la tabla, de modo que la tabla resultante contenga únicamente las filas que cumplen la condición porque el resultado de la evaluación es *«Verdadero»*. En otras palabras, el proceso que ejecuta el DBMS inicia tomando cada fila de la tabla y evaluando si cumple la condición. Si la cumple, se incluye en el conjunto de filas resultante.

En la consulta presentada en el *Script* 2.13 se observa que se utiliza una expresión condicional compuesta por dos partes: la primera, si el valor de la columna `duración` es mayor o igual a «00:04:00», y la segunda, si la duración es menor o igual a «00:05:00». La columna `duración` es de tipo *Time*, el cual permite almacenar valores de tiempo en el formato *hh:mm:ss[.nnnnnn]*.

También se observa el uso de operadores lógicos, como `AND`, `OR` y `NOT`, para construir condiciones compuestas. En el ejemplo planteado se utiliza el operador `AND` puesto que el valor de la columna `duración` debe cumplir ambas condiciones para que una fila sea incluida en la tabla resultante. En la Tabla 2.14 se resaltan las filas —es decir, las canciones— que cumplen la expresión condicional.

Los operadores relacionales, como el operador igual que *(=)*, mayor que *(>)* o menor que *(<)*, pueden utilizarse para formar todas las expresiones condicionales que se requieran. En este caso, una de las expresiones condicionales se define para evaluar si la duración de la canción es menor o igual *(<=)* a cuatro minutos.

En esta consulta también se observa que las dos expresiones condicionales comparan el valor de la columna `duración` con, aparentemente, las cadenas de caracteres *«00:04:00»* y *«00:05:00»*. Sin embargo, lo que realmente sucede es que el DBMS hace primero una conversión de tipos

para transformar estos valores almacenados como texto en datos del tipo *Time* y luego evalúa las expresiones. El resultado de la ejecución del *Script* 2.13 es el conjunto de datos presentado en la Tabla 2.15.

Tabla 2.14. Filas de la tabla Canciones que cumplen la expresión condicional

identificador	título	duración	género	idioma	artista principal	álbum original
10001	La tierra del olvido	00:04:25	1	Español	50001	900001
10002	Ojos así	00:03:57	3	Español	50003	900002
10003	Mi gente	00:03:05	4	Español	50005	*NULL*
10004	Ambiente	00:04:08	4	Español	50005	900004
10005	Cali pachanguero	00:04:51	2	Español	50002	900005
10006	La creciente	00:03:04	1	Español	50004	900006
10007	Sueños de conquista	00:04:02	1	Español	50004	900008
10009	Carito	00:03:39	3	Español	50001	900009
10011	Una aventura	00:05:16	2	Español	50002	900011
10012	Ginza	00:04:39	4	Español	50005	NULL
10013	Octavo día	00:04:32	3	Español	50003	900002
10014	Quiero verte sonreír	00:03:18	3	Español	50001	900009

Tabla 2.15. Resultado de la ejecución del *Script* 2.13

título	duración	género
La tierra del olvido	00:04:25	1
Ambiente	00:04:08	4
Cali pachanguero	00:04:51	2
Sueños de conquista	00:04:02	1
Ginza	00:04:39	4
Octavo día	00:04:32	3

El SQL proporciona operadores que permiten simplificar las condiciones combinando más de un criterio. Uno de estos operadores es el definido con la palabra `BETWEEN`, mediante el cual se pueden especificar expresiones condicionales para evaluar que un valor se encuentre en el rango definido por otros dos valores: un límite inferior y un límite superior. El resultado será verdadero o TRUE cuando el valor comparado se encuentre en el rango; de lo contrario, se obtendrá FALSE.

Con el operador `BETWEEN` puede construirse una expresión condicional para, por ejemplo, determinar si el número 80 está en el rango entre 50 y 200; en otras palabras, si el valor 80 es mayor e igual que 50 y menor o igual que 200. Para este caso, como el valor evaluado, es decir el número 80, está entre 50 y 200, la evaluación del operador `BETWEEN` genera un resultado TRUE. Si el valor evaluado fuera 49, se obtendría FALSE, y con el valor 50 o el valor 200 se obtendrá TRUE.

El operador `BETWEEN` funciona con valores numéricos, fechas, tiempos y cadenas de caracteres. En el caso de las fechas, se comparan cronológicamente, y en el de las cadenas de

caracteres, alfabéticamente. En este sentido, la operación BETWEEN puede utilizarse para construir una consulta SQL alternativa a la presentada en el *Script* 2.13 con el fin de simplificar la forma de la expresión condicional para filtrar las filas, tal y como se presenta en el *Script* 2.14.

***Script* 2.14**

```
SELECT título,
       género,
       duración
FROM Canciones
WHERE duración BETWEEN '00:04:00' AND '00:04:51'
```

El criterio de filtrado puede variar en complejidades dependiendo de lo que se desea especificar. Para observar esto se propone abordar la siguiente pregunta:

¿Cuál es el nombre y el año de lanzamiento de los artistas solistas con tiempo de vida artística entre diez y veinte años?

Para resolver estar pregunta se requiere una expresión condicional compuesta en la cual se evalúen datos almacenados en dos columnas. Tal y como se observa en la Figura 2.8, la columna año de lanzamiento debe mostrarse en el resultado y también sirve de base para calcular el valor de los años de vida artística, el cual es requerido para filtrar las filas que van a incluirse en el resultado. La consulta que da respuesta a la pregunta se presenta en el *Script* 2.15. El *Script* 2.16, por su parte, presenta una solución alternativa, y la Tabla 2.16 es el resultado, que es igual para ambas consultas.

***Script* 2.15**

```
SELECT nombre,
       "año de lanzamiento"
FROM Artistas
WHERE tipo = 'Solista' AND
       (2021 - "año de lanzamiento" >= 10) AND
       (2021 - "año de lanzamiento" <= 20)
```

***Script* 2.16**

```
SELECT nombre,
       "año de lanzamiento"
FROM Artistas
WHERE tipo = 'Solista' AND
       2021 - "año de lanzamiento" BETWEEN 10 AND 20
```

Figura 2.8. Columnas requeridas para resolver la pregunta

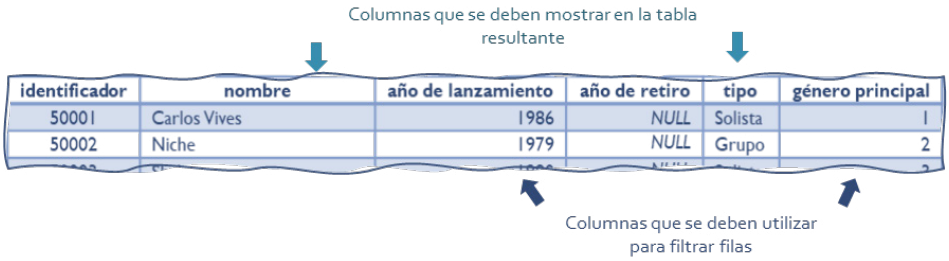


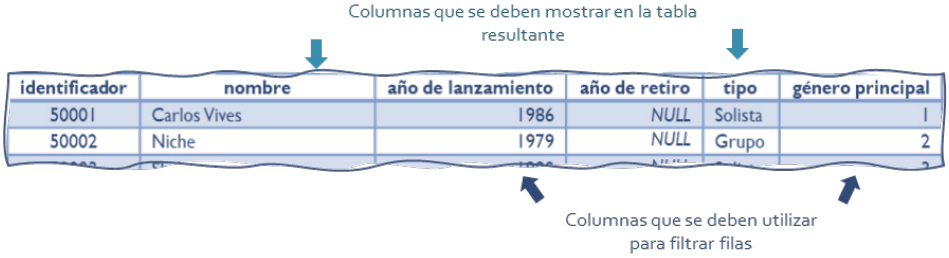
Tabla 2.16. Resultado de la ejecución del *Script* 2.16

nombre	año de lanzamiento
J Balvin	2006

Para continuar con el filtrado de filas con expresiones condicionales compuestas, se propone abordar la siguiente pregunta:

¿Cuál es el nombre y el tipo de todos los artistas del género vallenato o del género pop que tienen más de cuarenta años de vida artística?

Figura 2.9. Columnas requeridas para responder la pregunta



En esta pregunta se destaca la frase «los artistas del género vallenato o del género pop que tienen más de cuarenta años de vida artística». Allí se está indicando una expresión condicional compuesta en la que el género debe ser igual a *«Vallenato»* o igual a *«Pop»* y que la diferencia entre el año actual (2021) y el año de lanzamiento del artista debe ser mayor a cuarenta años. En la Figura 2.9 se presentan las columnas requeridas para responder la pregunta con los datos almacenados en la tabla Canciones. Al estar trabajando con una sola tabla, Artistas, resolver la pregunta implica definir cómo referirse a los géneros *«Vallenato»* y *«Pop»* pues estos no están almacenados en ella. En lugar de estas cadenas de caracteres, dicha tabla tiene la columna género principal, en la cual se almacena un número entero.

Con esta restricción, la única forma de completar los datos requeridos para escribir las expresiones condicionales relacionadas con los géneros principales de los artistas es observar la tabla Géneros para determinar el valor de la columna identificador en las filas correspondientes a los géneros solicitados. Como se muestra en la Tabla 2.17, el género «*Vallenato*» corresponde al identificador «*1*», y el género «*Pop*», al identificador «*3*».

Tabla 2.17. Filas de la tabla Géneros que contienen los identificadores requeridos

Géneros	
identificador	**nombre**
1	Vallenato
2	Salsa
3	Pop
4	Urbano Latino

La consulta que permite responder la pregunta se presenta en el *Script* 2.17, con el cual se obtiene como resultado el conjunto de datos de la Tabla 2.18.

***Script* 2.17**

```
SELECT nombre,
       tipo
FROM Artistas
WHERE (2021 - "año de lanzamiento") > 40 AND
       ("género principal" = 1 OR
       "género principal" = 3)
```

Tabla 2.18. Resultado de la ejecución del *Script* 2.17

nombre	**tipo**
Binomio de Oro de América	Grupo

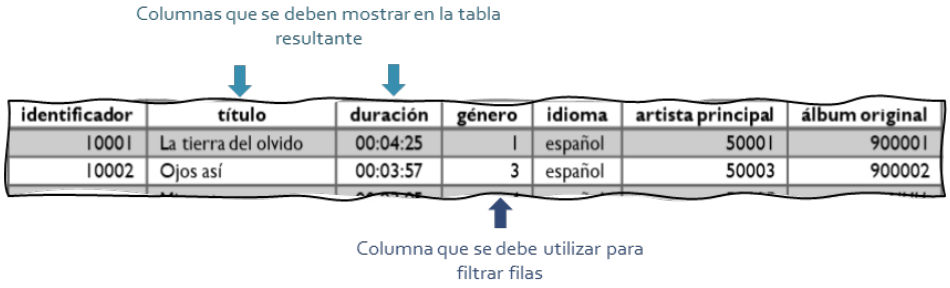
Otro operador diseñado para simplificar las tareas de comparación es el operador `IN`, el cual se utiliza para comprobar que un valor hace parte de una lista de valores. Con el fin de observar el uso de `IN`, se propone la siguiente pregunta:

¿Cuál es el título y la duración de las canciones del género de vallenato, pop o salsa?

Al analizar la pregunta se observa que debe evaluarse el valor de la columna género en cada fila de la tabla Canciones con relación a tres opciones: «*Vallenato*» (identificador 1 en la tabla Géneros), «*Salsa*» (identificador 2 en la tabla Géneros) y «*Pop*» (identificador 3 en la tabla Géneros). Si el valor de la columna género coindice con al menos una, dicha canción

debe incluirse en la tabla resultante. En la Figura 2.10 se muestra el análisis requerido para responder la pregunta.

Figura 2.10. Columnas requeridas para responder la pregunta



Para responder la pregunta se propone el *Script* 2.18, en el cual se plantea una expresión condicional compuesta de tres predicados o condiciones. Sin embargo, en este caso se podría utilizar el operador `IN` para comprobar si el género de una canción es «*1*», «*2*» o «*3*». Dicho operador compara el valor con todos los de la lista y, si al menos uno es igual, genera como resultado *verdadero* o TRUE; en caso contrario, será *falso* o FALSE. En el *Script* 2.19 se utiliza `IN` para responder la pregunta.

***Script* 2.18**

```
SELECT título,
       duración
FROM Canciones
WHERE género = 1 OR
       género = 2 OR
       género = 3
```

***Script* 2.19**

```
SELECT título,
       duración
FROM Canciones
WHERE género IN (1, 2, 3)
```

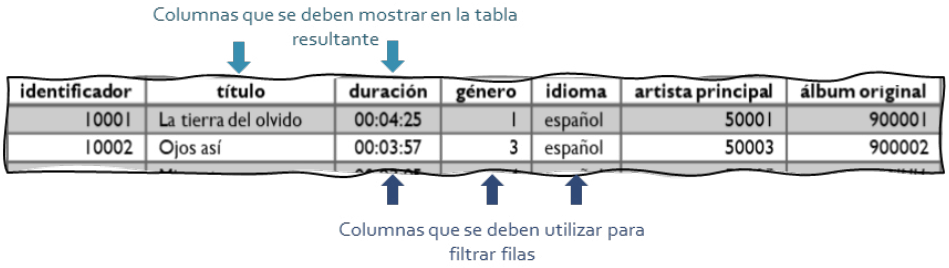
Todos estos operadores pueden ser combinados para construir expresiones condicionales tan complejas como sea necesario, siempre y cuando, como se ha mencionado, puedan evaluarse para llegar a un valor TRUE o FALSE. Para demostrar la combinación de diferentes condiciones, se propone abordar la siguiente pregunta:

¿Cuál es el título y la duración de las canciones en español de los géneros vallenato, pop o salsa que duran entre tres y cuatro minutos?

Mostrar primero la canción de mayor duración y finalizar con la de menor duración.

Para resolver la pregunta debe consultarse la tabla Canciones y utilizar las condiciones definidas en el enunciado para filtrar las filas: 1) idioma español, 2) el valor del género podrá ser *«Vallenato»* (identificador 1 en la tabla Géneros), *«Salsa»* (identificador 2 en la tabla Géneros) o *«Pop»* (identificador 3 en la tabla Géneros), y 3) el valor de la duración de la canción debe ser mayor o igual a tres minutos y menor o igual a cuatro minutos. La Figura 2.11 ilustra el análisis realizado, y en el *Script* 2.20 se presenta la consulta que permite responder la pregunta, la cual genera como resultado el conjunto de datos de la Tabla 2.19.

Figura 2.11. Columnas requeridas para responder la pregunta



***Script* 2.20**

```
SELECT título,
       duración
FROM Canciones
WHERE idioma = 'español' AND
       género IN (1, 2, 3) AND
       duración BETWEEN '00:03:00' AND '00:04:00'
ORDER BY duración DESC
```

Tabla 2.19. Resultado de la ejecución del *Script* 2.20

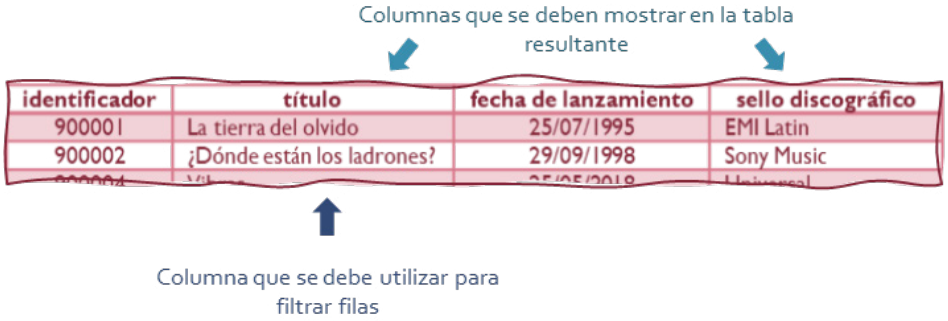
título	duración
Ojos así	00:03:57
La creciente	00:03:04
Carito	00:03:39
Quiero verte sonreír	00:03:18
Ojos así	00:03:57

Por otro lado, hay consultas que requieren expresiones condicionales basadas en los caracteres contenidos en una cadena, tal y como se pide en la siguiente pregunta:

¿Cuál es el título y el sello discográfico de los álbumes cuyos títulos inician con la vocal e?

Para responder la pregunta debe utilizarse la tabla Álbumes, aplicando un filtro basado en los valores almacenados en la columna título, tal y como se presenta en la Figura 2.12.

Figura 2.12. Columnas requeridas para responder la pregunta



Para un caso como este, el SQL tiene el operador `LIKE`, el cual es utilizado principalmente para realizar comparaciones de cadenas de texto con dos caracteres comodines: el porcentaje (%) y el guion bajo (_). El primero representa una cadena cualquiera (0 o más caracteres), mientras que el segundo representa un solo carácter.

Los caracteres comodines deben incluirse en la cadena de texto utilizada en la expresión condicional. Por ejemplo, si se quiere filtrar las filas de los artistas que inician con la letra a, la expresión condicional sería `nombre LIKE 'a%'`, indicando que el resultado debe iniciar con la letra a y luego puede aparecer cualquier cadena (%). Este operador no es sensible a las mayúsculas.

En esta pregunta se pide incluir aquellos álbumes cuyo título inicia con la letra e. Esto puede lograrse combinando el operador `LIKE` con el comodín %, tal y como se observa en la consulta del *Script* 2.21, con la cual se obtiene la Tabla 2.20.

Tabla 2.20. Resultado de la ejecución del *Script* 2.21

título	sello discográfico
El binomio de oro	Codiscos

***Script* 2.21**

```
SELECT título,
       "sello discográfico"
FROM Álbumes
WHERE título LIKE 'E%'
```

En algunos casos deben responderse preguntas relacionadas con datos desconocidos o inexistentes, es decir, datos representados con el valor NULL. Para abordar esta situación se propone la siguiente pregunta:

¿Cuál es el título y el sello discográfico de los álbumes con fecha de lanzamiento desconocida?

Hay que recordar que en una base de datos puede darse el caso de que una tabla contenga valores nulos porque no existen los datos o se desconocen en el momento de almacenar la fila correspondiente. Si en la columna en la que se encuentra un valor NULL se almacenan datos numéricos, podría pensarse que la nulidad significa lo mismo que el valor cero, pero no es así. De igual forma, si la columna almacena cadenas de caracteres, podría pensarse que NULL es equivalente a un espacio en blanco, pero esto también es un error.

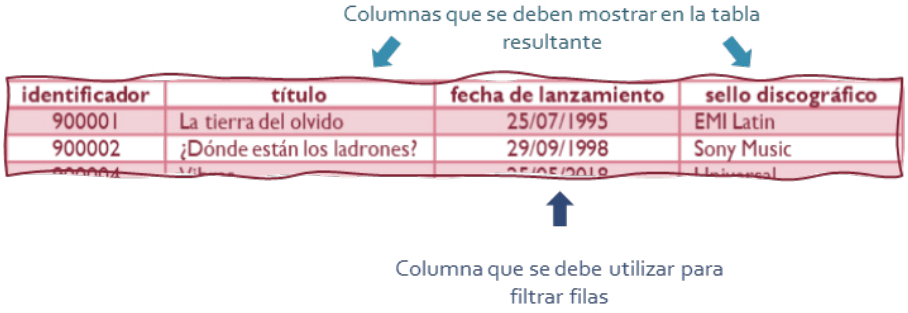
Tanto el valor cero como el espacio en blanco pueden tener significado en el contexto de los datos que se están almacenando. Por ejemplo, una columna que representa el número de hijos de una persona podría tomar el valor cero para representar que no tiene hijos, mientras que NULL significaría que se desconoce el número de hijos.

Para resolver esta pregunta se requiere trabajar con la tabla Álbumes. Como se muestra en la Figura 2.13, los datos resultantes están almacenados en las columnas título y sello discográfico. Además, es necesario realizar un filtrado de filas con base en los datos de la columna fecha de lanzamiento; específicamente, las filas que almacenan valores NULL.

Cuando se evalúa una expresión condicional haciendo referencia a alguna columna que tenga valores nulos, debe tomarse la precaución de evitar usar los operadores de comparación que se emplean para el caso de valores específicos porque el resultado será desconocido. En otras palabras, si se utiliza una expresión del tipo columna = `NULL`, el resultado de la evaluación lógica será siempre desconocido; nunca será TRUE ni FALSE. NULL no es igual a NULL, pero tampoco es diferente de NULL.

En este sentido, para evaluar que un valor es NULL se utilizará la palabra reserva `IS` acompañada de la palabra NULL, y para indicar que no es NULL se agregará la palabra `NOT`. En la resolución de la siguiente pregunta, se muestra el filtro de valores NULL. En este caso debe comprobarse de manera directa si el valor de la fecha de lanzamiento es desconocido, lo cual se logra con la consulta del *Script* 2.22.

Figura 2.13. Columnas requeridas para responder la pregunta



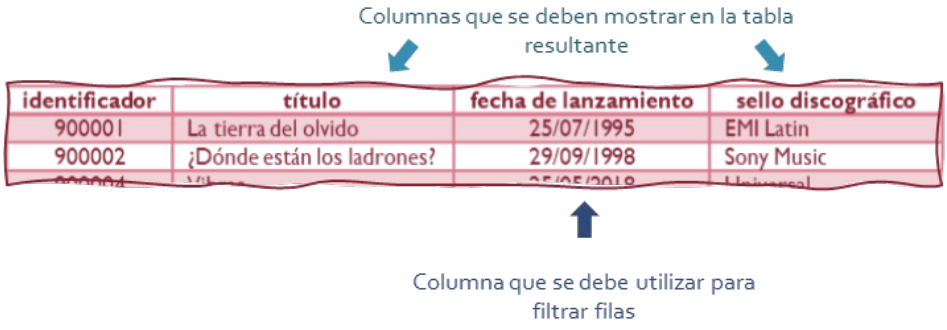
***Script* 2.22**

```
SELECT título,
       "fecha de lanzamiento",
       "sello discográfico"
FROM Álbumes
WHERE "fecha de lanzamiento" IS NULL
```

El trabajo con valores NULL exige un análisis detallado pues pueden generarse errores de interpretación. Para ilustrar esto se propone la siguiente pregunta:

¿Cuál es el título y el sello discográfico de los álbumes lanzados antes del año 2001?

Figura 2.14. Columnas requeridas para responder la pregunta



Al igual que la pregunta resuelta con la consulta del *Script* 2.22, en este caso se requieren datos de la tabla Álbumes, tal y como se muestra en la Figura 2.14. Sin embargo, debe establecerse una condición donde la fecha de lanzamiento sea menor o igual que «*31/12/2000*».

La respuesta a esta pregunta puede obtenerse con la consulta del *Script* 2.23. De esta forma se obtendrá como resultado la Tabla 2.21, que muestra los cinco álbumes que fueron lanzados antes de la fecha indicada en la condición.

***Script* 2.23**

```
SELECT título,
       "fecha de lanzamiento",
       "sello discográfico"
FROM Álbumes
WHERE "fecha de lanzamiento" <= '31/12/2000'
```

Tabla 2.21. Resultado la ejecución del *Script* 2.23

título	fecha de lanzamiento	sello discográfico
La tierra del olvido	25/07/1995	EMI Latin
¿Dónde están los ladrones?	29/09/1998	Sony Music
El binomio de oro	03/11/1976	Codiscos
Por lo alto	02/11/1976	Codiscos
Cielo de tambores	20/12/1990	Codiscos

Sin embargo, este resultado no significa que las demás filas de la tabla correspondan a álbumes lanzados a partir del año 2001. En otras palabras, es un error asumir que, dado que la tabla Álbumes tiene ocho filas y el resultado de la consulta del *Script* 2.23 tiene cinco filas, entonces hay tres álbumes lanzados después del año 2000.

El error de interpretación recae en que la fecha de lanzamiento del álbum «No hay quinto malo» es desconocida y por ende tiene registrado el valor NULL; por lo tanto, no es mayor ni menor que «*31/12/2000*». En tal sentido, el resultado de la consulta del *Script* 2.22 será una tabla con una sola fila: la correspondiente al álbum «No hay quinto malo» del sello discográfico «Internacional Records».

Es importante aclarar que, como se mencionó en la presentación de este material, el propósito del libro *Bases de datos relacionales: Un enfoque aplicado y orientado a resultados de aprendizaje* no es servir como manual de SQL, sino, a través de cada lección, conocer, entender y practicar las acciones más relevantes dentro de una base de datos. Para descargar y conocer más acerca del funcionamiento de algunos de los DBMS más utilizados, se pueden visitar los siguientes enlaces:

- Descarga PosgreSQL: https://www.postgresql.org/download/
- Manual PostgreSQL: https://www.postgresql.org/docs/current/index.html
- Descarga MySQL: https://dev.mysql.com/downloads/mysql/
- Manual MySQL: https://dev.mysql.com/doc/refman/8.0/en/

- Descarga SQL Server: https://www.microsoft.com/es-es/sql-server/sql-server-downloads
- Manual SQL Server: https://learn.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver16

2.4. Aprendizajes más importantes del capítulo 2

Lo tratado en este capítulo debería haberle permitido aprender lo expresado en la siguiente lista de ideas:

- SQL es el lenguaje que utilizan las bases de datos relacionales para permitir la manipulación de los objetos y de los datos almacenados en ellas.
- SQL es un lenguaje declarativo, de alto nivel, robusto y poderoso, el cual está compuesto por tres sublenguajes: DDL, DML y DCL.
- DML es el lenguaje utilizado para especificar las consultas de acceso a los datos. Como resultado de una consulta en este lenguaje, siempre se obtiene una estructura en forma de tabla.
- La palabra reservada FROM indica desde cuál tabla provienen los datos por utilizar.
- La palabra reservada SELECT se utiliza para especificar las columnas que harán parte del conjunto de datos resultante.
- La palabra reservada WHERE permite indicar qué condición deben cumplir las filas que se incluirán en el conjunto resultante.
- Para establecer las condiciones que deben cumplir las filas puede utilizarse cualquier combinación de operadores y operandos, siempre que la expresión formada por estos pueda reducirse a verdadero o falso.
- Los valores desconocidos o faltantes se representan con una palabra especial: NULL. Las pruebas lógicas donde esta palabra interviene se reducen al valor desconocido.

2.5. Actividades de aplicación para evidenciar lo aprendido

1. Utilizando la nueva versión de la base de datos de la colección de canciones, proponga cinco preguntas que expresen necesidades de datos que puedan responderse consultando una sola tabla. Todas las preguntas deben tener un nivel de complejidad que demande la utilización de filtros con condiciones compuestas y las demás operaciones explicadas en este capítulo.
2. Construir las consultas SQL y mostrar el conjunto de datos resultante para dar respuesta a las cinco preguntas propuestas en la actividad 1.
3. ¿Cuál es la pregunta que se está resolviendo con la siguiente consulta?

```
SELECT título,
       idioma
FROM Canciones
WHERE "álbum original" IS NULL AND
       idioma = 'español'
```

4. Proponga una consulta SQL para responder: ¿cuál es el título y la duración de las canciones que contienen la palabra *La* al inicio de su título?
5. ¿Cuál es la pregunta que se resuelve con la siguiente consulta?

```
SELECT DISTINCT
       "sello discográfico"
FROM Álbumes
WHERE
       "fecha de lanzamiento" NOT BETWEEN '01/01/1990' AND '31-12-2020' OR "fecha de
       lanzamiento" IS NULL
```

6. ¿Cuál es la pregunta o necesidad de datos que se resuelve con la siguiente consulta? ¿Cuál es el conjunto de datos resultante?

```
SELECT nombre,
       "año de lanzamiento",
       tipo
FROM Artistas
WHERE((2021 - "año de lanzamiento") BETWEEN 20 AND 40) AND
       "año de retiro" IS NULL
ORDER BY nombre DESC
```

7. Proponga una consulta SQL que permita responder: ¿cuáles son los sellos discográficos que han lanzado álbumes el mismo año de la fundación de la empresa Apple?
8. Seleccione la opción que expresa mejor lo que se obtiene con la siguiente consulta SQL. Argumente las razones de su decisión:

```
SELECT identificador,
       nombre,
       "año de lanzamiento"
FROM Artistas
WHERE nombre NOT LIKE '%A' OR
       nombre NOT LIKE '%E' OR
       nombre NOT LIKE '%I' OR
       nombre NOT LIKE '%O' OR
       nombre NOT LIKE '%U'
```

a. El identificador, el nombre y el año de lanzamiento de los artistas cuyo nombre contiene alguna vocal.
b. El identificador, el nombre y el año de lanzamiento de los artistas cuyo nombre termina con alguna vocal.
c. El identificador, el nombre y el año de lanzamiento de los artistas cuyo nombre no termina con alguna vocal.
d. El identificador, el nombre y el año de lanzamiento de todos los artistas.

9. ¿Cuál será el resultado luego de ejecutar la siguiente consulta? Explique su respuesta:

```
SELECT DISTINCT
       título
FROM Álbumes
WHERE "fecha de lanzamiento" < '31-10-2000' OR
       "fecha de lanzamiento" = NULL
```

10. Proponga una consulta SQL que permita responder: ¿cuál es el nombre y la duración de las canciones de genero vallenato, salsa o pop que son interpretadas por solistas?

Capítulo 3

Funciones y agrupamiento

Resultados de aprendizaje

- *Construye consultas en SQL utilizando funciones escalares, funciones de agregación, agrupamiento de filas y filtrado por grupos.*
- *Identifica alternativas de implementación de consultas en SQL para responder a necesidades que impliquen la agregación de datos de una tabla.*

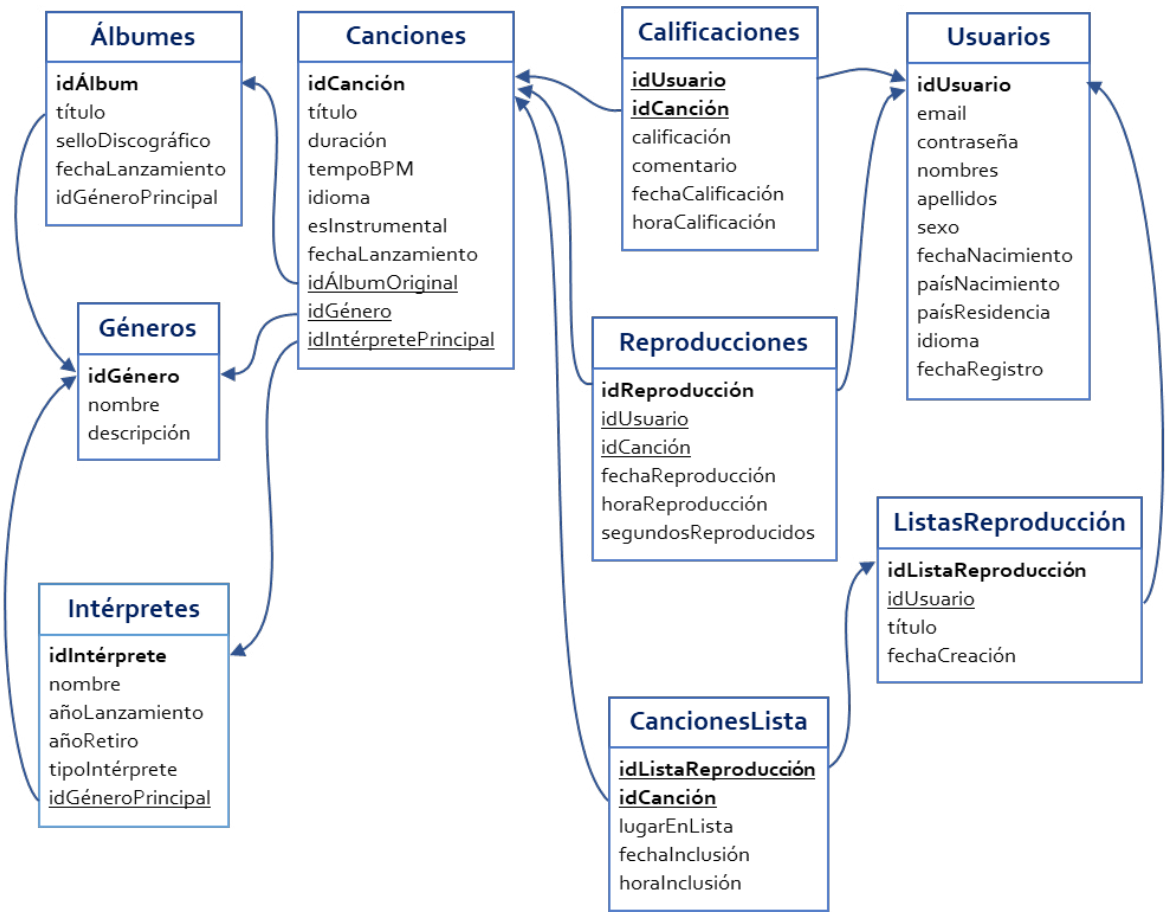
Los elementos del SQL permiten sacar provecho de los datos de una tabla para responder preguntas y suplir necesidades de datos en un contexto particular. Con la sentencia o comando `SELECT` pueden construirse, de forma sencilla y precisa, consultas que al ejecutarse en el DBMS generan los conjuntos de datos requeridos. Por otra parte, la operación `DISTINCT` hace que las filas de la tabla resultante sean distintas, es decir, evita que existan filas duplicadas en el resultado de una consulta. Además, con la cláusula `WHERE` dentro de una sentencia `SELECT` pueden filtrarse filas a partir de expresiones condicionales, también conocidas como predicados, que podrían tener gran complejidad si se utilizaran combinaciones de los diversos operadores disponibles.

Las posibilidades que ofrece el SQL van mucho más allá de lo expresado en el párrafo anterior. Hasta este momento apenas se ha tenido una pequeña degustación que deja una agradable sensación sobre la potencia y sencillez del lenguaje: solamente hay que decir lo que se quiere, y el DBMS lo entregará en forma de una tabla resultante. Ahora bien, para avanzar en el aprendizaje en este capítulo se continuará utilizando el caso de la colección de canciones, pero introduciendo una nueva versión de la base de datos con elementos que se adicionaron luego de la evaluación y mejora del diseño que se tenía en la versión de cuatro tablas presentada en el capítulo 2.

Al igual que en las situaciones reales, lo más común, normal y necesario es que las bases de datos no solo crezcan en cantidad de filas, sino que tengan una evolución en su diseño para satisfacer las nuevas necesidades o aprovechar las oportunidades que surgen de los cambios en estrategias, procesos, criterios de decisión y demás elementos del contexto de aplicación. En este orden de ideas, se procederá a explicar esta nueva versión de la base de datos de la plataforma *«Mis Canciones»*, la cual se resume en el diagrama de la Figura 3.1. El *script* de creación y carga de datos para el DBMS PostgreSQL está disponible como material complementario de este capítulo.

En el diagrama de la base de datos, las tablas se representan con rectángulos. Dentro de cada rectángulo se distinguen el nombre de la tabla y las columnas que la componen. Las columnas que están en negrilla son las que cumplen la función de ser la clave principal o PK (de la expresión en inglés *Primary Key*) de la tabla, es decir, los datos que permiten identificar cada fila de manera única. Debe recordarse que estos valores se mantendrán distintos aun cuando se agreguen nuevas filas.

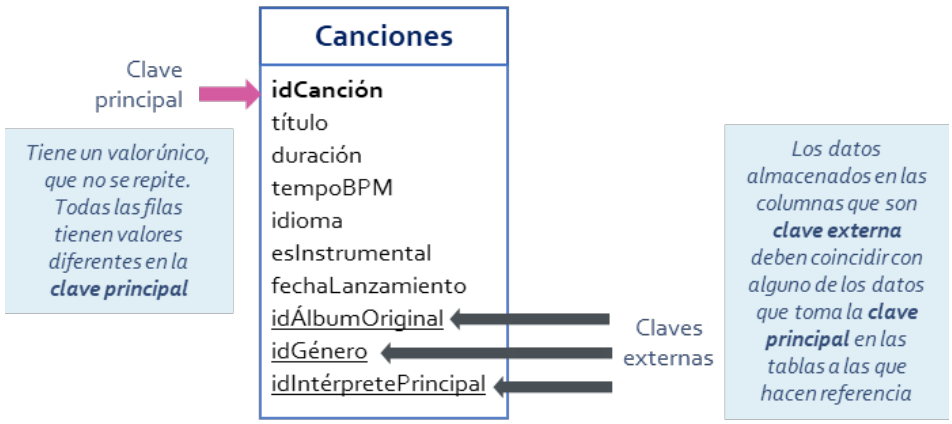
Figura 3.1. Versión 3 de la base de datos de la plataforma *«Mis Canciones»*



No es posible que dos filas tengan el mismo dato en la columna definida como clave principal. Esto se aplica también en el caso de claves principales de más de una columna, es decir, claves principales compuestas. Aunque en una columna, de manera independiente, pueden repetirse datos, en una clave principal compuesta dos filas no tendrán las mismas combinaciones de datos en las columnas que la componen.

Las columnas que aparecen subrayadas establecen las relaciones que tiene una fila de una tabla con filas almacenadas en otras tablas, es decir, cumplen la función de ser claves externas, claves foráneas o FK (de la expresión en inglés *Foreign Key*). Estas columnas hacen referencia a alguna clave principal de otra tabla; por eso en el diagrama hay conectores que van desde la clave externa hasta la clave principal. En algunos casos las columnas que son clave externa también son clave principal de la tabla. Para ilustrar esto, en la Figura 3.2 se muestra la tabla `Canciones`, resaltando su clave principal y sus tres claves externas que hacen referencia a las tablas `Álbumes`, `Géneros` e `Intérpretes`.

Figura 3.2. Identificación de clave principal y claves externas de la tabla `Canciones`



Esta versión incluye cinco nuevas tablas en las cuales se almacenan los datos de los usuarios de la plataforma y las interacciones que tienen con la colección de canciones. Los usuarios pueden interactuar con las canciones al reproducirlas, lo cual queda registrado en la tabla `Reproducciones`; al calificarlas, consignándose en la tabla `Calificaciones`, y al incluirlas en alguna de sus listas de reproducción, en cuyo caso se evidencia en las tablas `ListasReproducción` y `CancionesLista`.

Un usuario puede reproducir muchas veces la misma canción. En unas ocasiones es posible que la reproduzca completa, y en otras, solamente algunos segundos. En todos los casos se registra la cantidad de segundos que se reprodujo la canción. Asimismo, una canción solo puede tener una calificación por cada usuario. De este modo, si el usuario asigna una calificación y luego cambia su opinión, en la tabla `Calificaciones` se modificarán la calificación otorgada, la fecha de calificación, la hora de calificación e incluso el comentario realizado,

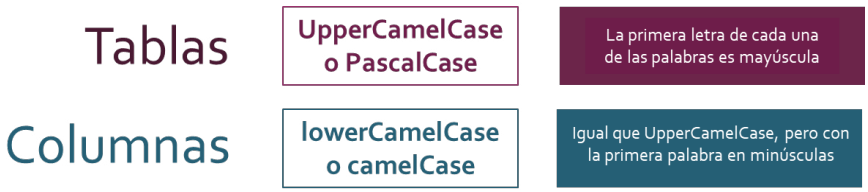
pero no se agregarán nuevas filas. La clave principal de la tabla `Calificaciones` se compone de dos columnas, lo cual significa que la combinación de valores no puede repetirse.

También, un usuario puede crear todas las listas de reproducción que desee, y en cada una tiene la capacidad de agregar múltiples canciones; incluso es posible tener listas de reproducción sin canciones, esto es, listas vacías. Las canciones se adicionan a una lista de acuerdo con las preferencias musicales del usuario, pero solamente pueden aparecer una vez por cada lista. En otras palabras, no es posible conformar una lista de reproducción agregando varias veces la misma canción.

En la tabla `Canciones` se adicionaron dos columnas: una para registrar el tempo de la canción en pulsos *(beats)* por minuto o BPM, y otra que identifica con un valor booleano si la canción es instrumental o no. También hubo un cambio en la tabla `Artistas`, que tomó un nombre más apropiado a su contenido: `Intérpretes`. Adicionalmente, es necesario precisar que son pocas las columnas que permiten el registro de valores nulos (`NULL`); específicamente, estas son `descripción`, de la tabla `Géneros`; `añoRetiro`, de la tabla `Intérpretes`; `tempoBPM` e `idÁlbumOriginal`, de la tabla `Canciones`, y `comentario`, de la tabla `Calificaciones`.

Varias tablas y columnas de esta nueva versión tienen nombres nuevos, y se observa el uso de los estilos de escritura denominados *Camel Case* o «mayúscula de camello» que se describen en la Figura 3.3. De este modo se simplifica la escritura de las consultas porque se evita utilizar caracteres para denotar el inicio y el fin de la frase cuando los nombres están conformados por frases con espacios en blanco. Es decir, no se requerirán comillas dobles para referirse a esas tablas o columnas en una consulta.

Figura 3.3. Estilos de escritura aplicados a los nombres de tablas y columnas



Trabajando con esta versión de la base de datos, en este capítulo se abordarán nuevos elementos del SQL como son las funciones escalares, las funciones de agregación y las operaciones de agrupamiento. Con una función escalar puede calcularse un nuevo dato a partir de otro almacenado en una fila de una tabla. La función de agregación, por su parte, permite calcular un valor a partir de operaciones aplicadas sobre todos los datos almacenados en una columna para un conjunto de filas. Finalmente, las operaciones de agrupamiento se emplean para definir grupos de filas que contengan el mismo valor en alguna columna, lo cual puede ser útil para aplicar alguna función de agregación de manera separada por cada

grupo y generar resultados resumidos por algún criterio, como podría ser el cálculo de la cantidad de canciones por artista o por idioma.

La utilización de funciones y las operaciones de agrupamiento multiplican las posibilidades de aprovechamiento de los datos en los contextos de uso. De tal manera se podrán construir consultas que generen, por ejemplo, resúmenes de datos comúnmente utilizados en reportes para apoyar la toma de decisiones.

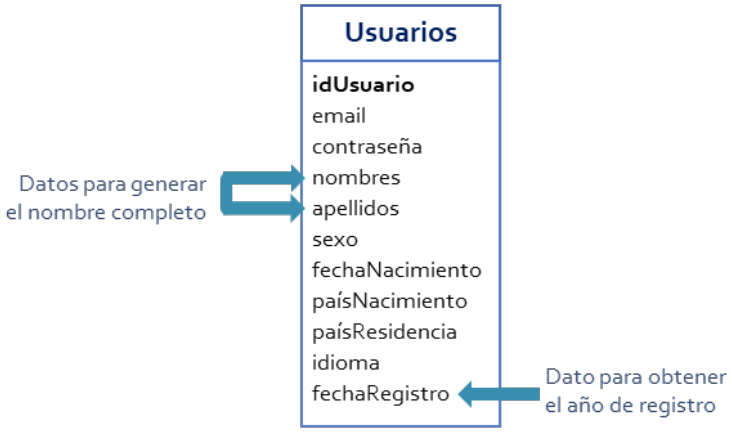
3.1. Funciones escalares

Los datos almacenados en las tablas son la base para obtener las respuestas y satisfacer las necesidades definidas en el contexto de aplicación. Sin embargo, es muy común que deba realizarse algún tipo de operación o de procesamiento sobre los datos almacenados con el fin de obtener nuevos datos necesarios para satisfacer las necesidades definidas o para ponerlos en algún formato requerido. Para ilustrar este caso de uso y abordar la forma de trabajarlo, se propone la siguiente pregunta:

¿Cuál es el nombre completo y el año de registro de todos los usuarios?

El análisis de la pregunta permite identificar que para responderla debe trabajarse con los datos almacenados en la tabla `Usuarios`. Ahora, a pesar de que en esa tabla no existen columnas con el nombre completo ni con el año de registro, sí existen las columnas `nombres`, `apellidos` y `fechaRegistro`. En este sentido, puede decirse que están disponibles todos los insumos, pero se requiere de un procesamiento adicional, tal y como se muestra en la Figura 3.4.

Figura 3.4. Columnas requeridas para responder la pregunta



El nombre completo se obtendrá al anexar o concatenar las cadenas de texto almacenadas en las columnas `nombres` y `apellidos`, tomando la precaución de incluir un espacio

entre ellas dos. Por su parte, el año de registro debe extraerse de los datos almacenados en la columna `fechaRegistro`. La respuesta a la pregunta puede obtenerse con la consulta presentada en el *Script* 3.1.

***Script* 3.1**

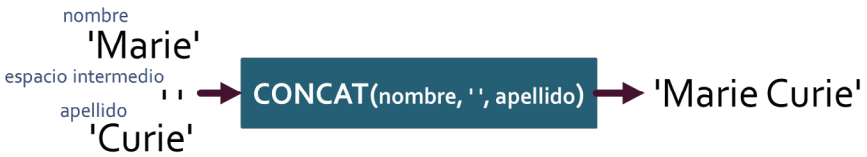
```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       EXTRACT(YEAR FROM fecharegistro) AS añoRegistro
FROM Usuarios
ORDER BY añoregistro DESC,
       nombreCompleto
```

En el *Script* 3.1 se observan nuevos elementos dentro de una consulta SQL, esto es, dos funciones escalares que se identifican con los nombres CONCAT y EXTRACT. La primera se utiliza para conformar una cadena de texto con los nombres y los apellidos de los usuarios, mientras que la segunda permite obtener el valor entero con el año de la fecha almacenada en la columna `fechaRegistro`.

Las funciones escalares en el SQL son las que más se asemejan a las que se implementan en cualquier lenguaje de programación estructurada. Estas funciones reciben un dato de un tipo particular, realizan operaciones con este y como resultado retornan un nuevo dato. Los DBMS tienen implementadas y disponibles una cantidad significativa de funciones que ayudan a llevar a cabo procesamientos de este tipo.

Volviendo a la función CONCAT, la cual hace parte del conjunto de funciones para procesamiento de cadenas de texto, se observa que recibe como entrada un conjunto de cadenas que desean anexarse y genera una nueva cadena con la concatenación. Las cadenas por concatenar pueden referenciarse con los nombres de las columnas que almacenan estos datos o pueden declararse explícitamente, tal y como se ve en el *Script* 3.1 para la cadena del espacio en blanco que debe ubicarse entre los nombres y los apellidos para mantener la separación de palabras. En la Figura 3.5 se ilustra el funcionamiento de esta función.

Figura 3.5. Conceptualización del funcionamiento de la función escalar CONCAT



Un aspecto que debe quedar claro antes de continuar es la forma en que operan estas funciones escalares. Intuitivamente, podría pensarse que al invocar una función como se hace con la función CONCAT en el *Script* 3.1 se estarían concatenando todos los nombres y todos los apellidos en una gran cadena de texto porque no se está especificando la fila a la

que se aplicaría. Esta, sin embargo, es una idea equivocada debido a que la operación sí se aplica a todas las filas de la tabla, pero fila por fila.

La función EXTRACT, por otra parte, permite extraer partes de una fecha, para lo cual recibe como entrada el componente de la fecha que desea extraerse. Por ejemplo, para el año se utiliza la palabra YEAR y el nombre de la columna que almacena la fecha. EXTRACT es un caso de una función incluida en el estándar SQL pero que no está implementada en todos los DBMS. En la Figura 3.6 se ilustra su funcionamiento.

Figura 3.6. Conceptualización del funcionamiento de la función escalar EXTRACT

El hecho de que las funciones estándar no estén implementadas no necesariamente es un problema porque normalmente los DBMS incorporan alguna función equivalente. Por ejemplo, en Microsoft SQL Server existe la función YEAR, la cual se utilizaría como se muestra en el *Script* 3.2. El resto es exactamente igual al *Script* 3.1.

***Script* 3.2**

```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       YEAR(fechaRegistro) AS añoRegistro
FROM Usuarios
ORDER BY añoregistro DESC,
       nombreCompleto
```

En otras palabras, la implementación de las funciones escalares no es igual en todos los DBMS. Es probable que, para realizar alguna operación, PostgreSQL proporcione una función; Microsoft SQL Server, otra, y así por cada DBMS disponible. Por ejemplo, en PostgreSQL, además de implementar la función estándar EXTRACT, también existe una función llamada DATE_PART que cumple el mismo objetivo y que podría utilizarse de la forma en que se presenta en el *Script* 3.3.

***Script* 3.3**

```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       DATE_PART('year', fechaRegistro) AS añoRegistro
FROM Usuarios
ORDER BY añoregistro DESC,
       nombreCompleto
```

En definitiva, el uso de las funciones escalares permite sacar mayor provecho de los datos almacenados y abre la posibilidad de realizar consultas para responder preguntas más complejas. Para continuar el aprendizaje, se propone la siguiente pregunta que tiene elementos similares a la resuelta con el *Script* 3.1:

¿Cuál es el *e-mail*, el *sexo*, la edad actual, la edad en el momento de registrarse en *Mis Canciones*, el tiempo que lleva como usuario y el país de residencia de todos los usuarios?

En esta pregunta nuevamente debe consultarse la tabla `Usuarios` sin aplicar filtros de filas. No obstante, además de solicitar datos almacenados allí, como el *e-mail*, el sexo y el país de residencia, se está requiriendo datos que deben calcularse a partir de las columnas `fechaNacimiento` y `fechaRegistro`, tal y como se presenta en la Figura 3.7.

Figura 3.7. Columnas requeridas para responder la pregunta

La edad actual debe calcularse tomando como base la fecha actual —es decir, la fecha en el momento de ejecución de la consulta— y el dato almacenado en la columna `fechaNacimiento`. De forma similar, el cálculo de la edad del usuario a la fecha en que se registró en la plataforma debe utilizar los datos almacenados en las columnas `fechaRegistro` y `fechaNacimiento`. Por último, para establecer el tiempo que lleva como usuario debe tomarse la fecha actual —de nuevo, aquella en que se hace la consulta— y el dato almacenado en la columna `fechaRegistro`.

Hay varias formas de realizar las operaciones con fechas. La más sencilla implica utilizar la función AGE, disponible en PostgreSQL, tal y como se presenta en el *Script* 3.4.

***Script* 3.4**

```
SELECT email,
       sexo,
       AGE(fechaNacimiento) AS edadActual,
       AGE(fechaRegistro, fechaNacimiento) AS edadAlRegistrarse,
       AGE(fechaRegistro) AS tiempoComoUsuario,
       paísResidencia
FROM Usuarios
```

Esta función recibe dos fechas y genera el número de años, meses y días entre ellas. Si solamente se proporciona una fecha como entrada, la función AGE realiza el cálculo con relación a la fecha actual, es decir, la fecha del momento de ejecutar la consulta. Esto se observa en el *Script* 3.4, en donde para calcular la edad del usuario simplemente se utiliza la función AGE tomando como entrada el valor almacenado en la columna `fechaNacimiento`. Aunque la función se denomina AGE o edad en español, su funcionalidad es amplia porque permite determinar el intervalo entre dos fechas.

Un grupo de funciones que tiene amplia utilización son las que realizan operaciones sobre cadenas de texto. Además de la función CONCAT, que sirve para anexar dos o más cadenas, existen funciones como la función UPPER, que permite poner un texto en mayúsculas, o la función LOWER, para dejarlo en minúsculas. También hay funciones que permiten manipular las cadenas de texto, obtener partes o subcadenas, determinar el tamaño, identificar si un carácter o una cadena está contenida en otra cadena, entre otras operaciones. Para ilustrar esto se propone la siguiente pregunta:

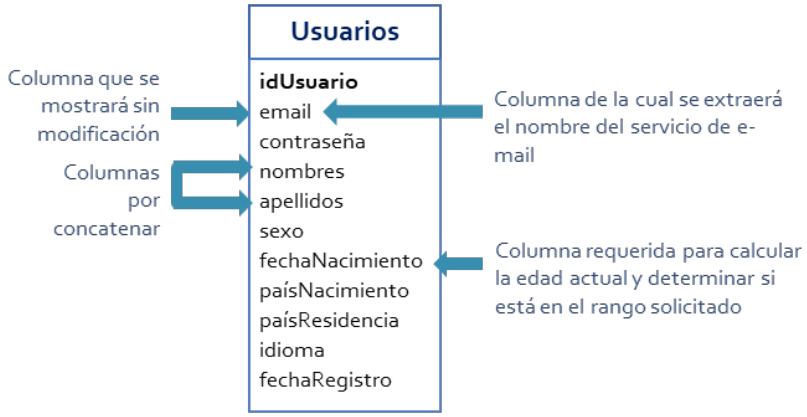
¿Cuál es el nombre completo, el *e-mail* y el nombre de dominio del servicio de *e-mail* de los usuarios que están en el rango de edad entre 20 y 39 años?

Para responder esta pregunta debe consultarse la tabla `Usuarios` utilizando varias funciones que permitan obtener los datos solicitados. En la Figura 3.8 se identifican las columnas requeridas para ese fin.

El nombre completo se obtiene concatenando las columnas `nombres` y `apellidos`. Para determinar si el usuario está dentro del rango de edad definido, es decir, entre 20 y 39 años, debe calcularse la edad actual a partir de la columna `fechaNacimiento`. Dicho valor no será incluido en la respuesta; solo se utilizará para filtrar las filas.

Para obtener el nombre de dominio del servicio de *e-mail*, es decir, el texto que está después del símbolo @ en cualquier dirección de *e-mail* válida, deben realizarse varias operaciones con los datos almacenados en la columna `email`. En principio, es preciso extraer la parte de la cadena de texto que está ubicada después de @, lo cual implica que se determine en cuál posición de la cadena está ubicado dicho símbolo.

Figura 3.8. Columnas requeridas para responder la pregunta



También es necesario determinar la longitud de la cadena para establecer cuántos caracteres hay después del símbolo @, es decir, la cantidad de caracteres que deben extraerse. La consulta que permite responder la pregunta se presenta en el *Script* 3.4, y el resultado obtenido de su ejecución es la Tabla 3.1.

***Script* 3.5**

```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       email,
       SUBSTRING(email,
       POSITION('@' in email) + 1,
       CHAR_LENGTH(email) - POSITION('@' in email)
       ) AS servicioEmail
FROM Usuarios
WHERE EXTRACT (YEAR FROM AGE(fechaNacimiento)) BETWEEN 20 AND 39
ORDER BY servicioEmail, email
```

Tabla 3.1. Resultado de la ejecución del *Script* 3.5

nombreCompleto	email	servicioEmail
Alexander Buenahora	alex@gmail.com	gmail.com
Marie Curie	marie.curie@gmail.com	gmail.com
René Descartes	rene.metodo@gmail.com	gmail.com
Ana Frank	ana.frank@hotmail.com	hotmail.com
Jean-Jacques Rousseau	elcontratosocial@yahoo.com	yahoo.com

La función `SUBSTRING` utilizada en el *Script* 3.4 obtiene una porción de una cadena de texto. Recibe como entradas la cadena original, que en este caso es el contenido de la columna `email`, la posición desde la cual se extraerán los caracteres y la cantidad de caracteres por extraer. En la Figura 3.9 se ilustra este funcionamiento.

Figura 3.9. Conceptualización del funcionamiento de la función escalar SUBSTRING

Para obtener la posición desde la cual se empezará a extraer la porción de la cadena de texto se utiliza la función escalar POSITION, indicando que debe ubicarse el símbolo @. Por su parte, para determinar la cantidad de caracteres por extraer se utiliza una expresión en la que se invoca a la función escalar CHAR_LENGTH.

La resolución de las tres preguntas anteriores involucró varias funciones escalares, pero esto apenas es una mínima muestra de lo que está incluido en el SQL. En las versiones del SQL que se implementan en cada DBMS existe un amplio inventario de funciones escalares aplicables a diferentes tipos de datos. En la medida en que se requiera alguna operación escalar, es conveniente consultar la documentación oficial del DBMS en uso, de forma que se identifiquen las funciones escalares estándar o de implementación propia que podrían servir para realizar el procesamiento de datos. No obstante, en caso de no existir alguna función aplicable al problema específico, el SQL contempla la opción de programar funciones personalizadas o a la medida para aplicarlas sobre los datos. Esto será un elemento de trabajo en capítulos posteriores.

3.2. Funciones de agregación

Obtener valores que resuman los datos almacenados en las tablas es una demanda común de parte de los usuarios de una base de datos porque les proporciona niveles de comprensión diferentes a los que pueden lograrse si solamente se miran los datos de manera individual. En algunos casos, esto significa realizar una operación tan sencilla como contar la cantidad de filas existentes o que cumplan cierta condición. También puede representar la acumulación o el cálculo de un valor total a partir de cantidades almacenadas en cada fila. Incluso puede pensarse en una operación que permita sumar los datos almacenados en cada fila y luego dividir el resultado de la suma entre el número de filas sumadas, es decir, el cálculo de la media aritmética.

Este tipo de operaciones son significativas porque facilitan la generación de resúmenes que permiten construir rápidamente interpretaciones generales para sustentar la toma de decisiones y la ejecución de alguna actividad. En muchas oportunidades es muy significativo conocer datos agregados, como el promedio de ingresos de los habitantes de una ciudad, para tomar medidas sin necesidad de revisar uno a uno los datos.

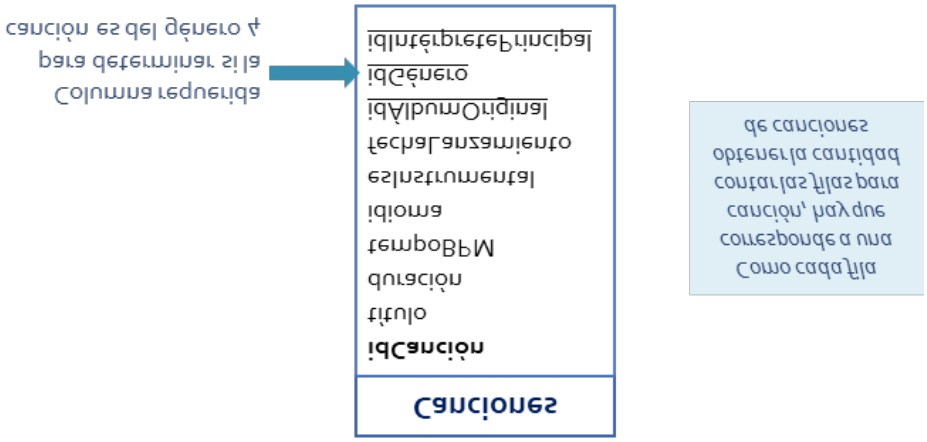
Problemas o necesidades de este tipo pueden abordarse con las funciones de agregación incluidas en el SQL. Como su nombre lo sugiere, este tipo de funciones permiten agregar en

un solo valor los datos almacenados en una columna. Para iniciar el aprendizaje de ellas se propone la siguiente pregunta:

¿Cuántas canciones clasificadas en el género 4 se tienen almacenadas en la colección?

El análisis para resolver la pregunta permite identificar que los datos requeridos están en la tabla `Canciones`. Sin embargo, el dato solicitado no está almacenado en alguna tabla de la base de datos, sino que hace referencia al valor que se obtendría con el conteo de las canciones de género 4. Por lo tanto, deben obtenerse las canciones del género 4 y luego contarlas para obtener un número entero positivo, o cero en caso de que no estén almacenadas canciones de ese género. En la Figura 3.10 se identifican las columnas requeridas para responder la pregunta.

Figura 3.10. Columnas requeridas para responder la pregunta



Para iniciar, es conveniente plantear una consulta que permita obtener las canciones del género 4, la cual se presenta en el *Script* 3.6. Al ejecutarla utilizando la nueva versión de la base datos, se obtiene el conjunto de datos presentado en la Tabla 3.2.

***Script* 3.6**

```
SELECT idCanción,
       título,
       idGénero
FROM Canciones
WHERE idGénero = 4
```

Tabla 3.2. Resultado de la ejecución del *Script* 3.6

idCanción	título	idGénero
9	Amarte más no pude	4
10	Sin medir distancias	4
19	La creciente	4
20	Olvídala	4

El resultado presentado en la Tabla 3.2, en la que aparecen listadas las canciones que pertenecen al género 4, es fácil de procesar manualmente por cualquier persona. Basta una mirada rápida para determinar que hay cuatro canciones del género 4. Sin embargo, no siempre es sencillo llegar a resultados de este tipo, y es en esos escenarios donde intervienen las funciones de agregación. En este sentido, la consulta que permite obtener el resultado requerido es la que se presenta en el *Script* 3.7.

***Script* 3.7**

```
SELECT COUNT(*) AS cantidadCancionesGénero4
FROM Canciones
WHERE idGénero = 4
```

Para resumir los datos contando filas se utiliza la función `COUNT` con el parámetro *, el cual representa que el conteo se realiza sobre la fila completa. En esta consulta también se requirió la cláusula `WHERE` para filtrar las filas por contar. El resultado es una tabla con una columna llamada `cantidadCancionesGénero4` y una sola fila con el número 4, tal como se muestra en la Tabla 3.3.

Tabla 3.3. Resultado de la ejecución del *Script* 3.7

cantidadCancionesGénero4
4

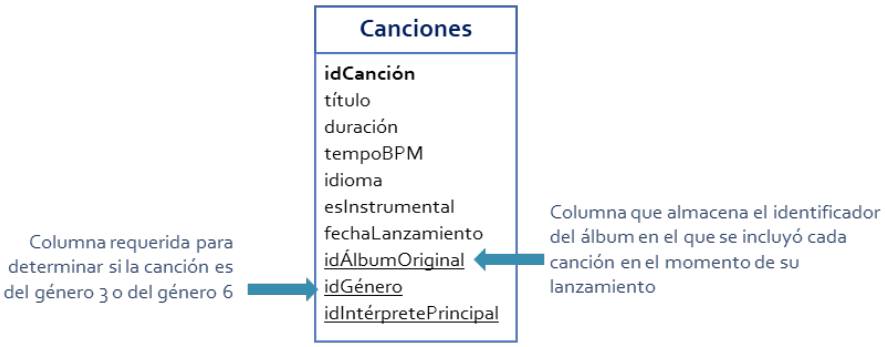
La función de conteo de filas puede generar resultados diferentes en escenarios aparentemente equivalentes y debe utilizarse con precaución. Para mostrar esto se propone abordar la siguiente pregunta:

¿Cuántos álbumes tienen canciones almacenadas en la colección que estén clasificadas en el género 3 o en el género 6?

Esta pregunta involucra dos conceptos que tienen representación en dos tablas de la base de datos: los álbumes y las canciones. Sin embargo, para responderla debe utilizarse

únicamente la tabla `Canciones` porque en la tabla `Álbumes` no existen datos relacionados con las canciones que contienen ni con los géneros en los que está clasificada cada canción. En la tabla `Canciones`, tal y como se muestra en la Figura 3.11, existen las columnas `idÁlbumOriginal` e `idGénero`, con las cuales puede obtenerse una respuesta a la pregunta.

Figura 3.11. Columnas requeridas para responder la pregunta



Inicialmente, puede plantearse la consulta del *Script* 3.8 para obtener los datos de los álbumes a partir de los datos contenidos en la tabla `Canciones`, cuyo resultado se presenta en la Tabla 3.4. Allí se observa que hay seis canciones, pero cuatro valores distintos del identificador del álbum original debido a que una canción tiene el identificador de álbum con valor `NULL` y el identificador *15* aparece dos veces.

***Script* 3.8**

```
SELECT título, idÁlbumoriginal
FROM Canciones
WHERE idgénero IN (3,6)
```

Tabla 3.4. Resultado de la ejecución del *Script* 3.8

título	idÁlbumoriginal
Una aventura	3
Gotas de lluvia	4
Las acacias	15
Oropel	15
Ne me quitte pas	16
Cali es sabrosura	*NULL*

Al utilizar la función `COUNT` con el parámetro *, tal y como se muestra en el *Script* 3.9, el DBMS contará las filas y generará como resultado el número 6, tal y como se muestra en la

Tabla 3.5. Este valor, no obstante, es incorrecto porque, como ya se analizó antes, existen cuatro valores diferentes para el identificador del álbum. Para resolver esto podría utilizarse la función `COUNT` solicitándole que cuente las filas, pero tomando únicamente la columna `álbum`, tal y como se presenta en el *Script* 3.10. De esta forma se genera como resultado la Tabla 3.6.

***Script* 3.9**

```
SELECT COUNT(*) AS cantidadÁlbumes
FROM Canciones
WHERE idgénero IN (3, 6)
```

Tabla 3.5. Resultado de la ejecución del *Script* 3.9

cantidadÁlbumes
6

***Script* 3.10**

```
SELECT COUNT(idÁlbumoriginal) AS cantidadÁlbumes
FROM Canciones
WHERE idgénero IN (3, 6)
```

Tabla 3.6. Resultado de la ejecución del *Script* 3.10

cantidadÁlbumes
5

Ahora bien, este resultado también es incorrecto porque, como se mostró en la Tabla 3.4, solo existen cuatro valores distintos de los identificadores de álbumes. Para realizar la operación correcta, es decir, contar los identificadores distintos de los álbumes con canciones de los géneros *3* y *6*, debe utilizarse la palabra `DISTINCT`. Mediante esta instrucción se eliminan los duplicados y los valores nulos se excluyen del conteo. En este sentido, la consulta para resolver correctamente la pregunta se presenta en el *Script* 3.11, con el cual se genera el resultado presentado en la Tabla 3.7.

***Script* 3.11**

```
SELECT COUNT(DISTINCT idÁlbumoriginal) AS cantidadÁlbumes
FROM Canciones
WHERE idgénero IN (3, 6)
```

Tabla 3.7. Resultado de la ejecución del *Script* 3.11

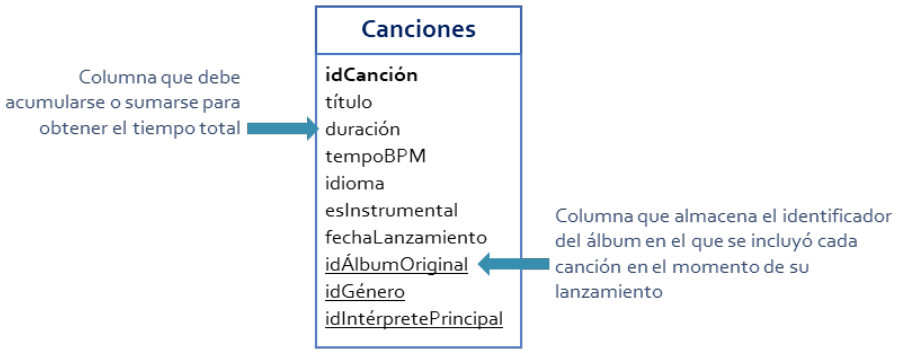
cantidadÁlbumes
4

En el SQL existen otras funciones de agregación, tales como `AVG` para calcular promedios, `SUM` para calcular totales, y las funciones `MAX` y `MIN` para conocer los valores máximos y mínimos respectivamente. Con el fin de avanzar en la comprensión y el uso de las funciones de agregación, se propone abordar la siguiente pregunta:

¿Cuánto tiempo tomaría la reproducción consecutiva de todas las canciones de la colección que tienen registrado como álbum original el álbum identificado con el número 15?

En esta pregunta se pide hallar la suma de la duración de todas las canciones que pertenecen al álbum con identificador *«15»*. En la Figura 3.12 se identifican las columnas requeridas para responder a esta solicitud.

Figura 3.12. Columnas requeridas para responder la pregunta



Para resumir los datos haciendo una suma de un conjunto de valores, el SQL proporciona la función `SUM`, la cual recibe como parámetro la columna o expresión que contiene los valores por sumar. Por lo tanto, para resolver la pregunta debe invocarse la función `SUM` utilizando como parámetro la columna `duración` de la tabla `Canciones`, tal y como se presenta en el *Script* 3.12.

***Script* 3.12**

```
SELECT SUM(duración) AS duraciónTotal
FROM Canciones
WHERE idÁlbumOriginal = 15
```

La consulta generará como resultado los datos presentados en la Tabla 3.8 puesto que solo hay dos canciones que tienen registrado como álbum original el que tiene identificador «*15*». La duración total será la suma de la de las canciones «*Las acacias*» y «*Oropel*», es decir, la suma de «*00:04:04*» y «*00:02:28*».

Tabla 3.8. Resultado de la ejecución del *Script* 3.12

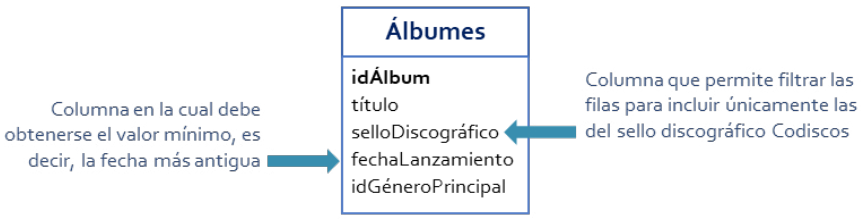
duraciónTotal
00:06:32

En varios casos se pide identificar cuáles son los valores extremos en una lista de valores, es decir, el máximo y el mínimo. Para esto, SQL proporciona la función `MAX` para el máximo y la función `MIN` para el mínimo. En ambas debe especificarse la columna o expresión correspondiente a la lista de valores en la cual se busca el valor extremo. Para comprender la aplicación de estas funciones, se proponen las dos preguntas que se trabajan a continuación:

¿Cuál es la fecha de lanzamiento del álbum más antiguo del sello discográfico Codiscos?

En esta pregunta se requiere hallar el valor mínimo o más antiguo para la fecha de lanzamiento de los álbumes del sello discográfico *Codiscos*. En la Figura 3.13 se identifican las columnas requeridas para responder la pregunta, mientras que el *Script* 3.13 presenta la consulta que permite responder este interrogante.

Figura 3.13. Columnas requeridas para responder la pregunta



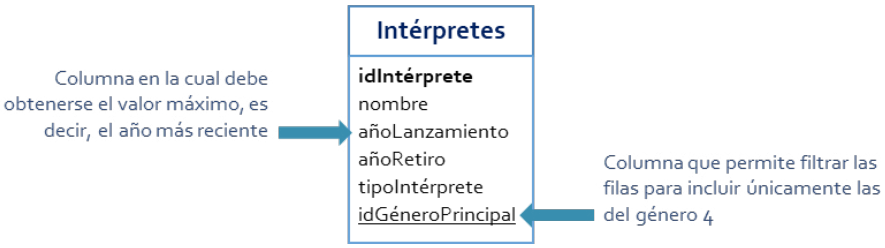
***Script* 3.13**

```
SELECT MIN(fechaLanzamiento) AS fechaLancamientoAlbumMásAntiguo
FROM Álbumes
WHERE sellodiscográfico = 'Codiscos'
```

¿Cuándo inició la carrera musical del intérprete del género 4 con menor tiempo de actividad?

Para resolver esta pregunta debe obtenerse el año más reciente en que un intérprete del género 4 inició su carrera. Todos los datos necesarios están almacenados en la tabla `Intérpretes`, tal y como se muestra en la Figura 3.14. La consulta para resolver la pregunta se presenta en el *Script* 3.14.

Figura 3.14. Columnas requeridas para responder la pregunta



***Script* 3.14**

```
SELECT MAX(añoLanzamiento) AS fechaInicioCarreraMásReciente
FROM Intérpretes
WHERE idGéneroPrincipal = 4
```

Una medida de tendencia central ampliamente utilizada para resumir datos es la media aritmética o simplemente el promedio. Un caso de uso de este tipo de agregación se aborda con la siguiente pregunta:

¿Cuál es la calificación promedio que obtuvo la canción con identificador 5 durante el año 2020?

En esta pregunta se necesita obtener un valor único que resuma todas las calificaciones que asignaron los usuarios a una canción. Los datos requeridos para resolverla están almacenados en la tabla `Calificaciones`, tal y como se describe en la Figura 3.15.

Figura 3.15. Columnas requeridas para responder la pregunta

Para resolver la pregunta puede utilizarse la función de agregación AVG, que permite hallar la media aritmética del conjunto de datos almacenados en la columna `calificación` de

la tabla `Calificaciones`. Además, debe contemplarse que el valor de columna `idCanción` debe ser igual a *5* y que la fecha de calificación debe estar entre el primer día de enero del año 2020 y el último día de diciembre del mismo año. Con base en este análisis, se construyó la consulta presentada en el *Script* 3.15, cuya ejecución arroja que la calificación promedio para la canción en el periodo especificado es *2*.

***Script* 3.15**

```
SELECT AVG(calificación) AS calificaciónPromedioCanción5Año2020
FROM Calificaciones
WHERE idCanción = 5 AND
       (fechaCalificación BETWEEN '01/01/2020' AND '31/12/2020')
```

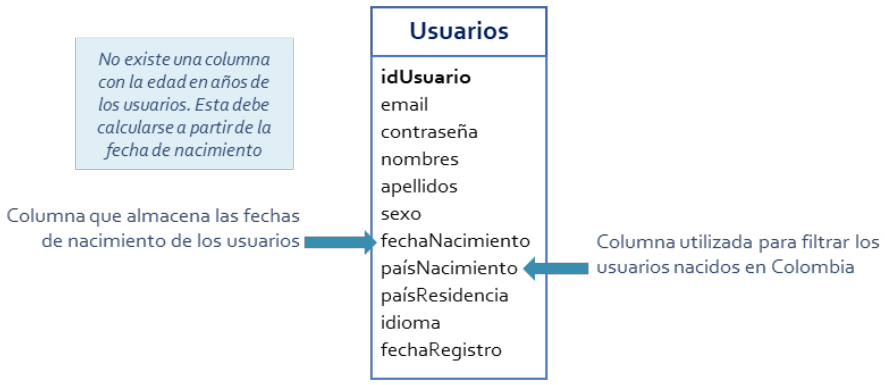
Otro ejemplo del uso de la función promedio se presenta con la siguiente pregunta:

¿Cuál es la edad promedio, en años, de los usuarios que nacieron en Colombia?

Para abordar la solución a esta pregunta es conveniente descomponer el problema y comprender los elementos que se deben tomar en consideración. En primer lugar, la operación de agregación debe restringirse a los usuarios nacidos en Colombia, para lo cual debe incluirse la cláusula `WHERE` con una condición donde la columna `paisNacimiento` de la tabla `Usuarios` sea igual a «*Colombia*».

En segundo lugar, en relación con los datos por resumir con la función `AVG`, es decir, las edades de los usuarios, debe identificarse que en la tabla `Usuarios` no hay columnas que almacenen estos valores, pero hay una con la fecha de nacimiento. En la Figura 3.16 se identifican estas columnas requeridas para responder la pregunta.

Figura 3.16. Columnas requeridas para responder la pregunta



La forma más simple de calcular la edad promedio en años es la implementada en la consulta del *Script* 3.16.

***Script* 3.16**

```
SELECT AVG(EXTRACT(YEAR FROM AGE(fechaNacimiento))) AS Promedio
FROM Usuarios
WHERE paísNacimiento = 'Colombia'
```

En el *Script* 3.16 se utiliza la función de agregación AVG para resumir los datos obtenidos al ejecutar la función escalar EXTRACT a partir del valor hallado tras ejecutar la función escalar AGE sobre los datos almacenados en la columna `fechaNacimiento`. En otras palabras, se calcula la edad del usuario con la función AGE, luego se extrae el año de esa edad utilizando la función EXTRACT con el parámetro YEAR. Ese número entero que representa los años de un usuario es el que se resumirá con la función de agregación AVG. La Tabla 3.9 presenta el resultado obtenido.

Tabla 3.9. Resultado de la ejecución del *Script* 3.16

Edad promedio
51,5

Las funciones de agregación, a diferencia de las funciones escalares, tienen un mayor nivel de estandarización e implementación en los diferentes DBMS. Las principales funciones de agregación tienen los mismos nombres y reciben los mismos parámetros en todos los DBMS de amplio uso en la industria.

3.3. Agrupamiento

Las funciones de agregación permiten utilizar los datos para generar nuevos datos que resumen o que denotan algún tipo de tendencia. Hasta el momento se han aplicado sobre todas las filas de una tabla o sobre las filas que cumplan las expresiones condicionales especificadas en la cláusula WHERE para generar un único valor como resultado. Sin embargo, este uso puede ampliarse introduciendo el concepto de agrupamiento, el cual es, simplemente, conformar subgrupos con las filas de la tabla que tengan el mismo dato en alguna columna. Para iniciar con esta operación se propone la siguiente pregunta:

¿Cuántos intérpretes hay por cada tipo?

Tabla 3.10. Datos de la tabla `Intérpretes`

nombre	añolanzamiento	tipointérprete	idgéneroprincipal
Carlos Vives	1986	Solista	4
Grupo Niche	1979	Grupo	3
Shakira	1990	Solista	1
Aterciopelados	1990	Grupo	2
Diomedes Díaz	1975	Solista	4
ChocQuibTown	2000	Grupo	1
J Balvin	2006	Solista	5
Silva y Villalba	1966	Grupo	7
Yuri Buenaventura	1996	Solista	3
El Binomio de Oro de América	1976	Grupo	4

En esta pregunta se requieren los datos de la tabla `Intérpretes`, haciendo especial énfasis a la columna `tipoIntérprete`, la cual almacena una cadena de texto que determina si un intérprete es *«Solista»* o *«Grupo»*. En la Tabla 3.10 se presentan los datos almacenados en la tabla `Intérpretes` de la versión en uso de la base de datos. En este caso se presentan solamente cuatro de las columnas que posee dicha tabla y todas las filas.

En la Tabla 3.10 se observa que la columna `tipoIntérprete` almacena alguno de los dos valores que denotan el tipo correspondiente. Con observación directa puede determinarse que las filas 1, 3, 5, 7 y 9 tienen el texto *«Solista»*, mientras que las demás indican *«Grupo»*. En tal sentido, para responder la pregunta bastará con contar las filas de la tabla en las que coincidan los valores de la columna `tipoIntérprete`, para lo cual puede utilizarse la función de agregación `COUNT` con el parámetro *.

Una primera opción para resolver la solicitud sería utilizar la cláusula `WHERE` para filtrar las filas y utilizar la función `COUNT` para contarlas, de la forma en que se muestra en el *Script* 3.17 para el tipo de intérprete *«Solista»*. Así se obtiene el resultado presentado en la Tabla 3.11.

***Script* 3.17**

```
SELECT COUNT(*) AS cantidadDeSolistas
FROM Intérpretes
WHERE tipointérprete = 'Solista'
```

Tabla 3.11. Resultado de la ejecución del *Script* 3.17

cantidadDeSolistas
5

Este enfoque de solución implica construir una consulta por cada tipo, lo cual en esta oportunidad no demanda tanto trabajo porque son solamente dos valores diferentes. No

obstante, en otros casos en los cuales la cantidad de valores diferentes sea más grande o desconocida este enfoque no es viable.

El SQL incluye la cláusula `GROUP BY` para construir consultas basadas en grupos. Con esta cláusula se procesan los datos en grupos definidos con base en uno o varios criterios determinados por los valores que tengan las filas en las columnas por las cuales desea agruparse. Para este caso, el agrupamiento con base en la columna `idgéneroprincipal` organiza dos grupos de filas, como se muestra en la Tabla 3.12.

Tabla 3.12. Identificación de grupos de filas por datos de la columna `tipointérprete`

nombre	añolanzamiento	tipointérprete	idgéneroprincipal	
Aterciopelados	1990	Grupo	2	**Grupo 1** Intérpretes tipo «*Grupo*»
ChocQuibTown	2000	Grupo	1	
El Binomio de Oro de América	1976	Grupo	4	
Grupo Niche	1979	Grupo	3	
Silva y Villalba	1966	Grupo	7	
Carlos Vives	1986	Solista	4	**Grupo 2** Intérpretes tipo «*Solista*»
Diomedes Díaz	1975	Solista	4	
J Balvin	2006	Solista	5	
Shakira	1990	Solista	1	
Yuri Buenaventura	1996	Solista	3	

Luego de haber agrupado las filas se aplica la función de agregación indicada, que en este caso corresponde a la función `COUNT`, con el parámetro *. Con esto se obtendrían como resultado dos valores numéricos: uno con el conteo de las filas que tienen el dato «*Grupo*» en la columna `tipoIntérprete`, que en este caso es cinco, y otro con el conteo de las filas que tienen el dato «*Solista*», también cinco. La consulta SQL para responder la pregunta sería la que se presenta en el *Script* 3.18, con la cual se genera como resultado la Tabla 3.13.

***Script* 3.18**

```
SELECT tipoIntérprete, COUNT(*) AS cantidad
FROM Intérpretes
GROUP BY tipoIntérprete
```

Tabla 3.13. Resultado de la ejecución del *Script* 3.18

tipoIntérprete	cantidad
Solista	5
Grupo	5

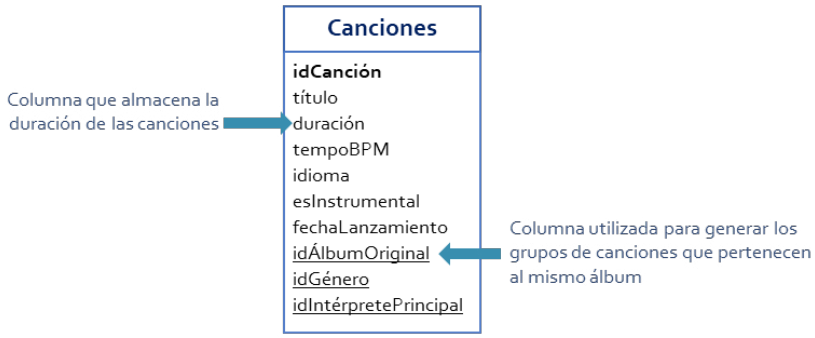
Con este ejemplo sencillo se ilustra el funcionamiento general de la creación de grupos y de la aplicación de las funciones de agregación sobre los grupos: en primer lugar, deben especificarse las columnas de la tabla o expresiones que se utilizarán para crear los grupos y,

después de especificar el criterio de agrupamiento, se procede a aplicar la función de agregación sobre cada grupo. Las columnas o expresiones que se toman como base para crear los grupos se especifican empleando la cláusula `GROUP BY`. Para profundizar en su funcionamiento se propone abordar la siguiente pregunta:

¿Cuál es la duración total de cada álbum?
Los resultados deben presentarse ordenados de mayor a menor duración.

Para iniciar es clave identificar que se solicita la duración total de cada álbum, para lo cual debe sumarse la duración de cada una de las canciones incluidas en cada caso. En la Figura 3.17 se identifican las columnas requeridas para responder la pregunta.

Figura 3.17. Columnas requeridas para responder la pregunta



En el análisis se deduce que deben agruparse las canciones de acuerdo con el álbum al que pertenecen, es decir, la columna `idÁlbumoriginal` de la tabla `Canciones`, para luego utilizar la función de agregación `SUM` sobre la columna `duración`. La consulta se presenta en el *Script* 3.19, la cual genera el resultado mostrado en la Tabla 3.14.

***Script* 3.19**

```
SELECT idÁlbumoriginal AS Álbum,
       SUM(duración) AS duraciónTotal
FROM Canciones
GROUP BY idÁlbumoriginal
ORDER BY duraciónTotal DESC
```

Si una consulta utiliza agrupación de filas, todas las acciones posteriores a la acción de la cláusula `GROUP BY`, incluidas `SELECT` y `ORDER BY`, deben operar en grupos en lugar de hacerlo en filas individuales. Cada grupo está representado por una única fila en el resultado final de la consulta, lo que implica que todas las expresiones que se especifiquen en las otras

cláusulas deben garantizar la obtención de un escalar, es decir, un valor particular por cada grupo. Para ilustrar esto se propone la siguiente pregunta:

Tabla 3.14. Resultado de la ejecución del *Script* 3.19

álbum	duraciónTotal
15	00:06:32
4	00:05:54
16	00:05:37
3	00:05:16
18	00:05:04
10	00:04:58
9	00:04:49
1	00:04:25
12	00:04:21
8	00:04:09
6	00:03:55
11	00:03:51
2	00:03:47
7	00:03:45
5	00:03:38
NULL	00:03:14
13	00:03:09
17	00:03:02
14	00:02:51

¿Cuántas canciones conforman cada una de las listas de reproducción de los usuarios?

Los resultados deben estar ordenados de mayor a menor cantidad de canciones.

Para responder la pregunta hay que contar las canciones que tienen el mismo identificador de lista de reproducción en la tabla `CancionesLista`, es decir, aquellas que están incluidas en la misma lista. Específicamente, deberían agruparse los datos de la tabla `CancionesLista` utilizando como criterio la columna `idListaReproducción` para contar los `idCanción` de cada grupo. Por último, el listado debe ordenarse de manera que la primera fila de la tabla resultante contenga la lista de reproducción con el mayor número de canciones. En la Figura 3.18 se identifican las columnas requeridas para responder la pregunta.

La consulta que implementa lo descrito en el análisis se presenta en el *Script* 3.20. De esta forma se obtiene como resultado la Tabla 3.15.

Figura 3.18. Columnas requeridas para responder la pregunta

***Script* 3.20**

```
SELECT idListaReproducción AS Lista,
       COUNT(idCanción) AS "Cantidad de canciones"
FROM CancionesLista
GROUP BY idListaReproducción
ORDER BY "Cantidad de canciones" DESC
```

Tabla 3.15. Resultado de la ejecución del *Script* 3.20

Lista	Cantidad de canciones
7	14
10	12
16	11
17	11
4	11
20	10
6	10
1	9
5	8
8	8
14	8
2	8
11	7
18	7
19	6
12	6
13	6
15	5
3	4
9	2

En las consultas de este tipo es posible que en ocasiones se requiera realizar un filtrado de filas antes de realizar el agrupamiento de los datos. Para ilustrar este caso se propone abordar la siguiente pregunta:

¿Cuál es la calificación promedio de las canciones con identificador 1, 2, 3, 4, 5, 6, 7, 8, 9 o 10?
Mostrar los resultados ordenados de la mejor a la peor calificada.

Todos los datos necesarios para responder a esta pregunta están en la tabla `Calificaciones`. Para resolver la solicitud debe trabajarse únicamente con las canciones cuyo identificador sea alguno de los valores de la lista, es decir, entre 1 y 10. Esto obliga a realizar un filtrado de filas para trabajar solo con las canciones que cumplen ese criterio.

También se indica que debe hallarse un promedio de calificación por canción, lo cual implica crear grupos para cada identificador de canción. En la Figura 3.19 se señalan las columnas requeridas para responder la pregunta.

Figura 3.19. Columnas requeridas para responder la pregunta

Una consulta para resolver la pregunta se presenta en el *Script* 3.21, donde se utiliza la cláusula `WHERE` con el operador `BETWEEN` para filtrar las filas antes de organizar los grupos requeridos para calcular la calificación promedio. También se puede observar el uso de los alias `canción` y `calificación promedio` para las columnas de la tabla resultante, con el fin de mejorar la presentación.

***Script* 3.21**

```
SELECT idCanción AS canción,
       AVG(calificación) AS "calificación promedio"
FROM Calificaciones
WHERE idCanción BETWEEN 1 AND 10
GROUP BY idCanción
ORDER BY "calificación promedio" DESC
```

En el conjunto de datos resultante, presentado en la Tabla 3.16, se aprecia que la canción mejor calificada es la que tiene como identificador el número «8», la cual tiene una valoración promedio de «3.833». Asimismo, se puede advertir que la peor valorada es la identificada con el número «3» la cual tiene un valor de «2.000».

En la cláusula `GROUP BY` puede incluirse más de una columna o expresión para crear los grupos. Con la siguiente pregunta se ejemplifica este caso de uso:

¿Cuántas reproducciones se hicieron por cada canción y por cada mes durante los meses de noviembre y diciembre del año 2020?

En esta pregunta se combinan todos los elementos presentados hasta este punto. Las columnas requeridas para resolverla se identifican en la Figura 3.20.

Tabla 3.16. Resultado de la ejecución del *Script* 3.21

canción	calificación promedio
8	3.833
2	3.250
1	3.000
9	2.833
4	2.714
6	2.600
10	2.2857
7	2.2857
5	2.1250
3	2.0000
8	3.8333
2	3.2500
1	3.0000
9	2.8333
4	2.7142
6	2.6000
10	2.2857
7	2.2857
5	2.1250
3	2.0000

Figura 3.20. Columnas requeridas para responder la pregunta

Específicamente, se requiere un filtro (`WHERE`) para trabajar únicamente con las reproducciones realizadas entre el 1 de noviembre y el 31 de diciembre del 2020. También se necesita un agrupamiento de datos utilizando el identificador de la canción y el mes en que se hizo la reproducción (el mes en que se hizo la reproducción debe obtenerse utilizando la función escalar `EXTRACT`). La expresión para extraer el mes debe incluirse tanto en la cláusula `GROUP BY` como en la cláusula `SELECT`. Finalmente, es preciso presentar los datos ordenados por la cantidad de reproducciones. En el *Script* 3.22 se plantea la consulta SQL que implementa los elementos descritos y genera la solución a la pregunta.

***Script* 3.22**

```
SELECT idCanción AS Canción,
       EXTRACT(MONTH FROM fechaReproducción) AS MES,
       COUNT(idReproducción) AS Reproducciones
FROM Reproducciones
WHERE fechaReproducción BETWEEN '01/11/2020' AND '31/12/2020'
GROUP BY idCanción,
       EXTRACT(MONTH FROM fechaReproducción)
ORDER BY Reproducciones DESC
```

3.4. Filtrado de datos agrupados

La cláusula `WHERE` permite filtrar filas en una consulta, y su ámbito de ejecución se da antes de aplicar el agrupamiento de datos, tal y como se muestra, por ejemplo, en el *Script* 3.22. Sin embargo, es muy común que sea necesario filtrar filas con base en los datos generados con las funciones de agregación luego de efectuar el agrupamiento. Para abordar este escenario se propone la siguiente pregunta, que es una variación de la pregunta resuelta con el *Script* 3.20:

¿Cuántas canciones conforman cada una de las listas de reproducción de los usuarios?
Mostrar únicamente las listas con más de 10 canciones y los resultados ordenados de la lista con mayor número de canciones a la lista con menor número de canciones.

En la pregunta se aclara que deben incluirse únicamente aquellos grupos donde la función de agregación arroje un valor superior a 10, es decir, solamente aquellas filas en las que la función `COUNT` aplicada a la columna `idCanción` genera un resultado mayor a 10. En la Figura 3.21 se identifican las columnas requeridas.

Figura 3.21. Columnas requeridas para responder la pregunta

La consulta que implementa la solución se presenta en el *Script* 3.23, con la cual se genera como resultado los datos que se presentan en la Tabla 3.17. Este resultado es parecido al presentado en la Tabla 3.15, con la diferencia de que en esta consulta se mantienen únicamente los grupos donde la función de agregación genera un valor mayor a 10.

***Script* 3.23**

```
SELECT idListaReproducción AS Lista,
       COUNT(idCanción) AS "Cantidad de canciones"
FROM CancionesLista
GROUP BY idListaReproducción
HAVING COUNT(idCanción) > 10
ORDER BY "Cantidad de canciones" DESC
```

Tabla 3.17. Resultado de la ejecución del *Script* 3.23

Lista	Cantidad de canciones
7	14
10	12
17	11
4	11
16	11
7	14
10	12
17	11
4	11
16	11

En el *Script* 3.23 se muestra que, para realizar el filtrado de filas luego del agrupamiento de datos, existe la cláusula `HAVING`. Mientras que la cláusula `WHERE` es un filtro a nivel de fila, la cláusula `HAVING` es un filtro a nivel de grupo. En esta última se especifican los criterios que deben cumplir todos los grupos que serán mostrados en el resultado final. Así, solo los grupos para los que la evaluación de los predicados de la cláusula `HAVING` es *verdadera* son incluidos en el resultado. En cambio, los grupos para los que los predicados se evalúan como *falso* o *desconocido* se descartan.

Un elemento de análisis en la sintaxis presentada en el *Script* 3.23 es el uso del alias para la columna resultante de la función de agregación, definido en la cláusula `SELECT` con la expresión *cantidad de canciones* e invocado en la cláusula `ORDER BY`. Sin embargo, en la cláusula `HAVING` no se utiliza debido al cumplimiento de lo especificado en el estándar SQL en relación con el orden de procesamiento de las cláusulas de una consulta. Según el estándar, las cláusulas son procesadas lógicamente en un orden diferente a aquel en el cual aparecen en la consulta, esto es:

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `SELECT`
6. `ORDER BY`

De acuerdo con lo anterior, la asignación de un alias a una columna de la tabla resultante mediante la cláusula `SELECT` sucede después de haber hecho el agrupamiento y antes de

hacer el ordenamiento. Es decir, el alias no existe cuando se procesa el agrupamiento, pero sí en el momento en que se realiza el ordenamiento al final.

Es posible que en alguna consulta deban utilizarse los filtros a nivel de filas y los filtros a nivel de grupos. Para abordar este escenario se propone la siguiente pregunta:

¿Cuántos comentarios positivos tiene cada una de las canciones de la colección?

Mostrar aquellas canciones que tengan más de cinco comentarios positivos.

Como en cualquier problema, es necesario entender lo que se está requiriendo. Para esta consulta en particular, se utilizará un enfoque que podría calificarse como «ingenuo» para identificar los comentarios positivos. Se entenderá por comentario positivo todo aquel que contenga las palabras «buena» o «excelente». Por consiguiente, debe establecerse un filtro para dejar únicamente las filas de la tabla `Calificaciones` que tienen comentarios con ese contenido. Las columnas requeridas para resolverla se identifican en la Figura 3.22.

Figura 3.22. Columnas requeridas para responder la pregunta

Adicionalmente, debe considerarse que el operador `LIKE` es sensible a las mayúsculas y minúsculas, por lo que deben homogeneizarse todos los comentarios a minúsculas, usando la función `LOWER`, antes de buscar si contienen las dos palabras clave. El resto del problema es similar a lo abordado hasta este punto: agrupar por el identificador de la canción y contar la cantidad de filas por cada grupo indicando en la cláusula `HAVING` que solo incluya aquellas en las que el conteo o `COUNT` (*) sea mayor o igual a cinco. La consulta que implementa lo descrito se presenta en el *Script* 3.24.

***Script* 3.24**

```
SELECT idcanción AS Canción,
       COUNT(*) AS "Cantidad comentarios positivos"
FROM Calificaciones
WHERE(LOWER(comentario) LIKE '%buena%')
       OR (LOWER(comentario) LIKE '%excelente%')
GROUP BY idcanción
HAVING COUNT(*) >= 5
```

En algunos escenarios puede surgir la duda entre aplicar el filtro deseado en la fase de ejecución de la cláusula `WHERE` o en la fase de ejecución de la cláusula `HAVING`. Para recrear este caso se propone la siguiente pregunta:

¿Cuál es la duración promedio de las canciones instrumentales por género e idioma?

Considerar únicamente las canciones de los géneros 1 y 3.

Esta consulta puede abordarse como está en el *Script* 3.25, es decir, manteniendo únicamente las canciones instrumentales (`esInstrumental` ='0'), agrupando por las columnas `idGénero` e `idioma`, hallando el promedio, eliminando los grupos que no pertenecen a los géneros 1 o 3 y aplicando agregación para mostrar los resultados.

***Script* 3.25**

```
SELECT idGénero AS Género,
       idioma,
       AVG(duración) AS "Duración promedio"
FROM Canciones
WHERE esInstrumental = '0'
GROUP BY idGénero,
       idioma
HAVING idGénero IN(1, 3)
```

De manera análoga, la pregunta puede ser resuelta filtrando los datos en la cláusula `WHERE` no solo por si es instrumental la canción, sino también por si pertenece a los géneros 1 o 3, tal como se muestra en el *Script* 3.26. De esta forma se obtendrán los mismos resultados presentados en la Tabla 3.18.

Este comportamiento equivalente debe ser analizado de forma detallada para evitar errores que comprometan la integridad y la fiabilidad de los datos generados. Lo clave es determinar el lugar preciso para realizar el filtrado de acuerdo con el orden de ejecución de las cláusulas.

***Script* 3.26**

```
SELECT idGénero AS Género,
       idioma,
       AVG(duración) AS "Duración promedio"
FROM Canciones
WHERE esInstrumental = '0'
       AND idGénero IN (1, 3)
GROUP BY idGénero,
       idioma
```

Tabla 3.18. Resultado de la ejecución del *Script* 3.26

Género	idioma	Duración promedio
I	Español	00:03:55
I	Inglés	00:03:38
3	Español	00:04:48
3	Francés	00:05:37

3.5. Aprendizajes más importantes del capítulo 3

- Las funciones escalares permiten manipular los valores de las columnas presentes en cada una de las filas incluidas en una consulta.
- La implementación de las funciones escalares puede variar dependiendo del DBMS utilizado. Por consiguiente, es clave revisar la documentación oficial del DBMS que se esté empleando en el momento.
- Las funciones de agregación permiten resumir en un solo valor los datos almacenados en una columna de una tabla.
- Con la cláusula `GROUP BY` pueden crearse grupos a partir de los valores presentes en las columnas de una tabla.
- Es posible establecer filtros a nivel de grupo con la cláusula `HAVING`.
- El orden de procesamiento lógico de las cláusulas no tiene una correspondencia directa con el orden en que son escritas. Este orden de procesamiento lógico impone algunas restricciones en cuanto a la forma para acceder a las nuevas columnas creadas.
- Es un error típico intentar hacer referencia al alias de una columna en cláusulas que se procesan antes de la cláusula `SELECT`.
- Según el estándar, la fase `ORDER BY` es la única en la que puede hacerse referencia a los alias de las columnas creadas en la fase `SELECT` porque es la única fase procesada después de esta.

3.6. Actividades de aplicación para evidenciar lo aprendido

1. Construya una consulta SQL que dé respuesta a la necesidad de datos expresada con la pregunta «¿Cuáles son las canciones de mayor y menor duración por cada género?».
2. Explique cuál es la necesidad de datos que se busca satisfacer con la consulta presentada en el siguiente *script*:

```
SELECT idCanción,
       AVG(calificación),
       COUNT(idUsuario)
FROM Calificaciones
GROUP BY idCanción
HAVING AVG(calificación) > 3
```

3. Construya una consulta SQL que dé respuesta a la necesidad de datos expresada con la pregunta «¿Cuántos usuarios residen en el mismo país en el que nacieron?».
4. Explique cuál es la necesidad de datos que se busca satisfacer con la consulta presentada en el siguiente *script*:

```
SELECT idIntérpretePrincipal,
       COUNT(*)
FROM Canciones
GROUP BY idIntérpretePrincipal
HAVING COUNT(DISTINCT idioma) > 1
```

5. Construya una consulta SQL que dé respuesta a la necesidad de datos expresada con la pregunta «¿Cuál es promedio del tempo en pulsos de las canciones de cada intérprete por cada uno de sus álbumes?».
6. Modifique la consulta presentada en el siguiente script de modo que se mejore la forma en que se presenta la tabla resultante:

```
SELECT idGénero,
       idioma,
       COUNT(*),
       MIN(tempoBPM),
       MAX(tempoBPM)
FROM Canciones
GROUP BY idGénero,
       idioma
HAVING COUNT(*) > 1
ORDER BY idGénero,
       idioma
```

7. Responda las preguntas planteadas en los siguientes literales tomando como base las dos consultas presentadas a continuación:

```
SELECT idIntérpretePrincipal,
 COUNT(*)
FROM Canciones
WHERE idIntérpretePrincipal <> 1
 AND esInstrumental <> '1'
 AND idGénero NOT IN (1, 2)
GROUP BY idIntérpretePrincipal
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC
```

```
SELECT idIntérpretePrincipal,
 COUNT(*)
FROM Canciones
WHERE idGénero > 2
 AND esInstrumental IN ('0')
GROUP BY idIntérpretePrincipal
HAVING COUNT(*) > 1
 AND idIntérpretePrincipal <> 1
ORDER BY COUNT(*) DESC
```

a. ¿Se obtiene el mismo resultado?
b. ¿Son equivalentes?

c. ¿Cuál de las dos le parece mejor? ¿Por qué?
d. ¿Qué mejora le haría a la que le parece mejor?

8. Construya una consulta SQL que dé respuesta a la necesidad de datos expresada con la pregunta «¿Cuántas veces por cada día de la semana se reproduce cada una de las canciones?». Mostrar las canciones que tienen al menos tres reproducciones cada día y listar los resultados ordenados de la canción con más reproducciones a la canción con menos reproducciones.
9. Explique cuál es la necesidad de datos que se busca satisfacer con la consulta presentada en el siguiente *script*:

```
SELECT EXTRACT("year" FROM fechaReproducción) año,
       EXTRACT("month" FROM fechaReproducción) mes,
       idCanción, COUNT(*), COUNT(DISTINCT idUsuario),
       AVG(segundosReproducidos)
FROM Reproducciones
WHERE segundosReproducidos > 30
GROUP BY año, mes, idCanción
ORDER BY año DESC, mes DESC, COUNT(*) DESC,
       COUNT(DISTINCT idUsuario) DESC
```

10. Utilizando la nueva versión de la base de datos de la colección de canciones presentada en este capítulo, defina cinco preguntas que le permitan construir cinco consultas SQL en las que se utilicen los datos de una sola tabla. En todas debe aplicarse agrupamiento con filtrado basado en los datos obtenidos con funciones de agregación, filtrado de filas y ordenamiento de resultados.

Capítulo 4

Subconsultas

Resultados de aprendizaje

– *Construye consultas en SQL que usan datos de diferentes tablas de una base de datos aplicando subconsultas autónomas y subconsultas correlacionadas.*
– *Determina alternativas de uso de subconsultas para responder a necesidades que impliquen el uso de datos almacenados en varias tablas.*

Todas las consultas SQL de los capítulos 2 y 3 han utilizado como fuente de datos una única tabla. Solamente se ha requerido una sentencia `SELECT` para obtener las respuestas a las preguntas planteadas. No obstante, la mayoría de las preguntas o necesidades de datos que surgen en el contexto de aplicación de una base de datos deben suplirse con consultas que necesitan utilizar más de una tabla.

Como se ha mostrado hasta el momento, el SQL es un lenguaje sencillo, pero potente. También es muy versátil en su sintaxis ya que permite integrar elementos que van generando consultas cada vez más complejas. Puede darse el caso de consultas que retornen datos de una única tabla, pero que requieren uno a varios datos de otras tablas como parte de expresiones condicionales para filtrar las filas resultantes. Asimismo, es posible que los datos esperados en la respuesta provengan de varias tablas. En este capítulo se abordarán consultas como la del primer caso y se utilizará la misma versión de la base de datos *Mis Canciones* del capítulo 3.

El nuevo elemento del lenguaje que se introduce en este capítulo se denomina *subconsulta*. Una subconsulta o consulta anidada es, simplemente, una consulta dentro de otra.

Puesto en términos simples, una subconsulta es una sentencia `SELECT` que contiene otra sentencia `SELECT` en alguna de sus cláusulas.

La primera puede denominarse la consulta principal, es decir, la más externa, y las demás se conocen como consultas secundarias o subconsultas. La consulta *externa* utiliza los datos obtenidos en la consulta *interna* para realizar alguna operación. El uso de las subconsultas abre nuevas posibilidades para utilizar de forma efectiva los datos de múltiples tablas.

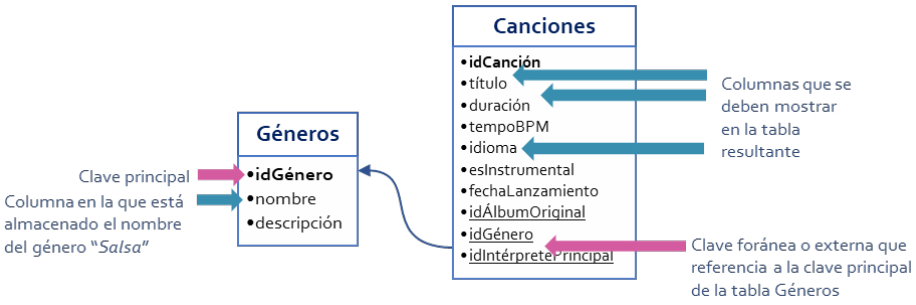
4.1. Subconsultas autónomas

Una subconsulta autónoma, autocontenida o independiente es aquella que genera un valor o un conjunto de valores requeridos por la consulta principal para realizar alguna operación. Para iniciar el aprendizaje se propone la siguiente pregunta:

¿Cuáles son los títulos, la duración y el idioma de las canciones del género «*Salsa*»?

A diferencia de las preguntas abordadas hasta el momento, en esta se plantea la necesidad de filtrar las filas de la tabla `Canciones` utilizando un dato que no está en ella. Se requieren los datos de las canciones del género «*Salsa*», para lo cual debe considerarse que en la tabla `Canciones` está almacenado el identificador del género o `idGénero` de cada canción y no el `nombre` de dicho género. Por consiguiente, no puede aplicarse un filtro con una expresión condicional asociada a la cadena de texto «Salsa». En la Figura 4.1 se presenta un diagrama con las tablas de la base de datos requeridas para resolver la pregunta.

Figura 4.1. Tablas y columnas requeridas para responder la pregunta



Una manera de responder esta pregunta es ejecutar dos pasos. El primero es buscar el identificador del género «*Salsa*» en la tabla `Géneros` y luego, en segundo lugar, usar este valor para filtrar las filas de la tabla `Canciones` con una expresión condicional en la columna `idGénero`.

Para el primer paso puede utilizarse la consulta mostrada en el Script 4.1, que genera como resultado la Tabla 4.1, es decir, una tabla de una fila y una columna `idGénero` con

el valor «3». Una vez obtenido este último, puede utilizarse en otra consulta que permita obtener las canciones que pertenecen a dicho género, tal como se muestra en el Script 4.2.

***Script* 4.1**

```
SELECT idGénero
FROM Géneros
WHERE nombre = 'Salsa'
```

Tabla 4.1. Resultado de la ejecución del Script 4.1

idGénero
3

***Script* 4.2**

```
SELECT título,
       duración,
       idioma
FROM Canciones
WHERE idGénero = 3
```

El Script 4.2 genera los resultados presentados en la Tabla 4.2, y con esto se resuelve la pregunta planteada. Sin embargo, son notorias las dificultades de este enfoque. ¿Qué pasa si cambia el identificador utilizado para el género «Salsa»?, ¿qué sucede si al utilizar el identificador del género en la segunda consulta hay un error de digitación? Estas y otras preguntas permiten llegar a la conclusión de que no es conveniente este procedimiento porque el código es altamente dependiente de los datos. Esta consulta podría calificarse como de «codificación dura» o «hard-code».

Tabla 4.2. Resultado de la ejecución del Script 4.2

título	duración	idioma
Una aventura	00:05:16	Español
Gotas de lluvia	00:05:54	Español
Ne me quitte pas	00:05:37	Francés
Cali es sabrosura	00:03:14	español

Las subconsultas son una alternativa para resolver este tipo de preguntas porque las dos consultas pueden ser integradas de la forma en que se muestra en el Script 4.3. La consulta del Script 4.2 se convierte así en la consulta externa, y la del Script 4.1 pasa a ser la consulta interna o subconsulta. La consulta externa es la que utiliza el valor obtenido por la subconsulta como parte de una expresión condicional dentro de la cláusula `WHERE`.

***Script* 4.3**

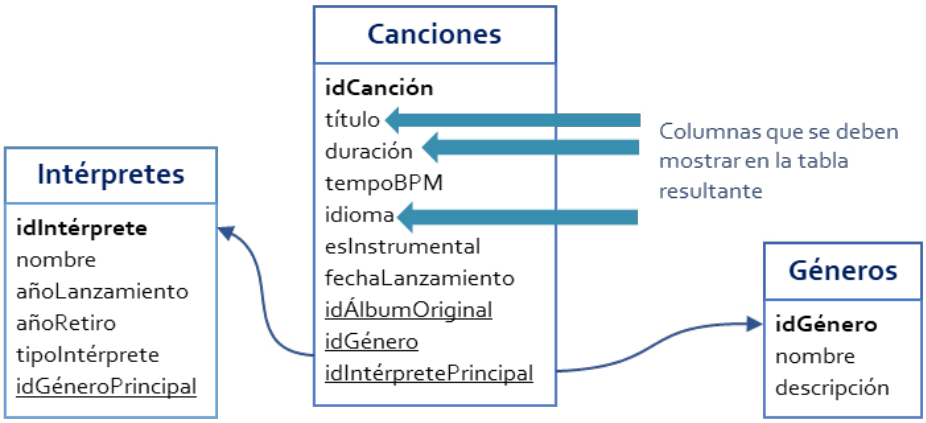
```
SELECT título,
       duración,
       idioma
FROM Canciones
WHERE idGénero = (SELECT idGénero
                  FROM Géneros
                  WHERE nombre = 'Salsa')
```

Este tipo de subconsulta autónoma genera una tabla de una fila con una columna, la cual puede utilizarse en cualquier lugar de la consulta principal en donde se requiera un valor escalar. Para ampliar el aprendizaje de este tipo de subconsultas, se propone trabajar en la siguiente pregunta:

¿Cuáles son los títulos, la duración y el idioma de las canciones del género «*Salsa*» cuyo intérprete principal es Yuri Buenaventura?

Al analizar la pregunta se nota que es muy parecida a la resuelta con el Script 4.3, solo que en esta se agrega una condición. Además de pertenecer al género «Salsa», las canciones por mostrar deben ser interpretadas por «Yuri Buenaventura». En la Figura 4.2 se presenta un diagrama con las tablas requeridas para resolver la pregunta.

Figura 4.2. Tablas y columnas requeridas para responder la pregunta



En este sentido, para construir la solución se requiere una subconsulta que obtenga el `idGénero` del género «*Salsa*», igual a la del *Script* 4.1, y otra que obtenga el `idIntérprete` de «*Yuri Buenaventura*», tal y como se muestra en el *Script* 4.4. Con esas subconsultas puede construirse la consulta que soluciona la pregunta, descrita en el *Script* 4.5, la cual genera como resultado el conjunto de datos presentado en la Tabla 4.3.

***Script* 4.4**

```
SELECT idIntérprete
FROM Intérpretes
WHERE nombre = 'Yuri Buenaventura'
```

***Script* 4.5**

```
SELECT título,
       duración,
              idioma
FROM Canciones
WHERE idGénero = (SELECT idGénero
                  FROM Géneros
                  WHERE nombre = 'Salsa') AND
       idIntérpretePrincipal = (SELECT idIntérprete
                                FROM Intérpretes
                                WHERE nombre = 'Yuri Buenaventura')
```

Tabla 4.3. Resultado de la ejecución del *Script* 4.5

Título	duración	idioma
Ne me quitte pas	00:05:37	Francés
Cali es sabrosura	00:03:14	Español

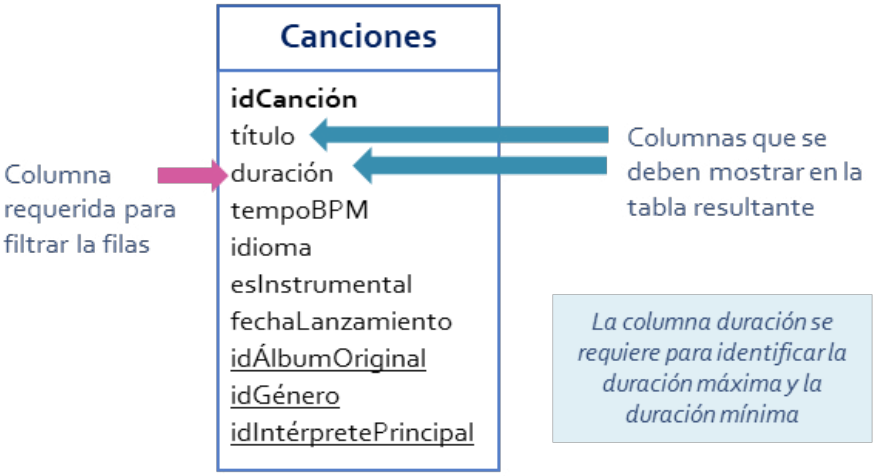
En las subconsultas pueden utilizarse cualquiera de las operaciones que se han trabajo hasta el momento, como es el caso de las funciones de agregación. Para ilustrar este uso, se propone abordar la siguiente pregunta:

¿Cuál es el título y la duración de las canciones más «largas» y más «cortas»?

Al analizar esta pregunta se identifica que los datos necesarios para responderla están almacenados en la tabla `Canciones`, como se muestra en la Figura 4.3.

Inicialmente, debe identificarse cuál es el mayor tiempo de duración que tienen las canciones, lo cual puede obtenerse con la función de agregación `MAX` sobre todas las filas, es decir, sin crear grupos, tal como se aprecia en el Script 4.6. De igual forma, en el Script 4.7 se busca el menor tiempo de reproducción de todas las canciones.

Figura 4.3. Tablas y columnas requeridas para responder la pregunta



***Script* 4.6**

```
SELECT MAX(duración)
FROM Canciones
```

***Script* 4.7**

```
SELECT MIN(duración)
FROM Canciones
```

El siguiente paso es encontrar esas canciones que tienen en la columna de duración alguno de los valores obtenidos con las dos consultas anteriores. Esto puede realizarse con el operador `IN` de la forma en que se presenta en el Script 4.8.

***Script* 4.8**

```
SELECT título,
       duración
FROM Canciones
WHERE duración IN((SELECT MIN(duración)
                   FROM Canciones),
                  (SELECT MAX(duración)
                   FROM Canciones))
```

Las subconsultas que devuelven una fila pueden incluirse en cualquier expresión donde se acepte un escalar. Para ilustrar esta posibilidad se propone la siguiente pregunta:

¿Cuál es el identificador y la cantidad de reproducciones de las canciones cuya duración promedio de reproducción durante el año 2020 es menor a la duración promedio de reproducción de todas las canciones?

Para resolver esta necesidad de datos deben utilizarse únicamente los datos almacenados en la tabla `Reproducciones`, tal como se observa en la Figura 4.4. La consulta requerida puede pensarse inicialmente calculando la duración promedio por reproducción de todas las canciones, según se plantea en el Script 4.9.

Figura 4.4. Tablas y columnas requeridas para responder la pregunta

***Script* 4.9**

```
SELECT AVG(segundosreproducidos)
FROM reproducciones
```

También debe calcularse el tiempo de reproducción promedio para cada una de las canciones. Esto se consigue agrupando las reproducciones por el identificador de la canción o `idCanción` y aplicando la función AVG, como se muestra en el *Scrip*t 4.10.

***Script* 4.10**

```
SELECT idcanción,
       AVG(segundosreproducidos)
FROM reproducciones
WHERE DATE_PART('year', fechareproducción) = 2020
GROUP BY idcanción
```

Una vez obtenido el promedio de los segundos reproducidos para cada una de las canciones, el siguiente paso es dejar aquellas en donde el promedio no supere el valor obtenido con el *Script* 4.9. Esto puede hacerse filtrando los grupos en la cláusula HAVING con el resultado de la subconsulta, como se muestra en el *Script* 4.11.

***Script* 4.11**

```
SELECT idcanción,
       COUNT(*) AS cantidadReproducciones
FROM reproducciones
WHERE DATE_PART('year', fechareproducción) = 2020
GROUP BY idcanción
HAVING AVG(segundosreproducidos) < (SELECT AVG(segundosreproducidos)
                                    FROM reproducciones)
```

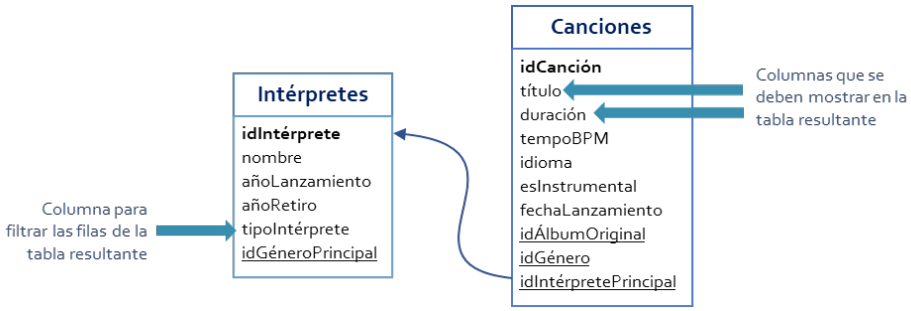
Hasta este punto se han realizado subconsultas para obtener un escalar que es utilizado en una expresión condicional. En tal sentido, todas las consultas internas generan una tabla compuesta por una fila y una columna. No obstante, hay otros casos en los cuales se requieren subconsultas que generen más de un valor; por ejemplo, una lista de valores. Para abordar esta posibilidad, se propone la siguiente pregunta:

¿Cuál es el título y la duración de las canciones interpretadas por solistas?

Mostrar los resultados ordenados alfabéticamente por el título de las canciones.

Desde el inicio se ha argumentado que antes de responder cualquier necesidad de datos es recomendable un análisis para determinar lo que se está solicitando y si es posible descomponer el problema en partes más pequeñas. En este caso, se buscan datos de canciones, pero no de todas; solamente aquellas que sean interpretadas por solistas. En la Figura 4.5 se presenta un diagrama con las tablas requeridas para resolver la pregunta.

Figura 4.5. Tablas y columnas requeridas para responder la pregunta



Para la pregunta anterior lo primero sería obtener el identificador de aquellos intérpretes del tipo «*Solista*». Los datos necesarios están almacenados en la tabla `Intérpretes`, y puede utilizarse la consulta del *Script* 4.12 para obtener esa lista de identificadores de intérpretes.

***Script* 4.12**

```
SELECT idIntérprete
FROM Intérpretes
WHERE tipointérprete = 'Solista'
```

El resultado de ejecutar el Script 4.12 es una tabla con una columna y cinco filas con los identificadores de intérprete 1, 3, 5, 7 y 9. Como este resultado obtenido es una lista de valores, no puede incluirse en un lugar de la consulta principal en donde se espere un escalar, como es el caso de una expresión condicional con operadores de comparación como igual (=), mayor que (>), menor o igual que (<=), entre otros. Sin embargo, hay que recordar que en el SQL también existen operadores que reciben una lista de valores, como el operador `IN`.

En este sentido, una forma de responder la pregunta es ejecutando la consulta del *Script* 4.13, en la cual se utilizan directamente los valores 1, 3, 5, 7, 9 en la expresión condicional conformada para el operador `IN`. El resultado de ejecutar el *script* se presenta en la Tabla 4.4.

***Script* 4.13**

```
SELECT título,
       duración
FROM Canciones
WHERE idIntérpretePrincipal IN (1, 3, 5, 7, 9)
ORDER BY título
```

Una mejor alternativa para responder la pregunta es utilizar la consulta del *Script* 4.12 como una subconsulta que permite obtener los valores definidos en el operador `IN`. En otras palabras, la consulta del *Script* 4.13 se tomaría como la principal, y la del *Script* 4.12 sería la consulta interna que genera los identificadores de los intérpretes de tipo «*Solista*». Esta solución del caso se presenta en el *Script* 4.14.

Tabla 4.4. Tabla resultante del *Script* 4.13

título	duración
Amarte más no pude	00:04:49
Cali es sabrosura	00:03:14
Ginza	00:02:51
Hips don't lie	00:03:38
La bicicleta	00:03:47
La tierra del olvido	00:04:25
Mi gente	00:03:09
Ne me quitte pas	00:05:37
Ojos así	00:03:55
Sin medir distancias	00:04:58

***Script* 4.14**

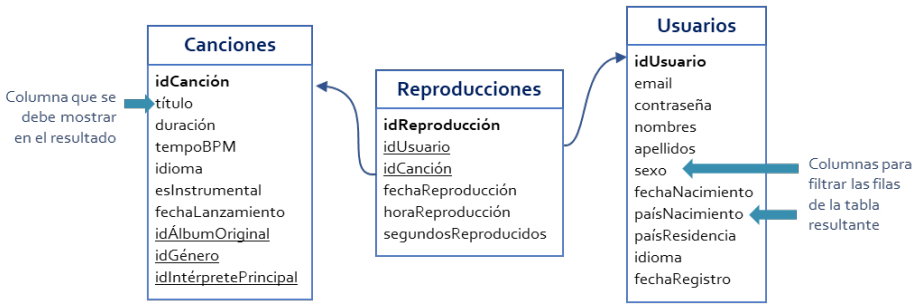
```
SELECT título,
       duración,
       idIntérpretePrincipal
FROM Canciones
WHERE idIntérpretePrincipal IN (SELECT idIntérprete
                                FROM Intérpretes
                                WHERE tipoIntérprete =     'Solista')
ORDER BY título
```

Hasta el momento se han tratado casos en los que hay una consulta externa y una o más consultas internas al mismo nivel de anidamiento o profundidad. Sin embargo, en ocasiones se requieren subconsultas que también tienen subconsultas. En otras palabras, hay situaciones en las cuales es posible tener una consulta interna que a su vez tiene otra consulta interna. Incluso pueden tenerse varios niveles de anidamiento o de profundidad. Para ilustrar esto, se propone la siguiente pregunta:

¿Cuál es el título de las canciones reproducidas por mujeres nacidas en Colombia?

En esta pregunta se pide únicamente un dato de las canciones: su título, pero se requieren datos de otras tablas para filtrar las canciones que deben incluirse. En la Figura 4.6 se presenta un diagrama con las tablas requeridas para resolver la pregunta.

Figura 4.6. Tablas y columnas requeridas para responder la pregunta



Para responder a esta necesidad de datos puede descomponerse el problema en tres niveles. En primer lugar, se requiere identificar los *usuarios* de sexo femenino nacidos en Colombia, lo cual puede hacerse con la consulta del *Script* 4.15.

***Script* 4.15**

```
SELECT idUsuario
FROM Usuarios
WHERE sexo = 'F' AND paísNacimiento = 'Colombia'
```

Con esa lista de identificadores de `Usuarios` se procede a obtener las `Reproducciones` que han realizado, enfocándose específicamente en los identificadores de las canciones. Para esto puede utilizarse el *Script* 4.16, en donde se encuentra la subconsulta del *Script* 4.15 para generar los `idUsuario` requeridos por la consulta externa. Así se obtienen los `idCanción` de las canciones reproducidas por mujeres nacidas en Colombia.

***Script* 4.16**

```
SELECT idCanción
FROM Reproducciones
WHERE idUsuario IN (SELECT idUsuario
                    FROM Usuarios
                    WHERE sexo = 'F' AND
                          paísNacimiento = 'Colombia')
```

Por último, para obtener los títulos de las canciones se consulta la tabla `Canciones` utilizando los `idCanción` obtenidos con el *Script* 4.16. En este sentido, la consulta completa para responder la pregunta se presenta en el *Script* 4.17. Este es un ejemplo de una estructura de anidamiento de tres niveles de profundidad, en donde se observa que los conceptos «*consulta externa*» y «*consulta interna*» se aplican de forma relativa: una consulta que puede considerarse como *interna* con respecto a otra puede ser *externa* para un nivel mayor de anidamiento o profundidad.

***Script* 4.17**

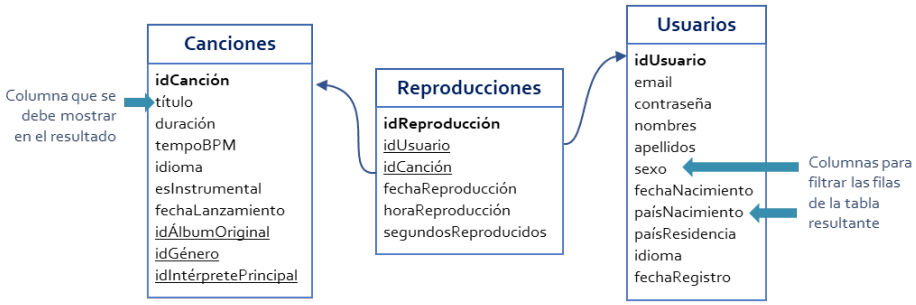
```
SELECT título
FROM Canciones
WHERE idCanción IN (SELECT idCanción
                    FROM Reproducciones
                    WHERE idUsuario IN (SELECT idUsuario
                                        FROM Usuarios
                                        WHERE sexo = 'F' AND
                                              paísNacimiento = 'Colombia'))
```

En una consulta pueden utilizarse subconsultas que obtienen un valor y subconsultas que obtienen una lista de valores. Para analizar este caso se planteará una variación a la pregunta resuelta con el *Script* 4.17 en los siguientes términos:

¿Cuál es el título de las canciones del género salsa reproducidas por mujeres nacidas en Colombia?

En esta pregunta también se pide únicamente un dato de las canciones, su título, pero se requieren datos de otras tablas para filtrar las canciones que deben incluirse. En la Figura 4.7 se presenta un diagrama con las tablas necesarias para resolver la solicitud.

Figura 4.7. Tablas y columnas requeridas para responder la pregunta



Para resolver esta pregunta puede tomarse como base el *Script* 4.17 y agregar el *Script* 4.1 como una subconsulta para obtener el `idGénero` correspondiente al género de nombre «Salsa». La consulta SQL resultante se presenta en el *Script* 4.18, cuya ejecución genera la Tabla 4.5.

El nivel de anidamiento puede ampliarse tanto como sea necesario. Cuando se tienen consultas con varios niveles de profundidad, puede entenderse como si se estuviesen recorriendo las tablas utilizando las claves foráneas o externas y las claves principales en las expresiones condicionales con las que se filtran las filas.

***Script* 4.18**

```
SELECT título
FROM Canciones
WHERE idCanción IN (SELECT idCanción
                    FROM Reproducciones
                    WHERE idUsuario IN (SELECT idUsuario
                                        FROM Usuarios
                                        WHERE sexo = 'F' AND
                                            paísNacimiento = 'Colombia'))
          AND idGénero = (SELECT idGénero
                          FROM Géneros
                          WHERE nombre = 'Salsa')
```

Tabla 4.5. Tabla resultante del *Script* 4.18

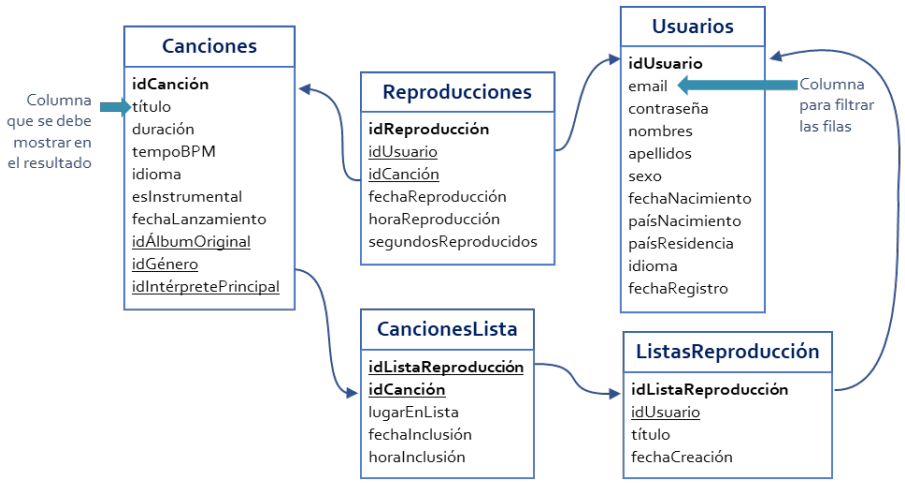
título
Una aventura
Gotas de lluvia
Ne me quitte pas
Cali es sabrosura

En algunos casos el nivel de anidamiento puede ser alto, generando una estructura de subconsultas que podría llegar a ser confusa a primera vista. Sin embargo, este tipo de consultas son fáciles de entender si se aplica la noción de descomposición de los problemas y se van obteniendo resultados parciales con datos que permiten avanzar hacia la solución completa para responder adecuadamente a la necesidad de datos. Para ilustrar esta situación se propone abordar la siguiente pregunta:

¿Cuál es el título de las canciones que están incluidas en alguna lista de reproducción del usuario identificado con el *e-mail* alex@gmail.com, pero que este nunca ha reproducido?

Esta pregunta representa la base de una funcionalidad muy común en diversas aplicaciones de este tipo. Normalmente, se le recomienda al usuario reproducir las canciones que en algún momento agregó a alguna de sus listas de reproducción pero que no ha escuchado hasta el momento. El resultado esperado es, simplemente, un listado de canciones que cumplen dos condiciones: la primera es estar incluidas en alguna de las listas de reproducción del usuario, y la segunda es que dicho usuario nunca las haya reproducido. En la Figura 4.8 se presenta un diagrama con las tablas requeridas para resolver la pregunta.

Figura 4.8. Tablas y columnas requeridas para responder la pregunta



La primera condición demanda una subconsulta para obtener los `idCanción` que están incluidos en algunas de las listas de reproducción creadas por el usuario. Para la segunda, se requiere otra subconsulta que indique los `idCanción` de las canciones que han sido reproducidas por el usuario. La consulta se presenta en el *Script* 4.19.

***Script* 4.19**

```
SELECT título
FROM Canciones
WHERE idCanción IN (SELECT idCanción
                    FROM CancionesLista
                    WHERE idListaReproducción IN
                      (SELECT idListaReproducción
                       FROM ListasReproducción
                       WHERE idUsuario IN
                         (SELECT idUsuario
                          FROM Usuarios
                          WHERE email = 'alex@gmail.com'))) AND
      idCanción NOT IN (SELECT idCanción
                        FROM Reproducciones
                        WHERE idUsuario IN
                          (SELECT idUsuario
                           FROM Usuarios
                           WHERE email = 'alex@gmail.com'))
```

En el SQL se incluyen los constructos `ANY`, `SOME` y `ALL`, los cuales pueden combinarse con los operadores de comparación en una expresión para evaluar una condición con relación a un conjunto o lista de valores. Así, si se requiere verificar que un valor está contenido en una lista de valores, puede utilizarse el operador `=` en conjunto con la palabra `ANY`. Asimismo, cuando se necesita comprobar que un valor no está contenido en una lista de valores, puede utilizarse el operador `<>` (diferente de) en conjunto con la palabra `ALL` o con la palabra `SOME`, dado que son equivalentes.

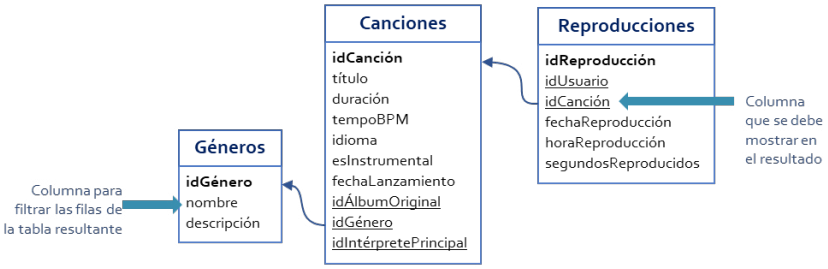
En este sentido, la consulta del *Script* 4.19 podría modificarse reemplazando la expresión `IN` con `= ANY` (*es igual a alguno*), y la expresión `NOT IN`, con la expresión `<> ALL` *(es diferente de todos)*, lo que arrojaría el mismo resultado. Los constructos `ANY`, `SOME` y `ALL` abren muchas posibilidades de comparación de un valor en relación con un conjunto o lista de valores escalares.

La combinación de los constructos `ANY`, `SOME` y `ALL` con operadores de comparación permite especificar expresiones condicionales para determinar, por ejemplo, si un valor es mayor que alguno de los valores resultantes de una subconsulta. Para aplicar esto se propone abordar la siguiente pregunta:

¿Cuáles son los *idCanción* de las canciones del género «*Salsa*» que tienen más reproducciones que alguna de las canciones del género «*Pop*»?

Para resolver esta pregunta, debe calcularse la cantidad de reproducciones que ha tenido cada canción del género «*Salsa*» y determinar si es mayor que la cantidad de reproducciones de alguna de las canciones del género «*Pop*». En la Figura 4.9 se presenta un diagrama con las tablas requeridas para resolver la pregunta.

Figura 4.9. Tablas y columnas requeridas para responder la pregunta



En este sentido, el problema puede dividirse para ir construyendo la consulta por partes. En primer lugar, es posible construir una consulta para obtener las cantidades de reproducciones que han tenido las canciones del género «*Salsa*», tal y como se presenta en el *Script* 4.20.

***Script* 4.20**

```
SELECT idCanción, COUNT(*)
FROM Reproducciones
WHERE idCanción IN (SELECT idCanción
                    FROM Canciones
                    WHERE idGénero = (SELECT idGénero
                                      FROM Géneros
                                      WHERE nombre = 'Salsa'))
GROUP BY idCanción
```

Luego hay que agregar un filtro aplicado luego del agrupamiento, es decir, con la cláusula `HAVING`, con una subconsulta que obtenga las cantidades de reproducciones que han tenido las canciones del género «*Pop*», tal y como se presenta en el *Script* 4.21. La consulta principal, basada en la consulta del *Script* 4.20, tiene una ligera modificación al no incluir en la tabla resultante la cantidad obtenida con la función de agregación `COUNT`. Dicha cantidad solamente se utiliza en la cláusula `HAVING`.

Ahora bien, aunque el uso dado a las subconsultas hasta este punto ha sido el de obtener un valor escalar o una lista de valores, su utilidad no se restringe a esto. También es posible utilizar una subconsulta en la cláusula `FROM` dado que el resultado de toda consulta, según lo especificado en el modelo relacional, es una tabla. Para ilustrar este uso de las subconsultas se propone la siguiente pregunta:

***Script* 4.21**

```
SELECT idCanción
FROM Reproducciones
WHERE idCanción IN (SELECT idCanción
                    FROM Canciones
                    WHERE idGénero = (SELECT idGénero
                                      FROM Géneros
                                      WHERE nombre = 'Salsa'))
GROUP BY idCanción
HAVING COUNT(*) > ANY (SELECT COUNT(*)
                       FROM Reproducciones
                       WHERE idCanción IN (SELECT idCanción
                                           FROM Canciones
                                           WHERE idGénero =
                                            (SELECT idGénero
                                             FROM Géneros
                                             WHERE nombre = 'Pop'))
                       GROUP BY idCanción)
```

¿Cuántas canciones tiene la lista de reproducción con mayor número de canciones?

Para resolver esta pregunta se hace necesario calcular la cantidad de canciones de cada lista y luego seleccionar el número mayor. En la Figura 4.10 se presenta un diagrama con la tabla requerida para resolver la pregunta.

Figura 4.10. Tabla y columnas requeridas para responder la pregunta

Estos datos solicitados pueden obtenerse fácilmente utilizando la función de agregación COUNT y agrupando las filas de la tabla `CancionesLista`, tal y como se presenta en el *Script* 4.22. El resultado de la ejecución de esta consulta se encuentra en la Tabla 4.6.

Tomando como base el resultado de ejecutar el *Script* 4.22 —es decir, la Tabla 4.6—, debe obtenerse el valor máximo de la columna `cantidadCanciones`. Normalmente, para ese fin se utiliza la función de agregación MAX; sin embargo, este caso es diferente porque la Tabla 4.6 no existe en la base de datos. En ese sentido, y con miras a completar la solución a esta

pregunta, debe recordarse que el resultado de una consulta es tratado por el SQL como una tabla derivada. Es decir, puede pensarse como si se creara una tabla temporal que contiene los datos resultantes de la consulta. Por ende, es posible incluir la consulta del *Script* 4.22 como una subconsulta en la cláusula `FROM` asignándole un alias, y la consulta principal la puede utilizar de la misma forma en que lo hace con una tabla que existe físicamente en la base de datos.

***Script* 4.22**

```
SELECT idListaReproducción,
       COUNT(idCanción) AS cantidadCanciones
FROM CancionesLista
GROUP BY idListaReproducción
```

Tabla 4.6. Resultado de la ejecución del *Script* 4.22

idListaReproducción	cantidadCanciones
14	8
17	11
8	8
12	6
15	5
1	9
10	12
11	7
4	11
18	7
16	11
6	10
19	6
2	8
3	4
20	10
13	6
5	8
9	2
7	14

En el *Script* 4.23 se muestra cómo se construye la consulta principal utilizando el *Script* 4.22 como una consulta interna o subconsulta en la cláusula `FROM` con el alias `Canciones-Porlista`. Con esto se obtiene el máximo valor de la columna `cantidadCanciones` de la Tabla 4.6, es decir, el número «14».

***Script* 4.23**

```
SELECT MAX(cantidadCanciones)
FROM (SELECT idListaReproducción,
             COUNT(idCanción) AS cantidadCanciones
      FROM CancionesLista
      GROUP BY idListaReproducción) AS CancionesPorlista
```

Otro caso común en el que se requieren subconsultas en la cláusula `FROM` es cuando deben aplicarse funciones de agregación a diferentes niveles. Para ilustrar esto se propone la siguiente pregunta:

¿Cuántas listas de reproducción tienen más de diez canciones?

Para resolver esta pregunta se hace necesario calcular la cantidad de canciones de cada lista y luego filtrar aquellas en las que el resultado sea mayor a diez. En la Figura 4.11 se presenta un diagrama con la tabla requerida para este caso.

Figura 4.11. Tabla y columnas requeridas para responder la pregunta

Si la pregunta iniciara con la palabra «cuáles», la solución sería relativamente sencilla; bastaría con utilizar la función `COUNT` y aplicar un filtro con la cláusula `HAVING`, tal y como se presenta en el *Script* 4.24.

***Script* 4.24**

```
SELECT idListaReproducción,
       COUNT(idCanción) AS cantidadCanciones
FROM Cancioneslista
GROUP BY idListaReproducción
HAVING COUNT(idCanción) > 10
```

No obstante, la respuesta esperada no es un listado de los `idListaReproducción` y la cantidad de canciones de cada lista, que debe ser mayor que diez, sino un número entero que indique la cantidad de listas de reproducción que cumplen la condición. En otras palabras,

para obtener la respuesta deben contarse las filas resultantes obtenidas con el *Script* 4.24, lo cual se logra con la consulta presentada en el *Script* 4.25.

***Script* 4.25**

```
SELECT COUNT(*) AS cantidadListas
FROM (SELECT idListaReproducción,
             COUNT(idCanción) AS cantidadCanciones
       FROM Cancioneslista
       GROUP BY idListaReproducción
       HAVING COUNT(idCanción) > 10) AS ListasConCantidadCanciones
```

Una alternativa es ubicar en la consulta principal el filtrado de filas. Es decir, no usar la cláusula `HAVING` y aplicar la cláusula `WHERE`, como se muestra en el *Script* 4.26.

***Script* 4.26**

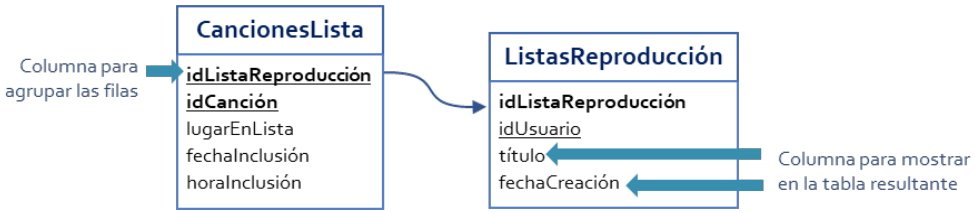
```
SELECT COUNT(*) AS cantidadListas
FROM (SELECT idListaReproducción,
             COUNT(idCanción) AS cantidadCanciones
       FROM Cancioneslista
       GROUP BY idListaReproducción) AS ListasConCantidadCanciones
WHERE cantidadCanciones > 10
```

Todos los usos de subconsultas autónomas pueden aplicarse en una misma consulta. Para mostrar eso se propone la siguiente pregunta:

¿Cuál es el título y la fecha de creación de las listas de reproducción con mayor número de canciones?

Para resolver esta pregunta puede utilizarse como punto de partida el *Script* 4.23, con el cual se obtuvo la máxima cantidad de canciones que tiene alguna lista de reproducción. Al respecto, es necesario precisar que varias listas podrían tener la misma cantidad de canciones y coincidir con que dicha cantidad es la máxima. En la Figura 4.12 se presenta un diagrama con las tablas requeridas en esta solicitud.

Figura 4.12. Tablas y columnas requeridas para responder la pregunta



En este sentido, el *Script* 4.23 se utilizará como una subconsulta, y el valor que arroje se incluirá dentro de una expresión condicional para filtrar las filas de una consulta que mostrará los identificadores de las listas de reproducción que tengan la cantidad de canciones mencionada. En este caso, la subconsulta se utilizará dentro de la cláusula `HAVING` de la consulta externa, tal y como se presenta en el *Script* 4.27.

***Script* 4.27**

```
SELECT idListaReproducción
FROM Cancioneslista
GROUP BY idListaReproducción
HAVING COUNT(idCanción) = (SELECT MAX(cantidadCanciones)
                           FROM (SELECT idListaReproducción,
                                        COUNT(idCanción) AS cantidadCanciones
                                   FROM CancionesLista
                                   GROUP BY idListaReproducción
                                  ) AS CancionesPorlista)
```

El resultado del *Script* 4.27 es un listado de identificadores de listas de reproducción que pueden utilizarse en una expresión condicional para filtrar filas de la tabla `ListasReproducción`. Esto es necesario porque los datos solicitados, es decir, el título y la fecha de creación, están almacenados en esa tabla. Por consiguiente, el *Script* 4.27 será empleado como subconsulta en la cláusula `WHERE` de la consulta principal, tal y como se presenta en el en el *Script* 4.28.

***Script* 4.28**

```
SELECT título,
       fechaCreación
FROM ListasReproducción
WHERE idListaReproducción IN (SELECT idListaReproducción
                              FROM Cancioneslista
                              GROUP BY idListaReproducción
                              HAVING COUNT(idCanción) =
                                     (SELECT MAX(cantidadCanciones)
                                      FROM (SELECT idListaReproducción,
                                                   COUNT(idCanción) AS cantidadCanciones
                                            FROM CancionesLista
                                            GROUP BY idListaReproducción
                                       ) AS CancionesPorlista))
```

4.2. Subconsultas correlacionadas

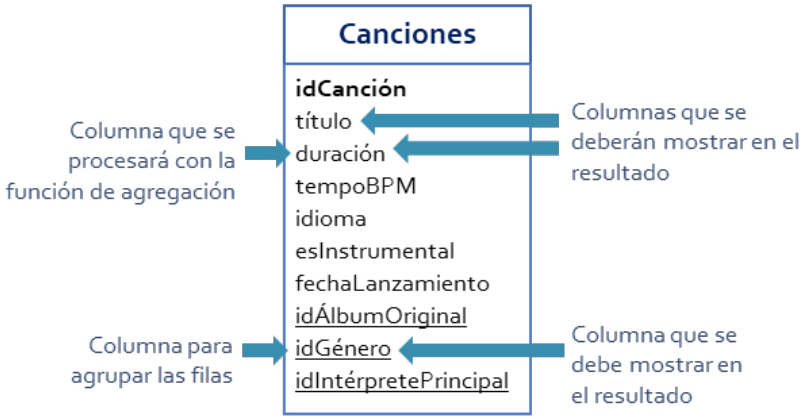
Las subconsultas creadas hasta este punto no han dependido de la consulta externa para su ejecución. Es decir, la consulta interna puede ejecutarse de manera autónoma y siempre

generará un resultado independiente de lo que suceda en la consulta principal. Sin embargo, hay casos en los cuales esto no es suficiente para obtener la respuesta esperada. Para ilustrar esta situación se propone abordar la siguiente pregunta:

¿Cuál es el identificador del género, el título y la duración de las canciones de mayor duración en cada género?

Un primer acercamiento a la solución puede darse pensando en términos de subconsultas autónomas. Específicamente, cabría la posibilidad de utilizar una subconsulta que indique la duración máxima que tienen las canciones de un género en particular y, así, filtrar las filas de la tabla `Canciones` que tengan almacenado ese valor en la columna `duración`. En la Figura 4.13 se presenta un diagrama con la tabla requerida para resolver la pregunta.

Figura 4.13. Tabla y columnas requeridas para responder la pregunta



El *Script* 4.29 muestra la forma de implementar esta consulta y obtener las canciones de mayor duración del género con identificador «4».

***Script* 4.29**

```
SELECT idGénero,
       título,
       duración
FROM Canciones
WHERE duración = (SELECT MAX(duración)
                  FROM Canciones
                  WHERE idGénero = 4)
```

La solución parcial del *Script* 4.29 permite responder la pregunta para un solo género; específicamente, el `idGénero` con valor *4*. Con este enfoque de solución, para procesar los

demás géneros tendría que cambiarse manualmente el valor *4* por el de cada `idGénero` y así registrar las soluciones parciales. No obstante, esto, además de ser impráctico, sería insostenible y estaría en contradicción con el paradigma desde el cual está planteado el SQL, es decir, definir qué se quiere sin especificar cómo obtenerlo.

En este caso es posible implementar un concepto que permite que la subconsulta tenga un comportamiento diferente para distintos valores de las filas de la consulta principal. Particularmente, se requiere que el valor obtenido con la subconsulta corresponda a la duración máxima de las canciones con `idGénero` igual al `idGénero` de la canción por la que se está evaluando la expresión condicional de la cláusula `WHERE` en la consulta principal. Esto se conoce como correlacionar la subconsulta a algún valor de cada fila de la consulta principal. Así pues, el *Script* 4.30 está basado en el *Script* 4.29 e implementa la correlación entre la subconsulta y la consulta principal con la expresión condicional de la subconsulta en relación con las columnas `idGénero`.

***Script* 4.30**

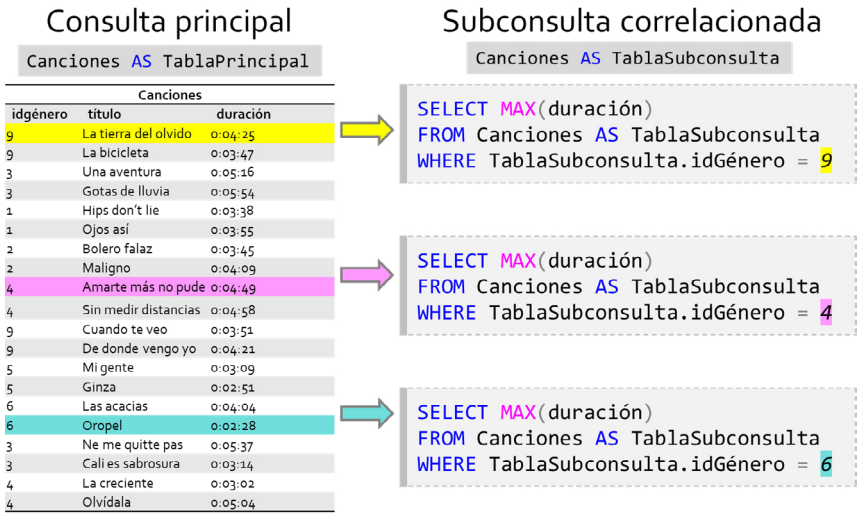
```
SELECT idGénero,
       título,
       duración
FROM Canciones AS TablaPrincipal
WHERE duración = (SELECT MAX(duración)
                  FROM Canciones AS TablaSubconsulta
                  WHERE TablaSubconsulta.idGénero = TablaPrincipal.idGénero)
```

En la consulta del *Script* 4.30 se aprecia que se han utilizado alias para renombrar las tablas porque en la subconsulta se hace referencia al `idGénero` dos veces. La expresión condicional de la cláusula `WHERE` de la subconsulta filtra las filas de la tabla `Canciones`, identificada con el alias `TablaSubconsulta`, antes de obtener el valor máximo de la duración con la función de agregación `MAX`. La subconsulta se ejecuta por cada fila de la tabla `Canciones` bajo el alias `TablaPrincipal`, es decir, en el ámbito de ejecución de la consulta principal.

En la Figura 4.14 se ilustra el proceso de ejecución de esta consulta y la subconsulta correlacionada. Cuando se va a evaluar la expresión condicional para determinar si la primera fila de la tabla se incluye o no en el resultado, debe ejecutarse la subconsulta tomando el `idGénero` de esa fila, es decir, 9.

Hay que recordar que asignar el alias no implica que se esté cambiando el nombre de la tabla. También debe estar claro que utilizar la misma tabla en la subconsulta y asignarle otro alias no significa que se modifique la tabla o que se esté haciendo referencia a un conjunto de datos diferentes. Simplemente, son dos ámbitos de ejecución distintos en los que se está utilizando la misma tabla.

Figura 4.14. Ejecución de la subconsulta correlacionada del *Script* 4.30



De forma general, puede observarse que en las subconsultas correlacionadas la consulta interna es dependiente de la consulta externa y no puede ser invocada de forma autónoma. En consecuencia, para las subconsultas correlacionadas, el DBMS evalúa la consulta interna una vez por cada fila de la consulta externa.

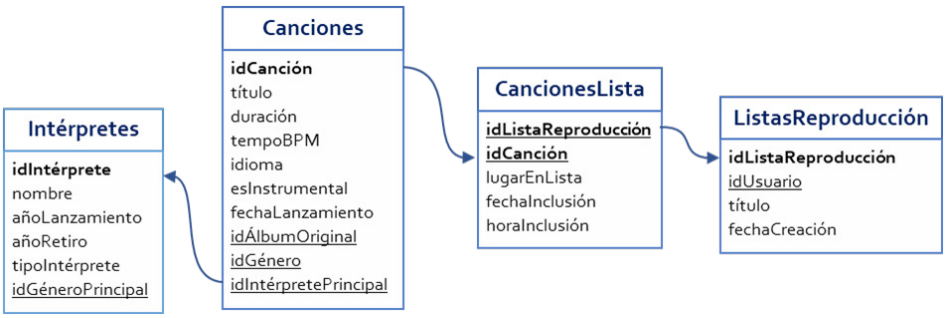
Las subconsultas correlacionadas también pueden utilizarse para obtener datos de más de una tabla. Un caso de aplicación se da al resolver la siguiente pregunta:

¿Cuáles son los títulos de las canciones interpretadas por Carlos Vives que han sido agregadas a listas de reproducción y los nombres de dichas listas?

La respuesta esperada es una tabla con dos columnas: en la primera se presentará el título de la canción, y en la segunda, el nombre de la lista de reproducción en la cual está agregada dicha canción. Si una canción interpretada por *«Carlos Vives»* está incluida en varias listas de reproducción, entonces deberá aparecer en varias filas de la tabla resultante. De igual forma debe ocurrir con el nombre de una lista de reproducción si en ella hay varias canciones interpretadas por *«Carlos Vives»*.

En la tabla `CancionesLista` están almacenados los identificadores de las listas de reproducción y los de las canciones que pertenecen a las listas. En esta tabla, no obstante, no se encuentran los títulos de las canciones ni los de las listas de reproducción. Por consiguiente, es necesario utilizar un enfoque que permita obtener datos de diferentes tablas —en este caso, las tablas `Canciones` y `ListasReproducción`— para conformar la tabla resultante con base en los datos almacenados en la tabla `CancionesLista`. En la Figura 4.15 se presenta un diagrama con la tabla requerida para resolver la pregunta.

Figura 4.15. Tablas y columnas requeridas para responder la pregunta



En la tabla `CancionesLista` están los datos de todas las canciones que conforman las listas de reproducción. Sin embargo, la pregunta se centra únicamente en las canciones interpretadas por *«Carlos Vives»*. En tal sentido, el primer paso hacia la solución es filtrar las filas de la tabla `CancionesListas` para dejar únicamente aquellas que correspondan a canciones interpretadas por dicho artista.

***Script* 4.31**

```
SELECT idListaReproducción,
       idCanción
FROM CancionesLista
WHERE idCanción IN (SELECT idCanción
                    FROM Canciones
                    WHERE idIntérpretePrincipal IN (SELECT idIntérprete
                                                    FROM Intérpretes
                                                    WHERE
                                                    nombre='Carlos Vives'))
```

Lo anterior puede obtenerse con la consulta del *Script* 4.31, en la cual se utilizan dos niveles de subconsulta para generar una tabla con dos columnas: el identificador de la lista de reproducción y el de la canción interpretada por *«Carlos Vives»* que está contenida en dicha lista. Ahora solamente falta presentar los títulos de las listas de reproducción y los de las canciones en lugar de mostrar los identificadores.

Para llegar a esta solución puede pensarse inicialmente en el problema de obtener el título de una canción a partir del identificador, el cual se resuelve con una consulta como la presentada en el *Script* 4.32 para la canción con `idCanción` igual a 5. De la misma forma, cabe considerar la forma de obtener el título de una lista de reproducción a partir del identificador, como se muestra en el *Script* 4.33 para la lista de reproducción con `idListaReproducción` igual a 3.

Las dos consultas presentadas antes pueden ser incluidas como subconsultas correlacionadas de la consulta principal presentada en el *Script* 4.31 en lugar de las columnas `idListaReproducción` e `idCanción`, reemplazando el valor escalar específico por la referencia a la columna de la consulta principal.

***Script* 4.32**

```
SELECT título
FROM Canciones
WHERE idCanción = 5
```

***Script* 4.33**

```
SELECT título
FROM ListasReproducción
WHERE idListaReproducción = 3
```

Finalmente, en el *Script* 4.34 se presenta la solución, en la cual se generan las columnas `títuloLista` y `títuloCanción` a partir del resultado de ejecutar las subconsultas correlacionadas en las que se evalúa la expresión condicional para buscar la coincidencia entre el valor del identificador de las filas de la tabla del ámbito de ejecución de la subconsulta y cada fila de la tabla del ámbito de ejecución de la consulta principal. En este caso son tablas diferentes.

***Script* 4.34**

```
SELECT
          (SELECT título
           FROM ListasReproducción
           WHERE ListasReproducción.idListaReproducción =
                 Cancioneslista.idlistareproducción) AS títuloLista,
           (SELECT título
            FROM Canciones
     WHERE Canciones.idCanción = CancionesLista.idCanción) AS títuloCanción
FROM CancionesLista
WHERE idCanción IN (SELECT idCanción
                           FROM Canciones
                           WHERE idIntérpretePrincipal IN (SELECT idIntérprete
                                                           FROM Intérpretes
                                                           WHERE
                                                           nombre='Carlos Vives'))
```

4.3. Aprendizajes más importantes del capítulo 4

- Las subconsultas permiten «navegar» entre las tablas de una base de datos.
- Con las subconsultas se evitan pasos intermedios en el momento de utilizar el resultado de una consulta como entrada de otra.
- Las subconsultas pueden ser autónomas o correlacionadas.
- Las subconsultas que devuelven una tabla de una fila con una columna pueden incluirse en cualquier lugar donde se requiera un valor escalar.

- Las subconsultas pueden insertarse en cláusulas con operadores que utilicen listas de valores. En este caso no se restringe a que deban devolver una sola fila, sino que pueden arrojar múltiples filas, pero siempre una sola columna.
- El resultado de una consulta siempre es una tabla. Por lo tanto, una subconsulta también puede insertarse en la cláusula en donde se definen las tablas participantes de una consulta. En tal caso, la subconsulta puede retornar cualquier cantidad de filas y de columnas.
- Las consultas correlacionadas permiten ejecutar una consulta secundaria por cada una de las filas de la consulta principal. De este modo es posible que la consulta secundaria utilice valores de la consulta principal.

4.4. Actividades de aplicación para evidenciar lo aprendido

1. Explique cuál es la necesidad de datos resuelta con la siguiente consulta:

```
SELECT título,
       idioma
FROM Canciones
WHERE idGénero = (SELECT idGénero
                  FROM Géneros
                  WHERE nombre = 'Pop') AND
            idIntérpretePrincipal = (SELECT idIntérprete
                                     FROM Intérpretes
                                     WHERE nombre = 'Shakira');
```

2. Escriba la consulta que permita obtener la duración promedio por idioma de las canciones que han sido reproducidas en algún momento.
3. Escriba la consulta que permita obtener el nombre y el año de lanzamiento de los intérpretes de todas las canciones que, durante el primer semestre del año 2020, hayan tenido al menos una reproducción con duración de al menos 30 segundos por usuarios de sexo femenino registrados en dicho periodo.
4. ¿Cuáles son los elementos que impiden que la siguiente consulta pueda ejecutarse correctamente?

```
SELECT título FROM Canciones WHERE idcanción IN(SELECT idcanción FROM reproducciones
WHERE idusuario = (SELECT idusuario FROM Usuarios WHERE sexo = 'M' AND paísnacimiento
IN 'Ecuador' OR 'Colombia'))
```

5. Explique cuál es la necesidad de datos que se busca satisfacer con la consulta de la actividad 4 una vez corregida.
6. Explique cuál es la necesidad de datos resuelta con la siguiente consulta:

```
SELECT *
FROM (SELECT idAlbum,
                    título,
                   (SELECT SUM(segundosReproducidos)
                    FROM Reproducciones
                    WHERE segundosReproducidos > 30 AND
                       idCanción IN (SELECT idCanción
                                     FROM Canciones
                                     WHERE idÁlbumOriginal =
                                           Álbumes.idAlbum)
                                     ) AS cantidad
              FROM Álbumes) ReproduccionesAlbum
WHERE cantidad = (SELECT MIN(cantidad)
                         FROM (SELECT idAlbum,
                                      título,
                                      (SELECT SUM(segundosReproducidos)
                                       FROM Reproducciones
                                       WHERE segundosReproducidos > 30 AND
                                       idCanción IN (SELECT idCanción
                                                     FROM Canciones
                                                    WHERE idÁlbumOriginal =
                                                          Álbumes.idAlbum)
                                       ) AS cantidad
                                  FROM Álbumes) reproduccionesAlbum)
```

7. ¿Cómo modificaría la consulta de la actividad 6 para que no se consideren los álbumes del sello discográfico «*Codiscos*»?
8. Escriba una consulta que muestre, por cada género, el nombre, la cantidad de canciones que existen en la colección, la cantidad de canciones que han sido reproducidas y la calificación promedio que han obtenido las canciones.
9. Escriba una consulta donde se obtenga el nombre del intérprete o de los intérpretes cuyas canciones tengan el mayor número de reproducciones.
10. Proponga tres casos en los que deban utilizarse los constructos ANY y ALL en conjunto con operadores de expresiones condicionales.

Capítulo 5

Combinaciones

Resultados de aprendizaje

– *Construye consultas en SQL que generan resultados combinando datos almacenados en diferentes tablas de una base de datos.*
– *Determina cuáles tipos de operaciones de combinación de tablas deben utilizarse y son los más adecuados en diferentes casos de uso.*

En las consultas para dar respuesta a las necesidades de datos normalmente deben utilizarse varias tablas. En algunos casos, todos los datos requeridos están en una sola tabla, pero se necesita utilizar datos de otras tablas para realizar ciertas operaciones, como puede ser el filtrado de filas en la tabla resultante a partir del resultado de una subconsulta. En otros casos, los datos requeridos no están en una sola tabla, y es preciso incluir en la consulta algunas operaciones que permitan el uso de ellos.

Una alternativa viable para presentar datos de varias tablas es la inclusión de una subconsulta correlacionada en la lista de columnas, es decir, entre la sentencia `SELECT` y la cláusula `FROM`, para generar un único valor escalar por cada fila a partir de datos de otra tabla y crear una nueva columna en el resultado de la consulta. Sin embargo, el SQL también posee operaciones de combinación de tablas que amplían las posibilidades de aprovechar efectivamente los datos almacenados.

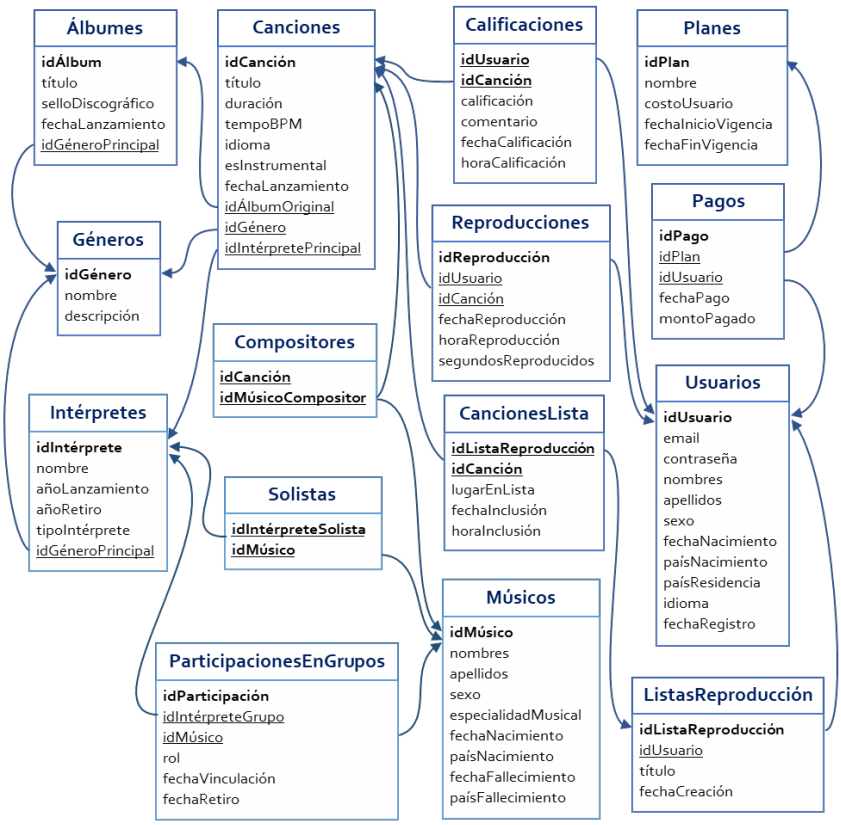
Las combinaciones de tablas se conocen como operaciones `JOIN`. Un `JOIN` opera sobre dos tablas y produce como resultado una nueva tabla. En este sentido, en la cláusula `FROM` pueden incluirse tantas tablas o subconsultas como se requiera, pero debe definirse la forma en que se realizarán las combinaciones. Hay tres tipos de combinaciones, denominadas

`INNER JOIN`, `OUTER JOIN` y `CROSS JOIN`. En este capítulo se trabajará con estas operaciones, iniciando con la operación `INNER JOIN`.

Para realizar las actividades de aprendizaje de las operaciones de combinación se utilizará una nueva versión de la base de datos *Mis Canciones*. En ella se incluyen varias mejoras en comparación con la versión 3 y se amplía el alcance de los datos almacenados, cubriendo nuevos elementos que comúnmente se encuentran en plataformas de este tipo. En la Figura 5.1 se presenta un diagrama del esquema relacional de la versión 4 de esta base de datos, y en los siguientes párrafos se explican los nuevos elementos. El *script* de creación y carga de datos en PostgreSQL está disponible como material complementario de este capítulo.

En la nueva versión se introducen las tablas `Planes` y `Pagos`. En la primera se almacenan los datos de los diferentes tipos de afiliación a los que pueden optar las personas al registrarse en la plataforma, y en la segunda se registran los datos de las transacciones de pago realizadas por los usuarios de acuerdo con el plan que tengan vigente. El valor almacenado en la columna `costoUsuario` de la tabla `Planes` refleja el costo vigente o actual para cada plan, es decir, el valor de referencia utilizado en el momento de registrar un pago de un usuario, cifra que puede cambiar en el tiempo.

Figura 5.1. Versión 4 de la base de datos de la plataforma *«Mis Canciones»*



Por su parte, el valor almacenado en la columna `montoPagado` de la tabla `Pagos` indica la cantidad efectivamente pagada por cada usuario en la fecha correspondiente. Al respecto, cabe tener presente que un usuario puede tener el mismo plan durante varios meses, pero el monto pagado en una fecha podría ser diferente al pagado en una fecha anterior, lo cual significaría que el costo del plan cambió en algún punto.

También se introdujeron varias tablas para tener mayores detalles relacionados con los intérpretes, las personas y los diferentes roles que pueden tener en el contexto de la creación o interpretación de canciones. En primer lugar, se agregó la tabla `Músicos` para almacenar los datos de las personas que participan en las producciones musicales. Así, un músico puede ser un intérprete solista o puede hacer parte de un grupo. Además, es posible que sea el compositor de una o varias canciones.

En la tabla `Compositores` se almacenan los identificadores de los músicos que compusieron las canciones. En muchos casos la composición es colectiva; por lo tanto, puede suceder que existan varias filas por cada canción, una por cada compositor.

Asimismo, la tabla `Solistas` se incluyó para almacenar el identificador del músico —de la persona— que es un intérprete solista. En este sentido, la tabla `Solistas` sirve como especialización de la tabla `Intérpretes`, permitiendo incorporar datos personales almacenados en la tabla `Músicos`. Un intérprete solista, por ejemplo «*Shakira*», tendrá una fila en la tabla `Músicos` en la cual estarán los datos personales, como sus nombres: «*Shakira Isabel*»; sus apellidos: «*Mebarak Ripoll*» y su fecha de nacimiento: «*02/02/1977*».

En la tabla `ParticipacionesEnGrupos` se almacena el historial de participaciones de un músico durante su carrera. Por esta razón podría darse el caso de que una persona haga parte de varios grupos en diferentes momentos de su vida, e incluso podría tener varias participaciones en el mismo grupo, algo que sucede cuando el integrante se retira temporalmente y después de un tiempo vuelve a vincularse. En esta tabla también podrían almacenarse los datos de los músicos que acompañan o hacen parte de las agrupaciones de los intérpretes solistas.

Con la inclusión de estas tablas se representa de mejor forma la relación entre las personas y las canciones. Una persona registrada en la tabla `Músicos` podría ser compositor de un conjunto de canciones y, al mismo tiempo, solista y además hacer parte de una agrupación. Es importante que la base de datos permita almacenar todos estos detalles para facilitarles a los usuarios la búsqueda de canciones.

5.1. Combinación interna

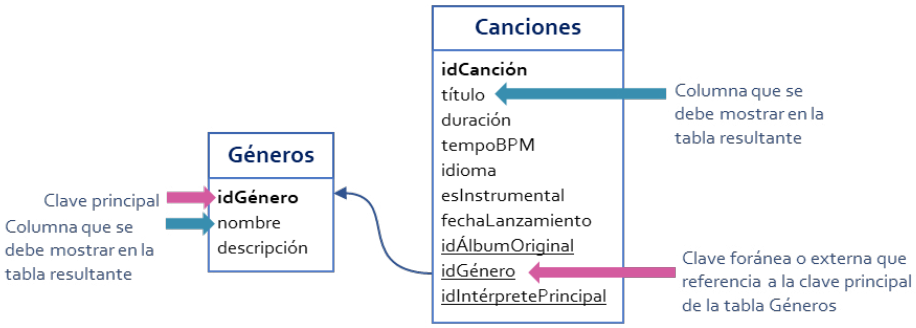
Los datos en una base de datos se almacenan en varias tablas, y es común que las necesidades que deben satisfacerse demanden, a su vez, la obtención y la presentación de datos de varias tablas. Con las subconsultas se abrió la posibilidad de utilizar datos de diversas tablas de diferentes formas. Sin embargo, hay otra forma de utilizar los datos combinando las filas de varias tablas que tengan relación, de modo que los datos entregados por el DBMS en la tabla resultante sean más significativos y ofrezcan mayores detalles al usuario.

Para iniciar el aprendizaje de este nuevo elemento de las consultas en el SQL se propone la siguiente pregunta:

¿Cuál es el título de las canciones y el nombre del género en el cual está clasificada cada una?

Siempre es clave identificar la tabla o las tablas en las que están almacenados los datos necesarios para responder a las necesidades de datos. En este caso, los datos que se requieren para generar el resultado están almacenados en dos tablas diferentes: `Canciones` y `Géneros`. En la Figura 5.2 se presenta un diagrama que ilustra dicho razonamiento.

Figura 5.2. Tablas requeridas para responder la pregunta



Los títulos de las canciones están almacenados en la tabla `Canciones`, y los nombres de los géneros, en la tabla `Géneros`. En `Canciones` se tiene la columna `idGénero`, con un número entero que corresponde al identificador del género de la canción según lo guardado `Géneros`. Por lo tanto, para dar respuesta a la pregunta debe utilizarse alguna operación que permita combinar los datos de las dos tablas de forma tal que la tabla resultante tenga dos columnas: una con el título de cada canción y otra con el género correspondiente.

Antes de avanzar al código en el SQL, es necesario precisar que la combinación de tablas se fundamenta en la operación de conjuntos denominada producto cartesiano. En este caso, los conjuntos participantes son las tablas, entendiendo que cada una de ellas es un conjunto de datos; específicamente, un conjunto de filas. Por consiguiente, se podrán obtener diferentes combinaciones entre las filas de las dos tablas participantes en el producto cartesiano.

En la Figura 5.3 se muestra un ejemplo de la operación producto cartesiano para el caso de dos versiones simplificadas de las tablas `Canciones` y `Géneros`, cada una de las cuales tendría dos filas de datos. En otras palabras, se tienen dos conjuntos de filas, es decir, las dos tablas, y en cada conjunto hay dos elementos, esto es, dos filas.

Figura 5.3. Ejemplo de un producto cartesiano entre dos tablas

Tabla ***Canciones***

idCanción	*título*	*idGénero*
1004	Azul	3
1005	Cielo	4

Tabla ***Géneros***

idGénero	*nombre*
3	Pop
4	Rock

Canciones producto cartesiano ***Géneros***

(*Canciones* **X** *Géneros*)

idCanción	*título*	*idGénero*	*idGénero*	*nombre*
1004	Azul	3	3	Pop
1004	Azul	3	4	Rock
1005	Cielo	4	3	Pop
1005	Cielo	4	4	Rock

Al realizar la operación indicada con el símbolo **X** se obtiene una nueva tabla con todas las columnas de las dos tablas participantes en el producto cartesiano y con la combinación de cada fila de la primera tabla con todas las filas de la segunda tabla. Para el ejemplo de la Figura 5.3, la primera fila de la tabla `Canciones`, que se identifica con el dato *1004* en la columna `idCanción`, se combina con las dos filas de la tabla `Géneros` y se obtienen las dos primeras filas de la tabla resultante.

En la Figura 5.3 puede identificarse que no todas las filas de ese producto cartesiano tienen coherencia o significado porque la relación entre las dos tablas está establecida por la columna común, es decir, `idGénero`. En este caso, las filas que tienen sentido son aquellas en las cuales coinciden los datos almacenados en las dos tablas, como sucede en la primera y última filas, en donde el dato almacenado en la columna `idGénero` es *3* y *4*, respectivamente.

En la primera fila, correspondiente a la canción con título «*Azul*», se observa que las dos columnas `idGénero` tienen el mismo valor, asociado al género «*Pop*». Por el contrario, en la segunda fila de la tabla resultante los valores de las columnas `idGénero` no coinciden. En este orden de ideas, para asegurar la coherencia de los datos debería aplicarse una operación de filtrado de filas para excluir aquellas que no cumplen la condición de coincidencia en las columnas comunes. En el *Script* 5.1 se presenta la consulta con la implementación de una de las formas con las cuales puede responderse la pregunta.

En el *Script* 5.1 se observa que en la cláusula `FROM` se indican las tablas por combinar separadas por una coma, lo cual corresponde al producto cartesiano de las dos tablas. Además, en la cláusula `WHERE` se define la expresión condicional para filtrar las filas en las cuales hay coincidencia en los valores de las columnas `idGénero`. Esta implementación corresponde a la primera operación de combinación que se aborda en este capítulo: la combinación interna. La cláusula `ORDER BY` se utiliza al igual que en las consultas que emplean una sola tabla.

El nombre de las columnas se presenta en conjunto con el nombre de la tabla a la que pertenecen porque las dos tablas tienen columnas con exactamente el mismo nombre y el

SQL exige que no existan ambigüedades causadas por dicha coincidencia en las tablas o las columnas involucradas en una consulta. Si se omitiera esta referencia completa, incluyendo el nombre de la tabla, la expresión condicional de la cláusula `WHERE` tendría la forma `idGénero = idGénero`, causando una ambigüedad para el DBMS que va a ejecutar la consulta.

Como se explicó con la Figura 5.3, la tabla resultante del producto cartesiano tiene todas las columnas de las tablas participantes. Esto significa que en la sentencia `SELECT` pueden especificarse las columnas que se requieran de las dos tablas. Si se solicitan todas las columnas de las tablas participantes en la combinación, es posible utilizar el símbolo *, de la misma forma en que se hace en consultas sobre una sola tabla.

Para este caso, se requiere mostrar únicamente la columna `título` —proveniente de la tabla `Canciones`, lo cual se especifica con la expresión `Canciones.título`— y la columna `nombre` —correspondiente a la tabla `Géneros`, tal como se especifica con la expresión `Géneros.nombre`—. En la Tabla 5.1 se presenta un ejemplo con algunas filas de lo que sería la tabla resultante al ejecutar el *Script* 5.1.

***Script* 5.1**

```
SELECT Canciones.título,
       Géneros.nombre AS género
FROM Canciones,
       Géneros
WHERE Géneros.idGénero = Canciones.idGénero
ORDER BY Canciones.título
```

Tabla 5.1. Resultado de la ejecución del *Script* 5.1

título	género
Amarte más no pude	Vallenato
Bolero falaz	Rock
Cali es sabrosura	Salsa
Cuando te veo	Latina fusión
De donde vengo yo	Latina fusión
Ginza	Reggaeton
Gotas de lluvia	Salsa
Hips don't lie	Pop
La bicicleta	Latina fusión

Esta forma de combinación de tablas era la única disponible en el SQL hasta la publicación de la versión de 1992 del estándar internacional para este lenguaje. En ese momento se introdujo una forma diferente, más precisa y organizada de especificar las combinaciones: la palabra `JOIN` con sus diferentes variaciones y el término `ON` para establecer las expresiones condicionales por utilizar en la combinación. En este sentido, una alternativa para solucionar la pregunta es la consulta del *Script* 5.2.

***Script* 5.2**

```
SELECT título,
       nombre AS género
FROM Canciones
       INNER JOIN Géneros ON Géneros.idGénero = Canciones.idGénero
ORDER BY Canciones.título
```

En el *Script* 5.2 se observa que en la cláusula FROM se especifican las tablas participantes en la consulta separadas por la expresión `INNER JOIN` en lugar de la coma utilizada en el *Script* 5.1. Esta expresión define el tipo de combinación que debe realizar el DBMS; para este caso, una combinación interna o `INNER`. Al ejecutar esta consulta se obtendrá exactamente el mismo resultado.

Cuando no se especifica explícitamente el tipo de combinación, el DBMS asume que es interna. En otras palabras, si en lugar de utilizar la expresión `INNER JOIN` se deja tan solo la palabra `JOIN`, el DBMS ejecutará una combinación interna o `INNER`. Sin embargo, es recomendado como buena práctica utilizar la notación completa `INNER JOIN` en lugar de insertar únicamente la palabra `JOIN` o de especificar las tablas por combinar separadas por coma junto con una cláusula `WHERE`.

Ahora bien, una alternativa para responder la pregunta sin utilizar la operación `INNER JOIN` es implementar una subconsulta correlacionada para generar una nueva columna con el nombre del género. Específicamente, esta alternativa de solución podría implementarse con la consulta presentada en el *Script* 5.3, en la cual el nombre del género de cada canción se obtiene con la subconsulta correlacionada.

***Script* 5.3**

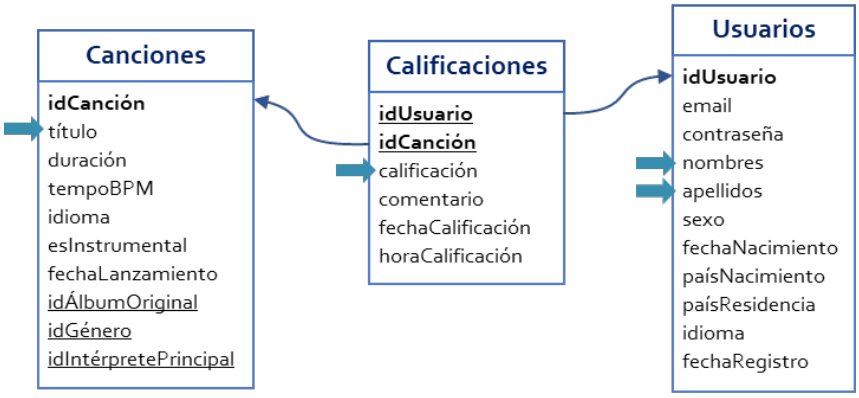
```
SELECT título,
       (SELECT nombre
       FROM Géneros
       WHERE Géneros.idGénero = Canciones.idGénero) AS género
FROM Canciones
ORDER BY título
```

En una consulta SQL pueden combinarse más de dos tablas para responder una necesidad de datos. Para abordar este caso se propone la siguiente pregunta:

¿Cuáles son los títulos de las canciones, los nombres y apellidos de los usuarios y las calificaciones asignadas por cada usuario a todas las canciones?

Los datos necesarios para responder la pregunta están almacenados en tres tablas: los títulos de las canciones, en `Canciones`; los nombres y apellidos de los usuarios, en `Usuarios`, y las calificaciones asignadas a las canciones, en `Calificaciones`. Por consiguiente, la consulta para obtener el resultado esperado debe combinar las filas de esas tres tablas cumpliendo los criterios de combinación que sean necesarios. En la Figura 5.4 se presenta un diagrama con las tablas de la base de datos requeridas para resolver la pregunta.

Figura 5.4. Tablas requeridas para responder la pregunta



La combinación interna o `INNER JOIN` es una operación que se aplica únicamente sobre dos tablas. Sin embargo, es posible encadenar operaciones de combinación sucesivas para ir ampliando el número de tablas combinadas. En otras palabras, se puede especificar dentro de la cláusula `FROM` una combinación de dos tablas con la expresión `INNER JOIN` y, con la palabra `ON`, definir las condiciones de combinación, para luego combinar ese resultado con una tercera tabla.

En este caso es necesario utilizar las columnas que son clave principal o *Primary Key* y las columnas que son claves foráneas o *Foreign Key* para especificar las expresiones condicionales de combinación. Por un lado, la tabla `Canciones` tiene como clave principal la columna `idCanción`. Por otro lado, la tabla `Calificaciones` cuenta también con una columna llamada `idCanción`, la cual es una clave foránea que referencia a la misma columna de la tabla `Canciones`.

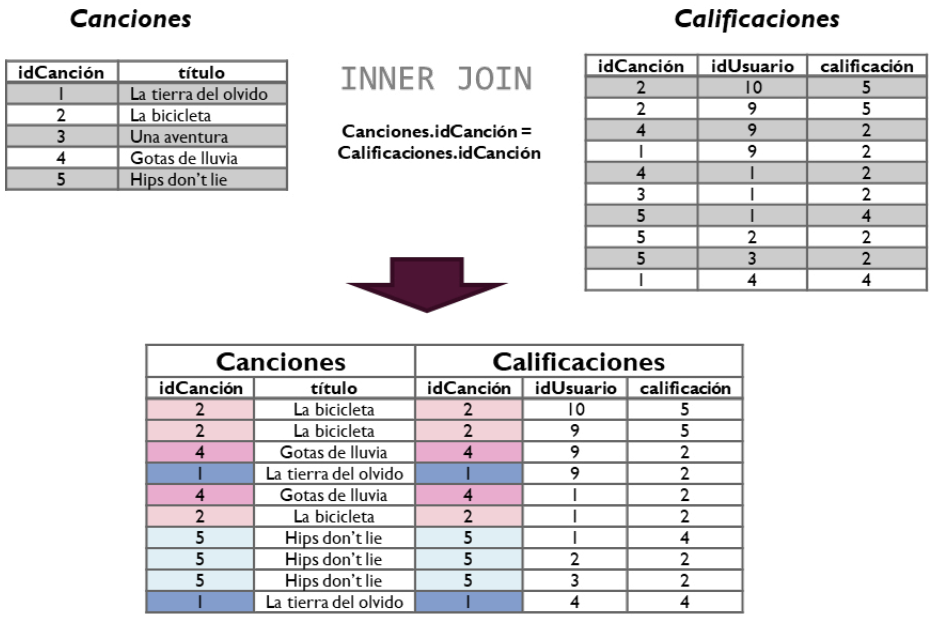
Esta relación entre la clave principal y la clave foránea sirve como criterio para combinar las filas de las dos tablas cuando el valor almacenado en la columna `idCanción` en alguna fila de la tabla `Canciones` coincida con el que se encuentra en la columna `idCanción` de alguna fila de la tabla `Calificaciones`. El *Script* 5.4 muestra una consulta SQL con esta combinación.

***Script* 5.4**

```
SELECT Canciones.título,
       Calificaciones.calificación
FROM Canciones
       INNER JOIN Calificaciones
       ON Canciones.idCanción = Calificaciones.idCanción
```

La consulta del *Script* 5.4 genera una tabla con dos columnas, `Canciones.título` y `Calificaciones.calificación`, y con el mismo número de filas que tiene la tabla `Calificaciones`. La cantidad de filas puede dimensionarse al hacer un sencillo análisis de la operación de combinación con relación a las columnas comunes, que en este caso son las columnas `idCanción` de ambas tablas.

Figura 5.5. Combinación interna de filas de las tablas `Canciones` y `Calificaciones`



idCanción	título
1	La tierra del olvido
2	La bicicleta
3	Una aventura
4	Gotas de lluvia
5	Hips don't lie

idCanción	idUsuario	calificación
2	10	5
2	9	5
4	9	2
1	9	2
4	1	2
3	1	2
5	1	4
5	2	2
5	3	2
1	4	4

Canciones		Calificaciones		
idCanción	título	idCanción	idUsuario	calificación
2	La bicicleta	2	10	5
2	La bicicleta	2	9	5
4	Gotas de lluvia	4	9	2
1	La tierra del olvido	1	9	2
4	Gotas de lluvia	4	1	2
2	La bicicleta	2	1	2
5	Hips don't lie	5	1	4
5	Hips don't lie	5	2	2
5	Hips don't lie	5	3	2
1	La tierra del olvido	1	4	4

En la tabla `Calificaciones` pueden almacenarse varias calificaciones para una misma canción, pero deben ser calificaciones asignadas por diferentes usuarios. Todas las filas de la tabla `Calificaciones` tendrán un valor válido en la columna `idCanción`, es decir, alguno de los valores guardados en la columna `idCanción` de la tabla `Canciones`. Por esto, al hacer la combinación con el criterio definido con la expresión `Canciones.idCanción = Calificaciones.idCanción`, el número de filas resultante será igual al número de filas de la tabla `Calificaciones` porque todas las demás combinaciones de filas no cumplirían ese criterio de igualdad. En la Figura 5.5 se ilustra esta combinación con datos de ejemplo en una versión simplificada de las tablas.

Si la tabla Canciones tuviese solamente esas dos columnas y esas cinco filas, y la tabla Calificaciones contara apenas con esas tres columnas y esas diez filas, el resultado de la combinación interna sería una tabla con cinco columnas (dos provenientes de Canciones y tres provenientes de Calificaciones) y diez filas (las mismas de la tabla Calificaciones combinadas con las filas correspondientes en la tabla Canciones).

Un detalle importante que muestra este ejemplo es el hecho de que la canción titulada «Una aventura» no está incluida en la tabla resultante. Esto sucede porque en la tabla Calificaciones no hay filas que tengan el valor 3 en la columna idCanción, por lo que el DBMS no encuentra filas coincidentes.

Para responder la pregunta falta combinar la tabla Usuarios porque se requieren los nombres y los apellidos de los usuarios que emitieron cada calificación. En otras palabras, se debe ejecutar una operación de combinación entre la tabla Usuarios y la obtenida tras combinar las tablas Canciones y Calificaciones.

En la tabla Usuarios existe la columna idUsuario, la cual está definida como clave principal. Por su parte, en la tabla Calificaciones también existe una columna llamada idUsuario, la cual es una clave foránea que referencia a dicha clave principal. Nuevamente, la relación clave principal-clave foránea sirve como criterio para combinar las filas de forma coherente. El *Script* 5.5 muestra una consulta SQL con esta operación.

***Script* 5.5**

```
SELECT Usuarios.nombres,
       Usuarios.apellidos,
       Canciones.título,
       Calificaciones.calificación
FROM Canciones
       INNER JOIN Calificaciones
       ON Canciones.idCanción = Calificaciones.idCanción
       INNER JOIN Usuarios
       ON Usuarios.idUsuario = Calificaciones.idUsuario
```

En el *Script* 5.5 se observa que la segunda operación INNER JOIN se ubica después de la expresión condicional con el criterio de combinación establecido para la primera operación INNER JOIN. En este caso, la relación entre la tabla Usuarios y la tabla Calificaciones permite que un usuario asigne una o varias calificaciones a diferentes canciones, pero una canción solamente puede ser calificada una vez por cada usuario. Por lo tanto, el número de filas de la tabla resultante será igual al número de filas de la tabla Calificaciones.

Para resumir la presentación de los resultados, puede utilizarse una función escalar que concatene los nombres y los apellidos de los usuarios en una sola cadena de texto. De este modo se generará una columna derivada llamada nombreCompleto, tal y como se presenta en la consulta del *Script* 5.6.

***Script* 5.6**

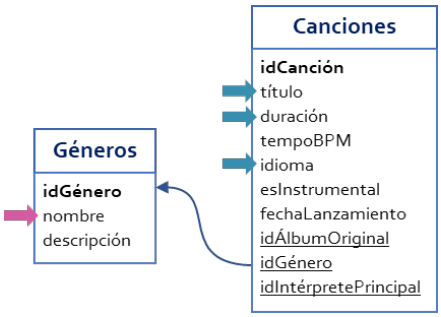
```
SELECT CONCAT(Usuarios.nombres, ' ', Usuarios.apellidos) AS nombreCompleto,
       Canciones.título,
       Calificaciones.calificación
FROM Canciones
       INNER JOIN Calificaciones
       ON Canciones.idCanción = Calificaciones.idCanción
       INNER JOIN Usuarios
       ON Usuarios.idUsuario = Calificaciones.idUsuario
```

En una consulta SQL pueden combinarse todas las tablas que sean necesarias para obtener el resultado esperado. Ahora bien, el hecho de que los datos de alguna tabla no se incluyan en la tabla resultante no es impedimento para utilizar operaciones de combinación de tablas. Para ilustrar esto, se propone abordar la siguiente pregunta:

¿Cuáles son los títulos, la duración y el idioma de las canciones del género «*Salsa*»?

La tabla resultante esperada para responder la pregunta requiere datos de una sola tabla: `Canciones`. Sin embargo, se necesita filtrar las filas para incluir únicamente las que corresponden al género «*Salsa*» tal y como se presenta en la Figura 5.6. Esta pregunta ya fue resuelta utilizando subconsultas con el código del *Script* 5.7.

Figura 5.6. Tablas requeridas para resolver la pregunta



***Script* 5.7**

```
SELECT título,
       duración,
       idioma
FROM Canciones
WHERE idGénero = (SELECT idGénero
       FROM Géneros
       WHERE nombre = 'Salsa')
```

Otra forma de resolver esta pregunta es realizando una combinación de tablas y luego aplicando un filtro para el género. El *Script* 5.8 presenta esta alternativa de solución implementada con una operación `INNER JOIN` entre las tablas `Canciones` y `Géneros`.

***Script* 5.8**

```
SELECT título,
       duración,
       idioma
FROM Canciones
       INNER JOIN Géneros
       ON Géneros.idGénero = Canciones.idGénero
WHERE Géneros.nombre = 'Salsa'
```

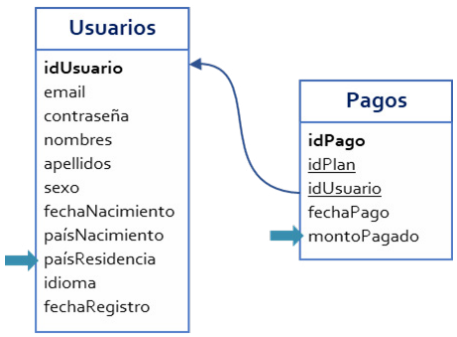
La combinación interna se realiza utilizando como criterio la igualdad en los datos de la columna `idGénero` de la tabla `Canciones` y la columna `idGénero` de la tabla `Géneros`. Nuevamente, se utiliza la relación clave principal-clave foránea para conformar las filas combinadas en la tabla resultante. Además, se incluye una expresión condicional en la cláusula `WHERE` para filtrar las filas y dejar únicamente las que tienen el valor *«Salsa»* en la columna `Géneros.nombre`.

Las columnas que componen la tabla resultante de una combinación interna pueden utilizarse como parte de la respuesta o, si es necesario, sobre estas pueden realizarse operaciones adicionales como la aplicación de funciones escalares o de agregación, entre las demás operaciones explicadas hasta este punto. Para ilustrar este caso, se propone la siguiente pregunta:

¿Cuántos usuarios hay y cuánto dinero se ha recaudado en cada país?

Los datos requeridos para responder esta pregunta están almacenados en las tablas `Usuarios` y `Pagos`. En la Figura 5.7 se presenta un diagrama con las tablas de la base de datos requeridas para resolver la pregunta.

Figura 5.7. Tablas requeridas para responder la pregunta



En la tabla `Usuarios` se tiene la columna `paísResidencia`, la cual se tomará como base de agregación para realizar los conteos de usuarios y las sumas de los dineros recaudados que han sido pagados por los usuarios que residen en cada país. Por otra parte, los datos de los pagos están almacenados en la tabla `Pagos`, cuya columna `idUsuario` sirve como una clave foránea que referencia a la columna `idUsuario` de la tabla `Usuarios`, mientras que la columna `montoPagado` representa la cantidad de dinero de cada transacción realizada por cada persona. La consulta que permite dar respuesta a la pregunta se presenta en el *Script* 5.9.

***Script* 5.9**

```
SELECT Usuarios.paísResidencia,
       COUNT(DISTINCT Usuarios.idUsuario) AS cantidadUsuarios,
       SUM(Pagos.montoPagado) AS dineroRecaudado
FROM Usuarios
       INNER JOIN Pagos
       ON Pagos.idUsuario = Usuarios.idUsuario
GROUP BY Usuarios.paísResidencia
ORDER BY dineroRecaudado DESC
```

En la consulta del *Script* 5.9 se utiliza la combinación interna tomando como criterio las columnas que establecen la relación entre las dos tablas al ser clave principal y la clave foránea. Una vez combinadas las filas, se tiene una tabla con todas las columnas de la tabla `Usuarios` y todas las columnas de la tabla `Pagos`. Esto permite que se utilice la columna `paísResidencia` de la tabla `Usuarios` en la cláusula `GROUP BY` y se ejecute la función de agregación `SUM` sobre los datos de la columna `montoPagado` de la tabla `Pagos`.

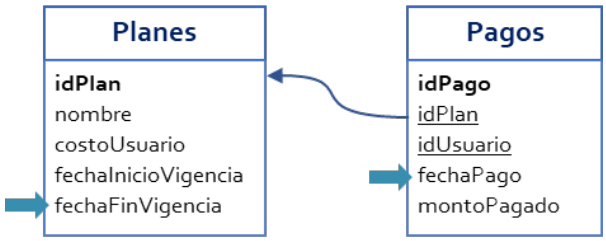
Para realizar el conteo de los usuarios, se utiliza la función de agregación `COUNT` sobre los datos de la columna `idUsuario` de la tabla `Usuarios` en conjunto con la palabra `DISTINCT`. Esto es necesario porque un usuario puede tener registrados varios pagos y su identificador aparecerá tantas veces como pagos haya realizado. Si se cuentan las filas sin excluir duplicados, es decir, sin utilizar `DISTINCT`, el conteo no correspondería a la realidad porque el mismo usuario se estaría contando varias veces.

Hasta este punto, las operaciones de combinación de tablas para resolver las preguntas se han realizado utilizando las claves foráneas y las claves principales de las tablas con el operador relacional =. Sin embargo, pese a que este es el uso más frecuente, no es la única forma de aprovechar las combinaciones. En la cláusula `ON`, por ejemplo, puede incluirse cualquier expresión que al evaluarse retorne un valor booleano. Para abordar esto, se propone la siguiente pregunta:

¿Cuáles pagos realizados por los usuarios se recibieron después del cierre de vigencia del plan correspondiente?

Esta pregunta se enfoca en identificar comportamientos fuera de lo normal dentro de los datos. Específicamente, se buscan los pagos realizados por los usuarios en alguna fecha posterior a aquella en la que el plan al que estaban afiliados perdiera vigencia. Los datos requeridos para resolver esta solicitud están almacenados en las tablas `Pagos` y `Planes`, tal como se muestra en la Figura 5.8.

Figura 5.8. Tablas requeridas para responder la pregunta



En la tabla `Planes` están las columnas `fechaInicioVigencia` y `fechaFinVigencia`, en las cuales se almacenan los datos del rango de fechas en que estaba vigente cada plan. En la tabla `Pagos` se tiene la columna `fechaPago`, donde se encuentra la fecha en que un usuario realizó un pago correspondiente a un plan. En el *Script* 5.10 se presenta la consulta que responde la pregunta, utilizando una condición no equivalente para únicamente combinar las filas en las que el pago se haya realizado en una fecha posterior a la fecha de finalización de vigencia del plan.

Este tipo de combinaciones en las cuales las expresiones condicionales no utilizan el operador = se denominan *JOIN no equivalente* o *NON EQUI JOINS* y tienen múltiples aplicaciones. Por ejemplo, para efectos de verificar la coherencia de los datos, podría utilizarse la consulta del *Script* 5.11 de manera que identifique si hay alguna fila en la tabla `Pagos` que tenga una fecha de pago anterior a aquella en que inició el plan correspondiente.

***Script* 5.10**

```
SELECT *
FROM Pagos
       INNER JOIN Planes ON Pagos.idPlan = Planes.idPlan AND
       Pagos.fechaPago > Planes.fechaFinVigencia
```

***Script* 5.11**

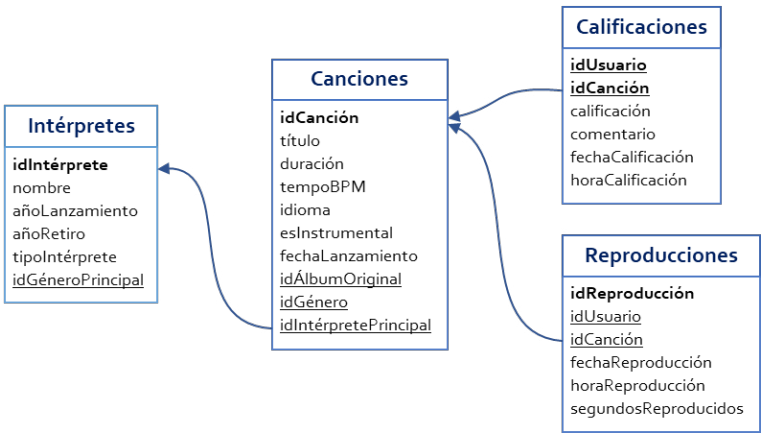
```
SELECT *
FROM Pagos
       INNER JOIN Planes ON Pagos.idPlan = Planes.idPlan AND
       (Pagos.fechaPago < Planes.fechaInicioVigencia OR
       Pagos.fechaPago > Planes.fechaFinvigencia)
```

En la mayoría de los casos se combinan tablas de la base de datos. Sin embargo, en otras ocasiones se requiere combinar tablas de la base de datos y tablas obtenidas a partir de una subconsulta autónoma, e incluso es posible que deban combinarse los resultados de dos subconsultas. Con la siguiente pregunta se ilustra este caso de uso:

¿Cuál es la cantidad de reproducciones que han tenido las canciones de cada intérprete y cuál es la calificación promedio que han recibido?

La respuesta esperada a esta pregunta es una tabla de tres columnas: el nombre del intérprete, la cantidad total de reproducciones de todas sus canciones y la calificación promedio de estas últimas. En tal sentido, para responder la pregunta se requieren datos de la tablas `Intérpretes`, `Canciones`, `Calificaciones` y `Reproducciones`, tal y como se muestra en la Figura 5.9.

Figura 5.9. Tablas requeridas para responder la pregunta



Para plantear la solución puede iniciarse obteniendo la cantidad de reproducciones de las canciones de cada intérprete, lo cual puede realizarse combinando las tablas `Canciones` y `Reproducciones`. Con esa combinación puede aplicarse el agrupamiento por `idIntérpretePrincipal` y la función de agregación `COUNT`, tal como se presenta en el *Script* 5.12.

***Script* 5.12**

```
SELECT idIntérpretePrincipal,
       COUNT(*) AS cantidadReproducciones
FROM Canciones
       JOIN Reproducciones ON Canciones.idCanción = Reproducciones.idCanción
GROUP BY idIntérpretePrincipal
```

Otra parte de la solución puede abordarse de forma similar, pero combinando las tablas `Canciones` y `Calificaciones`. En este caso puede aplicarse el agrupamiento por `idIntérpretePrincipal` y la función de agregación `AVG` en la columna `calificación` tal como se presenta en el *Script* 5.13.

***Script* 5.13**

```
SELECT idIntérpretePrincipal,
       AVG(calificación) AS calificaciónPromedio
FROM Canciones
       JOIN Calificaciones ON Canciones.idCanción = Calificaciones.idCanción
GROUP BY idIntérpretePrincipal
```

En este punto se tienen dos tablas: una con la cantidad de reproducciones hechas de las canciones de cada intérprete y otra con la calificación promedio de dichas canciones. En ambas tablas está la columna `idIntérprete`, la cual puede servir para realizar combinaciones del resultado obtenido con estas dos consultas y la tabla `Intérpretes`. En el *Script* 5.14 se presenta este uso.

***Script* 5.14**

```
SELECT nombre,
       cantidadReproducciones,
       calificaciónPromedio
FROM Intérpretes
INNER JOIN(SELECT idIntérpretePrincipal,
       AVG(calificación) AS calificaciónPromedio
       FROM Canciones
       JOIN Calificaciones
                ON Canciones.idCanción = Calificaciones.idCanción
       GROUP BY idIntérpretePrincipal) AS CalificacionesPromedio
       ON Intérpretes.idIntérprete = CalificacionesPromedio.idIntérpretePrincipal
       INNER JOIN(SELECT idIntérpretePrincipal,
       COUNT(*) AS cantidadReproducciones
       FROM Canciones
       JOIN Reproducciones
                ON Canciones.idCanción = Reproducciones.idCanción
       GROUP BY idIntérpretePrincipal) AS CantidadesReproducciones
ON Intérpretes.idIntérprete = CantidadesReproducciones.idIntérpretePrincipal
```

Cada una de las dos consultas creadas inicialmente se convierten en una subconsulta que hace las veces de una tabla en la cláusula `FROM`. Las tablas generadas por ambas subconsultas se combinan utilizando `INNER JOIN` y estableciendo como condición de combinación la columna `idIntérprete`.

Es preciso anotar que esta consulta puede dar el resultado, pero tiene una limitación. En caso de que algún intérprete tenga canciones que no hayan sido calificadas o que no hayan sido reproducidas, los datos de ese artista no aparecerán en la tabla resultante. En otras palabras, con esta consulta se muestran los datos de todos los intérpretes que tienen canciones calificadas y reproducidas.

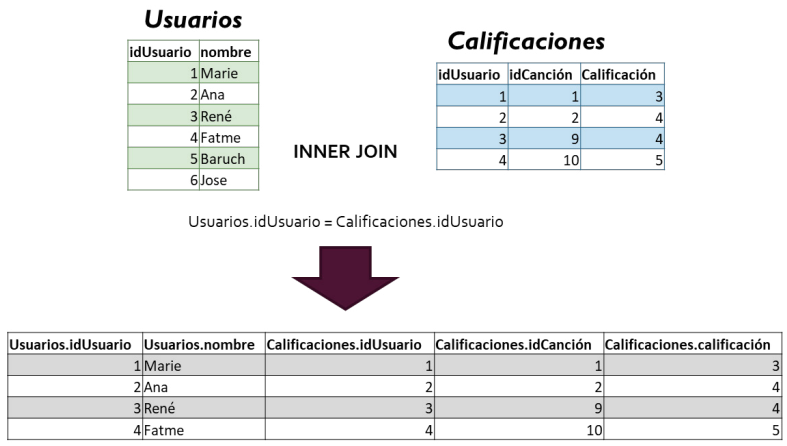
5.2. Combinación externa

En ciertos casos es necesario que la combinación de tablas se realice incluyendo las filas de alguna de las tablas participantes que no tienen coincidencia en la otra, es decir, que no cumplen la condición definida en la cláusula `ON`. Este planteamiento puede parecer contradictorio o confuso si se toma en consideración que una de las ideas que fundamentan la existencia de las bases de datos es, precisamente, la integridad y coherencia de los datos. Sin embargo, esta opción que ofrece el SQL permite que los datos obtenidos en una tabla resultante de una consulta estén realmente completos y ofrezcan una respuesta precisa a la pregunta que la motivó. Para abordar esta situación, se propone trabajar en la siguiente pregunta:

¿Cuántas canciones ha calificado cada usuario registrado en la plataforma?

Los datos requeridos para responder a esta pregunta están almacenados en las tablas `Usuarios` y `Calificaciones`. Así pues, puede utilizarse una combinación interna entre ellas o `INNER JOIN` que utilice la columna `idUsuario` —clave principal de `Usuarios` y clave foránea en `Calificaciones`— como criterio para determinar las filas que se incluirán en la operación. En la Figura 5.10 se ejemplifica esta aproximación con unas versiones simplificadas de las tablas.

Figura 5.10. Ejemplo de combinación interna



***Script* 5.15**

```
SELECT U.idUsuario,
       COUNT(C.calificación) AS cantidadCalificaciones
FROM Calificaciones AS C
       INNER JOIN Usuarios AS U
       ON C.idUSuario = U.idUsuario
GROUP BY U.idUsuario
ORDER BY cantidadCalificaciones DESC
```

A primera vista, puede parecer que se ha logrado el objetivo y que con esta combinación se tiene la base para responder la pregunta. Sin embargo, al observar detenidamente las tablas `Usuarios` y `Calificaciones` se nota que el usuario con identificador *«6»* y nombre *«Jose»* no ha calificado canciones o, lo que es lo mismo, su identificador aparece en `Usuarios`, pero no en `Calificaciones`. Por esta razón, el usuario *«Jose»*, al igual que el usuario *«Baruch»*, no se encuentran en la tabla resultante de la combinación y tampoco harían parte del resultado que se obtendría si se ejecutara la consulta del *Script* 5.15, en la cual se utiliza la función `COUNT` para establecer la cantidad de canciones que ha calificado cada usuario.

En el *Script* 5.15 se observa que la condición de la combinación está planteada con la igualdad de los datos de `idUsuario` en ambas tablas, lo cual garantiza que los usuarios que no han calificado canciones no se incluyan en la tabla resultante. Sin embargo, al volver a la pregunta que motiva la creación de esta consulta, podría inferirse que en el resultado se espera que aparezcan todos los usuarios *registrados en la plataforma* y no solamente los que *han calificado al menos una canción*. En otras palabras, si un usuario no ha calificado alguna canción, debería aparecer en la tabla resultante y el dato de la columna `cantidadCalificaciones` debería ser cero.

Para resolver esta situación, SQL cuenta con las combinaciones externas u `OUTER JOINS`, las cuales tienen el comportamiento normal de una combinación interna, pero además permiten incluir las filas que no cumplen la condición de combinación. Existen tres tipos: combinación externa izquierda, combinación externa derecha y combinación externa completa.

La combinación externa izquierda, formalmente definida en el SQL con la operación `LEFT OUTER JOIN` o simplemente `LEFT JOIN`, le indica al DBMS que, además de las filas coincidentes o que cumplen la condición de combinación, debe incluir todas las de la tabla ubicada en la parte izquierda de la operación que no cumplen la condición. A su vez, esta instrucción les asigna valores nulos a las columnas correspondientes a la tabla ubicada en la parte derecha de la combinación.

En la Figura 5.11 se ejemplifica el comportamiento de la combinación externa izquierda con los mismos datos presentados en la Figura 5.10. De manera general, si se hace una combinación izquierda o `LEFT JOIN` entre `Usuarios` y `Calificaciones`, se incluyen todas las filas que cumplan la condición de combinación (en este caso, la igualdad entre los datos

almacenados en las columnas que son clave principal y clave externa) y también las filas de `Usuarios` que no tienen coincidencia con alguna fila de `Calificaciones`.

Figura 5.11. Ejemplo de combinación externa izquierda

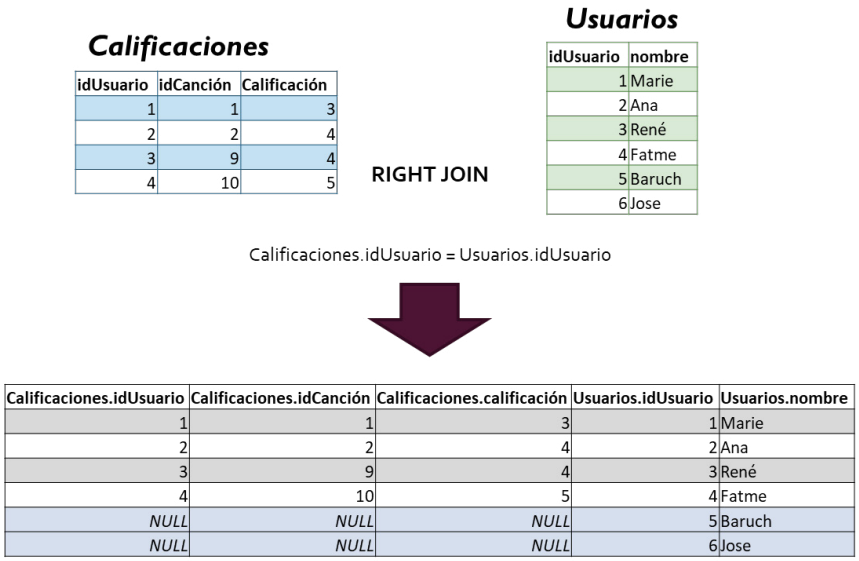
En comparación con el resultado obtenido con la combinación interna presentado en la Figura 5.10, la tabla resultante de la combinación izquierda que ilustra la Figura 5.11 incluye dos filas adicionales: las del usuario «*Baruch*» y la del usuario «*Jose*», que son los usuarios que aún no han calificado canciones. Como no hay filas coincidentes en `Calificaciones`, el DBMS pone en `NULL` los datos de las columnas `idUsuario`, `idCanción` y `calificación` de dicha tabla, que es la de la derecha.

En este orden de ideas, la consulta presentada en el *Script* 5.16 generará el resultado esperado, incluyendo el valor cero para aquellos usuarios que no han calificado canciones. Esto es posible porque la función de agregación `COUNT` se está aplicando sobre la columna `calificación` de la tabla `Calificaciones` y, en el caso de los usuarios que no han realizado calificaciones, esta columna tiene el valor `NULL`.

***Script* 5.16**

```
SELECT U.idUsuario,
       COUNT(C.calificación) AS cantidadCalificaciones
FROM Usuarios AS U
       LEFT JOIN Calificaciones AS C
       ON C.idUSuario = U.idUsuario
GROUP BY U.idUsuario
ORDER BY cantidadCalificaciones DESC
```

Figura 5.12. Ejemplo de combinación externa derecha



***Script* 5.17**

```
SELECT U.idUsuario,
       COUNT(C.calificación) AS cantidadCalificaciones
FROM Calificaciones AS C
        RIGHT OUTER JOIN Usuarios AS U
        ON C.idUSuario = U.idUsuario
GROUP BY U.idUsuario
ORDER BY cantidadCalificaciones DESC
```

Así como existe la combinación izquierda `LEFT OUTER JOIN`, también se encuentra la combinación externa derecha `RIGHT OUTER JOIN` o simplemente `RIGHT JOIN`, que realiza la misma operación, pero en el otro sentido. Es decir, esta combinación incluye todas las filas de la tabla ubicada en la parte derecha que no cumplen la condición y asigna valores nulos a las columnas correspondientes a la tabla ubicada en la parte izquierda, tal y como se ejemplifica en la Figura 5.12. Por consiguiente, una alternativa para resolver la pregunta es la consulta presentada en el *Script* 5.17.

Para seguir profundizando en el uso de las combinaciones externas, se propone la siguiente pregunta:

¿Cuál es el título de las canciones que han sido reproducidas en algún momento pero no hacen parte de alguna lista de reproducción?

Los datos necesarios para responder a esta consulta están en tres tablas que deben combinarse. El título de la canción se almacena en la tabla `Canciones`. Las canciones que ha sido reproducidas están en la tabla `Reproducciones`, y las canciones que han sido agregadas a una lista se encuentran en la tabla `CancionesLista`.

Teniendo claro lo anterior, se debe comenzar con la combinación de las tablas. Primero se combinarán `Canciones` y `Reproducciones` porque lo que se quiere inicialmente es saber el título de las canciones que han sido reproducidas; es decir, la canción debe existir en ambas tablas. La combinación puede hacerse con un `INNER JOIN` tal como se muestra en el *Script* 5.18.

***Script* 5.18**

```
SELECT C.idCanción,
       C.título,
       R.idReproducción
FROM Canciones AS C
       INNER JOIN Reproducciones AS R
       ON C.idCanción = R.idCanción
```

Habiendo combinado la tabla `Canciones` con la tabla `Reproducciones`, se puede continuar con la tabla `CancionesLista`. Ahora bien, es preciso determinar qué tipo de `JOIN` utilizar. Al revisar con detenimiento la pregunta, es claro que se solicitan las canciones que han sido reproducidas, pero no están en una lista, es decir, aquellas que, si bien hacen parte de la tabla `Reproducciones`, no están en `CancionesLista`. Así las cosas, si se ubica `Reproducciones` a la izquierda del `JOIN` y `CancionesLista` a la derecha, se trataría de un `LEFT JOIN`, como se muestra en el *Script* 5.19.

***Script* 5.19**

```
SELECT DISTINCT
       C.idCanción,
       C.título
FROM Canciones AS C
       INNER JOIN Reproducciones AS R
       ON C.idCanción = R.idCanción
       LEFT JOIN CancionesLista AS CL
       ON R.idCanción = CL.idCanción
WHERE CL.idListaReproducción IS NULL
```

Mediante el `LEFT JOIN` se obtienen todas las canciones que están en la tabla `Reproducciones`, incluidas tanto las que se encuentran en una lista como las que no. Por lo tanto, el siguiente paso es conservar solo las que no están en la tabla `CancionesLista`.

Para realizar la combinación, es necesario recordar que si no existe una fila que cumpla la condición en la tabla de la derecha (para el caso del `LEFT JOIN`), las nuevas columnas

adquieren el valor NULL. Por consiguiente, como resultado del LEFT JOIN todas las canciones que han sido reproducidas pero no agregadas a listas tendrán NULL en las columnas correspondientes a la tabla CancionesLista. Planteado lo anterior, para mantener únicamente las que no han sido agredas a una lista, se debe especificar en el WHERE que el identificador de la lista de reproducción sea nulo.

Asimismo, en el Script 5.19 se hace uso de DISTINCT debido a que, en el momento de combinar Canciones y Reproducciones, se puede obtener más de una fila por la misma canción si esta ha sido reproducida más de una vez. Dicho término permite, entonces, contar con una sola fila por cada canción.

¿Cuáles son los títulos de todas las canciones y de todos los álbumes presentes en la plataforma?

Esta pregunta plantea un contexto similar a las anteriores, pero diferente en sus detalles. Por una parte, en la tabla Álbumes se encuentran álbumes cuyas canciones aún no han sido agregadas a la plataforma y, por otra, en la tabla Canciones también se incluyen canciones cuyo álbum de origen es desconocido. Por consiguiente, se necesita mantener de la tabla Álbumes tanto los álbumes con canciones asociadas como aquellos que no aparecen en la tabla Canciones, y de la tabla Canciones, aquellas que no tienen álbum definido, al igual que las que sí lo tienen. En otros términos, se requieren las filas tanto de la parte izquierda del JOIN como de la parte derecha de este.

Para cumplir con lo anterior, SQL proporciona el FULL OUTER JOIN, una instrucción que permite mantener las filas de ambas tablas que intervienen en la combinación, cumplan o no la condición, rellenando con NULL las columnas correspondientes para las filas que no cumplan con la condición. El *Script* 5.20 muestra el SQL correspondiente para realizar el FULL OUTER JOIN entre estas dos tablas.

***Script* 5.20**

```
SELECT A.título tituloCanción,
       C.título tituloÁlbum
FROM Álbumes AS A
       FULL OUTER JOIN Canciones AS C
       ON A.idalbum = C.idÁlbumoriginal
```

La Tabla 5.2 muestra el resultado de esta consulta. Las filas con rellenos amarillos representan esos casos donde la canción no tiene asignado un álbum (como ocurre con «*Cali es sabrosura*») o de álbumes que no cuentan con al menos una canción incluida (es decir, «*Déjame entrar*» y «*Pies descalzos*»).

Tabla 5.2. Resultado obtenido al ejecutar el *Script* 5.20

tituloCanción	tituloÁlbum
La tierra del olvido	La tierra del olvido
Vives	La bicicleta
Cielo de tambores	Una aventura
Huellas del pasado	Gotas de lluvia
Oral Fixation, Vol. 2	Hips don't lie
¿Dónde están los ladrones?	Ojos así
El Dorado	Bolero falaz
Caribe atómico	Maligno
Título de amor	Amarte más no pude
Brindo con el alma	Sin medir distancias
El mismo	Cuando te veo
Oro	De donde vengo yo
Vibras	Mi gente
Energía	Ginza
Pescador, Lucero y Río	Las acacias
Pescador, Lucero y Río	Oropel
Herencia africana: Salsa de Colombia	Ne me quitte pas
NULL	Cali es sabrosura
El Binomio de Oro	La creciente
2000	Olvídala
Déjame entrar	NULL
Pies descalzos	NULL

5.3. Combinación cruzada

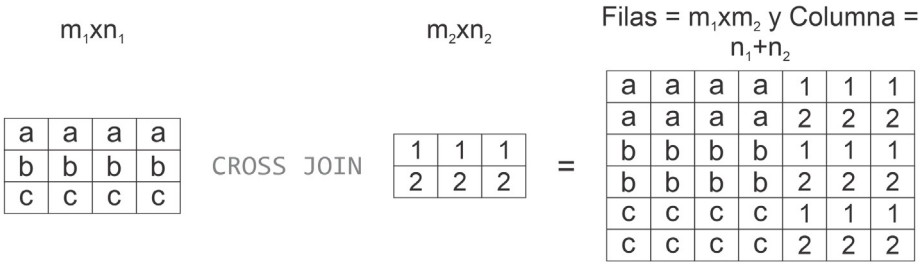
Las combinaciones realizadas con `INNER JOIN` y `OUTER JOIN` utilizan como referente una condición, de cuyo cumplimiento depende la operación. Por una parte, en el `INNER JOIN` se incluyen solo las filas de ambas tablas que cumplen la condición, mientras que en el `OUTER` se pueden mantener filas de las tablas que intervienen en la combinación aun si no cumplen la condición. Una tercera forma de combinación es la conocida como «producto cartesiano», «producto cruz» o simplemente `CROSS JOIN`.

El producto cartesiano opera de la siguiente forma: toma cada una de las filas de la primera tabla y la combina con cada una de las filas de la segunda tabla. Así, si se combinan por esta vía las tablas `ListasReproducción` y `Canciones`, se tomarán cada una de las listas de reproducción y se combinarán con cada una de las canciones sin tener en cuenta ninguna condición.

En ese orden de ideas, si `ListasReproducción` cuenta con 20 filas y `Canciones` incluye 20 filas, al final se obtendrá una nueva tabla con 400 filas puesto que por que cada fila de las listas de reproducción se harán 20 combinaciones. En cuanto a las columnas, la nueva tabla

tendrá tanto las columnas de `ListasReproducción` (suponga que sean cuatro) como las de `Canciones` (suponga que sean diez), es decir, 14 columnas en total. La Figura 5.13 ilustra esta combinación.

Figura 5.13. Resultado de combinación



El *Script* 5.21 muestra una forma para especificar el producto cartesiano entre dos tablas, y en el *Script* 5.22 se presenta una alternativa más explícita para realizar esta combinación a través de la palabra reservada `CROSS JOIN`, la cual es la recomendada como una buena práctica a nivel profesional porque mejora la legibilidad de las consultas.

***Script* 5.21**

```
SELECT *
FROM ListasReproducción, Canciones;
```

***Script* 5.22**

```
SELECT *
FROM ListasReproducción CROSS JOIN Canciones;
```

El `CROSS JOIN` puede emplearse para responder preguntas como la planteada a continuación:

¿Cuál es el título de las canciones que no están en la lista de reproducción con identificador *2*?
Mostrar también el nombre de la lista.

Con el *Script* 5.22 se obtuvo el producto cartesiano de todas las canciones con todas las listas o, lo que es lo mismo, todas las posibilidades de canciones que pueden pertenecer a cada una de las listas. Por consiguiente, el próximo paso sería conservar únicamente las

canciones que aún no han sido agregadas a la lista 2. Para dicho fin se puede emplear un condicional en el `WHERE`, tal como se indica en el *Script* 5.23.

***Script* 5.23**

```
SELECT LR.título,
       C.título
FROM ListasReproducción AS LR
       CROSS JOIN Canciones AS C
WHERE C.idcanción NOT IN(SELECT idcanción
                         FROM CancionesLista AS CL
                         WHERE CL.idListaReproducción = LR.idListaReproducción)
       AND LR.idListaReproducción = 2
ORDER BY LR.idlistareproducción
```

5.4. Aprendizajes más importantes del capítulo 5

- En la cláusula `FROM` se pueden realizar combinaciones de tablas de tres tipos: `INNER JOIN`, `OUTER JOIN` y `CROSS JOIN`.
- El `INNER JOIN` es útil en los escenarios donde la combinación se debe hacer entre las filas que cumplan una condición.
- El uso más común de las combinaciones mediante `INNER JOIN` es evaluar la igualdad de una clave foránea y una clave primaria. No obstante, en la condición se puede incluir cualquier expresión de la cual se pueda determinar su valor de verdad.
- El `OUTER JOIN` se emplea en casos en los que se desee mantener las filas que no cumplen la condición de combinación. Para conservar las de la izquierda, se usará el `LEFT OUTER JOIN`; para las de la derecha, el `RIGHT OUTER JOIN`, y para mantener ambas, el `FULL OUTER JOIN`.
- El `CROSS JOIN` se utiliza cuando es necesario combinar cada una de las filas de una tabla con cada una de las filas de la otra tabla. Esta operación también se conoce como producto cartesiano.
- La mayor parte de los DBMS soportan dos formas de escribir los `JOIN`: una siguiendo la sintaxis de 1986 y otra con la sintaxis de 1992. La comunidad recomienda como una buena práctica utilizar la de 1992 porque incrementa la legibilidad.

5.5. Actividades de aplicación para evidenciar lo aprendido

1. Escriba una consulta para obtener el título, la duración, el álbum original y el nombre del sello discográfico de este último para todas las canciones.
2. Escriba una consulta para calcular el dinero recaudado por país en el año 2020.
3. Proponga una pregunta que requiera obtener datos de al menos tres tablas y elabore la consulta que permita resolverla.

4. Escriba una consulta para obtener el título, el nombre del intérprete y la calificación promedio de cada una de las canciones en español.
5. Escriba una consulta para obtener un listado que contenga el título de todas las listas que hayan sido creadas por usuarios residentes en Colombia, el número de canciones incluidas en cada una, la duración total de las canciones incluidas y los *e-mails* de los usuarios que las crearon.
6. Modifique el siguiente *script* para obtener el mismo resultado sin utilizar subconsultas y argumente cuál es la necesidad de datos que se está resolviendo:

```
SELECT título,
       (SELECT nombre
        FROM Géneros
        WHERE Géneros.idGénero = Canciones.idGénero) AS género,
       duración,
       nombre,
       AVG(calificación) AS calificaciónPromedio
FROM canciones
       INNER JOIN Intérpretes
        ON Canciones.idIntérpretePrincipal = Intérpretes.idIntérprete
       LEFT JOIN Calificaciones
        ON Canciones.idCanción = Calificaciones.idCanción
WHERE Canciones.idCanción IN (SELECT idCanción
                              FROM Reproducciones
                              WHERE idUsuario IN
                              (
                                   SELECT IdUsuario
                                   FROM Usuarios
                                   WHERE paísResidencia = 'Francia'
                              ))
       AND idÁlbumOriginal IN (SELECT idalbum
                              FROM Álbumes
                              WHERE fechaLanzamiento < '01/01/2000')
GROUP BY título,
       duración,
       nombre,
       género
```

7. Escriba una consulta para obtener el nombre completo de los músicos que han hecho parte de más de un grupo durante su carrera y el nombre de los grupos en los que han participado.
8. Identifique los elementos lógicos o sintácticos que impiden que la siguiente consulta se pueda ejecutar correctamente. Explique la incidencia que tiene cada uno de estos elementos en la ejecución de la consulta:

```
SELECT DISTINCT
       C.idCanción,
       C.título
FROM Canciones C
       INNER JOIN Reproducciones R INNER JOIN CancionesLista CL
         ON Canciones.idCanción = R.idCanción AND R.idCanción =
CancionesLista.idCanción
WHERE CL.idListaReproducción IS NULL
```

9. Escriba una consulta para calcular la cantidad de canciones lanzadas como «sencillos» por cada intérprete. Si todas las canciones de un artista hacen parte de algún álbum, el nombre de ese músico también debe incluirse en el resultado y la cantidad de sencillos debe ser cero.
10. Proponga una consulta que utilice CROSS JOIN y explique por qué esta tendría sentido o utilidad dentro del funcionamiento de la plataforma *«Mis Canciones»*.

Capítulo 6

Operaciones de conjunto y funciones de ventana

Resultados de aprendizaje

– *Construye consultas en SQL que implementan la unión, la intersección y la diferencia de conjuntos de filas con estructuras similares.*
– *Construye consultas en SQL que operan a nivel de fila para producir datos que resumen subconjuntos o ventanas de filas.*

En los capítulos anteriores se abordó de manera incremental la obtención de datos desde una base de datos relacional y algunas operaciones de manipulación de los datos para satisfacer las necesidades expresadas en forma de preguntas. Las consultas construidas recuperan datos desde una o varias tablas, y estos pueden presentarse de forma resumida, agrupada o transformada utilizando funciones escalares y funciones de agregación.

Aunque estas operaciones presentadas son muy potentes y permiten resolver la mayoría de los requerimientos de datos, el SQL incluye otras operaciones que brindan distintas posibilidades para aprovechar los datos almacenados. Este capítulo está centrado en varias operaciones de manipulación de datos obtenidos con una o varias consultas. Específicamente, se trabajará con las operaciones de conjunto y las funciones de ventana, en la versión 4 de la base de datos *«Mis Canciones»*.

6.1. Operaciones de conjunto

Las bases de datos relacionales están fundamentadas en el álgebra relacional y la teoría de conjuntos. Hay que recordar que una tabla es un conjunto de filas, de forma que es natural

pensar la posibilidad de aplicar operaciones de conjuntos como la unión, la intersección o la diferencia a las tablas de una base de datos o a las obtenidas al ejecutar una o varias consultas.

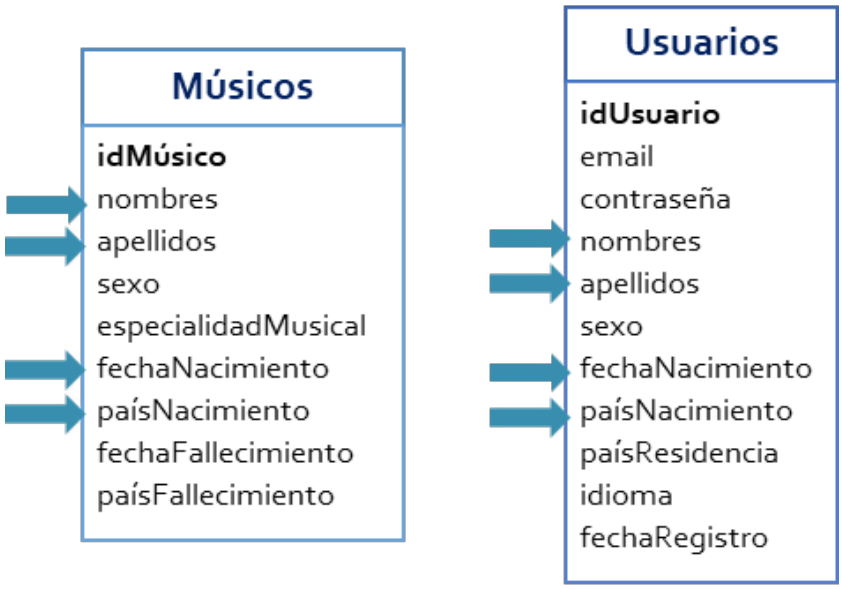
En el SQL, las operaciones de conjunto se aplican comúnmente sobre los conjuntos de filas obtenidos por dos consultas. Pueden unirse las filas de las dos tablas resultantes, hallar la intersección de las filas o identificar cuáles filas se obtuvieron en una consulta, pero no en la otra.

Ahora bien, el hecho de que las filas sean los elementos de los conjuntos que intervienen en estas operaciones impone dos restricciones que deben tener las tablas sobre las que se van a ejecutar. La primera es que los conjuntos de filas, es decir, las tablas participantes en la operación deben tener la misma cantidad de columnas. La segunda es que las columnas ubicadas en la misma posición en ambas tablas deben tener exactamente el mismo tipo de datos. Para iniciar el aprendizaje de estas operaciones, se propone la siguiente pregunta:

¿Cuáles son los nombres, los apellidos, el sexo, la fecha de nacimiento y el país de nacimiento de las personas registradas en la base de datos, bien sean usuarios o músicos?

Para responder a esta pregunta, puede identificarse que los datos requeridos están almacenados en las tablas `Usuarios` y `Músicos`, las cuales se presentan en la Figura 6.1. Específicamente, es necesario obtener algunos de los datos de los usuarios y unir ese resultado con lo que se obtenga de consultar los datos de los músicos.

Figura 6.1. Tablas y columnas requeridas para responder la pregunta



La operación unión toma las filas de los conjuntos de datos participantes y las entrega como una sola tabla en la que no hay filas repetidas. Por ejemplo, si se quisiera unir una tabla obtenida a partir de una consulta que devuelve el nombre y el país de músicos que son solistas con otra tabla que es el resultado de una consulta sobre el nombre y el país de músicos que participan en grupos, la tabla final tendría dos columnas: el nombre y el país, y todas las filas de las dos tablas participantes, sin filas duplicadas.

En la Figura 6.2 se muestra cómo sería la operación unión de las dos tablas de datos. Como se puede apreciar, la tabla resultante tendría siete filas porque las filas que están repetidas —los músicos de nombres «Marie» y «Baruch»— se presentan una sola vez.

Figura 6.2. Ejemplo de la operación `UNION`

Para responder la pregunta sobre los datos de los usuarios y de los músicos, se necesita generar una tabla con cinco columnas y con filas provenientes de ambas tablas. En este sentido, la solución puede abordarse dividiendo el problema en otros dos más pequeños. En primer lugar, hay que construir una consulta para obtener los datos de los usuarios, como la presentada en el *Script* 6.1, y unir ese resultado con lo que se obtenga al ejecutar otra consulta enfocada en los datos de los músicos, como la presentada en el *Script* 6.2.

***Script* 6.1**

```
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento
FROM Usuarios
```

***Script* 6.2**

```
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento
FROM Músicos
```

Como las dos tablas resultantes van a tener cinco columnas y sus tipos de datos serán correspondientes columna a columna, es viable realizar la operación `UNION`. En el *Script* 6.3 se presenta esta consulta.

***Script* 6.3**

```
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento
FROM Usuarios
UNION
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento
FROM Músicos
```

La tabla resultante al ejecutar la consulta del *Script* 6.3 tendrá cinco columnas y mostrará las filas de la tabla `Usuarios` seguidas de las de la tabla `Músicos`. En este caso se excluirán todas las filas de la tabla resultante de la consulta sobre la tabla `Músicos` que tengan los mismos datos que alguna de las filas de la tabla resultante de la consulta sobre la tabla `Usuarios`. Esta situación podría suceder si algún músico tiene un usuario registrado en la plataforma con los mismos datos con los que está registrado como músico.

No obstante, si en algún escenario deben mantenerse las filas duplicadas, puede añadirse la palabra `ALL` después de la cláusula `UNION`. También se recomienda utilizar `UNION ALL` si se tiene certeza de que no hay filas duplicadas, lo cual ofrecerá un mejor rendimiento que el conseguido al usar únicamente `UNION` porque se obtendría el mismo resultado, las mismas filas, ahorrándole la tarea al DBMS de realizar la comprobación de duplicidad.

Para mejorar la presentación de la tabla resultante, podría agregarse una columna con un valor escalar que denote el tipo de persona registrada en cada fila. En otras palabras, es posible sumar una columna para incluir una cadena de texto que diga si la fila proviene de la tabla `Usuarios` o de la tabla `Músicos`. De este modo se obtiene una nueva versión de la

consulta en el Script 6.4. Ahora, como hay certeza de que no existirán filas duplicadas dado que se está añadiendo la columna `tipoPersona` con valores diferentes en cada caso, es recomendable utilizar `UNION ALL`.

***Script* 6.4**

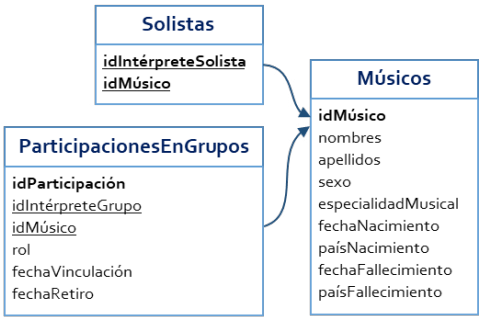
```
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento,
       'Usuario' AS tipoPersona
FROM Usuarios
UNION ALL
SELECT nombres,
       apellidos,
       sexo,
       fechaNacimiento,
       paísNacimiento,
       'Músico' AS tipoPersona
FROM Músicos
```

La operación `UNION` también puede utilizarse en subconsultas. Para observar esta aplicación, se resolverá la siguiente pregunta:

¿Cuáles son los nombres, los apellidos, el sexo y el país de nacimiento de los músicos que han sido solistas o que han pertenecido a grupos?

Del análisis de la pregunta puede identificarse que se requieren datos que están en la tabla `Músicos`, pero a partir de los datos de las tablas `Solistas` y `ParticipacionesEnGrupos`, las cuales se presentan en la Figura 6.3. Es preciso señalar que los músicos compositores que no han sido solistas o que no han pertenecido a grupos deben excluirse del resultado.

Figura 6.3. Tablas requeridas para responder la pregunta



Para resolver esta pregunta primero debe obtenerse un conjunto de datos que contenga los identificadores de los músicos que han sido solistas o que han participado en grupos. Para dicho fin pueden unirse los identificadores de los músicos que están almacenados en la tabla `Solistas` con los de los músicos de la tabla `ParticipacionesEnGrupos` mediante la consulta del Script 6.5.

Aunque para efectos del resultado final no habría ninguna diferencia, en el Script 6.5 se incluye `UNION ALL` porque no es necesario eliminar los duplicados. Esto se debe a que el resultado se utilizará como una subconsulta, con el operador `IN`, para filtrar las filas de la tabla `Músicos`, tal y como se observa en el *Script* 6.6.

***Script* 6.5**

```
SELECT idMúsico
FROM Solistas
UNION ALL
SELECT idMúsico
FROM ParticipacionesEnGrupos
```

***Script* 6.6**

```
SELECT CONCAT(M.nombres, ' ', M.apellidos) AS nombreCompleto,
       M.sexo,
       M.paísNacimiento
FROM Músicos AS M
WHERE idMúsico IN(SELECT idMúsico
       FROM Solistas
       UNION ALL
       SELECT idMúsico
       FROM ParticipacionesEnGrupos);
```

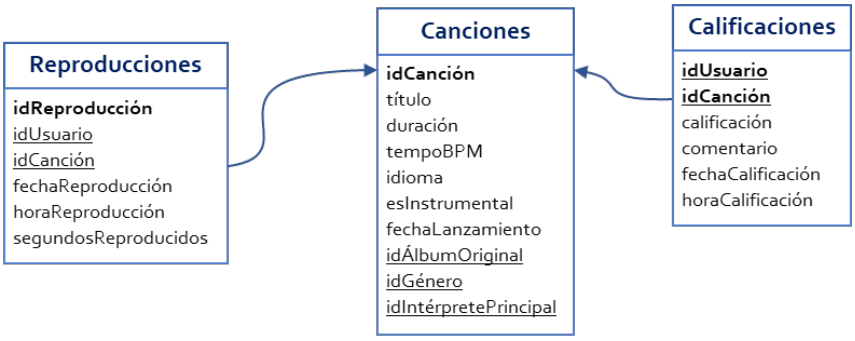
La operación `UNION` puede combinarse con las demás operaciones para responder preguntas como la que se plantea a continuación:

¿Cuál es título de la canción más reproducida y el de la canción con la calificación promedio más alta?

En esta pregunta se requieren datos almacenados en la tabla `Canciones`; específicamente, el título de la canción, pero se necesitan a su vez datos de las tablas `Reproducciones` y `Calificaciones` para obtener los criterios de filtrado. En la Figura 6.4 se presenta el diagrama con las tablas requeridas para responder la pregunta.

Lo primero que se realizará para resolver este interrogante será determinar cuál es la canción más reproducida mediante el *Script* 6.7. En esta consulta se agrega el texto «Más reproducida» en una nueva columna denominada `criterio`. Hay que anotar que varias canciones pueden ser las más reproducidas si tienen la misma cantidad de reproducciones.

Figura 6.4. Tablas requeridas para responder la pregunta



***Script* 6.7**

```
SELECT C.idCanción,
       título,
       'Más reproducida' AS Criterio
FROM Reproducciones R
       INNER JOIN Canciones C ON R.idCanción = C.idCanción
GROUP BY C.idCanción
HAVING COUNT(idReproducción) =
             (SELECT MAX(CantidadReproducciones)
              FROM
              (SELECT COUNT(idReproducción) CantidadReproducciones
              FROM Reproducciones
              GROUP BY idCanción) ReproduccionesPorCanción)
```

***Script* 6.8**

```
SELECT idCanción,
       (SELECT título
       FROM Canciones C
       WHERE C.idCanción = Cal.idCanción),
       'Mejor Valorada' AS Criterio
FROM Calificaciones Cal
GROUP BY idCanción
HAVING AVG(Calificación) = (SELECT MAX(CalificaciónPromedio)
                            FROM
                           (SELECT AVG(Calificación) CalificaciónPromedio
                           FROM Calificaciones
                           GROUP BY idCanción) CalificacionesPorCanción)
```

***Script* 6.9**

```
SELECT C.idCanción,
       título,
       'Más reproducida' AS Criterio
FROM Reproducciones R
       INNER JOIN Canciones C ON R.idCanción = C.idCanción
GROUP BY C.idCanción
HAVING COUNT(idReproducción) =
       (SELECT MAX(CantidadReproducciones)
       FROM
       (SELECT COUNT(idReproducción) CantidadReproducciones
       FROM Reproducciones
       GROUP BY idCanción) ReproduccionesPorCanción)
UNION
SELECT idCanción,
       (SELECT título
       FROM Canciones C
       WHERE C.idCanción = Cal.idCanción),
       'Mejor Valorada' AS Criterio
FROM Calificaciones Cal
GROUP BY idCanción
HAVING AVG(Calificación) = (SELECT MAX(CalificaciónPromedio)
                            FROM
                            (SELECT AVG(Calificación) CalificaciónPromedio
                            FROM Calificaciones
                            GROUP BY idCanción) CalificacionesPorCanción)
```

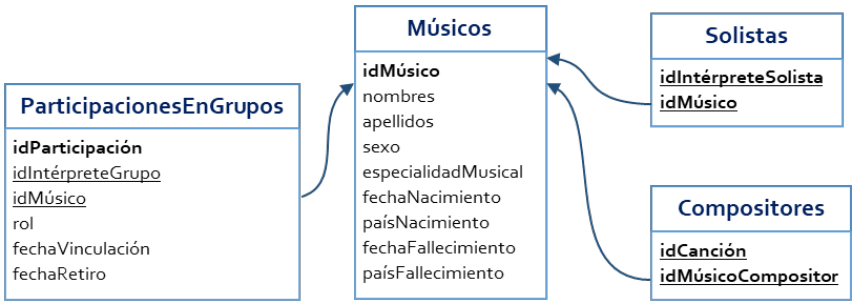
De manera similar, el *Script* 6.8 permite obtener la canción con mejor promedio de calificación. En este caso también se añade el texto «Mejor valorada» a una nueva columna denominada `criterio`, que permite etiquetar cuál es la canción con mejor calificación promedio dentro de la colección. Con estas dos consultas, puede utilizarse la operación `UNION` y obtener en una misma tabla las canciones con más reproducciones y las de mejor valoración, tal como se muestra en el *Script* 6.9.

Las operaciones de conjunto permiten resolver preguntas como la siguiente:

¿Cuál es la cantidad de músicos por país de nacimiento que han sido solistas, compositores y también han participado en grupos musicales?

Esta pregunta requiere datos de la tabla `Músicos`, pero para responderla se necesita realizar operaciones de filtrado basadas en los datos de las tablas `Compositores`, `Solistas` y `ParticipacionesEnGrupos`, tal como se ilustra en la Figura 6.5.

Figura 6.5. Tablas que dan respuesta al interrogante



En este orden de ideas, la respuesta a esta pregunta implica obtener los datos de los músicos, específicamente los identificadores, que estén almacenados en tres tablas. Para lograrlo, es posible utilizar tres subconsultas, pero también puede recurrirse a la operación intersección.

En la intersección se obtienen solo las filas que son comunes en los dos conjuntos, es decir, las filas que están presentes en las dos tablas participantes. Por ejemplo, al realizar la intersección entre los músicos que son solistas (conjunto A) y los músicos que han pertenecido a grupos musicales (conjunto B) que se muestran en la Figura 6.6, el resultado es una tabla de dos filas y dos columnas, con los datos de los músicos de nombres *«Marie»* y *«Baruch»*.

Figura 6.6. Ejemplo de la operación intersección

Regresando a la pregunta, el primer paso para responderla es obtener los identificadores de los músicos que aparecen en las tablas `Solistas`, `Compositores` y `ParticipacionesEnGrupos`. Hay que obtener solamente los que estén presentes en las tres: si un identificador está apenas en dos tablas, no debe incluirse. La consulta que permite obtener este conjunto de identificadores se presenta en el *Script* 6.10.

A su vez, el *Script* 6.10 puede utilizarse como una subconsulta que genera los datos requeridos para filtras las filas de la tabla `Músicos`. Además, es necesario agrupar por la columna `paisNacimiento` y contar con la función `COUNT` la cantidad de filas que tiene cada país (grupo), tal como se plantea en el *Script* 6.11.

***Script* 6.10**

```
SELECT idMúsico
FROM Solistas
INTERSECT
SELECT idMúsicoCompositor
FROM Compositores
INTERSECT
SELECT idMúsico
FROM ParticipacionesEnGrupos;
```

***Script* 6.11**

```
SELECT M.paísNacimiento,
       COUNT(*) AS CantidadMúsicos
FROM Músicos AS M
WHERE idMúsico IN(SELECT idMúsico
                  FROM Solistas
                 INTERSECT
                 SELECT idMúsicoCompositor
                 FROM Compositores
                 INTERSECT
                 SELECT idMúsico
                 FROM ParticipacionesEnGrupos)
GROUP BY M.paísNacimiento;
```

La tercera operación de conjuntos disponible en el SQL es la operación diferencia, la cual obtiene las filas del conjunto A que no están en el conjunto B. Como se iustra en la Figura 6.7, mediante EXCEPT se calcula la diferencia entre los músicos que son solistas (conjunto A, a la izquierda de la palabra EXCEPT) y los que han pertenecido a grupos (conjunto B, a la derecha de la palabra EXCEPT). Así, el conjunto de datos resultante contendrá únicamente los músicos solistas que nunca han estado en un grupo musical; en este caso: son los músicos: «Ana», «René», «Fatme» y «Jose».

Figura 6.7. Ejemplo de la operación diferencia

La operación diferencia es útil para responder preguntas como la siguiente:

¿Cuáles son los nombres y los apellidos de los solistas que no han compuesto canciones?

En esta oportunidad se están pidiendo los nombres y los apellidos de los músicos cuyo identificador esté almacenado en la tabla `Solistas` pero no en la tabla `Compositores`. Esto puede obtenerse con la operación diferencia o `EXCEPT`, tal como se presenta en el *Script* 6.12.

***Script* 6.12**

```
SELECT idMúsico
FROM Solistas
EXCEPT
SELECT idMúsicoCompositor
FROM Compositores;
```

El siguiente paso será incluir la consulta del *Script* 6.12 como una subconsulta dentro de una consulta principal a la tabla `Músicos`, como se muestra en el *Script* 6.13.

***Script* 6.13**

```
SELECT CONCAT(M.nombres, ' ', M.apellidos) AS nombreCompleto,
       M.sexo,
       M.paísNacimiento
FROM Músicos AS M
WHERE idMúsico IN(SELECT idMúsico
                  FROM Solistas
                  EXCEPT
                  SELECT idMúsicoCompositor
                  FROM Compositores);
```

6.2. Funciones de ventana

En el capítulo 3 se trabajó con las funciones escalares y las de agrupamiento. Las primeras están diseñadas para operar sobre un valor, y las segundas, para resumir un conjunto de valores. En el SQL existen además otro tipo de funciones, denominadas de ventana, las cuales reciben este particular nombre porque basan su funcionamiento sobre subconjuntos o ventanas de filas. Para iniciar el aprendizaje de este tipo de funciones, se propone la siguiente pregunta:

¿Cuál es el identificador, el título, la duración y el género de cada canción y la duración promedio de las canciones del mismo género?

Para responder esta pregunta deben utilizarse únicamente los datos almacenados en la tabla `Canciones`. Sin embargo, lo que se está solicitando tiene dos niveles diferentes de agregación. Los primeros cuatro datos son propios de cada canción: son columnas que están en la tabla y solamente deben mostrarse sin realizar algún procesamiento. En cambio, el quinto dato, la quinta columna de la tabla que se espera obtener con la consulta, corresponde a un valor que debe calcularse: la duración promedio de todas las canciones que están categorizadas en el mismo género que la canción presentada en cada fila.

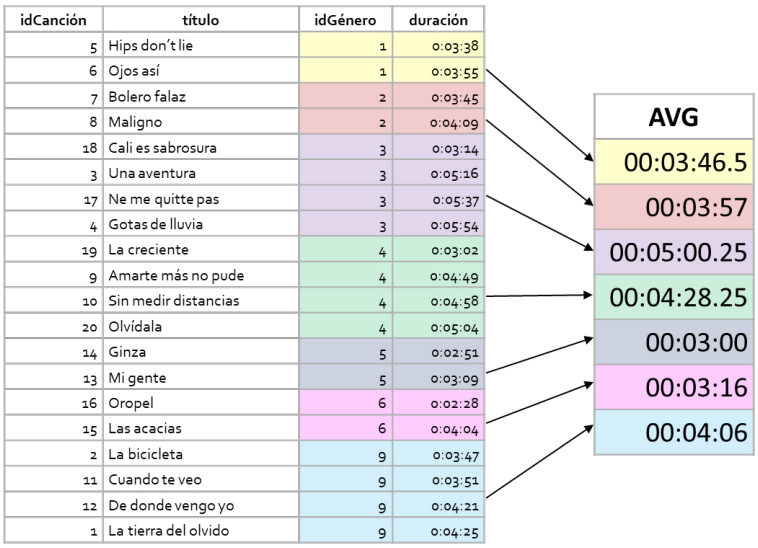
Una primera alternativa para implementar esta consulta es realizar una subconsulta para obtener la duración promedio por cada género y luego hacer una combinación interna o `INNER JOIN` de esta subconsulta con la tabla `Canciones`. En la consulta que se presenta en el *Script* 6.14 se calcula el promedio de los datos almacenados en la columna `duración`, pero se genera un valor para cada género, es decir, un promedio por cada valor diferente que esté almacenado en la columna `idGénero`.

***Script* 6.14**

```
SELECT AVG(duración)
FROM Canciones
GROUP BY idGénero
```

Agrupar por `idGénero` determina que el procesamiento de la función de agregación AVG no se realiza sobre todas las filas de la tabla `Canciones`, sino por subconjuntos o «ventanas» de filas de esta tabla. En otras palabras, se arman grupos o ventanas de procesamiento por cada valor distinto que se encuentre en dicha columna y se aplica la función, tal como se ilustra en la Figura 6.8.

Figura 6.8. Ejemplo de procesamiento por subconjuntos o grupos de filas



idCanción	título	idGénero	duración
5	Hips don't lie	1	0:03:38
6	Ojos así	1	0:03:55
7	Bolero falaz	2	0:03:45
8	Maligno	2	0:04:09
18	Cali es sabrosura	3	0:03:14
3	Una aventura	3	0:05:16
17	Ne me quitte pas	3	0:05:37
4	Gotas de lluvia	3	0:05:54
19	La creciente	4	0:03:02
9	Amarte más no pude	4	0:04:49
10	Sin medir distancias	4	0:04:58
20	Olvídala	4	0:05:04
14	Ginza	5	0:02:51
13	Mi gente	5	0:03:09
16	Oropel	6	0:02:28
15	Las acacias	6	0:04:04
2	La bicicleta	9	0:03:47
11	Cuando te veo	9	0:03:51
12	De donde vengo yo	9	0:04:21
1	La tierra del olvido	9	0:04:25

AVG
00:03:46.5
00:03:57
00:05:00.25
00:04:28.25
00:03:00
00:03:16
00:04:06

Para implementar la alternativa de solución planteada antes, se requiere entonces utilizar una consulta que tenga el identificador del género y la duración promedio de las canciones de cada género. Esta tabla de dos columnas puede obtenerse con la consulta que se presenta en el *Script* 6.15, la cual puede combinarse con la tabla `Canciones`, tal como se muestra en el *Script* 6.16.

***Script* 6.15**

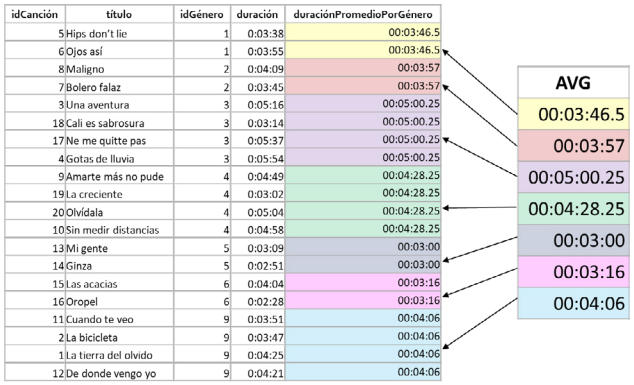
```
SELECT idGénero,
       AVG(duración) AS duraciónPromedioPorGénero
FROM Canciones
GROUP BY idGénero
```

***Script* 6.16**

```
SELECT idCanción,
       título,
       Canciones.idGénero,
       duración,
       duraciónPromedioPorGénero
FROM Canciones
       INNER JOIN(SELECT idGénero,
                  AVG(duración) AS duraciónPromedioPorGénero
                  FROM canciones
                  GROUP BY idgénero) AS DuracionesPromedioPorGénero
              ON Canciones.idGénero = DuracionesPromedioPorGénero.idGénero
ORDER BY idGénero
```

El resultado de ejecutar la consulta del *Script* 6.16 será una tabla con las cuatro columnas obtenidas directamente de la tabla `Canciones` y una columna denominada `duraciónPromedioPorGénero`, la cual presenta el valor obtenido con la función AVG dentro de la subconsulta. En las filas de canciones del mismo género, el valor incluido en la columna `duraciónPromedioPorGénero` es el mismo.

Figura 6.9. Ejemplo de agrupamiento de filas con utilizando cláusula GROUP BY



idCanción	título	idGénero	duración	duraciónPromedioPorGénero
5	Hips don't lie	1	0:03:38	00:03:46.5
6	Ojos así	1	0:03:55	00:03:46.5
8	Maligno	2	0:04:09	00:03:57
7	Bolero falaz	2	0:03:45	00:03:57
3	Una aventura	3	0:05:16	00:05:00.25
18	Cali es sabrosura	3	0:03:14	00:05:00.25
17	Ne me quitte pas	3	0:05:37	00:05:00.25
4	Gotas de lluvia	3	0:05:54	00:05:00.25
9	Amarte más no pude	4	0:04:49	00:04:28.25
19	La creciente	4	0:03:02	00:04:28.25
20	Olvídala	4	0:05:04	00:04:28.25
10	Sin medir distancias	4	0:04:58	00:04:28.25
13	Mi gente	5	0:03:09	00:03:00
14	Ginza	5	0:02:51	00:03:00
15	Las acacias	6	0:04:04	00:03:16
16	Oropel	6	0:02:28	00:03:16
11	Cuando te veo	9	0:03:51	00:04:06
2	La bicicleta	9	0:03:47	00:04:06
1	La tierra del olvido	9	0:04:25	00:04:06
12	De donde vengo yo	9	0:04:21	00:04:06

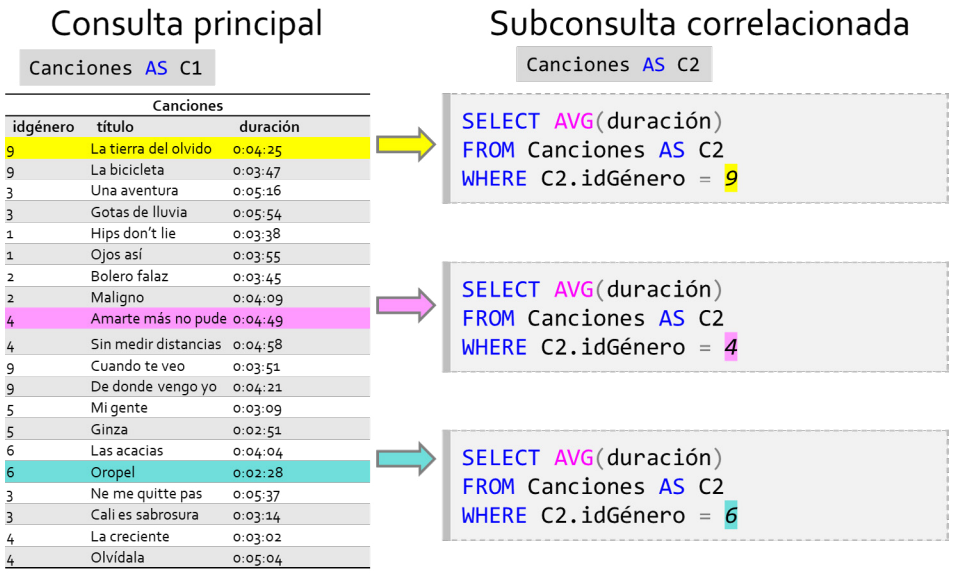
Una segunda alternativa para responder la pregunta es utilizar una subconsulta correlacionada, tal y como se presenta en el *Script* 6.17. Dicha subconsulta genera un único valor por cada fila de la tabla `Canciones` que tiene el alias `C1`, y dicho dato se muestra en la tabla resultante dentro de una nueva columna denominada `duraciónPromedioPorGénero`.

***Script* 6.17**

```
SELECT idCanción,
       título,
       idGénero,
       duración,
       (SELECT AVG(duración)
        FROM Canciones AS C2
        WHERE C2.idGénero = C1.idGénero) AS duraciónPromedioPorGénero
FROM Canciones AS C1
ORDER BY idGénero
```

La subconsulta se ejecuta por cada fila variando el valor utilizado en la cláusula `WHERE` para filtrar las filas de la tabla identificada con el alias `C2`, tomando los datos de la columna `idGénero` de cada fila de la tabla `C1`, tal y como se ilustra en la Figura 6.10.

Figura 6.10. Ejemplificación de la ejecución de una subconsulta correlacionada



Una tercera alternativa es utilizar el mismo concepto general del procesamiento por grupos que se hace con la cláusula `GROUP BY`, pero en lugar de grupos se definen particiones o ventanas de filas sobre las cuales se ejecuta una función; en este caso, la función promedio. En el *Script* 6.18 se presenta la consulta implementada de esta forma.

***Script* 6.18**

```
SELECT idCanción,
       título,
       idGénero,
       duración,
       AVG(duración) OVER(PARTITION BY idGénero) AS duraciónPromedioPorGénero
FROM Canciones
```

En este caso, la función AVG deja de ser una función de agregación tradicional, aplicada a todos los datos o a un grupo de datos, y pasa a ser una función que se calcula sobre una «partición» o ventana de filas de la tabla `Canciones`. La ventana de datos para calcular el promedio que se va a presentar en la columna con el alias `duraciónPromedioPorGénero` se conforma por las filas que tengan los mismos valores en la columna `idGénero`.

La palabra OVER indica que la función AVG se va a aplicar sobre un conjunto de datos específicos, sin que deba incluirse la cláusula de agrupamiento GROUP BY, porque el resultado final debe mostrar todas las filas de la tabla `Canciones`. Luego de OVER, entre paréntesis se utiliza la expresión PARTITION BY, con la cual se determina que las particiones o subconjuntos de datos sobre los que se va a ejecutar la función AVG serán aquellos que tengan el mismo valor en la columna `idGénero`.

En general, las funciones de ventana permiten obtener un escalar por cada fila de la tabla o de las combinaciones de tablas de la consulta principal. La función se ejecuta sobre (OVER) los subconjuntos o ventanas de filas que se declaran utilizando la expresión PARTITION BY. Las ventanas o particiones estarán constituidas por filas que tengan los mismos valores en una columna o las mismas combinaciones de valores en varias columnas.

Si no se indica alguna columna para realizar la partición, es decir, si no se incluye el nombre de alguna columna después de la expresión PARTITION BY, la función se ejecutará sobre todas las filas de la tabla resultante de la consulta principal. Este comportamiento es similar a lo que sucede cuando se utilizan las funciones de agregación sin la cláusula GROUP BY.

Dentro de OVER también puede definirse el orden en que las filas que conforman la ventana serán procesadas. Para ese fin se utiliza la cláusula ORDER BY seguida de los nombres de las columnas que se tendrán en cuenta para ordenar las filas durante el procesamiento con la función de ventana.

En el SQL hay varias funciones que pueden aplicarse sobre ventanas o particiones de filas: las de agregación que ya se habían utilizado y otras que permiten obtener algunos datos útiles en varios casos de uso, principalmente en la generación de reportes. Para ilustrar esto, se propone la siguiente pregunta:

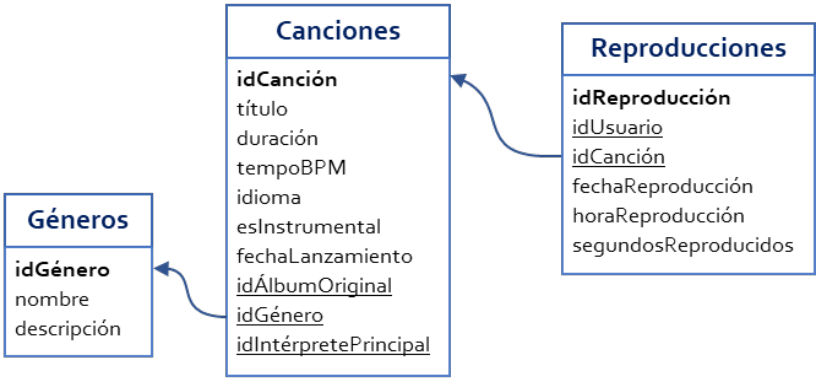
¿Cuál es el escalafón de las canciones con mayor número de reproducciones?

Mostrar el nombre del género y el título por cada canción.

En esta pregunta se pide una lista de las canciones, ordenada y enumerada en función de la cantidad de reproducciones que han realizado los usuarios para cada una, comenzando por el número uno para la canción con más reproducciones. Para lograr este resultado se requieren los datos almacenados en las tablas `Canciones`, `Géneros` y `Reproducciones`. En la Figura 6.11 se presentan los detalles de estas tablas.

Las tablas `Canciones`, `Géneros` y `Reproducciones` deben combinarse, y es preciso utilizar una función de agregación que permita contar el número de reproducciones, es decir, determinar el número de filas de la tabla `Reproducciones` en las que aparece el mismo dato en la columna `idCanción`. En este sentido, un primer paso para llegar a la respuesta requerida se logra con la consulta presentada en el *Script* 6.19, con la cual se genera una tabla con tres de las columnas solicitadas.

Figura 6.11. Tablas requeridas para responder la pregunta



***Script* 6.19**

```
SELECT G.nombre,
       C.título,
       COUNT(idReproducción) AS CantidadReproducciones
FROM Canciones C
     INNER JOIN Géneros G ON C.idGénero = G.idGénero
     LEFT JOIN Reproducciones R ON R.idCanción = C.idCanción
GROUP BY G.nombre,
         C.título
ORDER BY CantidadReproducciones DESC
```

La consulta del *Script* 6.19 utiliza una combinación izquierda para que en el listado se incluyan todas las canciones, incluso aquellas que no han sido reproducidas y cuyo identificador, por tanto, no esté almacenado en la tabla `Reproducciones`. La tabla resultante se presenta ordenada descendentemente por la cantidad de reproducciones.

Aunque el listado obtenido hasta aquí da la sensación de que ya ofrece la respuesta solicitada porque está ordenado, lo cierto es que cuando se utiliza el concepto escalafón o *ranking* se

espera que exista al menos una columna que indique la posición de cada canción en este. En otras palabras, se espera una columna con números naturales ordenados de menor a mayor, la cual representará el lugar que ocupa la canción en el escalafón. Para obtener esto puede recurrirse a un conjunto de funciones que ofrece el SQL y que permiten generar este tipo de columnas de «posiciones», tal como se hace en el *Script* 6.20.

***Script* 6.20**

```
SELECT G.nombre,
       C.título,
       COUNT(idReproducción) AS CantidadReproducciones,
       ROW_NUMBER() OVER(ORDER BY COUNT(idReproducción) DESC) fila,
       RANK() OVER(ORDER BY COUNT(idReproducción) DESC) puesto,
       DENSE_RANK() OVER(ORDER BY COUNT(idReproducción) DESC) puestoDenso
FROM Canciones C
       INNER JOIN Géneros G ON C.idGénero = G.idGénero
       LEFT JOIN Reproducciones R ON R.idCanción = C.idCanción
GROUP BY G.nombre,
       C.título
ORDER BY CantidadReproducciones DESC
```

Tabla 6.1. Tabla resultante del *Script* 6.20

Género	Título	Cantidad	fila	puesto	puestoDenso
Latina fusión	La bicicleta	51	1	1	1
Pop	Hips don't lie	42	2	2	2
Rock	Bolero falaz	39	3	3	3
Pop	Ojos así	38	4	4	4
Latina fusión	De donde vengo yo	32	5	5	5
Andina colombiana	Las acacias	26	6	6	6
Latina fusión	La tierra del olvido	25	7	7	7
Vallenato	Sin medir distancias	23	8	8	8
Salsa	Una aventura	22	9	9	9
Vallenato	Amarte más no pude	21	10	10	10
Reggaeton	Mi gente	21	11	10	10
Reggaeton	Ginza	20	12	12	11
Vallenato	Olvídala	18	13	13	12
Latina fusión	Cuando te veo	18	14	13	12

En el *Script* 6.20, construido a partir de la consulta presentada en el *Script* 6.19, se adicionaron tres columnas a la tabla resultante. Para ese fin se utilizaron las funciones de ventana denominadas ROW_NUMBER, RANK y DENSE_RANK.

La Tabla 6.1 es un ejemplo de un resultado posible tras ejecutar esta consulta. De este modo es más fácil comprender el funcionamiento de las tres funciones introducidas, las

cuales a primera vista podrían verse como equivalentes dado que todas enumeran filas, es decir, asignan un número natural correspondiente a la posición de la fila. Sin embargo, si se observan las filas resaltadas, pueden identificarse las diferencias en los números generados.

La función `ROW_NUMBER` enumera todas las filas y simplemente asigna un número entero que se va incrementando de uno en uno de acuerdo con el orden en que estas aparecen. Dicho número entero representa la posición de la fila dentro de la tabla y es lo que conforma la columna denominada `fila` de la Tabla 6.1.

Por su parte, la función `RANK` también asigna números enteros que se van incrementando de uno en uno, pero, a diferencia de la función `ROW_NUMBER`, asigna el mismo número o posición a las canciones que tienen la misma cantidad de reproducciones. Por esta razón, en la Tabla 6.1 hay dos filas que tienen el mismo valor en la columna `puesto`; específicamente, las filas 10 y 11, que tienen asignado el número 10, y las filas 13 y 14, con el número 13. En otras palabras, si hay un empate entre dos filas, ambas recibirán la misma posición. Hay que notar además que la posición 11 no se asigna, sino que la siguiente ubicación del *ranking* será la 12, tal como se observa para la fila 12 de la tabla.

Finalmente, la función `DENSE_RANK` opera de forma similar a la función `RANK`, es decir, asigna la misma posición a las canciones con la misma cantidad de reproducciones, pero la numeración se mantiene continua así se tengan empates. En la Tabla 6.1 esto se traduce en que en las filas 10 y 11 se asignó el valor 10 dentro de la columna `puestoDenso`; en otras palabras, esas dos canciones están empatadas en la posición 10. Enseguida, para la canción de la fila 12 se asignó la posición 11.

En la consulta también puede observarse que cuando se declara que las funciones se apliquen sobre ventanas, es decir, cuando se incluye la expresión `OVER`, no aparece la instrucción `PARTITION BY`. Únicamente está definido que la función debe aplicarse sobre filas ordenadas de manera descendente de acuerdo con el valor que se obtenga al contar la cantidad de reproducciones.

Las funciones de ventana son especialmente útiles cuando se crean particiones y se hacen operar las funciones sobre cada una de ellas. Un caso de uso en esta línea se presenta con la siguiente pregunta:

¿Cuál es el escalafón de las canciones reproducidas por cada género?

La pregunta planteada es similar a la resuelta con el *Script* 6.20, con la diferencia de que la enumeración debe hacerse por cada uno de los géneros. Esto se logra con la expresión `PARTITION BY`, en este caso indicando que la partición se haga por el género y que, por lo tanto, la función `DENSE_RANK` se aplique a cada partición o, lo que es lo mismo, a cada ventana conformada por las canciones pertenecientes a cada género. En el *Script* 6.21 se muestra una implementación de este tipo, la cual generará un resultado como el que se muestra en la Tabla 6.2.

***Script* 6.21**

```
SELECT
  C.título,
  COUNT(idReproducción) AS CantidadReproducciones,
  DENSE_RANK() OVER(PARTITION BY G.idGénero ORDER BY COUNT(idReproducción) DESC)
FROM Canciones C
       INNER JOIN Géneros G ON C.idGénero = G.idGénero
       INNER JOIN Reproducciones R ON R.idCanción = C.idCanción
GROUP BY G.idGénero,
         C.idCanción,
         C.título
```

Tabla 6.2. Ejemplo de resultado obtenido con el *Script* 6.21

Género	Título	Cantidad	*DENSE_RANK*
Pop	Hips don't lie	42	1
Pop	Ojos así	38	2
Rock	Bolero falaz	39	1
Salsa	Una aventura	22	1
Salsa	Ne me quitte pas	14	2
Salsa	Cali es sabrosura	6	3
Vallenato	Sin medir distancias	23	1
Vallenato	Amarte más no pude	21	2
Vallenato	Olvídala	18	3
Vallenato	La creciente	13	4
Reggaeton	Mi gente	21	1
Reggaeton	Ginza	20	2
Andina colombiana	Las acacias	26	1
Andina colombiana	Oropel	7	2

En la Tabla 6.2 puede observarse que la enumeración se ha hecho por cada género. En el caso del género «*Pop*» se tiene que la canción «*Hips don't lie*» es la más reproducida y que la canción «*Ojos así*» ocupa el segundo lugar. Por otra parte, para el género «*Salsa*» se observa que la más reproducida es «*Una aventura*», seguida por «*Ne me quitte pas*» y luego «*Cali es sabrosura*».

Ya habiendo presentado las funciones de ventana, es tiempo para comentar que las funciones de agregaciones, como se vio en el capítulo 3, son útiles para derivar nuevos datos en forma de agregados o resúmenes de los datos base, lo cual es muy valioso, pero trae consigo

un costo: pérdida de detalle. Al crear grupos de datos utilizando la cláusula `GROUP BY`, todos los cálculos que se realicen deben estar en el contexto de los grupos que han sido definidos. Sin embargo, es común que en una misma consulta se quiera tener el valor que proporciona trabajar con grupos de datos sin perder el detalle de las filas.

Las funciones de ventana no tienen la misma limitación. Este es uno de los escenarios de aplicación de las funciones de ventana debido a que estas son evaluadas por cada una de las filas de la consulta, pero se aplican a un subconjunto de las filas que se deriva de la consulta subyacente. Para hacer este uso más evidente, se resolverá la siguiente pregunta:

¿Cuál es el porcentaje del total de reproducciones de las canciones de la plataforma que representa las reproducciones de cada canción?

Para resolver esta necesidad de datos se requiere la cantidad total de reproducciones realizadas (CRT) y la cantidad de reproducciones realizadas por cada canción (CRC), luego multiplicar por 100 la cantidad de reproducciones de una canción en particular y, por último, dividir el valor resultante entre la cantidad total de reproducciones (100 * CRC/CRT).

Hacer este proceso utilizando únicamente agrupamientos y subconsultas implicaría construir una consulta bastante extensa. No obstante, con el uso de las funciones de ventana se simplifica la consulta a unas cuantas líneas; solo sería necesario realizar un conteo de todas las filas de la tabla reproducciones, es decir, sin hacer particiones, para obtener CRT, y un conteo de la cantidad de reproducciones por canciones, esto es, haciendo la partición por canción, para obtener CCT. En el *Script* 6.22 se presenta una consulta que ofrece un primer acercamiento a la solución.

***Script* 6.22**

```
SELECT
  R.idCanción,
  COUNT(*) OVER() AS CRT,
  COUNT(*) OVER(PARTITION BY R.idCanción) AS CRC,
  100.0 * COUNT(*) OVER(PARTITION BY R.idCanción) / COUNT(*) OVER() AS porcentaje
FROM Reproducciones R
ORDER BY idCanción
```

Ahora bien, al analizar con detalle el *Script* 6.22 puede identificarse un riesgo de que se muestren filas repetidas, es decir, el mismo valor por cada canción dependiendo de si esta ha sido reproducida varias veces. Para evitar esto debe utilizarse `DISTINCT`, tal como se presenta en el *Script* 6.23.

***Script* 6.23**

```
SELECT DISTINCT
  R.idCanción,
  COUNT(*) OVER() AS CRT,
  COUNT(*) OVER(PARTITION BY R.idCanción) AS CRC,
  100.0 * COUNT(*) OVER(PARTITION BY R.idCanción) / COUNT(*) OVER() AS porcentaje
FROM Reproducciones R
ORDER BY idCanción
```

En el SQL hay otras funciones de ventana que deben estudiarse porque su aplicación es muy común en contextos empresariales.

6.3. Aprendizajes más importantes del capítulo 6

- Las operaciones de conjunto pueden ser aplicadas sobre el resultado de dos consultas SQL que tengan la misma estructura y donde las columnas con posición coincidente presenten igual tipo de datos.
- Las operaciones disponibles son la unión, la intersección y la diferencia.
- Las funciones de ventana permiten realizar operaciones sobre el conjunto de datos devuelto por una consulta, bien sea tomando todas las filas como una ventana o haciendo particiones sobre ella.
- Lo más sorprendente de las funciones de ventanas es que pueden ejecutarse operaciones de agregación o resumen sin perder el detalle, lo que sí ocurre con las funciones de agregación y con la operación de agrupamiento.

6.4. Actividades de aplicación para evidenciar lo aprendido

1. Escriba una consulta que permita obtener la cantidad de canciones reproducidas por cada mes del año, de manera que cada nombre del mes se muestre en una columna, es decir, una columna para enero, otra para febrero, y así sucesivamente.
2. Utilizando operaciones de conjunto, obtenga el nombre completo y el año de nacimiento de los usuarios que han reproducido al menos una canción, pero no han calificado ninguna.
3. Escriba una consulta SQL, sin hacer uso de subconsultas, que permita identificar cuál es el nombre de la lista de reproducción con más canciones. Si la cantidad máxima de canciones es compartida por más de una lista, debe obtenerse solo una de estas.
4. Obtenga la diferencia entre la duración de cada canción y la duración de la canción más corta de su mismo género.
5. Escriba una consulta que muestre el monto total que se ha pagado por cada tipo de plan, el monto que corresponde al total pagado por cada uno de los sexos por tipo de plan y el monto que corresponde dentro de cada plan a cada sexo por país.

Capítulo 7

Otras operaciones de consulta

Resultados de aprendizaje

– *Construye consultas en SQL en las que se procesan expresiones condicionales compuestas con las que se derivan datos a partir de lo contenido en alguna columna de una tabla.*
– *Controla la cantidad de filas que se obtienen al ejecutar una consulta con base en el número de la fila y la posición de la fila dentro de la tabla.*
– *Construye consultas en SQL que pivotan filas en columnas para generar nuevas dimensiones de agregación o presentación de los resultados.*

El SQL ofrece una gran cantidad de opciones para el desarrollador que debe concebir, diseñar e implementar consultas en una base de datos relacional. En la mayoría de los casos, las respuestas a las necesidades de datos pueden obtenerse utilizando los elementos del lenguaje que se han trabajado hasta el momento. Sin embargo, hay necesidades que demandan otras operaciones que potencian mucho más la capacidad de procesamiento que alberga una consulta. En este sentido, este capítulo introduce las expresiones `CASE`, `LIMIT` y `OFFSET`, y las operaciones para generar columnas al pivotar datos de filas en nuevas columnas. En esta oportunidad se trabajará con la versión 4 de la base de datos *«Mis Canciones»*.

7.1. Expresión CASE

`CASE` es una expresión escalar de gran utilidad que permite obtener un valor basado en una condición lógica. Sin embargo, hay que tener presente que `CASE` es una expresión, no una sentencia, es decir, su función no es controlar el flujo de ejecución de la consulta, sino retornar un valor dependiendo de una o varias condiciones. Adicionalmente, como `CASE`

es una expresión escalar, es permitida en cualquier lugar en el que se acepte un escalar, es decir, en las cláusulas `SELECT`, `WHERE`, `HAVING` y `ORDER BY`. Para ejemplificar el uso de esta expresión, se resolverá la siguiente pregunta:

¿Cuál es el nombre completo de los usuarios y la cantidad de canciones que han calificado?
Clasificar los usuarios en cuatro categorías en función de la cantidad de canciones calificadas: más de ocho canciones, entre cuatro y ocho canciones, entre una y tres canciones y ninguna canción.

La novedad de la pregunta anterior con respecto a las otras que se han trabajado es que se pide derivar un escalar de tipo texto dependiendo de la cantidad de canciones que un usuario haya calificado. Esto es lo que se puede realizar con la expresión `CASE`: establecer que, cuando la cantidad de calificaciones cumpla una condición que se especifica con la palabra reservada `WHEN`, se evalúe una expresión que se indica con la palabra reservada `THEN`.

En este caso, cuando (`WHEN`) la cantidad de canciones calificadas (`COUNT`(calificación)) sea mayor a ocho, entonces (`THEN`) debe obtenerse el texto «más de ocho canciones». Cuando se encuentre entre cuatro y ocho, se indicará «entre cuatro y ocho canciones», y si es entre uno y tres, entonces se escribirá «entre una y tres canciones». Si ninguna de las condiciones anteriores se cumple (`ELSE`), se arrojará como resultado «Ninguna canción». El *Script* 7.1 refleja todo este razonamiento escrito en SQL.

***Script* 7.1**

```
SELECT CONCAT(U.nombres, ' ', U.apellidos) AS nombreCompleto,
       CASE
           WHEN COUNT(calificación) > 8
             THEN 'más de ocho canciones'
           WHEN COUNT(calificación) > 3 AND COUNT(calificación) <= 8
             THEN 'entre cuatro y ocho canciones'
           WHEN COUNT(calificación) >= 1 AND COUNT(calificación) <= 3
             THEN 'entre 1 y tres canciones'
           ELSE 'Ninguna canción'
       END AS clasificación
FROM Usuarios U LEFT JOIN Calificaciones C ON U.idUsuario = C.idUsuario
GROUP BY U.idUsuario, CONCAT(U.nombres, ' ', U.apellidos)
ORDER BY COUNT(calificación) DESC;
```

Como ya se mencionó, `CASE` puede ser incluida en cualquier lugar donde se acepte un escalar. Por lo tanto, cabe reformular la pregunta anterior para dejarla como se presenta a continuación:

¿Cuál es la cantidad de usuarios que han calificado más de ocho canciones, entre cuatro y ocho canciones, entre una y tres canciones o ninguna canción?

Para resolver esta pregunta se deben agrupar los datos utilizando como criterio de la creación de los grupos los valores que puede tomar el nuevo escalar creado a partir de la cantidad de calificaciones que ha realizado cada usuario. Es posible conseguir dicho objetivo fácilmente utilizando una expresión `CASE` en la cláusula `GROUP BY`, como se presenta en el *Script* 7.2.

***Script* 7.2**

```
SELECT CASE
            WHEN cantidadCalificaciones > 8
            THEN 'más de ocho canciones'
            WHEN cantidadCalificaciones BETWEEN 4 AND 8
            THEN 'entre cuatro y ocho canciones'
            WHEN cantidadCalificaciones BETWEEN 1 AND 3
            THEN 'entre 1 y tres canciones'
            ELSE 'Ninguna canción'
        END,
        COUNT(idUsuario) AS cantidadUsuarios
FROM(SELECT U.idUsuario,
             COUNT(calificación) cantidadCalificaciones
        FROM Usuarios U
             LEFT JOIN Calificaciones C ON U.idUsuario = C.idUsuario
        GROUP BY U.idUsuario) CalificacionesporUsuario
GROUP BY CASE
              WHEN cantidadCalificaciones > 8
              THEN 'más de ocho canciones'
              WHEN cantidadCalificaciones BETWEEN 4 AND 8
              THEN 'entre cuatro y ocho canciones'
              WHEN cantidadCalificaciones BETWEEN 1 AND 3
              THEN 'entre 1 y tres canciones'
              ELSE 'Ninguna canción'
        END;
```

7.2. LIMIT y OFFSET

Además de filtrar y ordenar las filas como se ha realizado hasta este punto, es común pretender obtener un número específico de filas o paginar los resultados para que empiecen a mostrarse a partir de una fila en específico. Esta funcionalidad, aunque está presente en la mayor parte de los DBMS existentes, puede variar en su implementación de uno a otro. En el caso de PostgreSQL existen las palabras reservadas `LIMIT` y `OFFSET`.

La primera palabra, `LIMIT`, permite utilizar un número para indicar cuántas filas se desea obtener; por ejemplo, 1 significaría que el conjunto de datos resultante debe estar compuesto

por una fila. La segunda palabra, `OFFSET`, señala a partir de cuál fila se quiere presentar los resultados. Es clave señalar que estas dos acciones son las que se ejecutan en último lugar, después del `ORDER BY`, y que el valor por defecto para `LIMIT` es `ALL` y para `OFFSET` es cero. Ahora se responderá la siguiente pregunta:

¿Cuál es el título de las últimas cinco canciones reproducidas?

Para resolver la solicitud, primero se ordenarán de forma descendente las reproducciones realizadas de las canciones utilizando como criterio de ordenamiento la fecha y la hora de reproducción, tal como se muestra en el *Script* 7.3.

***Script* 7.3**

```
SELECT C.título
FROM Reproducciones R
       INNER JOIN Canciones C ON R.idCanción = R.idCanción
ORDER BY R.fechaReproducción,
       R.horaReproducción
```

Luego de tener las canciones ordenadas en función de la fecha y la hora de reproducción, solo resta incluir la palabra reservada `LIMIT` seguida del número cinco para indicar al DMBS que arroje exclusivamente las primeras cinco canciones, tal como se muestra en el *Script* 7.4.

***Script* 7.4**

```
SELECT C.título
FROM Reproducciones R
       INNER JOIN Canciones C ON R.idCanción = R.idCanción
ORDER BY R.fechaReproducción,
         R.horaReproducción
LIMIT 5;
```

Es pertinente señalar que el enfoque anterior tiene el siguiente inconveniente: si las canciones devueltas por la consulta antes de alcanzar el `LIMIT`, en la quinta posición y la sexta, tienen la misma fecha y hora de reproducción, solo una se devolverá. Por lo tanto, en algunos escenarios no se puede asegurar que siempre se obtendrán los mismos resultados (es decir, la consulta adquiere un cierto grado de no determinismo) porque la canción ubicada en la fila seis puede estar, en una próxima ejecución, en la cinco, mientras que la que estaba en la cinco pasa a estar en la seis.

También es posible pretender que la cantidad de filas que se van a devolver no empiecen a contarse a partir de la primera fila, sino de alguna posición en particular. Este caso se puede ejemplificar con la siguiente pregunta:

¿Cuál es el nombre de la primera canción que se reprodujo después de las diez primeras?

La solicitud es similar a la respondida con el *Script* 7.4, con la diferencia de que en esta ocasión se está pidiendo omitir las diez primeras filas y mostrar solo la primera que sigue a esas; puesto en otras palabras: mostrar la fila número once. Para saltar las diez primeras filas, solo es necesario incluir la palabra reservada `OFFSET` seguida del valor diez, como se ve en el *Script* 7.5.

***Script* 7.5**

```
SELECT idListaReproducción,
       COUNT(idCanción) "Cantidad de canciones"
FROM CancionesLista
GROUP BY idListaReproducción
ORDER BY "Cantidad de canciones" DESC
LIMIT 1 OFFSET 10;
```

7.3. Pivoteo de filas con CROSSTAB

La presentación de los datos es un aspecto para tener siempre presente puesto que, finalmente, el propósito de extraerlos es que puedan ser utilizados en tareas de análisis y de toma de decisiones. Bajo esa premisa, en algunas ocasiones es necesario, por ejemplo, convertir los valores de una columna en columnas de manera que se mejore la comprensión de los datos por parte de quien los visualiza. El escenario de la siguiente pregunta permitirá ejemplificar este uso:

¿Cuál es la cantidad de reproducciones que se hizo de cada canción en cada uno de los años?
Mostrar los años como columnas de la tabla resultante.

El primer paso para responder cualquier interrogante es tener la certeza de entender lo que se está pidiendo: tener claro el estado al que se debe llegar para asegurar que se cumplió con la tarea. En este caso lo que se pide es un resultado como el que se presenta en la Tabla 7.1.

Un primer enfoque para llegar a un resultado de esta clase es explotar las potencialidades de la expresión `CASE`. Así pues, se deriva una nueva columna por cada uno de los años, comprobando por cada fila si el año de la fecha de reproducción es igual al año en el cual se

desea contar la reproducción: si es igual, se le pasa a la función COUNT ese identificador de reproducción. La consulta resultante sería la del *Script* 7.6.

Tabla 7.1. Tabla resultante del *Script* 7.6

idCanción	2017	2018	2019	2020	2021
1	0	1	3	5	16
2	1	3	3	18	26
3	0	0	1	6	15

***Script* 7.6**

```
SELECT idCanción,
COUNT(CASE EXTRACT (year FROM fechaReproducción) WHEN 2017 THEN idReproducción END) AS "2017",
COUNT(CASE EXTRACT (year FROM fechaReproducción) WHEN 2018 THEN idReproducción END) AS "2018",
COUNT(CASE EXTRACT (year FROM fechaReproducción) WHEN 2019 THEN idReproducción END) AS "2019",
COUNT(CASE EXTRACT (year FROM fechaReproducción) WHEN 2020 THEN idReproducción END) AS "2020",
COUNT(CASE EXTRACT (year FROM fechaReproducción) WHEN 2021 THEN idReproducción END) AS "2021"
FROM Reproducciones
GROUP BY idCanción
ORDER BY 1
```

Otra forma, quizá un poco más sencilla, es emplear la función CROSSTAB que está incluida dentro de la extensión de PostgreSQL denominada `tablefunc`. Dicha extensión debe ser habilitada tal cual se muestra en el *Script* 7.7 antes de que pueda ser utilizada.

***Script* 7.7**

```
CREATE extension tablefunc;
```

La función CROSSTAB recibe dos parámetros: el primero es la consulta base o central que contiene todos los datos, incluyendo las filas que se van a pivotar. Esta consulta corresponde al *Script 7.8*, cuya ejecución arroja un resultado como el de la Tabla 7.2.

***Script* 7.8**

```
SELECT idCanción, EXTRACT (year FROM fechaReproducción), COUNT(idReproducción) AS
Cantidad
FROM Reproducciones
GROUP BY idCanción, EXTRACT (year FROM fechaReproducción) ORDER BY 1,2
```

En el conjunto resultante se puede ver que los años aparecen en la columna Año y que las cantidades para esos periodos se encuentran en la columna `Cantidad`. Recuerde que lo que se busca es que los valores distintos de Año se conviertan en columna. Precisamente, ese es el argumento que ahora espera la función CROSSTAB: cuáles son los valores que ahora pasarán

a ser columnas; en este caso, como se ha dicho, todos los posibles años, es decir, todos los valores distintos que puede tomar el año de reproducción, tal como se presenta en el *Script 7.9*.

Tabla 7.2. Tabla resultante del *Script* 7.9.

idCanción	Año	Cantidad
1	2018	1
1	2019	3
1	2020	5
1	2021	16
2	2017	1
2	2018	3
2	2019	3
2	2020	18
2	2021	26

***Script* 7.9**

```
SELECT DISTINCT EXTRACT (year FROM fechaReproducción)
FROM Reproducciones ORDER BY 1;
```

Con estos dos parámetros, PostgreSQL tomará la primera consulta y la dejará intacta, mientras que, en la segunda, de los años, los valores distintos definirán las nuevas columnas y los valores de la tercera columna serán asignados en la celda correspondiente; en este caso: la cantidad de reproducciones de una canción en un año específico. Finalmente, es necesario especificar los tipos de datos de cada una de las columnas con las que queda el conjunto de datos resultante, tal cual como se hace en el *Script* 7.10.

***Script* 7.10**

```
SELECT * FROM crosstab(
  -- Consulta centra
  'SELECT idCanción, EXTRACT (year FROM fechaReproducción), COUNT(idReproducción)::NUMERIC AS
  Cantidad
        FROM Reproducciones GROUP BY idCanción, EXTRACT (year FROM fechaReproducción) ORDER BY
        1,2',
  -- Consultas para generar las nuevas columnas
  'SELECT DISTINCT EXTRACT (year FROM fechaReproducción) FROM Reproducciones ORDER BY 1')
  AS ("idCanción" INTEGER,
        "2017" NUMERIC,
        "2018" NUMERIC,
        "2019" NUMERIC,
        "2020" NUMERIC,
        "2021" NUMERIC);
```

Tabla 7.3. Tabla resultante del *Script* 7.10

idCanción	2017	2018	2019	2020	2021
1	0	1	3	5	16
2	1	3	3	18	26
3	0	0	1	6	15
5	0	2	3	7	30
6	0	1	0	6	31
7	1	1	4	10	23
9	0	1	5	8	7
10	0	1	0	6	16
11	0	3	4	3	8
12	0	1	0	10	21
13	1	1	3	8	8
14	0	0	1	9	10
15	2	3	1	8	12
16	3	2	1	1	0
17	0	2	1	5	6
18	0	0	1	2	3
19	0	0	0	5	8
20	1	2	2	5	8

Los dos enfoques anteriores permiten obtener el resultado deseado, tal como se presenta en la Tabla 7.3. Así podemos ver fácilmente que durante el 2017 la canción con identificador «*5*» tuvo cero reproducciones.

7.4. Aprendizajes más importantes del capítulo 7

- La expresión `CASE` permite incluir lógica condicional en la determinación de los valores escalares. Así se abre un gran abanico de posibilidades puesto que es posible incluir dicha expresión en cualquier lugar donde una consulta acepte un escalar.
- Con la palabra reservada `LIMIT` se puede especificar cuál es la cantidad precisa de filas que se desea mostrar de una consulta.
- El operador `OFFSET` permite especificar a partir de qué posición se quiere empezar a mostrar los resultados.
- Se pueden reorganizar las filas y las columnas de una tabla pivoteándolas mediante la expresión `CASE` o la función `CROSSTAB`.

7.5. Actividades de aplicación para evidenciar lo aprendido

1. Escriba una consulta que permita obtener la cantidad de canciones reproducidas por cada mes del año, de manera que cada nombre del mes se muestre en una columna, es decir, una columna para enero, otra para febrero, y así sucesivamente.
2. Identifique los errores lógicos o de sintaxis en la siguiente consulta. Explique la incidencia que tiene cada uno de estos elementos en la ejecución:

```
SELECT CASE
            WHEN cantidadCalificaciones > 8
            THEN 'más de ocho canciones'
            WHEN cantidadCalificaciones BETWEEN 4 AND 8
            THEN 'entre cuatro y ocho canciones'
            WHEN cantidadCalificaciones BETWEEN 1 AND 3
            THEN 'entre 1 y tres canciones'
            ELSE 'Ninguna canción'
       END, paísNacimiento
       COUNT(idUsuario) AS cantidadUsuarios
FROM(SELECT U.idUsuario,
            COUNT(calificación) cantidadCalificaciones
       FROM Usuarios U
            CROSS JOIN Calificaciones C ON U.idUsuario = C.idUsuario
       GROUP BY U.idUsuario) CalificacionesporUsuario
GROUP BY CASE
             WHEN cantidadCalificaciones > 8
             THEN 'más de ocho canciones'
             WHEN cantidadCalificaciones IN 4 AND 8
             THEN 'entre cuatro y ocho canciones'
             WHEN cantidadCalificaciones BETWEEN 1 OR 3
             THEN 'entre 1 y tres canciones'
             ELSE 'Ninguna canción'
       END;
```

3. Escriba una consulta que muestre la cantidad de usuarios que han reproducido las canciones por cada uno de los intérpretes. De tal manera, la idea es obtener la cantidad de usuarios por cada una de las siguientes categorías: intérpretes con menos de dos usuarios, intérprete con entre tres y cinco usuarios, intérpretes con entre seis y ocho usuarios e intérpretes con más de nueve usuarios.

Capítulo 8

Diseño conceptual

Resultados de aprendizaje

- *Analiza las necesidades de almacenamiento de datos, identificadas en un contexto específico, que van a ser satisfechas con una base de datos relacional.*
- *Modela las entidades de datos que son relevantes y necesarias para resolver las necesidades de almacenamiento de datos identificadas.*
- *Modela las relaciones entre las entidades de datos identificadas.*

Las bases de datos relacionales están presentes en gran variedad de escenarios. Su uso se ha incrementado en la última década debido, entre otros factores, al abaratamiento de los dispositivos de almacenamiento y procesamiento de datos, la evolución y madurez de los DBMS y la aparición de herramientas que facilitan la construcción y administración de bases de datos desde entornos gráficos.

Sin embargo, crear una base de datos no es una tarea instrumental que se resuma en el uso de alguna herramienta para realizar una implementación rápida. Por el contrario, esta actividad demanda identificar y analizar las necesidades de almacenamiento de datos, especificar las características que debe exhibir la solución por implementar y determinar los elementos que conformarán la solución en un diseño que articule todos los componentes.

8.1. Análisis de necesidades

Un ingeniero automotriz no empieza la construcción de un nuevo vehículo ensamblando el motor. Primero analiza las necesidades que deben satisfacerse con ese nuevo vehículo e identifica las características que requiere; por ejemplo, el precio, la potencia o la comodidad. Esta

es la base para que el ingeniero inicie el proceso de diseño y modele las diferentes partes que conforman el vehículo para cumplir con las características requeridas.

El proceso de diseño en ingeniería generalmente se aborda de manera iterativa: se empiezan a probar diferentes diseños hasta obtener versiones que permitan realizar pruebas simuladas. Finalmente, se crean unos primeros vehículos y, si todo va bien, comienza la producción masiva.

Diseñar una base de datos tiene cierta similitud con este enfoque. Aunque con la base de datos no se busca que la solución creada se replique muchas veces, frecuentemente sí se busca que esta sea usada por muchos usuarios de manera simultánea. De hecho, empezar directamente por la implementación de la solución, salvo en escenarios extremadamente sencillos y poco usuales en el ámbito profesional, trae consecuencias negativas. Estas soluciones pueden generar problemas asociados a la redundancia de datos e ineficiencia en el almacenamiento y la recuperación de datos, lo que se traduce muchas veces en sistemas de software con demoras excesivas y datos incorrectos, desde el punto de vista técnico, y en pérdida de clientes y dinero, si se extrapola a un ámbito empresarial.

El hecho de que un aficionado que crea una base de datos para guardar los libros que está leyendo o para gestionar sus gastos mensuales no siga un proceso riguroso de análisis, diseño e implementación es normal y aceptable. Sin embargo, que no lo haga un profesional de la informática es, además de poco presentable, peligroso.

8.2. Niveles de diseño

En las bases de datos, como en otros elementos de software, se suele hablar de tres niveles de diseño: el conceptual, el lógico y el físico. El primero se centra en las entidades, las relaciones que se presentan entre ellas y los atributos que las entidades y las relaciones poseen. El segundo se enfoca en el modelo de datos, y el tercero se ocupa de los detalles de implementación en un DBMS. Estos tres niveles no deben percibirse aislados unos de otros y tampoco conviene abordarlos con rigidez secuencial. Como muestra la Figura 8.1, entre cada uno de ellos hay interacción. Este capítulo se centrará en el diseño conceptual.

Figura 8.1. Niveles del diseño

Para iniciar con el modelado conceptual de una base de datos, es condición ineludible identificar cuáles son los hechos que se está intentando modelar. Ahora bien, antes de llegar

a este punto se deben realizar previamente diferentes actividades como entrevistas, observación directa de la organización, revisión de documentos (formularios, hojas de cálculo, etc.), análisis de procesos, entre otras.

Puesto en otras palabras, es necesario especificar los requisitos que se deben satisfacer con la creación de la base de datos. Por ejemplo, conviene tener claro cuáles son esas preguntas que deben poderse responder con la base de datos relacional. Por consiguiente, es clave revisar que los requisitos sean consistentes, claros, medibles y alcanzables.

Para ejemplificar el modelado conceptual se seguirá utilizando el caso de la plataforma de canciones, pero en lugar de presentar las diferentes tablas empleadas hasta el capítulo anterior se indicará ahora cuáles fueron las decisiones de diseño que fueron llevando a su forma definitiva. Recuerde que, dentro del dominio musical, se ha trabajado en la base de datos de una plataforma denominada «*Mis Canciones*» que ofrece colecciones de canciones para que el público las reproduzca gratuitamente o pagando una tarifa de suscripción mensual. Los diferentes detalles de esta plataforma serán introducidos de manera incremental con el fin de reflejar cómo, de un modo iterativo, las bases de datos pueden ir evolucionando a través de la incorporación de nuevos requisitos, la corrección de errores o la introducción de mejoras, tres acciones muy comunes en el ámbito profesional.

El modelado conceptual de la plataforma «*Mis Canciones*» puede partir de la siguiente descripción: los datos esenciales para administrar la colección de la plataforma son los correspondientes a las canciones y a quienes las interpretan, es decir, los artistas. Hay que tener presente que estos últimos pueden ser solistas o grupos. También es importante considerar que una canción puede hacer parte de un álbum en el que se publican un número específico de canciones o puede ser un sencillo, es decir, una canción publicada o lanzada de forma individual.

8.3. Entidades y atributos

Para diseñar una base de datos relacional se empieza por identificar cuáles son esas entidades o grupo de objetos que hacen parte del dominio que se quiere modelar. Según la descripción anterior, las entidades a las que se está haciendo alusión son «Canción» y «Artista». Puesto en otras palabras, en la base de datos que se va a modelar existirá un conjunto de artistas y otro de canciones, y cada una de estas entidades tiene unos atributos que son importantes para el contexto particular que se está trabajando. En primer lugar, se analizará el caso de la entidad «Artista».

En la plataforma «*Mis Canciones*» se desea almacenar datos de cada uno de los artistas, es decir, saber su nombre, la cantidad de años de vida artística, si es un grupo o un solista y el género principal que interpreta.

En este caso se puede identificar que «Artista» es una entidad que posee atributos como el nombre y la cantidad de años de vida artística. Al respecto, cabe mencionar que un método para determinar las entidades es poner atención en los sustantivos (artistas) o en las frases nominales que se mencionan (por ejemplo, nombre del artista, cantidad de años de vida artística de cada artista, etc.). En cuanto a los atributos, es posible detectarlos cuando el sustantivo o el sintagma nominal es una propiedad, cualidad, identificador o característica de una de estas entidades. Así pues, en «el nombre del artista», «nombre» es un atributo de la entidad «Artista». Los atributos suelen responder a la pregunta: ¿qué datos se quiere mantener de una entidad? En este caso, ¿qué datos se quiere mantener de la entidad «Artista»?

En la Figura 8.2 se observa un ejemplo de un artista con valores para cada uno de los atributos identificador, nombre, años de vida artística y género principal que interpreta. La entidad «Artista» sirve de molde para representar diferentes instancias particulares de un músico. Sin embargo, es importante señalar que los atributos seleccionados para representar la abstracción de una entidad dependen de los datos que tengan sentido para el dominio que se está trabajando. Por ejemplo, para el caso de la plataforma «*Mis Canciones*» son irrelevantes los datos relacionados con la apariencia física del artista.

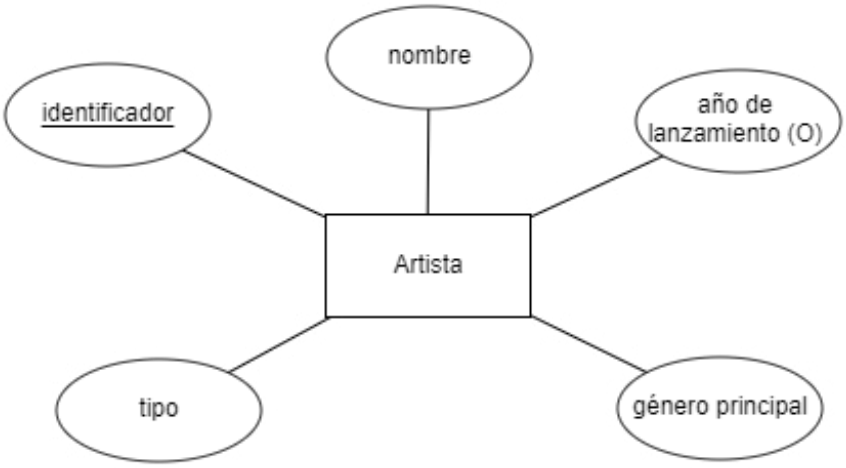
Una vez identificada la entidad que se desea modelar y los atributos que esta posee, se puede utilizar un diagrama que permita representar de manera visual el modelo que se está empezando a elaborar. Para realizar el modelado conceptual de una base de datos han surgido diferentes notaciones, y en este libro se seguirá la propuesta por Peter Chen, conocida como notación de Chen. A este autor se le atribuye el desarrollo del modelo Entidad Relación (ER) para el diseño de bases de datos en los 70, mientras trabajaba como profesor adjunto en la Escuela de Administración y Dirección de Empresas del Instituto Tecnológico de Massachusetts (MIT). El desarrollo de esta idea se publicó en el año 1976 bajo el título *Modelo entidad-relación: hacia una visión unificada de los datos*.

En la Figura 8.3 se observa cómo sería la representación de la entidad «Artista» con sus atributos utilizando la notación de Chen. La entidad cuenta con cinco atributos, que son el identificador, el nombre, el año de lanzamiento, el género y el tipo.

Figura 8.2. Abstracción de la entidad «Artista»

Un Artista
Identificador: 50001
Nombre: Carlos Vives
Años: 34
Tipo: Solista
Género: Vallenato

Figura 8.3. Atributos de la entidad «Artista»



Las entidades agrupan una colección de objetos. Es decir, la entidad «Artista» consiste en una abstracción que contiene los atributos que se consideran relevantes para representar a un músico dentro de la plataforma *«Mis Canciones»*. Por lo tanto, «Artista» constituye un molde que permitirá identificar artistas concretos cuando los atributos tomen un valor específico. De tal modo, por ejemplo, indicará que un artista tiene por identificador *«50001»*, su nombre es *«Carlos Vives»*, su año de lanzamiento es *«1986»*, corresponde al tipo *«solista»* y su género principal asignado es *«Vallenato»*. Así se puede utilizar este «molde» para representar tantos músicos como se desee.

En el diagrama, uno de los atributos (el identificador) está subrayado. Esto obedece al hecho de que una entidad debe garantizar que hay al menos una forma en que cada uno de los objetos que representa pueda ser identificado de manera única. El nombre no funciona para este fin porque existe la posibilidad de que haya más de un artista con el mismo nombre. Asimismo, varios músicos podrían tener el mismo año de lanzamiento, e igualmente con los demás atributos. Por lo tanto, que el atributo «identificador» esté subrayado significa que por cada artista toma un valor distinto.

El atributo o grupo de atributos seleccionados para identificar de forma indistinta a cada objeto reciben el nombre de llave principal de la entidad. Una entidad puede tener más de un atributo que permita identificarla de forma indistinta; todos los atributos que satisfacen esta condición se llaman llaves candidatas. Suponga, por ejemplo, que de los artistas también se quiere almacenar el número de pasaporte; entonces el identificador y el número del pasaporte serían dos llaves candidatas de la entidad «Artista». Cuando una de estas dos llaves candidatas se selecciona como llave principal, lo otra se convierte automáticamente en llave alternativa.

Adicionalmente, en la Figura 8.3 también se puede observar que uno de los atributos se representa incluyendo una letra «O» entre paréntesis luego del nombre. De tal forma se indica que este atributo es *opcional* o, lo que es lo mismo, en la plataforma «*Mis Canciones*» se pretende tener almacenados a los artistas, aunque se desconozca el año de lanzamiento.

También conviene mencionar que con el atributo «año de lanzamiento» sucede algo particular. Si se observa de nuevo el requisito, es evidente que lo que realmente se quiere saber es la cantidad de años de vida artística que tiene el músico, a pesar de que lo que se está almacenando es la fecha de lanzamiento. Esto se hace debido a que los años de vida artística son un valor calculado y cambiante, es decir, hoy «Carlos Vives» tendrá 34 años de vida artística, pero el próximo año serán 35.

A razón de lo anterior, es recomendable, si es posible, utilizar atributos a partir de los cuales se pueda calcular lo requerido. En este caso, con base en el «año de lanzamiento» es posible estimar los años de vida artística. Estos atributos cuyos valores se basan en los de otros atributos se conocen como atributos derivados. Otro ejemplo de un atributo derivado son la edad y el número de canciones. A menudo, estos atributos no se representan en el modelo de datos conceptual. Sin embargo, a veces el valor del atributo o atributos sobre los que se derivan se pueden eliminar o modificar. En dicho escenario, el atributo derivado debe mostrarse en el modelo de datos para evitar una posible pérdida de información.

8.4. Relaciones entre entidades

En la plataforma «*Mis Canciones*» también se quiere almacenar canciones y, de cada canción, el título, el artista que la interpreta, la duración y el álbum al que pertenece.

Una buena estrategia al modelar en un diagrama entidad-relación los requisitos planteados es traducirlos en restricciones o reglas semánticas. Por ejemplo, lo anterior se puede expresar así:

- Las canciones deben tener un identificador, un título, el nombre del álbum al que pertenecen, la duración, el género en el que están clasificadas y el artista que las interpreta.
- Un artista puede interpretar más de una canción.
- Una canción tendrá un artista principal que la interprete. Aunque una canción también puede ser interpretada por más de un artista, en este contexto solo se considerará el intérprete principal, es decir, cada canción tendrá solo un intérprete principal.

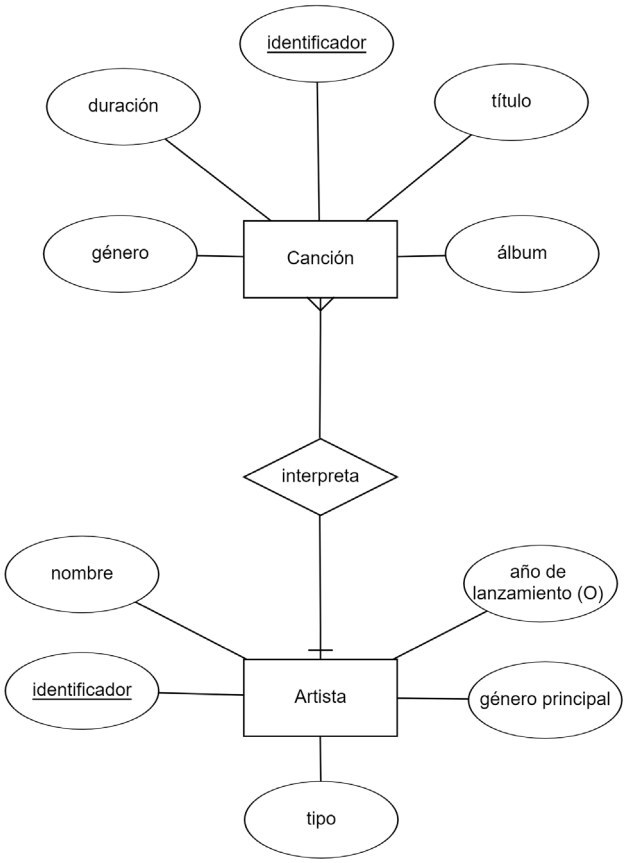
La primera restricción semántica solo indica que existirá una entidad de nombre «Canción» y unos atributos asociados a ella, tal como se ve en la parte derecha de la Figura 8.4. Las otras dos restricciones planteadas sugieren que se desea saber quién es el artista que interpreta una canción y cuáles canciones son las interpretadas por un artista. Una opción

en este sentido sería añadir un atributo del nombre del artista a la entidad «Canción», pero ya se cuenta con una entidad «Artista» que tiene sus propios atributos. Por lo tanto, lo que se puede hacer es establecer una relación entre estas dos entidades, denominada «interpreta». De esta manera, se reconoce que un artista *interpreta* una o muchas canciones. En la Tabla 8.1 se observa un resumen de lo que se desea modelar.

Tabla 8.1. Resumen de la relación entre las entidades «Artista» y «Canción»

	Entidad	**Artista**
	Atributos	identificador, nombre, año de lanzamiento, tipo, género principal
	Clave	identificador
	Entidad	Canción
	Atributos	identificador, título, álbum, duración y género
	Clave	identificador
Relación		Existe una relación entre los artistas y las canciones que interpretan

Figura 8.4. Diagrama ER de relación «interpreta»



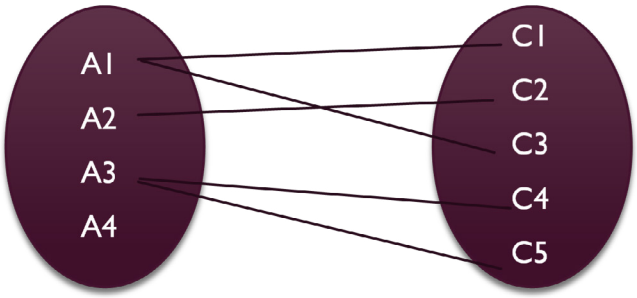
Otro aspecto que es importante acotar ahora que es claro que entre ambas entidades se puede plantear una relación es cuántos objetos de cada entidad pueden intervenir en dicho vínculo. Esto se conoce como cardinalidad e impone restricciones estructurales en el diseño.

Como se planteó en las dos últimas restricciones semánticas, un artista puede interpretar muchas canciones, mientras que una canción es interpretada en el rol de artista principal por un solo artista. Este tipo de relación se conoce como «de uno a muchos» y se representa como se establece en la Figura 8.4. Un ejemplo de una relación de este tipo en otro dominio sería una madre que puede tener cero o varios hijos, mientras que un hijo tiene una sola madre.

Para comprender cómo operan las relaciones de uno a muchos conviene analizar la Figura 8.5, donde se presenta un diagrama sagital que representa dos conjuntos: el de la izquierda corresponde a los artistas y tiene cuatro elementos —el artista «A1», el artista «A2», el artista «A3» y el artista «A4»—, y el de la derecha, asociado a las canciones, tiene cinco elementos.

En el mismo diagrama se evidencia que a un elemento del conjunto artistas le pueden corresponder cero o varias canciones, mientras que una canción solo puede estar enlazada a un artista. En la mayoría de los casos, las relaciones son binarias; en otras palabras, las relaciones existen entre exactamente dos entidades. Sin embargo, no siempre tiene que ser así: es posible involucrar más de dos entidades y también puede haber relaciones recursivas que involucren solo una entidad.

Figura 8.5. Relación uno a muchos



Como las entidades interactúan o se asocian entre sí, las relaciones pueden pensarse en términos de verbos. Por ejemplo, en el enunciado «Un artista *interpreta* canciones» el verbo «interpreta» indica cuál es la relación entre las entidades «Artista» y «Canción». En otro dominio, en la oración «Un estudiante puede *inscribirse* en diferentes asignaturas» las dos entidades serían el estudiante y el curso, y la relación representada es el acto de inscribirse, que conecta ambas entidades de ese modo. Como se puede ver en la Figura 8.4, estas relaciones se muestran, por lo general, como rombos.

Para cerrar con esta representación de la entidad «Canción», se observa que la llave principal es el identificador. Las llaves pueden ser naturales, es decir, pueden provenir de dominio que se desea modelar. Por ejemplo, el código estudiantil es útil para representar la llave principal de una entidad «Estudiante». Sin embargo, también pueden ser llaves artificiales

o subrogadas, es decir, añadidas a la entidad para facilitar y garantizar la unicidad de cada instancia de ella, aunque no existan en la «realidad» que se está modelando.

No siempre es obvio si un concepto en particular es una entidad, una relación o un atributo. De hecho, dependiendo de los requisitos reales, un mismo concepto podría clasificarse como cualquiera o todos estos. El diseño es en cierto nivel subjetivo y depende de cómo se haya pedido o querido modelar la «realidad». Esto se conoce como universo de discurso, es decir, la visión que tiene el diseñador de la realidad. Diferentes diseñadores pueden producir diferentes interpretaciones, pero igualmente válidas. Analice el siguiente escenario.

En la plataforma «*Mis Canciones*» también se quiere saber, para cada álbum, además del título, el sello discográfico que lo produjo y la fecha de lanzamiento. De igual forma, de los géneros también se requiere conocer el nombre y una breve descripción de cada uno. Adicionalmente, sería útil saber el año en que se retiró cada artista.

Según la descripción anterior, hay un cambio sustancial con respecto al manejo que se les había venido dando a los conceptos de álbum y género, los cuales se habían trabajado como atributos de las entidades porque era lo que tenía sentido para el diseño. No obstante, ahora se presentan unas exigencias adicionales que hacen que dicho diseño tenga que evolucionar: ahora se indica que cada uno de estos conceptos tiene sus propios atributos, es decir, esta vez son una entidad. En el caso de «Álbum», una entidad que se relaciona con «Canción», y en el caso de «Género», una entidad que se relaciona tanto con «Artista» como con «Canción». Las restricciones semánticas impuestas por estos requisitos son:

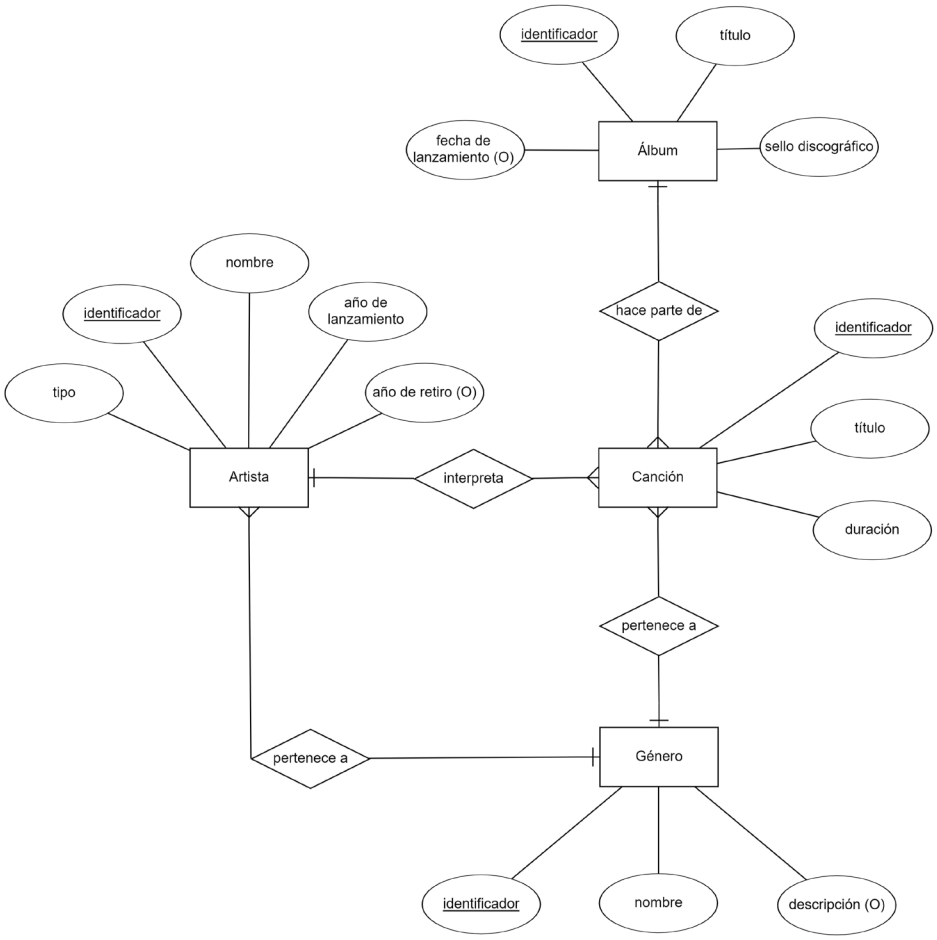
- Los álbumes tienen como atributos un identificador, un título, el año de lanzamiento y el sello discográfico.
- Cada álbum está compuesto por varias canciones, mientras que una canción puede pertenecer originalmente a un solo álbum.
- Los géneros tendrán un identificador, un nombre y una descripción. La descripción es un atributo opcional.
- Las canciones se clasifican dentro de un solo un género, mientras que un género puede agrupar más de una canción.
- Los artistas tienen un género principal que interpretan, pero un mismo género puede tener más de un artista que lo interpreta.

Todas estas relaciones descritas entre las entidades son de uno a muchos y se resumen en la Tabla 8.2. Por otra parte, en la Figura 8.6 se presenta el diagrama ER que satisface los nuevos requisitos planteados.

Tabla 8.2. Análisis de entidades y relaciones de uno a muchos

Relación	Entidades participantes	Cardinalidad
Hace parte de	Canción – Álbum	1:N
Pertenece	Género – Canción	1:N
Pertenece	Género – Artista	1:N

Figura 8.6. Diagrama ER de las entidades de la Tabla 8.2



También es clave mencionar que se han introducido cambios con respecto a algunos atributos. Por ejemplo, en la entidad «Artista» el año de lanzamiento dejó de ser opcional y se añadió el «año de retiro» del músico, el cual es opcional.

Finalmente, cabe destacar que las entidades pueden ser tangibles —o, lo que es lo mismo, tienen una correspondiente representación material en el mundo natural— o abstractas o conceptuales —es decir, son ideas definidas por el hombre—. En la Tabla 8.3 se aprecia un ejemplo de cada grupo.

Tabla 8.3. Ejemplificación de entidades concretas y abstractas

Entidades concretas	Entidades abstractas
Artista	Género Canción Álbum

No todas las relaciones que existen entre las entidades tienen que ser de uno a muchos como las que se han propuesto hasta este punto. Observe la siguiente descripción.

«*Mis Canciones*» también debe almacenar datos de los usuarios (nombre, *e-mail* y contraseña). Además, se busca saber las canciones que reproducen, en qué momento lo hacen y por cuánto tiempo.

Como se ha hecho con las descripciones anteriores, el primer paso es convertir la solicitud en restricciones semánticas que se deben modelar:

- Existen usuarios de los que se debe tener un identificador, correo electrónico, contraseña, nombre (primer nombre y segundo nombre), sexo, fecha de nacimiento, país de nacimiento, país de residencia, idioma y fecha de registro.
- Un usuario puede reproducir muchas canciones, inclusive varias veces la misma canción. De igual manera, una misma canción puede ser reproducida por muchos usuarios.

Esta última restricción plantea un escenario diferente al que se había modelado hasta este punto con las relaciones de uno a muchos. Ahora se está afirmando que un objeto de la entidad «Canción» puede ser reproducido por muchos objetos de la entidad «Usuario» y, a su vez, un objeto de esta última puede reproducir muchos objetos de la primera. Este tipo de relación se conoce como «muchos a muchos».

Con la Figura 8.7 se ilustra lo que sucede en una relación de muchos a muchos. En este caso cada uno de los ejemplos de un conjunto —por ejemplo, usuarios— puede relacionarse con uno o muchos elementos de otro conjunto —canciones—. Lo inverso también se cumple: un elemento del conjunto que contiene a las canciones puede estar relacionado con uno o muchos elementos del conjunto de usuarios. En la Figura 8.8, por otra parte, se aprecia cómo se modelaría utilizando el diagrama ER.

En el diagrama también se evidencia que el atributo «nombre» está compuesto por otros dos atributos simples, que son el «primer nombre» y el «segundo nombre». En un dominio diferente, otro ejemplo de atributo compuesto suele ser la dirección cuando se decide modelarla como una composición de atributos como calle, ciudad y código postal. La opción de representar los detalles de la dirección como un atributo simple o compuesto está determinada por el requisito del cliente o la conveniencia durante el diseño.

Figura 8.7. Ejemplificación de relación de muchos a muchos

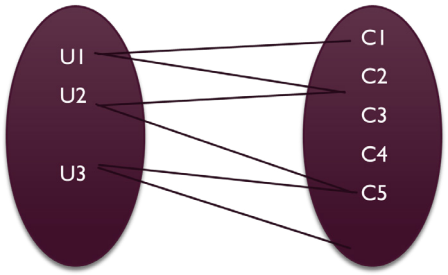


Figura 8.8. Diagrama ER de entidades «Canción» y «Usuario»

Además de ser simple o compuesto, un atributo también puede contener un solo valor o varios valores, como el caso del correo electrónico (*e-mail* en la Figura 8.8). Al decir que es un atributo multivaluado se indica que el usuario puede tener más de un correo electrónico, es decir, el atributo puede contener varios valores para una sola ocurrencia de entidad

(un usuario en este caso). Otro ejemplo típico suele ser el número de teléfono. También conviene considerar que una alternativa pudo haber sido representar los correos como una entidad separada de «Usuario». No obstante, como se verá en el capítulo 9, ambas formas producen el mismo resultado al ser traducidas al modelo relacional.

Para finalizar, se centrará la atención en la relación «reproduce» ya que hasta este punto ninguna de las relaciones había tenido atributos. En este caso se le han asignado tres: «segundos reproducidos», «hora de reproducción» y «fecha de reproducción». Una pregunta natural frente a esto es: ¿por qué se ha hecho de tal forma?

La respuesta es fácil de entender: los tres atributos mencionados solo tienen sentido cuando un usuario hace una reproducción; no para el usuario *per se* o la canción de manera aislada, sino solo cuando se configura una reproducción. Puesto en otras palabras, la hora de reproducción, por ejemplo, tiene sentido cuando un usuario U1 reproduce una canción C2. A continuación, se abordará otro ejemplo.

En «*Mis Canciones*» también se quiere que los usuarios califiquen las canciones. De tal forma, se busca saber la fecha y hora de la calificación, el puntaje otorgado y el comentario que tenga el usuario con respecto a la canción.

La calificación, al igual que la reproducción, es un evento que se produce cuando el usuario emite una valoración sobre una canción, por lo cual surge de la relación entre esas dos entidades. Al igual que para la reproducción, esta relación también tiene una cardinalidad de muchos a muchos (N:M) puesto que una canción puede ser calificada por cero o muchos usuarios y un usuario puede calificar muchas o ninguna canción.

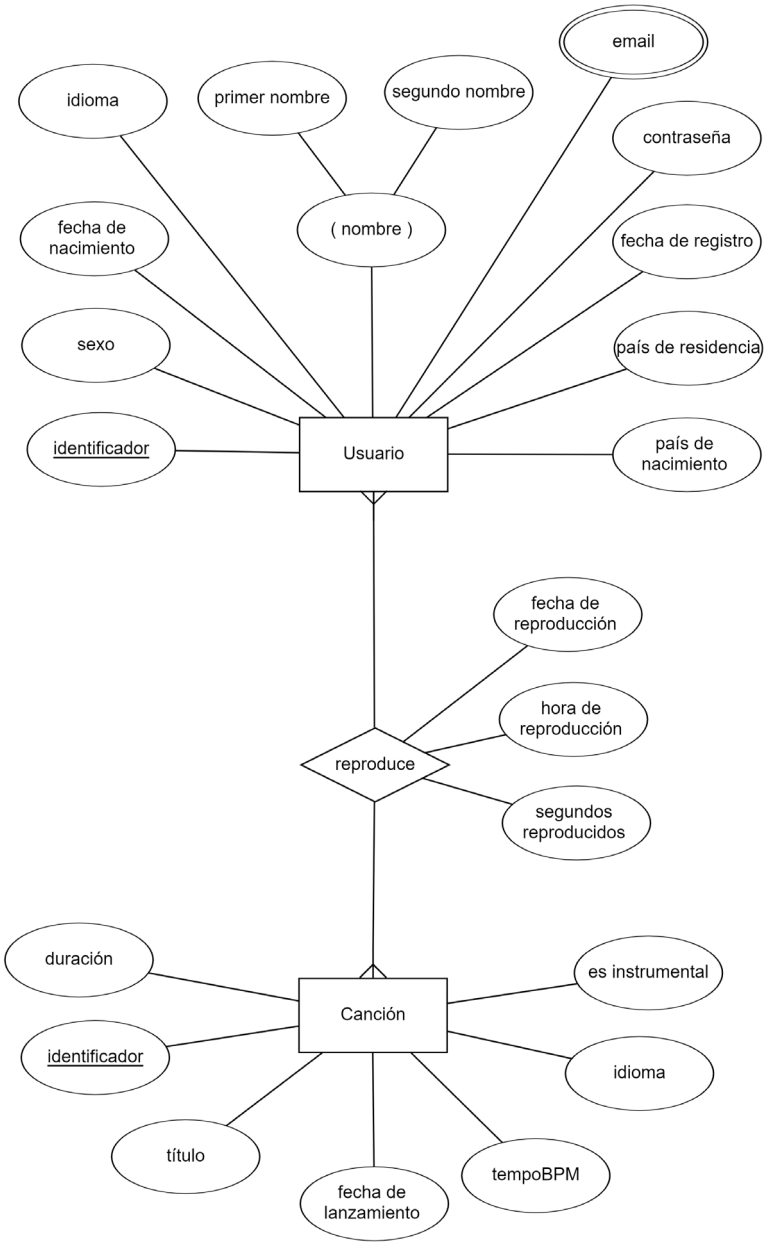
En cuanto a los atributos «comentarios», «calificación», «fecha de reproducción» y «hora de reproducción», puesto que solo tienen sentido cuando un usuario califica una canción en un instante específico, son asignados a la relación. En la Figura 8.9 se observa cómo este nuevo requisito es modelado. También se muestra que el atributo «comentario» es opcional: cuando se realiza una calificación de una canción, se puede dejar un comentario o no.

En «*Mis Canciones*» los usuarios tienen la posibilidad de guardar sus listas de reproducción, las cuales pueden tener un nombre y una fecha de creación.

Para modelar el requisito anterior, se considerarán las siguientes restricciones semánticas:

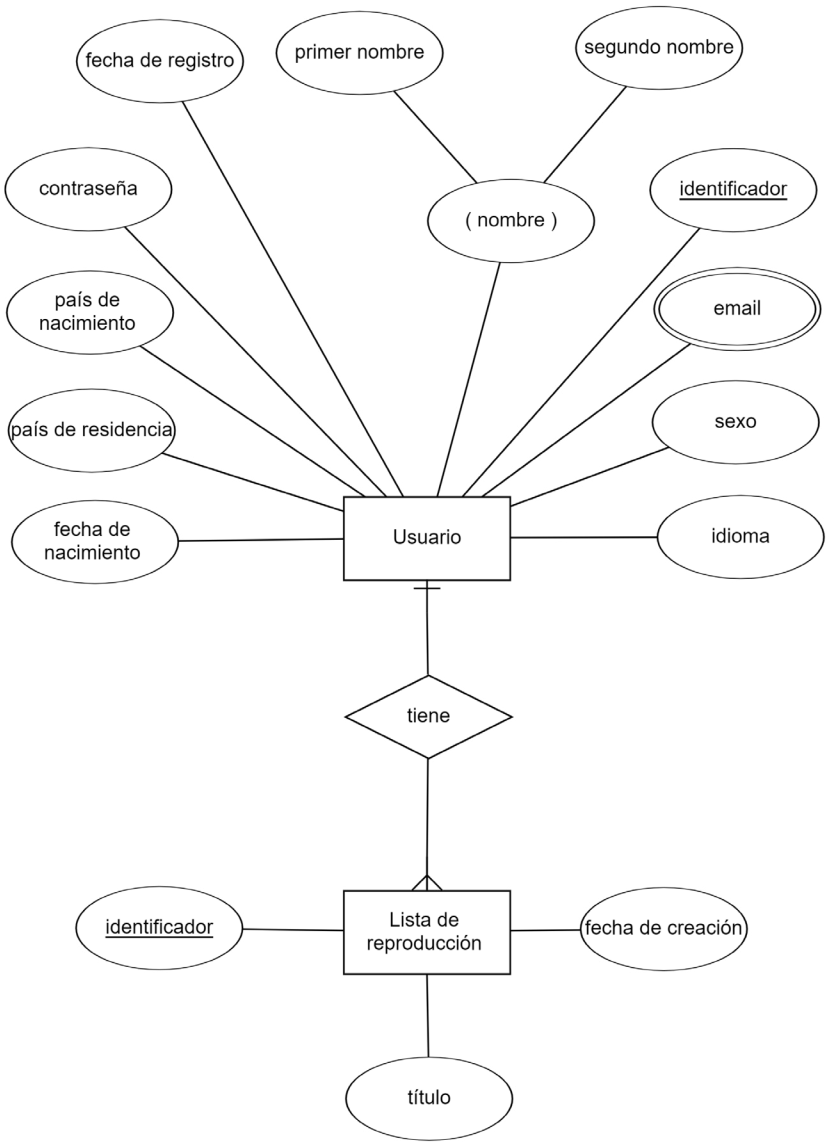
- Una lista de reproducción va a tener un identificador, un título y una fecha de creación.
- Un usuario puede tener cero o varias listas de reproducción, mientras que la lista de reproducción debe pertenecer exclusivamente a un usuario.

Figura 8.9. Relación entre las entidades «Canción» y «Usuario»



De las restricciones anteriores, se puede suponer que la lista de reproducción es una entidad con tres atributos: el identificador, que sirve de llave principal, el título y la fecha de creación. Por otra parte, entre las entidades «Lista de reproducción» y «Usuario» se establece una relación de uno a muchos porque un usuario puede tener cero o más listas de reproducción, pero una la lista de reproducción específica debe pertenecer exclusivamente a un usuario. Lo descrito se presenta gráficamente en el diagrama de la Figura 8.10.

Figura 8.10. Relación entre las entidades «Usuario» y «Lista de reproducción»



Para finalizar con esta primera versión del modelo, conviene abordar un requisito más, a continuación.

El usuario podrá añadir a las listas de reproducción las canciones que sean de su preferencia, indicando en cada caso el lugar de esa canción en la lista y la fecha y hora particular en que se realizó la inclusión.

Para modelar este requisito se debe tener claro que la relación que existe entre las entidades «Lista de reproducción» y «Canción» es que una canción puede pertenecer a cero o más listas de reproducción y que una lista de reproducción puede tener cero o más canciones. Por lo tanto, se configura una relación de muchos a muchos.

Teniendo claro lo anterior, también resulta evidente que la fecha y hora de inclusión y el lugar que ocupa una canción en la lista solo tienen relevancia y se configuran cuando se da la relación. Por este motivo los tres atributos son añadidos a la relación y no a ninguna de las entidades. En la Figura 8.11 se aprecia lo descrito.

Figura 8.11. Diagrama ER de las entidades «Usuario», «Lista de reproducción» y «Canción»

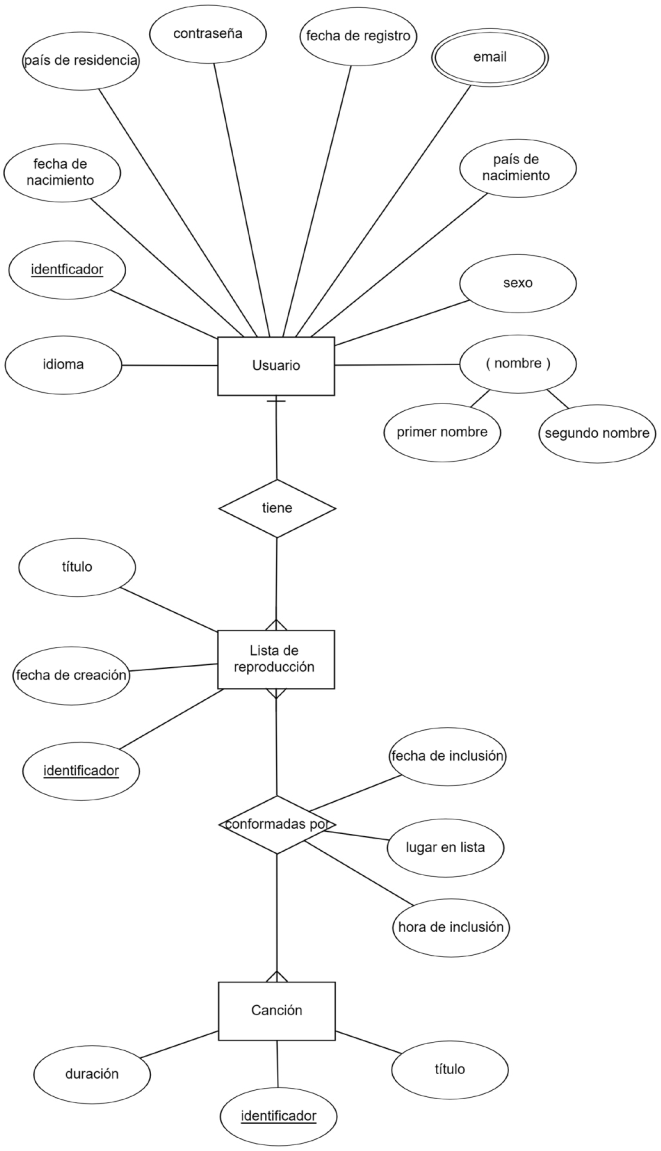
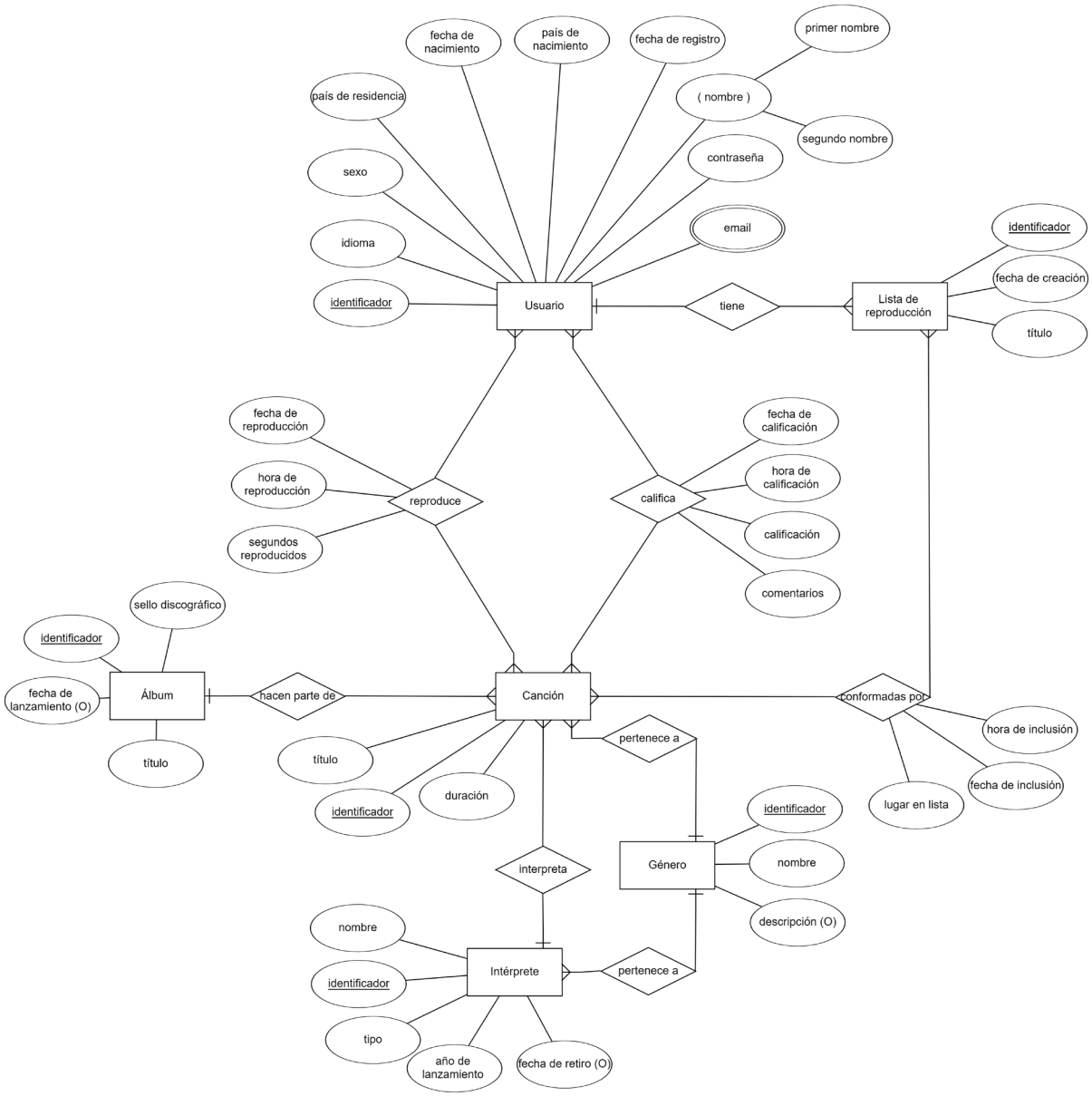


Figura 8.12. Diagrama ER con todas la entidades presentadas



La versión completa de esta primera parte del diagrama se presenta en la Figura 8.12. También, en la Tabla 8.4 se muestran todas las entidades añadidas al modelo junto con sus atributos, y en la Tabla 8.5, las relaciones entre las diferentes entidades con su respectiva cardinalidad. Es importante aclarar que, a partir de este momento, la entidad «Artista» ha sido renombrada como «Intérprete» debido a que se ajusta más al dominio conceptual, es decir, da cuenta de quién interpreta la canción.

Tabla 8.4. Consolidado de las entidades de la Figura 8.11

Entidades	Intérprete	Canción	Género	Lista de reproducción	Álbum	Usuario
Atributos	identificador	identificador	identificador	identificador	identificador	identificador
	nombre	título	nombre	título	título	email
	tipo	duración	descripción	fecha de creación	sello discográfico	contraseña
	año de lanzamiento	tempo (en *bpm)*			fecha de lanzamiento	nombre
	año de retiro	idioma				sexo
	tipo	fecha de lanzamiento				idioma
						país de nacimiento
						país de residencia
						fecha de nacimiento
						fecha de registro

Tabla 8.5. Consolidado de las relaciones entre las diferentes entidades

Relación	Entidades participantes	Cardinalidad
interpreta	Intérprete – Canción	1:N
hace parte de	Canción – Álbum	1:N
pertenece	Género – Canción	1:N
pertenece	Género – Intérprete	1:N
reproduce	Usuario – Canción	M:N
califica	Usuario – Canción	M:N
tiene	Usuario – Lista de reproducción	1:N
conformada por	Lista de reproducción – Canción	M:N

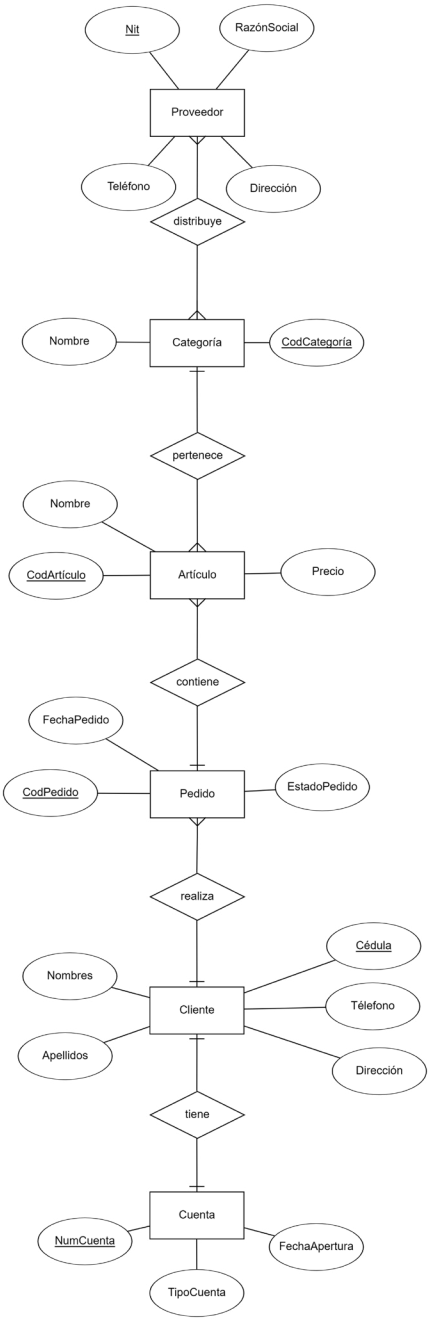
8.5. Aprendizajes más importantes del capítulo 8

- El diagrama ER permite obtener una representación visual de una base de datos en un nivel conceptual.
- Los elementos de un diagrama ER son tres: las entidades, las relaciones y los atributos de cada uno de estos.
- Las entidades serán cualquier objeto sobre el cual se están recogiendo datos; por ejemplo: persona, cosa, proceso, hecho, etc. Estas pueden ser tangibles o intangibles, como en el caso de un proveedor (tangible) o de una venta (intangible).
- Las relaciones serán las acciones que se realicen o las asociaciones que existan entre las entidades; por ejemplo: vende, tiene, representa, gobierna, etc. Los nombres de las relaciones no tienen que ser las palabras exactas que se encuentren en el texto de la restricción, pero sí deben ser similares o indicar el mismo tipo de enlace; por ejemplo: gobernar → gobierna o dirige.
- Los atributos serán las características que describen a la entidad o la relación; por ejemplo: códigos, nombres, fechas, direcciones y estados.

- Las relaciones tienen una cardinalidad que ayuda a saber la cantidad de objetos que pueden intervenir en estas.

8.6. Actividades de aplicación para evidenciar lo aprendido

1. Describa las restricciones semánticas que implementa el siguiente diagrama ER:



2. ¿Cómo mejoraría la representación propuesta en la actividad anterior? Justifique cada una de las mejoras planteadas e impleméntelas en un nuevo diagrama.
3. Teniendo en cuenta las siguientes restricciones semánticas, cree el modelo ER correspondiente:

    - Un centro comercial puede tener varias sedes, y cada sede pertenece a un solo centro comercial.
    - Los centros comerciales tienen un código que los identifica y un nombre.
    - Las sedes tienen un identificador, un nombre, un nombre de supervisor y el área total en metros cuadrados.
    - Cada sede está ubicada en una única ciudad, pero cada ciudad puede tener varias sedes.
    - Las ciudades tienen un código de ciudad, un nombre y un nombre de departamento.
    - Las sedes pueden tener múltiples trabajadores y, a su vez, cada trabajador puede trabajar en diferentes sedes.
    - Los trabajadores tienen un número de cédula, nombres, apellidos, dirección, fecha de nacimiento, estado civil y sexo.

4. Seleccione un sitio web —por ejemplo, Facebook— e imagine y diseñe un modelo ER que soporte algunas de sus funcionalidades.
5. Consulte la Ley 1273 de 2009 e identifique cuáles son los delitos relacionados con la manipulación y el acceso indebido a los datos que están tipificados en el Código Penal de Colombia.

Capítulo 9

Diseño lógico

Resultados de aprendizaje

– *Diseña las tablas que componen el esquema lógico relacional de una base de datos a partir de modelos conceptuales entidad-relación.*
– *Mejora el diseño del esquema lógico relacional de una base de datos a partir de la especificación de nuevos requisitos y la incorporación de elementos necesarios para la implementación y la operación.*

Con el modelado conceptual presentado en el capítulo anterior, se cumplió con el propósito de definir cómo las diferentes entidades identificadas en los requisitos de la base de datos por construir se deben relacionar entre sí para hacer posible la satisfacción de las necesidades de datos. Sin embargo, el hecho de que el modelo conceptual sea independiente del modelo de datos (estructuras lógicas) que se debe utilizar para representar las entidades, los atributos y las relaciones no permite tener una imagen detallada de la solución.

Por otra parte, dado que el modelo lógico tiene en cuenta el modelo de datos, es decir, cómo se organizarán los datos, ofrece un mayor nivel de detalle. Con lo dicho no se está afirmando la conveniencia de un nivel de diseño sobre el otro, sino dando a entender que estos se complementan entre sí.

El modelo lógico, aunque es independiente de la implementación física, define cuáles son las estructuras de datos que se utilizarán para representar los datos, qué elementos deben tener estas estructuras, cuáles operaciones se pueden realizar sobre ellas, etc. Las bases de datos relacionales están basadas, como se mencionó en el capítulo I, en el modelo relacional propuesto por Edgard Frank Codd, el cual está soportado en la teoría de conjuntos y en la lógica de predicado. La primera permite manipular las tablas y sus elementos de forma que sea posible

unirlas y combinarlas; la segunda, por otra parte, da la capacidad de expresar, por ejemplo, las condiciones para seleccionar y trabajar con elementos específicos de los conjuntos.

En este modelo todos los datos son lógicamente estructurados en relaciones. Ahora bien, es importante no confundir el concepto de relaciones del modelo ER con el del modelo relacional. En este último, para facilitar la comprensión, conviene recurrir al concepto de tabla como sinónimo. Dentro del modelo relacional, cada relación tiene un nombre y está conformada por atributos (los nombres de las columnas de la tabla), y a su vez cada relación es un conjunto de tuplas (filas). Los atributos (columnas) de cada tupla (fila) dentro de la relación (tabla) pueden tomar un valor específico. Algunas de las principales ventajas del modelo relacional son su simplicidad (todos los objetos son tablas) y que tiene un sustento matemático que lo reviste de formalidad (teoría de conjuntos y lógica de predicado).

La Tabla 9.1 muestra una relación de nombre «Artistas» con cinco atributos, que son el «identificador», el «nombre», el «año de lanzamiento», el «tipo» y el «género principal». Así, la relación «Artistas» contiene cinco tuplas. Un ejemplo de una tupla es la que tiene por identificador *«50001»*; por nombre, *«Carlos Vives»*; por año de lanzamiento, *«1986»*; por tipo, *«Solista»,* y por género principal, *«Vallenato»*. Desde este punto se utilizarán los términos de «tabla», «fila» y «columna» por simplicidad y por coherencia con la forma como se ha venido trabajando hasta ahora.

Tabla 9.1. Tabla o relación «Artistas»

Artistas				
identificador	**nombre**	**año de lanzamiento**	**tipo**	**género principal**
50001	Carlos Vives	1986	Solista	Vallenato
50002	Niche	1979	Grupo	Salsa
50003	Shakira	1990	Solista	Pop
50004	Binomio de Oro de América	1976	Grupo	Vallenato
50005	J Balvin	2006	Solista	Urbano Latino

Como síntesis, el modelo relacional permite organizar todos los datos en tablas y operar sobre estas para insertar, actualizar, eliminar y consultar datos. En este capítulo, primero se explicará cómo llegar a tener un modelo lógico partiendo del modelo entidad relación y, luego, se indicará cómo introducir algunas mejoras destinadas a mejorar la semántica del modelo.

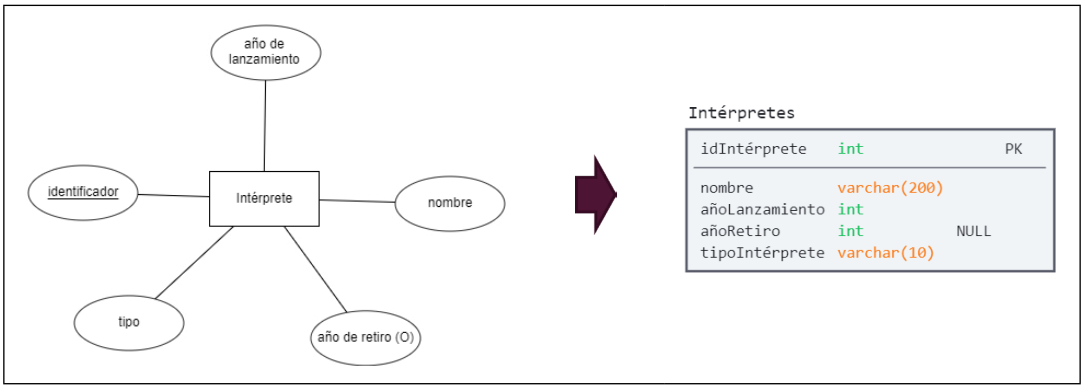
9.1. Modelo conceptual a modelo relacional

El primer pensamiento que aflora cuando surge la necesidad de transformar el modelo ER al modelo relacional es si existe un procedimiento que tenga como entrada el modelo conceptual y que, al seguirlo, arroje como salida el modelo relacional. La respuesta es sí: en efecto, existe un conjunto sencillo de reglas que indican las equivalencias que se deben hacer entre cada uno de los elementos del modelo ER y los del modelo relacional. Estas reglas son:

- Toda entidad se transforma en una tabla, y todo atributo de la entidad se transforma en una columna dentro de la tabla a la que pertenece.
- El identificador de la entidad se convierte en la clave primaria de la tabla.
- En las relaciones de uno a muchos la clave primaria de la entidad con cardinalidad 1 pasa a la tabla de la entidad cuya cardinalidad es muchos.
- Toda relación de muchos a muchos se convierte en una tabla que tendrá como columnas las llaves primarias de las entidades que intervienen en la relación.
- Los atributos asociados a una relación de muchos a muchos pasan a ser columnas de la nueva tabla.

Aunque estas reglas son muy sencillas de seguir, es mejor ilustrarlas haciendo la transformación paso a paso del modelo ER obtenido al final del capítulo 8 y plasmado en la Figura 8.12. Lo primero es transformar cada una de las entidades en una tabla. Por ejemplo, la entidad «Intérprete» será la tabla `Intérpretes`, y cada uno de los atributos de esta entidad será ahora una columna de la nueva tabla. Asimismo, la llave principal de la entidad «Intérprete», que según la figura es el atributo identificador, será ahora la llave principal (PK) de la tabla `Intérpretes` (en este caso se renombra como `idIntérprete` para facilitar la comunicación y posteriores operaciones como las de consulta). En la Figura 9.1 se presentan, en el lado izquierdo, la entidad que toma como entrada el procedimiento y, en el lado derecho, la tabla resultante.

Figura 9.1. Conversión de la entidad «Intérprete» a la tabla `Intérpretes`



Para el título de las tablas se ha decidido, por convención, utilizar el plural del sustantivo que da nombre a la entidad. También se ha optado por seguir la convención CamelCase para los nombres compuestos de las columnas y PascalCase para los nombres compuestos de las tablas. CamelCase opera de la siguiente forma: la primera palabra del nombre se escribe en minúsculas, y en el resto de las palabras la primera letra se escribe en mayúscula. En PascalCase, por su parte, la primera letra de todas las palabras se escribe en mayúscula. En ninguno de los dos casos se deja espacio entre las palabras.

Otro elemento importante de realizar la transformación del modelo ER al modelo relacional es considerar el dominio de valores que puede tomar cada uno de los atributos de la

entidad. Por ejemplo, la columna `idIntérprete` debe ser un número entero, el `nombre` y el `tipoIntérprete` solo puede contener caracteres alfabéticos, y el `añoLanzamiento` y el `añoRetiro` deben ser números. Esta restricción en el dominio de valores de cada columna se puede representar con el tipo de datos. Así, en el caso de `nombre` y `tipoIntérprete`, se ha señalado que debe ser una cadena de texto *(VARCHAR)*, mientras que para `idIntérprete`, `añoLanzamiento` y `añoRetiro` se asigna un número entero *(INT)*.

En la Figura 9.1 también se observa que los atributos opcionales de la entidad, como el `añoRetiro`, son marcados con la palabra `NULL` en la nueva tabla. Dicho término indica que para estas columnas no es necesario proporcionar un valor en el momento de la inserción de una nueva fila o que se puede utilizar el descriptor `NULL` al operar con dicha columna.

Es clave mantener presente la anterior descripción por cada columna en el momento de entender en su total extensión un modelo lógico. Por lo tanto, conviene seguir la buena práctica de ir creando un diccionario de datos paralelamente al modelo relacional que haga las veces de metadatos sobre los datos o, puesto en otros términos, «datos sobre los datos», orientados a mejorar la comprensión de cualquier persona que necesite trabajar con ellos.

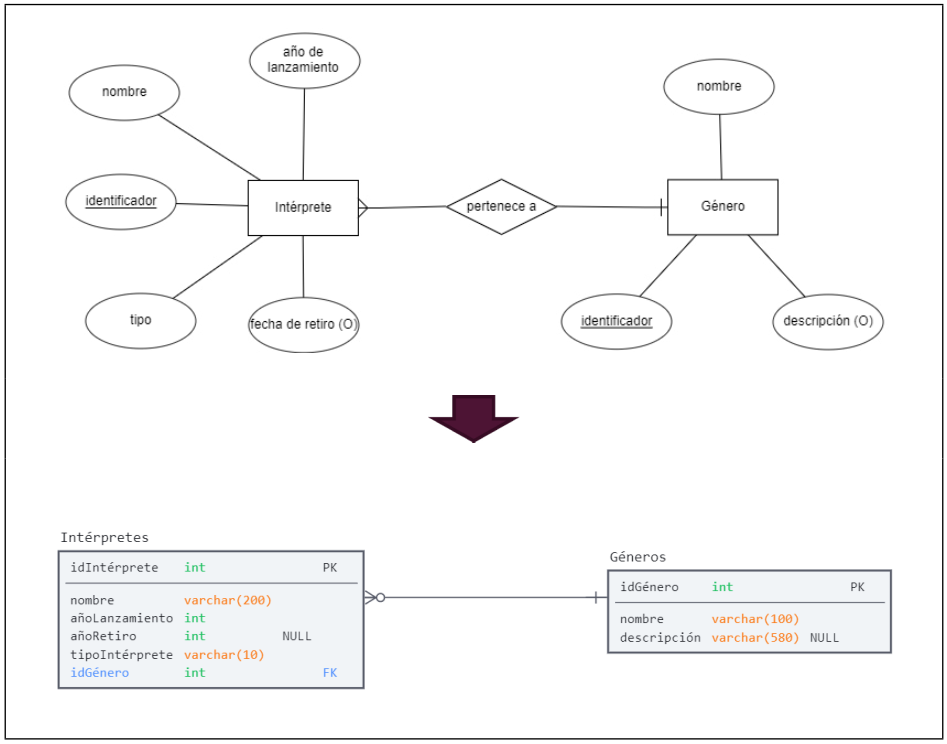
En la Tabla 9.2 se observa un resumen de la tabla `Intérpretes`. Este proceso se repite por cada una de las seis entidades que existen en el modelo ER, por lo que, al realizar este paso, se obtendrá la versión del modelo presentada en la Figura 9.2.

Tabla 9.2. Resumen de la tabla `Intérpretes`

Nombre	PK/FK	Acepta *NULL*	Descripción	Dominio
idIntérprete	PK	No	Un número independiente de dominio que representa el identificador de cada intérprete.	Un número entero positivo.
nombre		No	Representa el nombre artístico de cada intérprete.	Una cadena de texto.
añoLanzamiento		No	Un número entero de cuatro cifras que representa el año en que el intérprete inició su vida artística.	Un número entero positivo. Debe ser mayor que el año 1900.
añoRetiro		Sí	Un número entero de cuatro cifras que representa el año en que el intérprete finalizó su vida artística. En caso de no tener un valor, indicará que el intérprete no se ha retirado.	Un número entero positivo. Debe ser mayor que el año 1900.
tipoIntérprete		No	Una palabra que indica si el intérprete es solista o es un grupo musical.	Una cadena de texto.

Figura 9.2. Resultado de convertir todas las entidades en tablas

Figura 9.3. Conversión de una relación de uno a muchos del modelo conceptual al modelo lógico



Una vez creadas todas las tablas que tienen como equivalente una entidad en el modelo ER, surge la pregunta: ¿qué se debe hacer ahora con las relaciones? La respuesta, de acuerdo con el listado de las reglas, es: depende del tipo de relación. Para las relaciones uno a muchos como la que existe entre las entidades «Género» e «Intérprete», la clave primaria de la entidad con cardinalidad 1 pasa a la tabla de la entidad cuya cardinalidad es muchos. En el caso planteado, «Género» es la entidad con cardinalidad 1 e «Intérprete» es la de cardinalidad muchos, por lo que la llave principal de «Género» pasa como una columna a la tabla

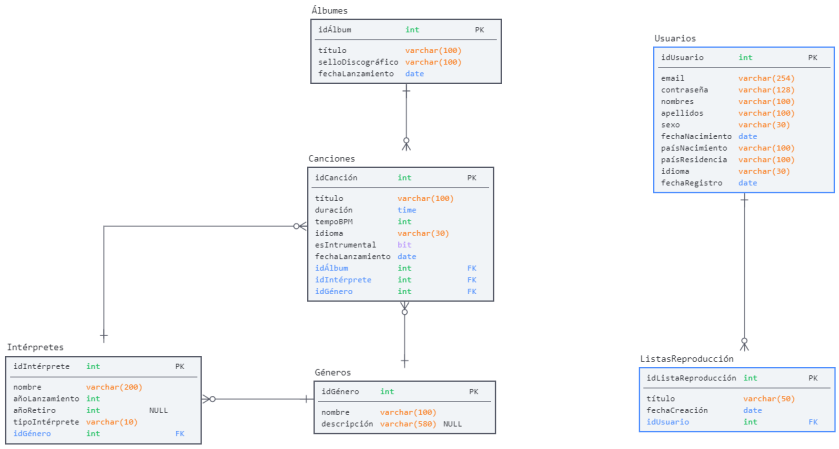
`Intérpretes`, convirtiéndose en llave foránea (FK, del término en inglés, *Foreign Key*). En la Figura 9.3 se muestra el resultado de esta transformación.

El paso recién descrito se repite por cada una de las relaciones de uno a muchos que existen en el diagrama relacional, lo que permitirá obtener una nueva versión del modelo lógico como la presentada en la Figura 9.4. Sobre esta nueva versión se debe resaltar cómo la tabla `Canciones` ahora tiene tres nuevas columnas: `idÁlbum`, `idIntérprete` e `idGénero`. Todas estas son llaves foráneas puesto que son producto de relaciones de uno a muchos con otras entidades donde la entidad «Canción» era la parte de la relación de cardinalidad muchos.

Las llaves foráneas, más allá del nombre y provenir de otra tabla, imponen una restricción sobre el dominio de valores que puede tomar la columna. Esta restricción indica que la columna solo puede tomar valores que existan previamente para esa misma columna en la tabla de la cual proviene o en la tabla en la cual es llave principal. Dicho de otro modo, los valores que puede tomar, por ejemplo, la columna `idGénero` de la tabla `Canciones` deben existir previamente como valores de la columna `idGénero` en la tabla `Géneros`.

Esta regla solo se exceptúa en el caso de que se declare que la llave foránea puede tomar valores nulos, como en el caso de la columna `idÁlbum`. En este caso el valor que para esta columna se proporcione a cualquier canción puede ser `NULL`, aunque si no tiene dicho valor, debe existir entonces previamente en la columna `idÁlbum` de la tabla `Álbumes`.

Figura 9.4. Conversión de todas las relaciones de uno a muchos del modelo conceptual a su representación en el modelo lógico



Siguiendo la recomendación brindada antes, se agrega la descripción de la tabla `Canciones` al diccionario de datos, como se observa en la Tabla 9.3.

Tabla 9.3. Descripción de la tabla `Canciones`

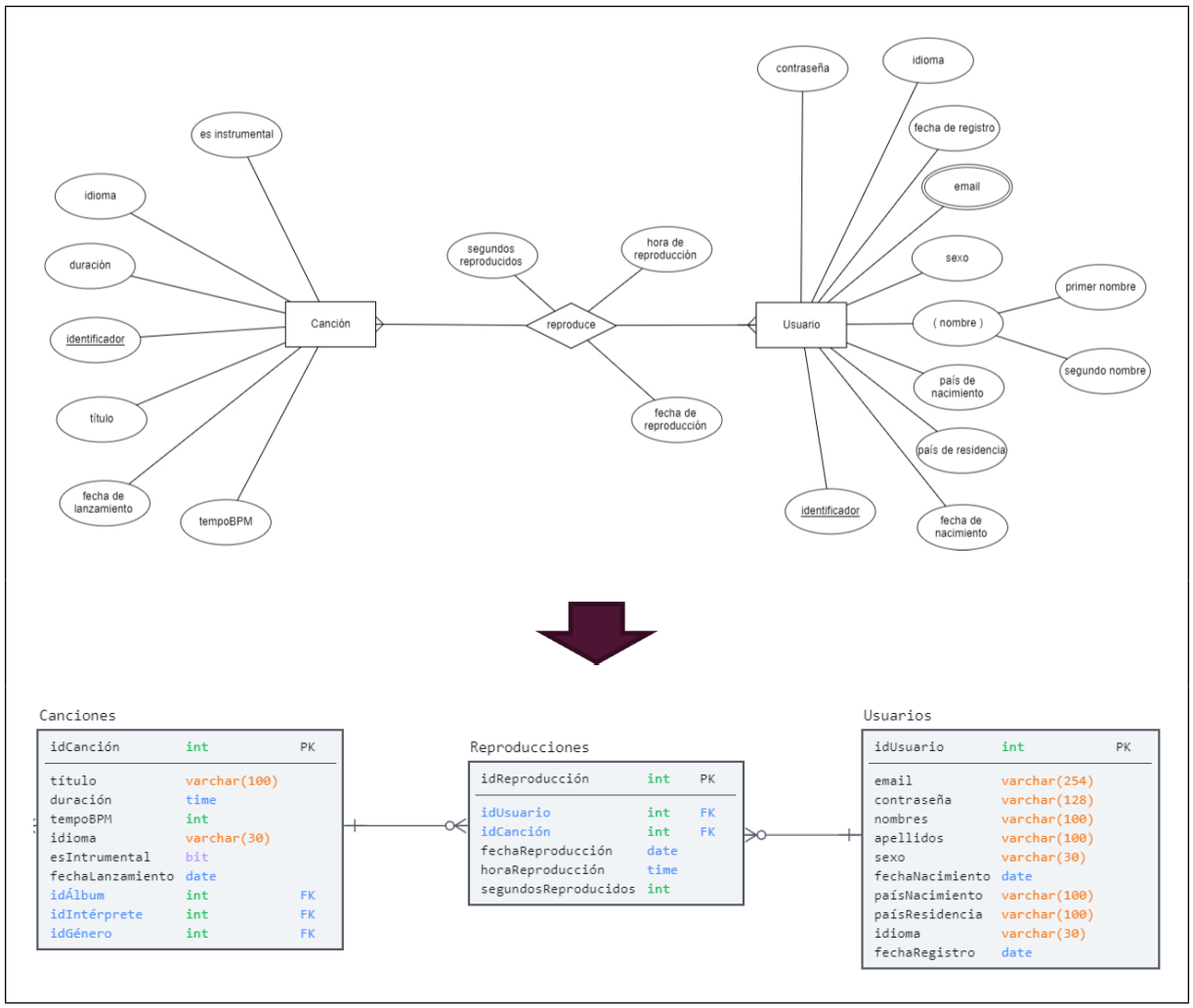
Nombre	PK/FK	Acepta *NULL*	Descripción	Dominio
idCanción	PK	No	El identificador de cada canción.	Un número entero positivo.
título		No	Corresponde al nombre de la canción.	Una cadena de texto.
duración		No	El tiempo que tarda la canción.	Un número en el formato «hh:mm:ss» que representa cuánto tiempo dura una canción.
idioma		No	El lenguaje en el que la canción es interpretada; por ejemplo, español o inglés.	Una cadena de texto.
esInstrumental		No	Un caracter que indica si la canción es instrumental. El «1» corresponde a instrumental; el «0» significa que no lo es.	Un carácter de dos posibles valores: 0 o 1.
fechaLanzamiento			Corresponde a la fecha en que la canción se dio a conocer al público.	Una fecha en el formato «DD/MM/YYYY».
idÁlbum	FK	Sí	Representa un álbum que existe en la tabla `Álbumes`.	Un número entero de los que están almacenados en la columna `idÁlbum` de la tabla `Álbumes`.
idIntérprete	FK	No	Representa un intérprete que existe en la tabla `Intérpretes`.	Un número entero de los que están almacenados en la columna `idIntérprete` de la tabla `Intérpretes`.
idGénero	FK	No	Representa un género que existe en la tabla `Géneros`.	Un número entero de los que están almacenados en la columna `idGénero` de la tabla `Géneros`.

El siguiente paso es trabajar con las relaciones de muchos a muchos. Siguiendo la recomendación, lo que se hace es crear una nueva tabla, la cual tendrá como llaves foráneas la llave principal de las entidades que intervienen en la relación, además de todos los atributos que la entidad posee *per se*. Por ejemplo, en la relación «reproduce» que hay entre las entidades «Usuario» y «Canción» se entiende que los usuarios reproducen canciones, por lo que la nueva tabla creada tendrá el nombre `Reproducciones` y tendrá como llaves foráneas `idUsuario` de la tabla `Usuarios` e `idCanción` de la tabla `Canciones`. También añade las columnas `fechaReproducción`, `horaReproducción` y `segundosReproducidos`.

En este punto es preciso tomar una decisión: ¿cuál es la llave principal de la nueva tabla? Si bien se puede optar por la combinación de `idCanción` e `IdUsuario`, es decir, una llave principal compuesta o superclave, en este caso no se podría dar una coincidencia del identificador de la canción con el identificador del usuario; puesto en otros términos, un mismo usuario no puede reproducir dos veces la misma canción, y esto no es lo que se desea.

Una alternativa sería incluir otra columna como la fecha y la hora a la llave principal, pero se ha decidido sumar una nueva columna, de nombre `idReproducción`, que permita identificar de forma inequívoca cada relación y, a su vez, tener más de una reproducción de la misma canción por el mismo usuario. En la Figura 9.5 se muestra el resultado de esta transformación.

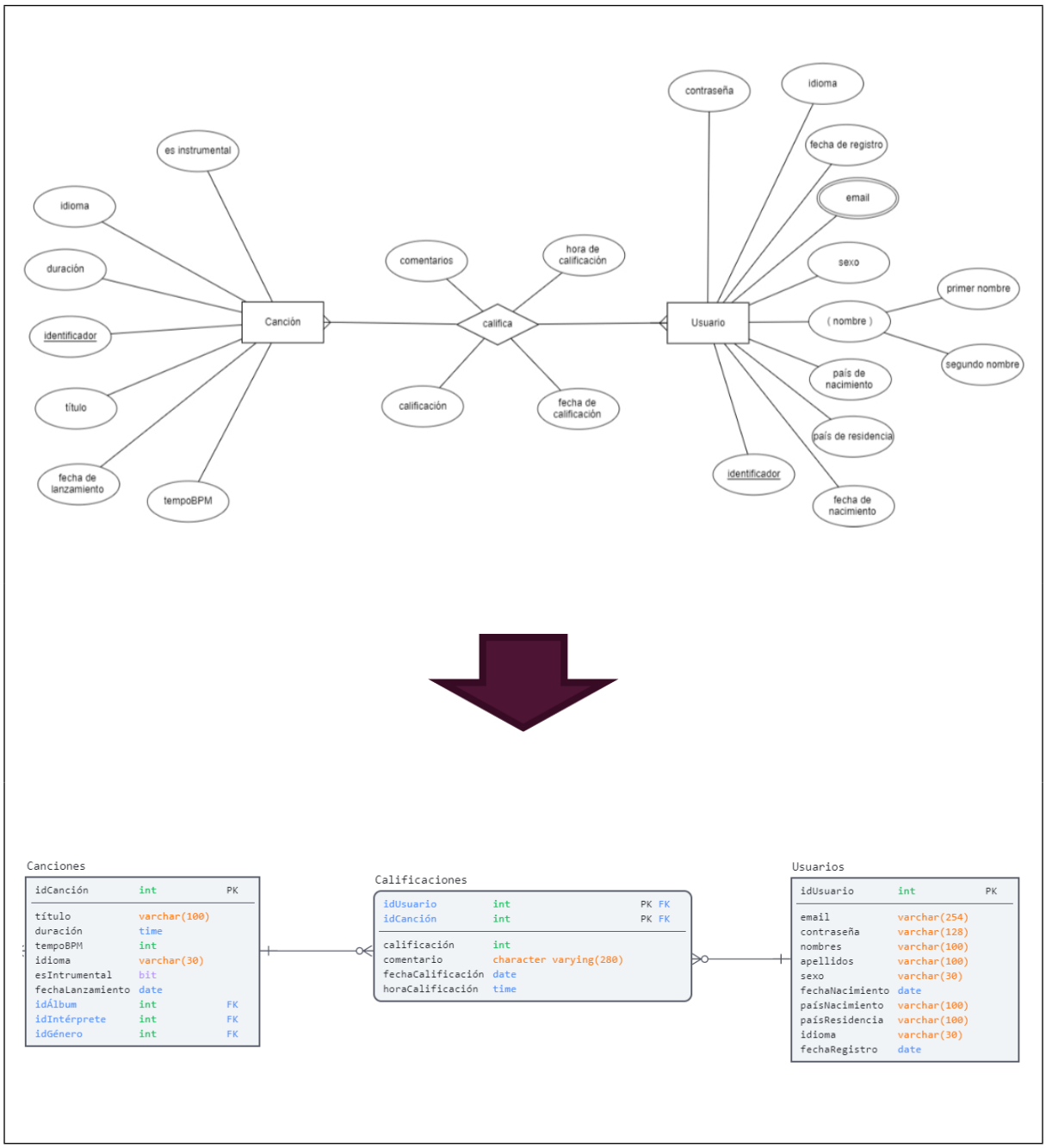
Figura 9.5. Relación «reproduce» entre las entidades «Usuario» y «Canción»



No siempre es necesario añadir una nueva columna para definir la llave principal de una tabla producto de una relación de muchos a muchos. Así lo demuestra, por ejemplo, el caso de la relación «califica» entre «Usuario» y «Canción», donde se ha definido una nueva tabla `Calificaciones` que contiene como llaves foráneas el identificador de la canción y el de los usuarios. Aquí la combinación de ambas llaves foráneas es la llave principal de la tabla, es decir, una llave compuesta, algo que resulta útil en este contexto porque se busca que un usuario solo pueda calificar una sola vez la misma canción.

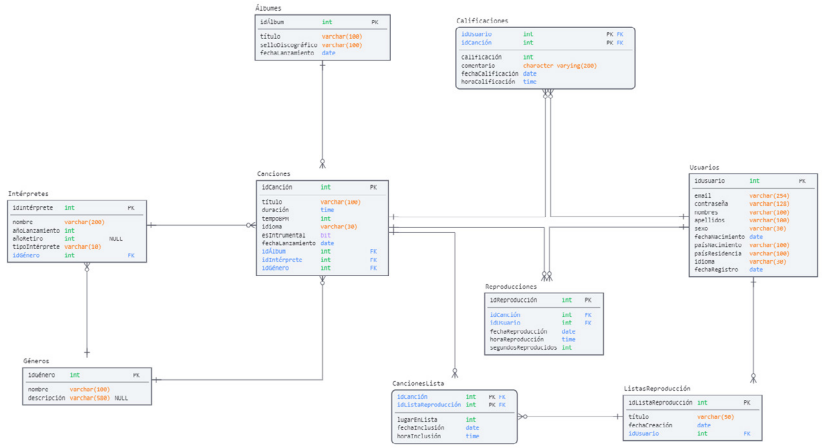
Luego de haber definido la llave principal, se añaden los atributos que hacen parte de la relación como columnas de la tabla, es decir: `fechaReproducción`, `horaReproducción` y `segundosReproducidos`. El resultado de la transformación se aprecia en la Figura 9.6.

Figura 9.6. Relación «califica» entre las entidades «Usuario» y «Canción»



Luego de seguir cada uno de los pasos del procedimiento explicado, se obtiene el modelo relacional equivalente al modelo ER presentado en la Figura 8.12 . El modelo resultante es el mostrado en la Figura 9.7.

Figura 9.7. Resultado de convertir el modelo conceptual al modelo lógico



9.2. Evolución del modelo relacional

En este punto se ha llegado a una versión del modelo relacional que representa la base de datos que se quiere construir para satisfacer las necesidades de datos planteadas. Sin embargo, esto no quiere decir que sea la versión definitiva. El cambio está inherente en el diseño de la base de datos: nuevos requisitos surgen, la normatividad vigente cambia, la necesidad de mejoras se hace evidente, etc. Por lo tanto, el cambio no debe ser visto con miedo, sino como una señal de que un modelo debe evolucionar para adaptarse a lo que el entorno organizativo está exigiendo. Ahora se planteará un nuevo requisito.

En la plataforma «*Mis Canciones*» existen unos planes a los cuales el usuario puede suscribirse. Por tanto, se desea llevar el registro de los pagos que realiza el cliente de cada plan.

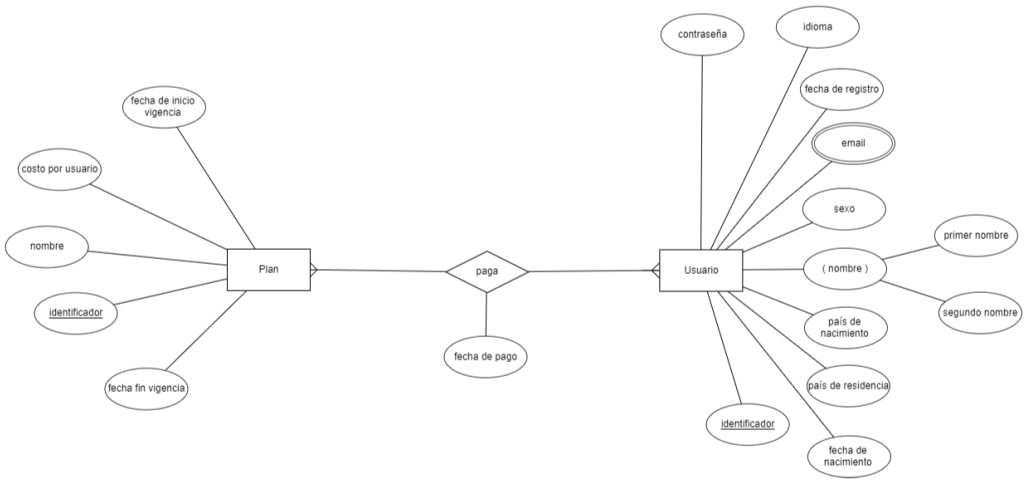
Al analizar el requisito planteado, es posible extraer las siguientes reglas semánticas que guiarán un nuevo cambio en el diseño:

- Los planes tienen un identificador, un nombre, un costo y una fecha de inicio y de fin de vigencia.
- Un usuario se suscribe a un plan a través de un pago mensual que realiza.
- Un mismo plan es pagado por diferentes usuarios en distintos momentos.
- Del pago es importante saber: la fecha en que se realizó y el monto.

Una alternativa para modelar lo que se acaba de describir es plantear una relación de muchos a muchos entre una entidad «Plan» y otra «Usuario». El nombre de esta relación sería «paga», en donde un usuario realiza uno o muchos pagos de un plan y un plan es pagado

por uno o muchos usuarios. En la Figura 9.8 se observa este planteamiento, asignándole a la relación «paga» el atributo de fecha de pago debido a que este tiene únicamente sentido cuando se realiza un pago.

Figura 9.8. Relación paga entre «Usuario» y «Plan»



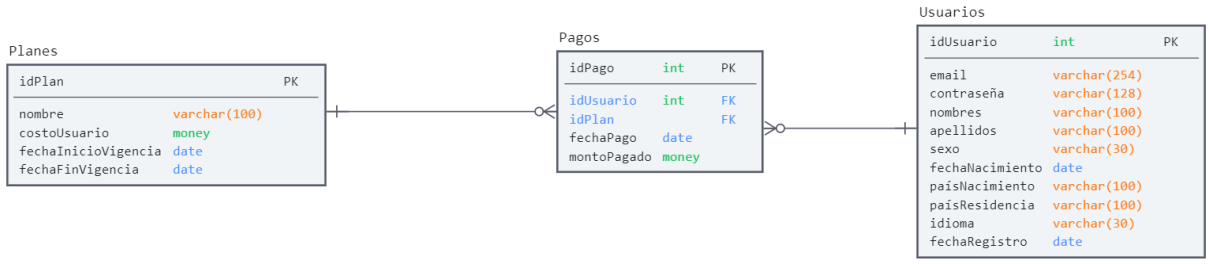
La transformación de lo modelado en la Figura 9.8 al modelo relacional plasmado en la Figura 9.9 se logra siguiendo las reglas ya descritas: la entidad «Plan» se convierte en la tabla `Planes` con los atributos de la entidad como columnas de la tabla. Luego, la relación que hay de muchos a muchos entre las entidades «Usuario» y «Plan» se vuelve una nueva tabla donde las llaves principales de estas dos entidades se convierten en llaves foráneas. A su vez, se define una nueva columna como llave principal, la cual contendrá un número entero que identifique cada pago.

En la nueva tabla `Pagos` también se ha añadido la columna `fechaPago` puesto que el atributo de fecha de pago pertenece a la relación «paga». Finalmente, de igual modo se ha incluido la columna `montoPagado` a la nueva tabla `Pagos` debido a que el valor de la columna `costoUsuario` puede cambiar y, al hacerlo, no habría forma de saber cuánto pagó un usuario por su plan, salvo que el pago haya sido posterior al último cambio del costo por usuario.

Analice ahora los requisitos planteados a continuación.

En la plataforma «*Mis Canciones*» se quiere tener registro de los miembros de los grupos y, de cada uno de ellos, detalles como el periodo de vinculación y el rol que ejerció en el grupo durante este lapso.

Figura 9.9. Resultado de converir la relación paga (de muchos a muchos) entre las entidades «Usuario» y «Plan»



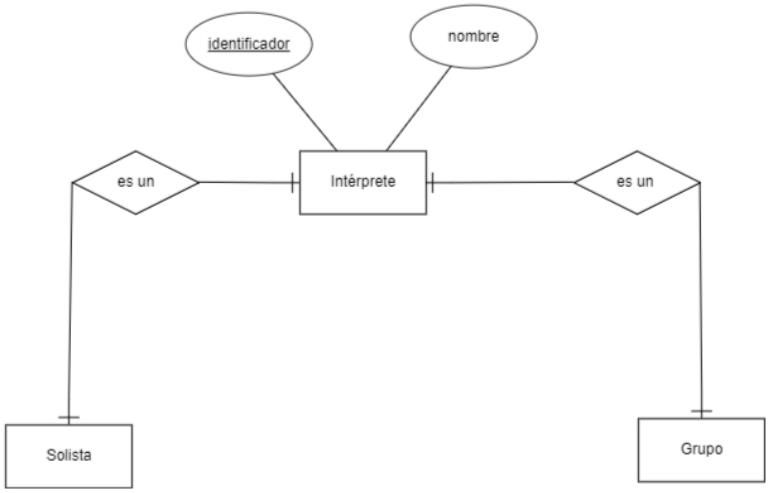
Es claro que los intérpretes de las canciones se han clasificado hasta aquí dentro de dos tipos: solistas o grupos, hasta ahora dentro de una misma entidad debido a que todos los atributos almacenados son comunes tanto para solistas como para los grupos. No obstante, con el requisito descrito se demanda almacenar, para los grupos en particular, la fecha en que cada miembro se vinculó y retiró del grupo y el rol que ejerció durante el periodo en que duró la vinculación, datos que no son necesarios para los solistas. Por consiguiente, deja de ser apropiado almacenar los dos tipos de intérpretes en la misma entidad. En este contexto es útil el concepto de especialización/generalización, con el cual se pueden representar relaciones entre entidades del tipo «La entidad "X" es una subclase de la entidad "Y"», «La entidad Y es una superclase de la entidad "X"».

La especialización permite que, a partir de aquellas entidades que incluyen la ocurrencia de uno o más subgrupos —por ejemplo, la entidad «Intérprete» incluye la ocurrencia de los subgrupos «Solista» y «Grupo»—, se puedan desprender tantas entidades como subgrupos haya. Siguiendo con el caso de los intérpretes, se tiene que de la entidad «Intérprete» se desprenden dos nuevas entidades: «Solista» y «Grupo», las cuales son también del tipo de la entidad de la que se desprendieron; puesto en otros términos, un «Solista» es un «Intérprete».

La razón de hacer lo anterior es básicamente que cada uno de estos subgrupos, como lo plantea el requisito recién presentado, puede tener atributos que lo distingan de otros subgrupos, lo que agrega información semántica al diseño. En la Figura 9.10 se observa cómo se puede plantear esta relación.

Las entidades entonces pueden tener distintas subclases (entidades hijas), y las subclases pueden contar con distintas superclases (entidades padres). En la Figura 9.10 la entidad «Intérprete» es la superclase y las entidades «Solista» y «Grupo» son las subclases. Como se mencionó, cada miembro de una subclase también lo es de la superclase, pero tiene un papel distinto. La relación entre una superclase y una subclase es uno a uno (1:1), es decir, un intérprete es un solista o un intérprete es un grupo. Este tipo de relación donde la entidad padre puede ser solo una de las entidades hijas se conoce como disjunta. En este punto conviene pausar por un instante y analizar el siguiente requisito.

Figura 9.10. Relaciones de uno a uno



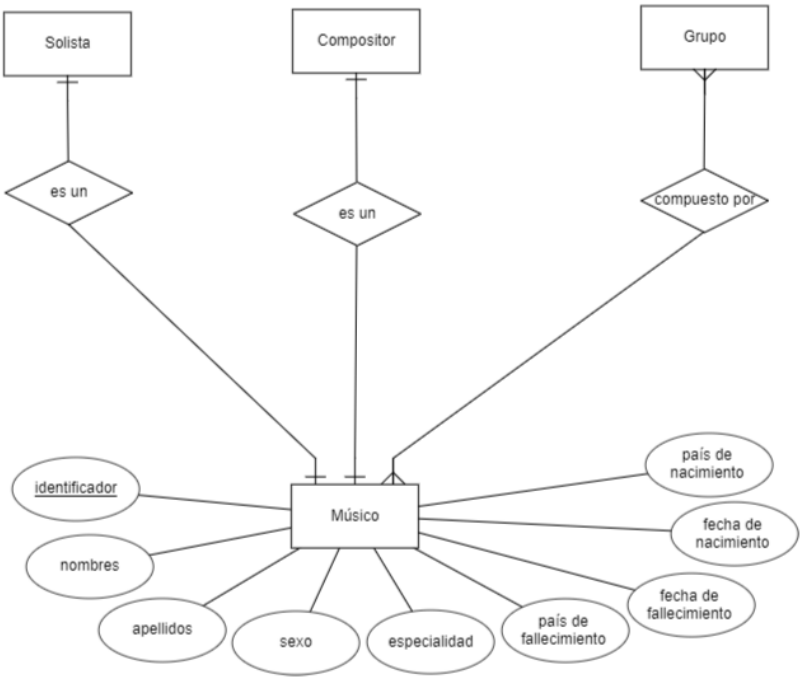
En la plataforma «*Mis Canciones*» también se desea almacenar los compositores de las canciones y mantener datos personales como el nombre real y la fecha de nacimiento tanto de compositores como de solistas y miembros de cada grupo.

Aunque esta solicitud se plantea como un solo requisito, al analizarla con detenimiento se puede inferir que hay mínimamente dos contenidos en ella: el primero centrado en saber quiénes son los compositores de las canciones, y el segundo, en tener datos personales de solistas, compositores y miembros de grupos.

Inicialmente, se partirá del segundo aspecto. Para darle solución es posible plantear tres entidades: «Solista», «Compositor» y «Miembros de grupo», y en cada una de estas incluir los atributos personales que se desea almacenar, es decir, atributos como nombres, apellidos, sexo, etc. Sin embargo, este enfoque para cada entidad, aunque intuitivo y útil, tiene algunos inconvenientes; por ejemplo, ¿qué sucede si una misma persona ha sido compositor, miembro de grupo y solista? Tendrían que repetirse datos, algo que, según se estableció, en las bases de datos relacionales con propósito transaccional es un síntoma de que algo no va bien. Es en este tipo de escenario donde se aplica la generalización.

El racionamiento en el que tiene sustento la generalización es que, si un grupo de entidades ubicadas en el mismo nivel conceptual tiene atributos comunes, se puede realizar una abstracción que consolide todos estos y se convierta en superclase o clase padre de las otras clases. Aplicado al requisito actual, se puede afirmar que tanto compositores como solistas y miembros de los grupos musicales son músicos, por lo que es posible asignar a esta nueva entidad «Músico» los nuevos atributos. Lo descrito está representado en la Figura 9.11.

Figura 9.11. Relaciones de generalización y especialización entre entidades

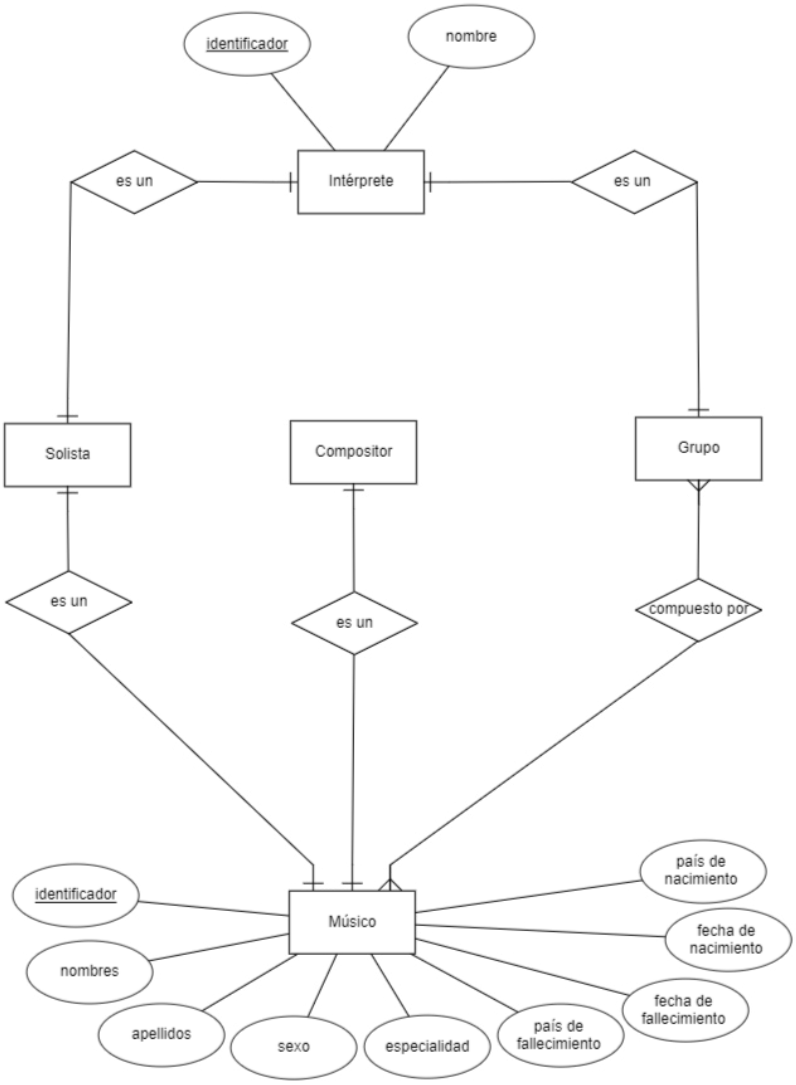


Algunas superclases pueden contener subclases superpuestas. Por ejemplo, en lo recién modelado un músico puede ser compositor y solista al tiempo. Tanto la especialización como la generalización permiten llegar al mismo punto utilizando enfoques metodológicos diferentes: la especialización sigue un enfoque *top-down*, donde se parte de tener una superclase y se identifica cuáles son las diferencias que hay entre los subgrupos que la superclase actualmente está representando. Por otro lado, la generalización sigue un enfoque *bottom-up*, es decir, parte de las subclases y busca los atributos comunes entre ellas para luego condensarlos en una superclase. En la Figura 9.12 se ha unificado la especialización que se realizó de los intérpretes y la generalización que se hizo de compositores, solistas y miembros de grupos musicales.

En la Figura 9.12 se puede ver cómo los solistas tienen una relación de uno a uno con intérpretes, es decir, un solista es un intérprete, y otra de uno a uno con «Músico». Así pues, un solista es un músico, por lo que un solista tendrá acceso tanto a los atributos de la entidad «Intérprete» como a los de «Músico». En el caso de los miembros de los grupos, la modelación tuvo presente que un músico puede pertenecer a muchas agrupaciones durante su vida musical y que un grupo musical puede estar conformado por varios músicos, es decir, se planteó una relación de muchos a muchos entre músicos y grupos. En cuanto a los atributos de la fecha de vinculación, la fecha de retiro y el rol, se incluyeron en la relación debido a que solo cobran sentido cuando un músico particular pertenece a un grupo específico en

un instante de tiempo; puesto en otras palabras, no son atributos del grupo, ni del músico, sino de la relación entre estos dos.

Figura 9.12. Unificación de las especializaciones y generalizaciones utilizadas

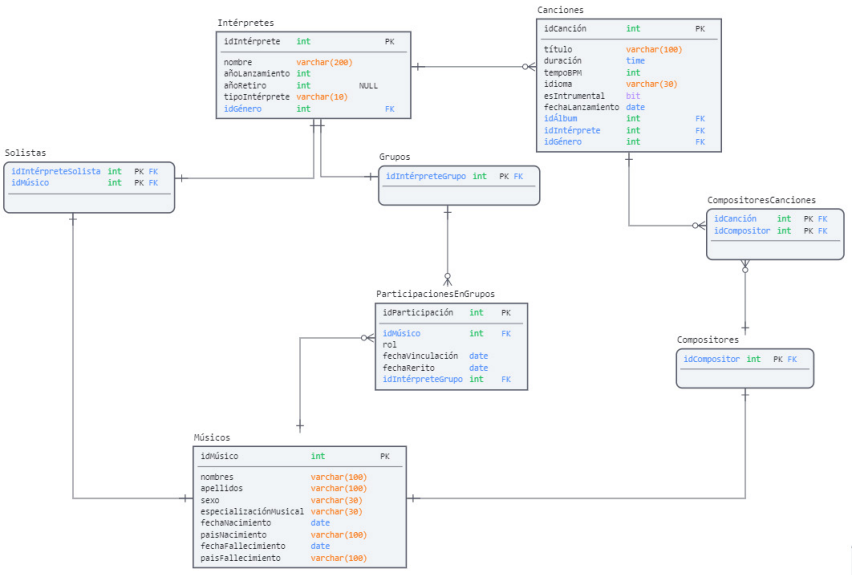


No olvide que uno de los requisitos es saber cuáles son las canciones que ha creado cada compositor. Como ya existe una entidad «Compositor» especializada y lo que se quiere modelar es que un compositor pueda crear varias canciones y que una canción pueda tener varios compositores, se puede plantear una relación de muchos a muchos entre las entidades «Canción» y «Compositor».

La Figura 9.13 representa la traducción de los nuevos requisitos modelados al modelo relacional. Para hacer esto posible se han seguido, además de las reglas ya presentadas, una adicional: en las relaciones uno a uno la llave primaria de la superclase es también la llave

primaria de la subclase, como se ve en la tabla `Solistas`. En el caso de «grupo» y «Músico», como existe una relación de muchos a muchos, se ha creado una tabla `ParticipacionesEnGrupos`, donde se almacenan los atributos de la relación. Lo mismo ocurre con la relación entre las entidades «Compositor» y «Canción», que al ser de muchos a muchos provoca la creación de una nueva tabla: `CompositoresCanciones`, la cual tiene como llave principal la combinación de las llaves principales de las tablas `Canciones` y `Compositores`.

Figura 9.13. Conversión de las especializaciones y generalizaciones del modelo conceptual a su representación en el modelo lógico

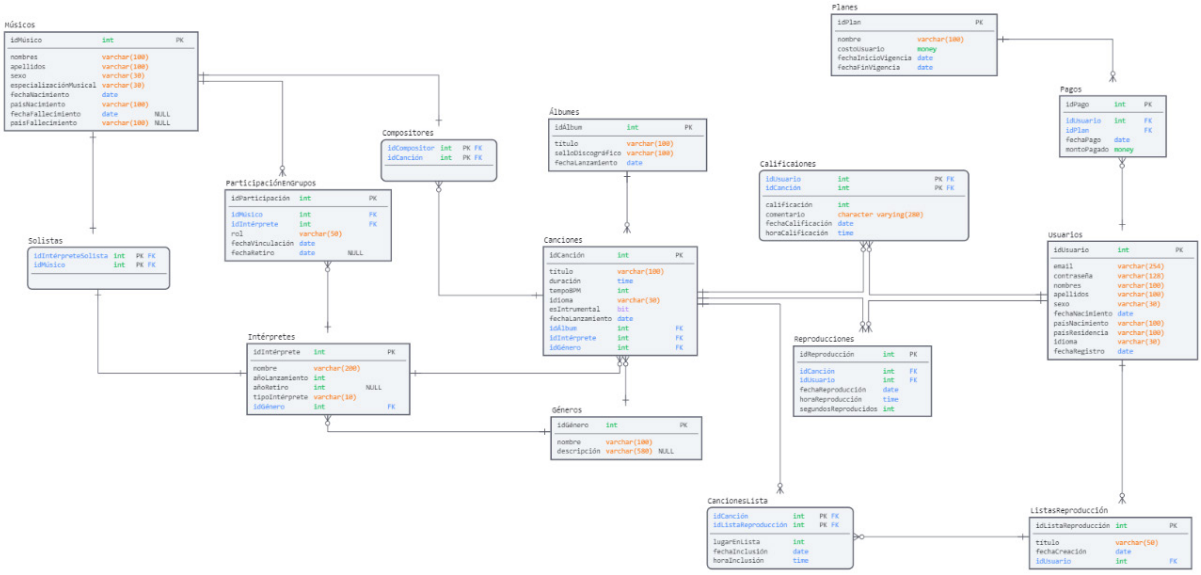


Para concluir, en la Figura 9.14 se presenta la versión que da respuesta a todos los requisitos presentados hasta este punto. En esta versión «final» también se han realizado unas mejoras, considerando que el diseño y la construcción de bases de datos son un proceso de mejora continua. Los cambios incorporados son:

- Se renombraron algunas columnas para aumentar el valor semántico y la legibilidad del modelo. Por ejemplo, en la tabla `Solistas` la llave foránea que proviene de la tabla `Intérpretes` ha cambiado de `idIntérprete` a `idIntérpreteSolista`.
- Se definió que algunas columnas acepten valores nulos, como es el caso de `fechaFallecimiento` y `paísFallecimiento` de la tabla `Músicos`.
- Se removieron las tablas `Grupos` y `Compositores` porque no tenían ningún atributo diferente al heredado de la tabla `Músicos`. Por consiguiente, la relación con canciones

se planteó directamente con «Músicos» y se le dio el nombre de «Compositores». De forma similar se procedió con «Grupos»: la relación se planteó directamente entre «Intérpretes» y «Músicos», manteniendo la relación entre estas dos con el nombre de la tabla `ParticipacionesEnGrupos`.

Figura 9.14. Versión completa del modelo lógico

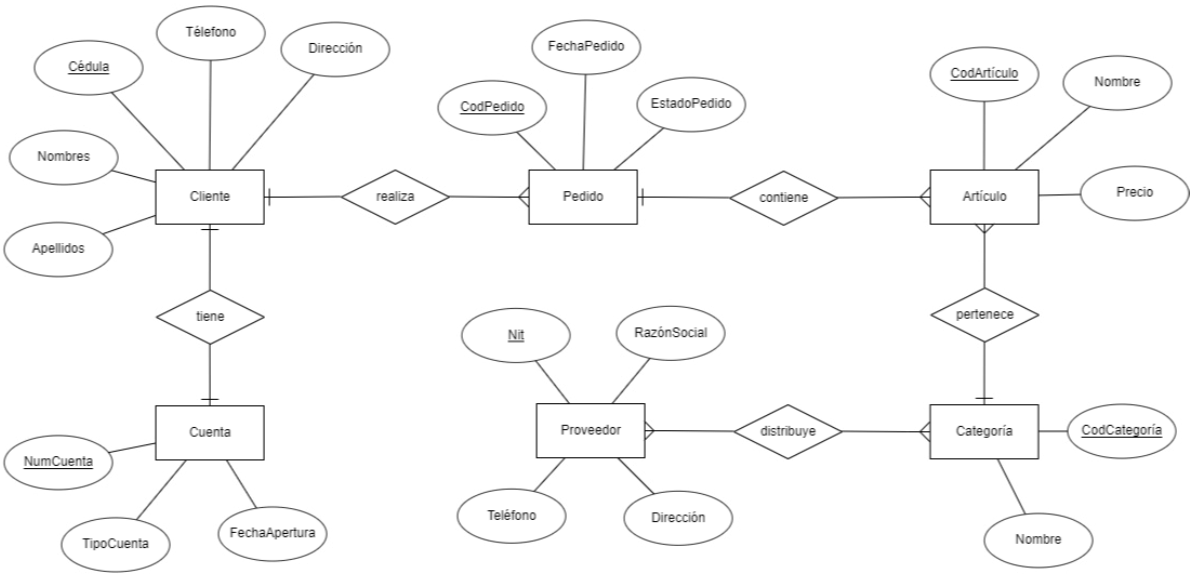


9.3. Aprendizajes más importantes del capítulo 9

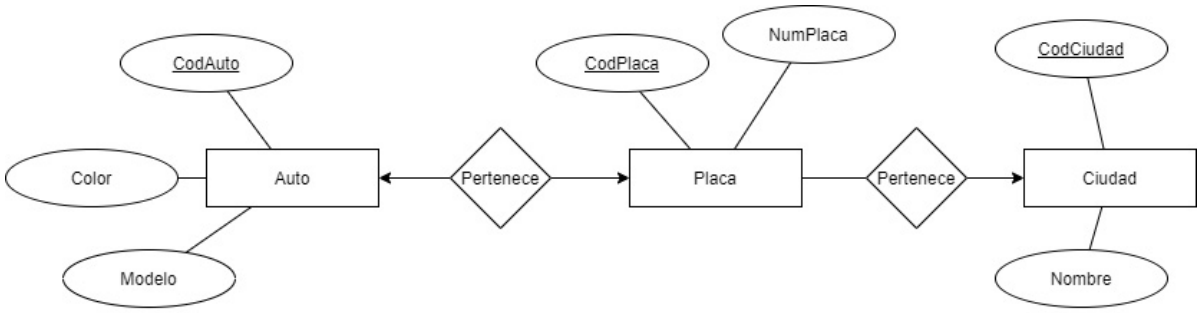
- Existe un conjunto claro de reglas que permite convertir fácilmente un modelo ER a un modelo relacional.
- A nivel lógico, las bases de datos relacionales están soportadas en el modelo relacional. Este trae consigo ventajas como la simplicidad y la formalidad.
- Es importante, además del diagrama del modelo relacional, ir creando un diccionario de datos que sirva de metadatos para los datos. Dicho en otros términos, conviene ir creando artefactos que faciliten la comprensión del modelo creado.
- Cuando se selecciona el enfoque de especialización, se intenta resaltar las diferencias entre entidades definiendo una o más subclases de una entidad superclase. Al optar por el enfoque de generalización, se busca identificar características comunes entre entidades para definir una entidad generalizadora.

9.4. Actividades de aplicación para evidenciar lo aprendido

1. Aplique el procedimiento para convertir el siguiente modelo ER en un modelo relacional:



2. Se ha argumentado que las tablas Grupos y Compositores se removieron del diagrama de la Figura 9.13 porque no se había añadido ningún atributo que justificara su inclusión. Proponga un atributo que haga necesario agregar estas dos tablas.
3. Indique si la conversión realizada en la siguiente figura es correcta o no. En caso de que no lo sea, plantee la solución correcta.



Auto

PK CodAuto	Color	Modelo	PK CodPlaca

Placa

PK CodPlaca	NumPlaca

Ciudad

PK CodCiudad	Nombre	FK CodPlaca

4. Añada una generalización o especialización al modelo ER definido en el punto 3 del capítulo 8. Luego realice la conversión del modelo ER resultante al modelo relacional.
5. Defina qué son los datos públicos, los datos privados, los datos semiprivados y los datos sensibles. Luego proporcione un ejemplo de cada uno de estos dentro del dominio de «*Mis Canciones*».

Capítulo 10

Normalización

Resultados de aprendizaje

– *Determina la forma normal en la que se encuentra una tabla de una base de datos y los problemas que pueden generarse por estar en dicho estado.*
– *Modifica el diseño de las tablas de una base de datos para resolver problemas de redundancia y reducir riesgos de pérdida de integridad.*

Al diseñar una base de datos para soportar las operaciones de una organización es necesario tomar en consideración aspectos que permitan asegurar la calidad de los datos, evitar las inconsistencias que pongan en riesgo la integridad y coherencia de los datos y garantizar que los esquemas relacionales diseñados no tengan elementos que puedan generar un bajo rendimiento en el acceso. Esto es de vital importancia para todas las partes interesadas.

La organización quiere tener datos consistentes en todo momento, de forma que le permitan realizar las actividades de forma efectiva. Los usuarios, además de la consistencia en los datos que consultan, quieren que las operaciones que realizan con la base de datos demoren lo menos posible. Por su parte, los diseñadores de la solución también tienen el interés de entregar una representación precisa de los datos, de las relaciones entre estos y de las restricciones sobre ellos, de forma que la base de datos sea pertinente para la empresa.

Para crear un producto que cumpla con las expectativas de clientes y usuarios, en el ámbito del desarrollo de software y de los sistemas de información se utilizan diferentes técnicas para probar si lo que se está construyendo se ajusta a unos estándares. Una de las técnicas aplicadas en el diseño de las bases de datos es la normalización.

La normalización se fundamenta en verificar la dependencia entre los atributos de las tablas, la cual se denomina dependencia funcional. De esta manera se busca que en el diseño

de una base de datos se utilicen la cantidad mínima de columnas necesarias para satisfacer los requisitos de datos. El objetivo es contar con un conjunto adecuado de tablas que facilite el acceso y el mantenimiento de los datos, y que ocupen un menor espacio de almacenamiento.

10.1. Tablas sin normalizar

Tabla 10.1. Tabla `Canciones` desnormalizada

Canciones						
identificador	**título**	**duración**	**género**	**nombre**	**tipo de intérprete**	**año de lanzamiento**
10001	La tierra del olvido	4:25	Vallenato	Carlos vives	Solista	1986
10002	Ojos así	3:57	Pop	Shakira	Solista	1990
10003	Mi gente	3:05	Urbano Latino	J. Balvin	Solista	2006
10004	Ambiente	4:08	Urbano Latino	J. Balvin	Solista	2006
10005	Cali pachanguero	4:51	Salsa	Niche	Grupo	1979
10006	La creciente	3:04	Vallenato	El Binomio de Oro	Grupo	1976
10007	Sueños de conquista	4:02	Vallenato	El Binomio de Oro	Grupo	1976
10009	Carito	3:39	Pop	Carlos vives	Solista	1986
10011	Una aventura	5:16	Salsa	Niche	Grupo	1979
10012	Ginza	4:39	Urbano Latino	J. Balvin	Solista	2006
10013	Octavo día	4:32	Pop	Shakira	Solista	1990
10014	Quiero verte sonreír	3:18	Pop	Carlos vives	Solista	1986

En la Tabla 10.1 se representa un conjunto de datos compuesto por siete columnas que almacenan el identificador, el título, la duración, el género y el intérprete de un grupo de canciones. De cada uno de los intérpretes también se tienen el tipo y el año de lanzamiento a la vida artística. Así, por ejemplo, la cuarta fila de esta tabla representa la canción con identificador «*10004*», título «*Ambiente*», duración «*4:08*», género «*Urbano Latino*», nombre del intérprete «*J. Balvin*», tipo de intérprete «*Solista*» y año de lanzamiento del intérprete «*2006*».La Tabla 10.1 cumple con el propósito básico de almacenar los datos de las canciones satisfactoriamente y permite responder preguntas como: ¿cuántas canciones han sido interpretadas por solistas?, o ¿cuál es la duración promedio de las canciones interpretadas por Carlos Vives? Sin embargo, se observan inconvenientes que se pueden desprender de esta forma de organizar los datos.

¿Qué sucede si debe insertarse la canción «*Fidelina*» interpretada por «*Carlos Vives*» en la tabla de canciones? En este primer escenario se estarían repitiendo nuevamente los valores del nombre, el tipo y el año de lanzamiento del intérprete. ¡Redundancia! Además, si hay una equivocación al ingresar el valor correcto en alguna de estas tres columnas —por ejemplo, el tipo de intérprete—, la tabla quedará en un estado de inconsistencia porque el mismo intérprete tendría dos tipos.

Asimismo, ¿qué sucede si debe cambiarse el año de lanzamiento a la vida artística del intérprete «*Carlos Vives*» de «*1986*» a «*1990*»? Tendrán que modificarse todas las filas que hacen referencia a Carlos Vives, en este caso tres. Si alguna fila no se modifica, existirán valores diferentes para un mismo hecho. ¡Inconsistencia!

Finalmente, ¿cómo debe procederse si se requiere almacenar datos de intérpretes que no tienen canciones registradas? Para hacer esto sería necesario insertar el valor `NULL` en todas las columnas que hacen referencia a las canciones, como el identificador, el título, la duración y el género. ¡Desperdicio de espacio! Sin embargo, si el identificador de la canción es la llave principal, entonces no puede realizarse la inserción requerida y no podrían insertarse intérpretes a menos que también se inserten canciones.

La normalización utiliza una serie de pruebas para identificar y corregir la ocurrencia de situaciones como las anteriores. El proceso permite evaluar una tabla respecto de una serie de reglas para comprobar si se está haciendo un buen manejo de la redundancia, de manera que se minimice el riesgo de estar o llegar a estados de inconsistencia. Luego, cuando las reglas no se cumplen, las tablas con anomalías deben ser descompuestas en tablas que sí sigan las formas normales.

El proceso de normalización está presente en las bases de datos relacionales desde que Codd presentó el modelo relacional en el artículo *«A relational model of data for large shared data Banks»*, donde introdujo el concepto de tablas normalizadas para referirse a aquellas que no tienen grupos repetidos. En este sentido, cabe precisar que cuando se habla de grupos, se refiere a valores de atributos que aparecen siempre juntos en las filas de la tabla. Por ejemplo, en la Tabla 10.1 los valores *«Carlos Vives»* para la columna `nombre`, el valor *«Solista»* para la columna `tipo de intérprete` y el valor *«1986»* para el `año de lanzamiento` representan un grupo: siempre aparecen juntos.

Situaciones como la anterior deben evitarse, y para ello Codd propuso inicialmente tres formas normales: la primera forma normal (1FN), la segunda forma (2FN) y la tercera forma normal (3FN). Luego se introdujo una versión más restrictiva de la tercera forma normal, denominada forma normal de Boyce-Codd (FNBC). Posteriormente, se añadieron las dos últimas formas normales: la cuarta forma normal (4FN) y la quinta forma normal (5FN). Debido a que estas tres últimas formas normales se aplican en situaciones poco comunes y que un diseño en 3FN cumple con los requisitos más generales para el manejo de la redundancia, el alcance del libro será hasta la 3FN.

A continuación, se mostrará cómo la normalización puede utilizarse como una técnica de validación para comprobar la estructura de las tablas. Para esto, se presentarán cada una de las formas normales, explicando en qué consisten, en qué escenario se deben aplicar y de qué forma.

10.2. Primera forma normal

Para abordar la 1FN se partirá de un análisis de la Tabla 10.2 para identificar si hay algún problema de redundancia o pérdida de integridad. Dicha tabla tiene por objetivo registrar las calificaciones que han realizado los usuarios a las diferentes canciones, para lo cual tiene asociadas las columnas del identificador de la canción (`idCanción`) y el identificador del usuario (`idUsuario`), el título de la canción (`título`), el identificador del intérprete (`idIntérprete`) y el nombre del intérprete (`intérprete`) de la canción, la calificación (`calificación`) que el

usuario le otorga a la canción, y el nombre y el apellido (`usuario`) y dirección(es) de correo electrónico (`correos`) del usuario que realiza la calificación.

De forma sucinta, se puede representar la tabla de la siguiente forma: `Calificaciones` (`idCanción`, `idUsuario`, `título`, `idIntérprete`, `intérprete`, `calificación`, `usuario`, `correos`). La llave principal de la tabla es compuesta ya que es la combinación del identificador de la canción con el identificador del usuario. De este modo se impone la restricción de que un usuario puede calificar diferentes canciones, pero no la misma canción dos veces.

Tabla 10.2. Tabla `Calificaciones` desnormalizada

Calificaciones							
idcanción	**idUsuario**	**título**	**idIntérprete**	**intérprete**	**calificación**	**usuario**	**correos**
14	6	Ginza	7	J. Balvin	3	Marie Curie	marie.curie@gmail.com madame@gmail.com
3	8	Una aventura	2	Niche	4	Piedad Bonett	bonett@gmail.com piedad@gmail.com
3	6	Una aventura	2	Niche	2	Marie Curie	marie.curie@gmail.com madame@gmai.com
7	8	Bolero falaz	4	Aterciopelados	5	Piedad Bonett	bonett@gmail.com piedad@gmail.com
6	5	Una aventura	2	Niche	2	Baruch Spinoza	spinoza@gmail.com
14	5	Ginza	7	J. Balvin	4	Baruch Spinoza	spinoza@gmail.com
8	6	Bolero falaz	4	Aterciopelados	4	Marie Curie	marie.curie@gmail.com madame@gmai.com
7	6	Bolero falaz	4	Aterciopelados	3	Marie Curie	marie.curie@gmail.com madame@gmai.com
14	8	Ginza	7	J. Balvin	1	Piedad Bonett	bonett@gmail.com piedad@gmail.com

En la Tabla 10.2 se empezará por verificar si cumple con la 1FN, que plantea que el dominio de todas las columnas debe ser «atómico», es decir, son unidades indivisibles. Recuerde que cuando se modeló la entidad «Usuario» se definió que el atributo `correos` podía aceptar múltiples valores o, lo que es lo mismo, que un usuario puede tener varios correos electrónicos. También se determinó que el nombre del usuario era un atributo compuesto por el primer nombre y el primer apellido. Si no se manejan con cautela estos tipos de atributos, pueden llevar a incumplir la 1FN.

En la tabla se observa que se está utilizando una misma columna (`usuario`) para representar la combinación del primer nombre y el primer apellido. Por ejemplo, *«Marie»* es el primer nombre, y *«Curie»*, el primer apellido de un usuario. También es evidente que la columna `correos` permite almacenar diferentes direcciones de correo electrónico. Continuando con el mismo usuario, *«Marie Curie»*, se muestran dos correos electrónicos registrados: *marie.curie@gmail.com* y *madame@gmail.com*.

Modelar los datos de esta forma impone que muchas tareas que se deben manejar desde el modelo de datos tengan que ser resueltas utilizando algún programa informático. Suponga, por ejemplo, que se desea enviar un correo electrónico a todos los usuarios con la siguiente estructura: «Señor(a) <primer apellido>». Para hacer esto posible, se tendría que

extraer el apellido de cada usuario, lo que implicaría un procesamiento del texto y puede desencadenar en inconvenientes, sobre todo si se considera que hay apellidos tanto compuestos como simples. En cambio, al almacenar el valor del primer nombre y el del primer apellido en columnas diferentes, la combinación de ellos es una tarea simple que se puede conseguir con una función como `CONCAT`.

De forma similar ocurre con el correo electrónico. Si por ejemplo se desea modificar uno de los correos electrónicos del usuario, se tiene que acceder a toda la cadena de texto, procesarla y extraer una de las direcciones de correo. En contraste, si se maneja de forma «atómica», en donde en una columna solo se pueda almacenar una dirección de correo electrónico, la tarea resulta muy sencilla.

Aunque los problemas planteados anteriormente son similares, las soluciones que se emplean para conseguir la «atomicidad» son diferentes. Para los atributos compuestos como el nombre se procede a crear una nueva columna por cada uno de los componentes del atributo compuesto. Así, para el caso de usuario, se asigna una columna al primer nombre y otra al primer apellido.

En el caso de las columnas que toman múltiples valores se procede de forma diferente: se crea una nueva tabla compuesta por una llave principal que identifique a quién pertenecen cada uno de los valores de la columna de múltiples valores y una columna que permita almacenar cada uno de los elementos de la columna con múltiples valores, en este caso cada uno de los correos electrónicos. Es así como se tendría una nueva tabla compuesta por columnas que contengan el correo electrónico y el identificador del usuario.

En este sentido, para resolver las anomalías detectadas en la Tabla 10.2 se requiere distribuir los datos en dos tablas: la Tabla 10.3 y la Tabla 10.4. Con el procedimiento realizado hasta este punto, se ha cumplido con la 1FN, es decir, cada intersección fila-columna contiene un valor único del dominio aplicable.

Tabla 10.3. Tabla `Calificaciones` en primera forma normal

Calificaciones							
idcanción	**idUsuario**	**título**	**idIntérprete**	**intérprete**	**calificación**	**primerNombre**	**primerApellido**
14	6	Ginza	7	J. Balvin	3	Marie	Curie
3	8	Una aventura	2	Niche	4	Piedad	Bonett
3	6	Una aventura	2	Niche	2	Marie	Curie
7	8	Bolero falaz	4	Aterciopelados	5	Piedad	Bonett
6	5	Una aventura	2	Niche	2	Baruch	Spinoza
14	5	Ginza	7	J. Balvin	4	Baruch	Spinoza
8	6	Bolero falaz	4	Aterciopelados	4	Marie	Curie
7	6	Bolero falaz	4	Aterciopelados	3	Marie	Curie
14	8	Ginza	7	J. Balvin	1	Piedad	Bonett

Tabla 10.4. Tabla `Correos` en primera forma normal

Correos	
idUsuario	**correo**
6	marie.curie@gmail.com
6	madame@gmail.com
8	bonett@gmail.com
8	piedad@gmail.com
5	spinoza@gmail.com

10.3. Segunda forma normal

Ahora se puede evaluar si el conjunto de tablas resultantes se encuentra en 2FN. Esta regla plantea que una tabla debe estar previamente en 1FN y que todos los atributos que no son clave principal deben depender de toda la clave principal, no de una parte de ella. Esta forma normal, a diferencia de la primera, tiene su sustento en la dependencia funcional entre columnas de una tabla, por lo que es pertinente explicar brevemente en qué consiste dicho concepto.

Para dos columnas A y B de una tabla Z, se dice que B es funcionalmente dependiente de A (A $\rightarrow$ B) si a cada valor de la columna A le corresponde siempre el mismo valor de la columna B. En otros términos, se puede decir que A determina funcionalmente a B o que con solo saber el valor de A se obtiene el valor de B.

Por ejemplo, en la Tabla 10.3 se puede afirmar que el valor de la columna `idCanción` determina funcionalmente el valor de la columna `título`. Así, el identificador de canción número «14» siempre tendrá el valor de «Ginza» en el título y, por tanto, `idCanción` $\rightarrow$ `título`. También se aprecia que el valor de la columna `idUsuario` determina funcionalmente el valor de las columnas `primerNombre` y `primerApellido`. Finalmente, `idIntérprete` determina funcionalmente el valor de la columna `intérprete`.

Es importante tener claro que la dependencia funcional se debe evaluar teniendo presentes todos los posibles valores que puede tomar una columna, no solo los que tenga en un momento específico. Por ejemplo, analizando la misma tabla `Calificaciones`, se puede afirmar que la columna `primerNombre` determina funcionalmente el `primerApellido` y el `idUsuario`, pero esto sería erróneo porque, pese a que hasta este momento solo existe un usuario con primer nombre «Marie» que siempre tendrá por primer apellido «Curie» y por identificador del usuario «6», el sentido de la columna indica que puede haber más de un usuario con el nombre «Marie». Por ende, no es correcto afirmar que la columna `primerNombre` determine funcionalmente al identificador del usuario y al primer apellido. En otras palabras, es fundamental que se comprenda bien el significado de los atributos y sus relaciones antes de iniciar el proceso de normalización.

Teniendo claro en qué consiste la dependencia funcional, ahora se evaluará la tabla `Calificaciones` (`idCanción`, `idUsuario`, `título`, `idIntérprete`, `intérprete`, `calificación`, `primerNombre`, `segundoNombre`) contra la 2FN. Como se puede observar, dicha tabla tiene

una llave principal compuesta, cuyas partes son el identificador de la canción y el identificador del usuario. Recuerde que la 2FN plantea que todas las columnas diferentes a las que hacen parte de la llave de principal deben depender funcionalmente de toda la llave, en este caso de ambas columnas, no solo de una de ellas. A continuación, se verificará si esto se cumple.

El título de la canción, el identificador del intérprete y el nombre de este dependen funcionalmente solo del identificador de la canción, no de ambas columnas. Es decir, solo con tener el valor de un identificador de la canción se puede saber cuáles son su título y su intérprete. Por lo tanto, no cumple la 2FN. La columna de calificación, en cambio, sí depende funcionalmente de ambas columnas puesto que solo tiene sentido cuando un usuario califica una canción en específico. El primer nombre y el segundo del usuario tienen una dependencia funcional del identificador de este, o sea, solo de una parte de la llave, por lo que no cumplen la 2FN.

Para solucionar estos inconvenientes, se deben ubicar en una nueva tabla las columnas que dependen funcionalmente de solo una parte de la llave principal junto a la parte de la llave de la cual dependen o que las determina funcionalmente. Puesto en otras palabras, se crearán dos nuevas tablas: una `Canciones`, que contendrá el identificador de la canción junto con el título, el identificador del intérprete y el nombre del intérprete, y otra `Usuarios`, con el identificador del usuario junto con el primer nombre y el primer apellido. En resumen, para cumplir los requisitos de 2FN las dos tablas deben convertirse en las siguientes cuatro tablas:

`Canciones` (`idCanción`, `título`, `idIntérprete`, `intérprete`): Tabla 10.5.
`Usuarios` (`idUsuario`, `primerNombre`, `primerApellido`): Tabla 10.6.
`Correos` (`idUsuario`, `correo`): Tabla 10.7.
`Calificaciones` (`idCanción`, `idUsuario`, `calificación`): Tabla 10.8.

Tabla 10.5. Tabla `Canciones` en segunda forma normal

Canciones			
idCanción	**título**	**idIntérprete**	**intérprete**
14	Ginza	7	J. Balvin
3	Una aventura	2	Niche
7	Bolero falaz	4	Aterciopelados
6	Una aventura	2	Niche
8	Bolero falaz	4	Aterciopelados

Tabla 10.6. Tabla `Usuarios` en segunda forma normal

Usuarios		
idUsuario	**PrimerNombre**	**SegundoNombre**
6	Marie	Curie
8	Piedad	Bonett
5	Baruch	Spinoza

Tabla 10.7. Tabla `Correos` en segunda forma normal

Correos	
idUsuario	**correo**
6	marie.curie@gmail.com
6	madame@gmail.com
8	bonett@gmail.com
8	piedad@gmail.com
5	spinoza@gmail.com

Tabla 10.8. Tabla `Calificaciones` en segunda forma normal

Calificaciones		
idcanción	**idUsuario**	**calificación**
14	6	3
3	8	4
3	6	2
7	8	5
6	5	2
14	5	4
8	6	4
7	6	3
14	8	1

La nueva tabla `Calificaciones` ahora solo tiene tres columnas, que son la llave principal compuesta y la calificación de la canción. La tabla `Canciones`, entretanto, cuenta con una llave principal simple, que es el identificador de la canción junto a las columnas que son determinadas funcionalmente por esta. De manera análoga sucede con la tabla `Usuarios`, y la tabla `Correos` se mantiene igual. Todas estas tablas están ahora tanto en 1FN como en 2FN.

10.4. Tercera forma normal

La normalización realizada hasta este punto para dejar las tablas en 1FN y 2FN permite afirmar que todos los dominios de las columnas de las tablas son «atómicos» y que no existen dependencias parciales con respecto a las columnas que conforman la llave principal de las tablas. Para dicho fin se generó una nueva tabla que incluye los atributos parcialmente dependientes de dicha llave, junto con una copia de su determinante. Aunque este procedimiento dio como resultado tablas que tienen menos redundancia que las de 1FN, aún pueden sufrir efectos secundarios en el momento de realizar las tareas de actualización. Por ejemplo, si se quiere actualizar el nombre de un intérprete como «J. Balvin» (con identificador «7»), se deberá hacer lo mismo con tres filas en la tabla `Canciones`.

En este contexto es útil la 3FN, que indica que no deben existir dependencias transitivas de la llave principal. En términos más claros, ningún atributo que no sea parte de la llave principal puede depender funcionalmente de otro atributo que tampoco conforme dicha llave.

Analice las dependencias funcionales de la tabla `Canciones`: el título de la canción, el identificador del intérprete y el nombre del intérprete (transitivamente) dependen funcionalmente del identificador de la canción (la llave principal). Sin embargo, el nombre del intérprete también está determinado funcionalmente por el identificador del intérprete (no es llave principal).

En un escenario como el anterior es cuando se habla de dependencia transitiva, porque $A \rightarrow B$ y $B \rightarrow C$, y por tanto, $A \rightarrow C$. Puesto en otras palabras, quien determina realmente en un primer grado el nombre del intérprete es el identificador de este, no el de la canción. No obstante, como el identificador del intérprete depende funcionalmente del de la canción, entonces este último determina transitivamente el nombre del intérprete. Por consiguiente, existe un atributo que no hace parte de la llave primaria (el nombre del intérprete) dependiendo funcionalmente de otro atributo que tampoco hace parte de la llave primaria (el identificador del intérprete).

La solución que se propone es definir una nueva tabla que incluya tanto los atributos que dependen transitivamente de la llave primaria como el determinante que no es llave primaria de la tabla. El resultado luego de hacer esta transformación genera una nueva versión de la tabla `Canciones` (Tabla 10.9) y una nueva tabla denominada `Intérpretes` (Tabla 10.10), donde están incluidos el atributo nombre, que hace referencia al nombre del intérprete, y el identificador del intérprete que es el que determina dicho nombre.

En este punto se puede afirmar que las tablas ya se encuentran en 3FN. Por lo general, las tablas en dicha instancia están lo suficientemente bien estructuradas para evitar los problemas asociados con la redundancia de datos. Sin embargo, no se debe olvidar que existen otras formas normales (la FNBC, la 4FN y la 5FN) que no se desarrollarán en este texto puesto que su utilidad se da en escenarios muy raros y, algunas veces, se debe balancear la practicidad de su implementación con los beneficios reales.

Tabla 10.9. Tabla `Canciones` en tercera forma normal

Canciones		
idCanción	**título**	**idIntérprete**
14	Ginza	7
3	Una aventura	2
7	Bolero falaz	4
6	Una aventura	2
8	Bolero falaz	4

Tabla 10.10. Tabla `Intérpretes` en tercera forma normal

Intérpretes	
idIntérprete	**nombre**
7	J. Balvin
2	Niche
4	Aterciopelados

10.5. Aprendizajes más importantes del capítulo 10

- La normalización es una técnica utilizada dentro de la comunidad de diseñadores de bases de datos para verificar si las tablas tienen un manejo aceptable de la redundancia.
- Las formas normales son seis: la 1FN, 2FN y 3FN, que fueron propuestas juntas; la FNBC, que es una versión más estricta de la 3FN, y la 4FN y la 5FN, que se plantearon posteriormente.
- La 1FN se centra en que los valores aceptados en las columnas tengan un dominio atómico. Se puede decir que permite manejar atributos compuestos y multivaluados de forma eficiente.
- La 2FN se centra en que en una tabla haya dependencia funcional total de la llave primaria por parte de todos los atributos que no hacen parte de esta.
- La 3FN pone su atención en las dependencias transitivas, es decir, que no haya dependencia funcional entre las columnas que no son parte de la llave principal.
- La 2FN y la 3FN tienen su sustento en el concepto de dependencias funcionales. No ocurre lo mismo con la 1FN.

10.6. Actividades de aplicación para evidenciar lo aprendido

1. Compruebe si el modelo relacional creado por usted en la actividad 4 del capítulo 9 está en 3FN. Si no lo está, llévelo a dicha norma.
2. Modifique la tabla `Reproducciones` del modelo presentado en la Figura 9.14 para que viole las tres formas normales: 1FN, 2FN y 3FN.
3. Lleve la siguiente tabla, que representa las capacitaciones presenciales ofrecidas por una institución de enseñanza, a la 3FN. Cada curso es de una única sesión y se ofrece varias veces durante el año, pero no el mismo día:

curso	fecha	nombreCurso	Salón	capacidadSalón	cuposCurso	profesores
SQL01	11/07/2021	**Fundamentos de SQL**	BN-301	40	39	Juan; Marta
BI01	09/11/2021	**Inteligencia de negocios**	MC-101	31	28	Luisa; Marta; Pedro
DM01	28/08/2021	*Data management*	MC-101	31	28	Mark
BI01	29/11/2021	**Inteligencia de negocios**	MC-101	40	33	Ernesto
DM01	28/07/2021	*Data management*	BS-202	20	18	Mark
TOGAF01	28/10/2021	**Arquitectura empresarial**	BN-301	40	25	Alex
SQL01	11/10/2021	**Fundamentos de SQL**	BS-202	20	20	Marta; José
DM01	28/10/2021	*Data management*	MC-101	31	30	Mark
BI01	12/12/2021	**Inteligencia de negocios**	BN-301	40	39	Piedad; Victoria; Juan

4. Consulte en qué consiste la auditoría de bases de datos e indique qué aspectos de ella considera importante tener presentes en el momento de diseñar una base de datos.

Capítulo 11

Implementación del diseño lógico

Resultados de aprendizaje

– *Crea las tablas de una base de datos usando los comandos del lenguaje DDL y cumpliendo las especificaciones del diseño lógico.*
– *Define restricciones sobre las tablas de una base de datos de acuerdo con las especificaciones y necesidades expresadas en el diseño lógico.*
– *Manipula los datos de las tablas de la base datos mediante operaciones de inserción, actualización y eliminación.*

Inicialmente, se debe tener presente que las tablas son las estructuras básicas de almacenamiento en el modelo relacional. Hasta este capítulo se ha mostrado cómo pueden ser consultadas, cómo pueden ser diseñadas y, en el capítulo anterior, cómo aplicar un grupo de reglas, denominadas formas normales, para comprobar que se está manejando bien la redundancia a nivel lógico.

Como resultado del diseño lógico se obtiene un conjunto de tablas que no existe físicamente en una base de datos. En ese sentido, es importante tener en cuenta que sin la existencia física de las tablas no puede ni almacenarse datos ni mucho menos consultarse los datos. Dicho de otra manera, el diseño lógico especifica la estructura de las tablas que se deben crear, pero luego es necesario crear estas en un DBMS.

La implementación del diseño lógico en un DBMS es bastante sencilla y, hasta cierto grado, mecánica. Esto se logra escribiendo sentencias en SQL utilizando el sublenguaje DDL *(Data Definition Language)*, con el cual se le indica al DBMS el nombre de las tablas, las columnas que las conforman y, por supuesto, las relaciones que hay entre las tablas. La gestión de estas últimas, en términos generales, es simple; sin embargo, existen algunos detalles que, si

no son presentados de forma correcta, pueden generar problemas y llevar a inconsistencias en los datos, sobreesfuerzo en la gestión de estas estructuras y conflictos de rendimiento.

Con el DDL pueden crearse, modificarse y eliminarse las tablas de una base de datos, pero estas no son el único objeto que puede gestionarse mediante dicho sublenguaje. Si bien las tablas son los más importantes, existen otros tipos de objetos como los índices, las vistas, las funciones o los procedimientos almacenados, que también pueden ser gestionados con este sublenguaje. Así, en el presente capítulo se abordará en concreto la implementación de índices, los cuales permiten mejorar el rendimiento del DBMS en el momento de ejecutar sentencias del SQL para la manipulación de datos.

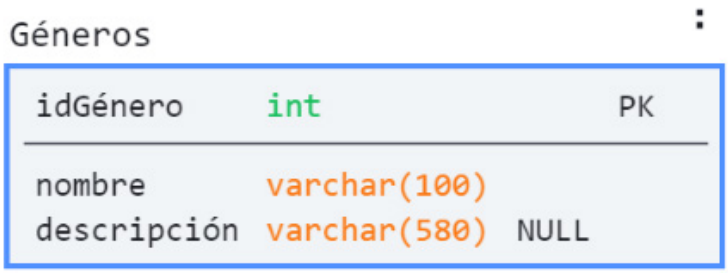
Luego de la creación de la tabla, es necesario poder insertar datos en ellas, modificarlos y eliminarlos. Por consiguiente, en este capítulo, además de la creación de las tablas, también se presentará la forma de trabajar con los datos almacenados en ellas, pues hasta el momento solamente se han realizado consultas que obtienen datos. El lenguaje de manipulación de datos (DML) va más allá de lo que se ha visto hasta el momento con la sentencia `SELECT`; existen otras sentencias para realizar la inserción (`INSERT`), la actualización (`UPDATE`) y la eliminación (`DELETE`) de datos.

11.1. Creación, eliminación y modificación de tablas

La estructura de una tabla está definida principalmente por las columnas que la conforman, por lo que la primera acción por realizar es la creación de una tabla con solo unas cuantas columnas. Para crear cualquier objeto utilizando el sublenguaje DDL se usa la palabra `CREATE` (crear, en inglés) seguida de la estructura que se quiere generar. Como en este caso es una tabla, se utilizará la palabra `TABLE` (tabla, en inglés). Luego debe indicarse el nombre del objeto; para este caso, el nombre de la tabla.

Como se dijo, la estructura de una tabla está definida por las columnas que contiene, por lo que el siguiente paso en el proceso de creación de una tabla es indicar cuáles son las columnas que la conforman. A continuación, con la creación de la tabla `Géneros`, mostrada en la Figura 11.1, se busca ejemplificar lo dicho hasta este punto.

Figura 11.1. Tabla `Géneros`



Así las cosas, para crear esta tabla se usó la palabra `CREATE`, seguida por la palabra `TABLE` y el nombre de la tabla; en este caso, `Géneros`. Luego se especificaron cada uno de

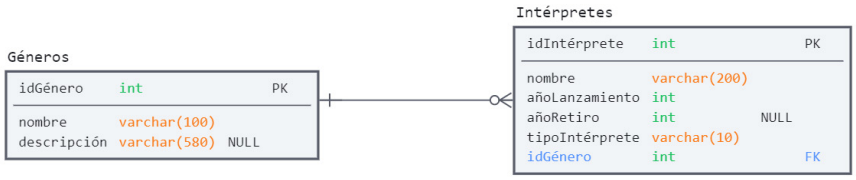
los nombres de las columnas que conforman la tabla: `idGénero`, `nombre` y `descripción`. Cada columna debe tener un tipo de datos, de manera que, según la figura, `idGénero` es de tipo `INT` (entero) y el `nombre` y la `descripción` son de tipo VARCHAR (cadena de texto). Lo descrito se aprecia en el *Script* 11.1.

***Script* 11.1**

```
CREATE TABLE Géneros
(idGénero INT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripción VARCHAR(500) NULL
);
```

En el *Script* 11.1 se presentan otros detalles además de los ya mencionados. Con las palabras `PRIMARY KEY` se indica cuál es la llave primaria de la tabla; en este caso, la columna `idGénero` (recuerde que una tabla solo puede tener una llave principal). También se establecen las columnas que permiten almacenar valores nulos, utilizando la palabra `NULL` luego de la especificación del tipo. En las columnas que deben almacenar algún valor diferente a nulo, la especificación se hace con las palabras reservadas `NOT NULL`.

Figura 11.2. Tablas `Intérpretes` y `Géneros`



***Script* 11.2**

```
CREATE TABLE Intérpretes
(idIntérprete INT PRIMARY KEY,
  nombre VARCHAR(200) NOT NULL,
  añoLanzamiento INT NOT NULL,
  añoRetiro INT,
  tipoIntérprete VARCHAR(10) NOT NULL,
  idGénero INT NOT NULL
);
```

La creación de la tabla `Géneros` representa un escenario bastante sencillo debido a que no tiene ninguna llave foránea. Ahora bien, en la Figura 11.2 se muestra la relación entre las tablas `Intérpretes` y `Géneros`, donde la primera hace uso de una llave foránea. En primera instancia puede procederse con la creación de la tabla `Intérpretes` de forma similar a como se hizo con la tabla `Géneros`, como se muestra en el *Script* 11.2.

Sin embargo, al crear la tabla `Intérpretes` de esta forma se ha olvidado un detalle importante: no se ha especificado la relación que existe entre las dos tablas. Puesto en otras palabras, no se le ha indicado al DBMS que la columna `idGénero` de la tabla `Intérpretes` es una llave que proviene de la columna `idGénero` de la tabla `Géneros`. ¿Qué se puede hacer ahora? Existen dos caminos: el primero es eliminar la tabla y corregir el error volviendo a crearla con todas sus columnas y especificaciones; el segundo consiste en modificar la estructura de la tabla y especificar que `idGénero` es una llave foránea.

Para proseguir con la primera alternativa, se usa la palabra reservada `DROP` (borrar, en inglés), seguida del tipo objeto que se desea borrar —en este caso una tabla (`TABLE`)— y el nombre del objeto en particular —`Intérpretes`—. La instrucción completa se muestra en el *Script* 11.3.

***Script* 11.3**

```
DROP TABLE Intérpretes;
```

***Script* 11.4**

```
CREATE TABLE Intérpretes
(idIntérprete INT PRIMARY KEY,
  nombre VARCHAR(200) NOT NULL,
  añoLanzamiento VARCHAR(200) NOT NULL,
  añoRetiro INT,
  tipoIntérprete VARCHAR(10) NOT NULL,
  idGénero INT NOT NULL,
  FOREIGN KEY(idGénero) REFERENCES Géneros(idGénero)
);
```

Una vez eliminada la tabla `Intérpretes`, se puede crear nuevamente, pero sin olvidar incluir la relación con `Géneros`. Para hacer esto, como se ve en el *Script* 11.4, además de crear la columna, se debe especificar que hay una llave foránea mediante las palabras reservadas `FOREING KEY`, seguidas por la columna que cumplirá dicha función —en este caso, `idGénero`—, acompañada de la palabra `REFERENCES` para indicar primero dónde está la columna a la cual la llave foránea hace referencia (la tabla de origen, es decir, `Géneros`) y luego la columna de esa tabla a la que se está haciendo referencia (`idGénero`).

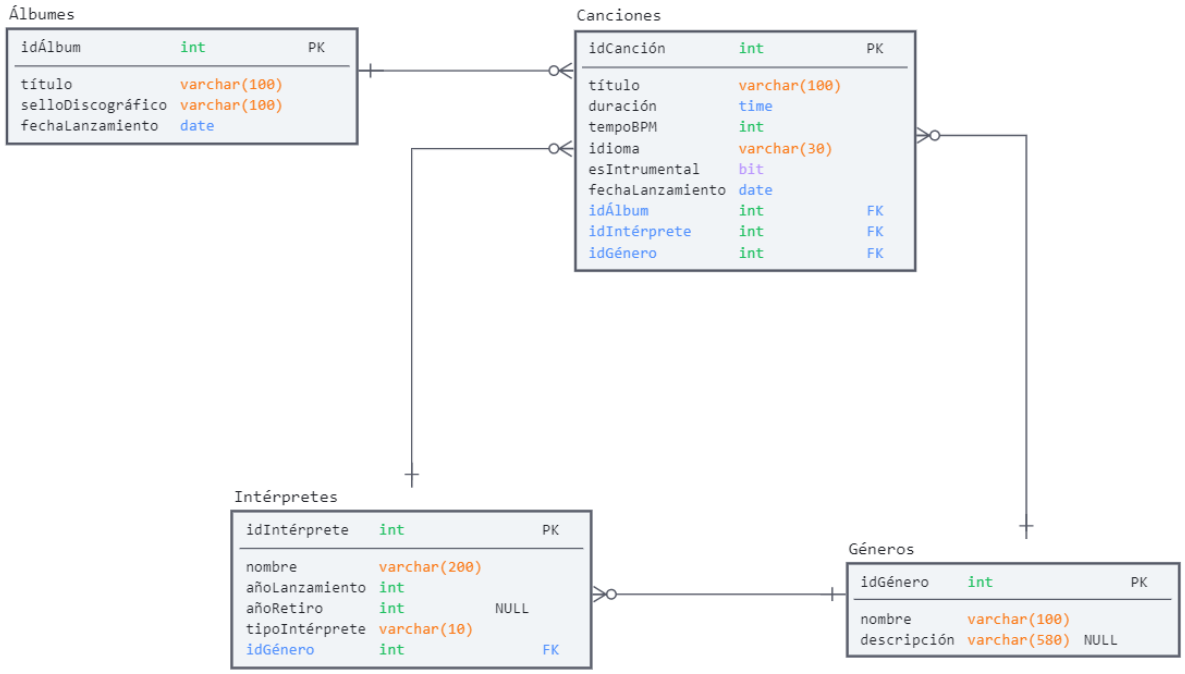
***Script* 11.5**

```
ALTER TABLE Intérpretes
ADD FOREIGN KEY (idGénero) REFERENCES Géneros(idGénero);
```

La primera alternativa, la de eliminar y crear nuevamente la tabla `Intérpretes`, aunque permite conseguir el objetivo buscado, puede resultar no plausible. Por esta razón conviene explorar la otra opción: modificar la tabla `Intérpretes`. Esto se puede hacer con las palabras reservadas `ALTER` (modificar, en inglés) y `TABLE`, como se muestra en el *Script* 11.5. En este caso lo que se hace es indicar cuál es el cambio que se quiere hacer en la tabla: `ADD` (añadir, en inglés) una llave foránea.

La cantidad de llaves foráneas que se pueden añadir a una tabla no tiene restricción y depende únicamente de lo que tenga sentido para la implementación que se esté realizando. En la Figura 11.3 se presenta un nuevo escenario.

Figura 11.3. Tablas `Canciones`, `Álbumes`, `Intérpretes` y `Géneros`



En esta oportunidad se introducen dos nuevas tablas: `Álbumes` y `Canciones`. La primera posee cuatro columnas, y ninguna de ellas es llave foránea; la segunda, por su parte, tiene diez columnas, de las cuales tres son llaves foráneas: el identificador del álbum (`idÁlbum`), que proviene de `Álbumes`; el identificador del intérprete (`idIntérprete`), que proviene de la tabla `Intérpretes`, y el identificador del género (`idGénero`), que proviene de la tabla `Géneros`. De las tres llaves foráneas, solo una hace referencia a una llave principal que aún no existe: el identificador del álbum. Si se desea crear la tabla `Canciones`, el mejor camino es generar primero la tabla `Álbumes` con todas sus columnas y tipos de datos, como se muestra en el *Script* 11.6.

***Script* 11.6**

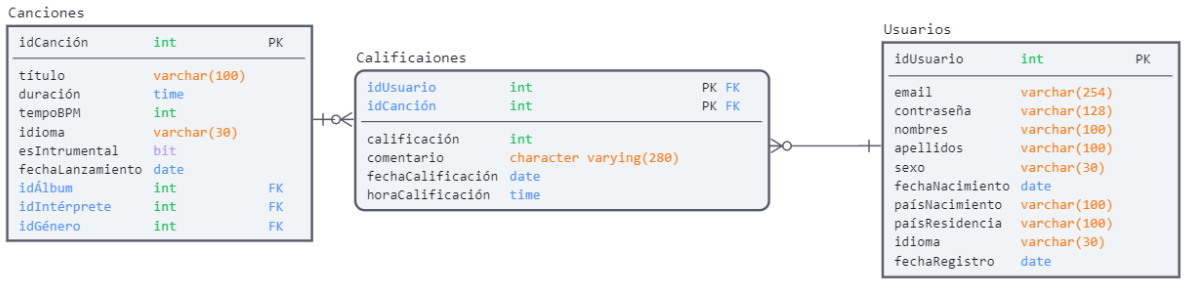
```
CREATE TABLE Álbumes (
  idAlbum INT PRIMARY KEY,
  título VARCHAR(100) NOT NULL,
  selloDiscográfico VARCHAR(100) NOT NULL,
  fechaLanzamiento DATE NOT NULL
);
```

Con la tabla `Álbumes` lista, ahora se puede crear la tabla `Canciones` sin ningún problema con las instrucciones mostradas en el *Script* 11.7. En este caso se ve que se especifican todos los elementos: columnas, tipos de datos, llave principal, llaves foráneas y si aceptan o no el descriptor `NULL`.

***Script* 11.7**

```
CREATE TABLE Canciones (
  idCanción INT NOT NULL PRIMARY KEY,
  título VARCHAR(100) NOT NULL,
  duración TIME(0) NOT NULL,
  tempoBPM INT,
  idioma VARCHAR(30) NOT NULL,
  esInstrumental BIT NOT NULL DEFAULT '0',
  fechaLanzamiento DATE NOT NULL,
  idÁlbum INT NULL,
  idIntérprete INT NOT NULL,
  idGénero INT NOT NULL,
  FOREIGN KEY (idÁlbum) REFERENCES Álbumes(idAlbum),
  FOREIGN KEY (idGénero) REFERENCES Géneros(idGénero),
  FOREIGN KEY (idIntérprete) REFERENCES Intérpretes(idIntérprete)
);
```

Figura 11.4. Tablas `Canciones`, `Calificaciones` y `Usuarios`



Una novedad que se incorporó en la definición de la tabla `Canciones` es establecer un valor por defecto para la columna `esInstrumental` (el valor *0*). Esto quiere decir que, cuando no se proporcione un valor para esta columna, automáticamente se ingresará cero. La

definición del valor por defecto se hizo con la palabra reservada `DEFAULT`, seguida del valor que se debe ingresar por defecto; en este caso, el cero. Ahora es momento de analizar la creación de la siguiente parte de la base de datos, presentada en la Figura 11.4.

De las tres tablas presentes en la Figura 11.4, solo dos son nuevas: `Calificaciones`, que se creó mediante el *Script* 11.7, y `Usuarios`. Debido a que la primera tiene una llave foránea proveniente de la tabla `Usuarios`, es necesario empezar por generar esta última. Para ese fin se utilizan las instrucciones del *Script* 11.8.

***Script* 11.8**

```
CREATE TABLE Usuarios (
  idUsuario INT NOT NULL PRIMARY KEY,
  email VARCHAR(254) UNIQUE NOT NULL,
  contraseña VARCHAR(128) NOT NULL,
  nombres VARCHAR(100) NOT NULL,
  apellidos VARCHAR(100) NOT NULL,
  sexo VARCHAR(30) CHECK(sexo IN ('M', 'F')) NOT NULL,
  fechaNacimiento DATE NOT NULL,
  paísNacimiento VARCHAR(100) NOT NULL,
  paísResidencia VARCHAR(100) NOT NULL,
  idioma VARCHAR(30) NOT NULL,
  fechaRegistro DATE NOT NULL,
  CONSTRAINT fechas_validas CHECK(fechaNacimiento<fechaRegistro)
);
```

En el *Script* 11.8 se introducen tres nuevas palabras reservadas que permiten incorporar restricciones: `UNIQUE`, `CHECK` y `CONSTRAINT`. Estas instrucciones ejercen como reglas que son comprobadas en el momento de realizar una inserción o una actualización de los datos.

Tabla 11.1. Resumen de las restricciones utilizadas

Restricción	Descripción
`PRIMARY KEY`	Identifica de forma única una fila y no permite valores repetidos.
`FOREIGN KEY`	Especifica que el valor en una columna debe coincidir con uno de los valores de la columna a la cual hace referencia la llave foránea. Esto se conoce como integridad referencial.
`NOT NULL`	Indica que la columna nunca debe aceptar el descriptor `NULL`.
`CHECK`	Permite especificar que el valor en cierta columna debe satisfacer una condición (una expresión de la cual se pueda determinar su valor de verdad).
`CONSTRAINT`	Permite especificar restricciones de forma similar a `CHECK`, pero no está restringido a una columna.
`UNIQUE`	Permite asegurar que los valores de una columna son únicos entre todas las filas de la tabla.

Anteriormente ya se había trabajado con otras restricciones: la de llave principal, la de llave foránea y la de valores nulos. Sin embargo, cuando se crean las tablas también es posible especificar, al nivel de columnas, unas reglas que apliquen a los valores insertados en una columna en particular o que relacionen más de una columna. Estas reglas se conocen

como restricciones (en inglés, CONSTRAINTS). En la Tabla 11.1 se presenta un resumen de las restricciones mencionadas.

Específicamente, las restricciones que se han añadido a la tabla Usuarios son:

- La palabra UNIQUE para especificar que la dirección de correo electrónico no se puede repetir.
- La palabra CHECK para indicar que la columna sexo solo puede tomar dos valores, bien sea «M» o «F».
- La palabra CONSTRAINT para indicar una regla que debe cumplir cada fila; en este caso, que el valor de la columna fechaNacimiento sea menor que el de la columna fechaRetiro.

***Script* 11.9**

```
ALTER TABLE Usuarios
ADD CONSTRAINT contraseña_valida CHECK (MD5(email) <> contraseña);
```

***Script* 11.10**

```
ALTER TABLE Usuarios
DROP CONSTRAINT fechas_validas;
```

Con la tabla Usuarios lista, se puede continuar con la creación de la tabla Calificaciones a través del *Script* 11.11.

***Script* 11.11**

```
CREATE TABLE Valoraciones (
  idUsuario INT NOT NULL,
  idCanción INT NOT NULL,
  valoración INT NOT NULL,
  comentario VARCHAR(280) NOT NULL,
  fechaCalificación DATE NOT NULL,
  horaCalificación TIME(0) NOT NULL,
  PRIMARY KEY (idUsuario, idCanción),
  FOREIGN KEY (idUsuario) REFERENCES Usuarios(idUsuario),
  FOREIGN KEY (idCanción) REFERENCES Canciones(idCanción)
);
```

Al igual que con los demás componentes de una tabla, las restricciones también se pueden establecer después de que se ha creado el objeto. Para esto se hace uso de las palabras reservadas ADD y CONSTRAINT. En el *Script* 11.9 se presenta, por ejemplo, cómo añadir una restricción que comprueba que el valor de la columna contraseña es diferente al de la

columna `email`. Asimismo, se ha utilizado la función `MD5` porque la contraseña ha sido guardada en este formato. De modo similar, puede eliminarse una restricción, tal como ocurre en el *Script* 11.10.

Se debe resaltar que en el *Script* 11.11 se ha creado una llave primaria compuesta utilizando la palabra `PRIMARY KEY` y proporcionando los nombres de las dos columnas que conformarán la llave; para el caso en mención, `idUsuario` e `idCanción`. Ahora bien, aunque el *Script* 11.11 se ejecuta correctamente, se cometieron algunos errores que deben corregirse. El primero de ellos fue nombrar la tabla `Valoraciones` en lugar de `Calificaciones`. El *Script* 11.12 muestra cómo hacer el ajuste.

***Script* 11.12**

```
ALTER TABLE Valoraciones
RENAME TO Calificaciones;
```

De forma análoga sucedió con la columna `valoración`. Esta nueva corrección se consigue con el *Script* 11.13.

***Script* 11.13**

```
ALTER TABLE Calificaciones
RENAME COLUMN valoración TO calificación;
```

Finalmente, se tiene que la columna `comentario` se ha dejado como obligatoria, lo que quiere decir que no recibe el descriptor `NULL` y, por ende, se debe eliminar esta restricción. Dicha eliminación se consigue con el *Script* 11.14.

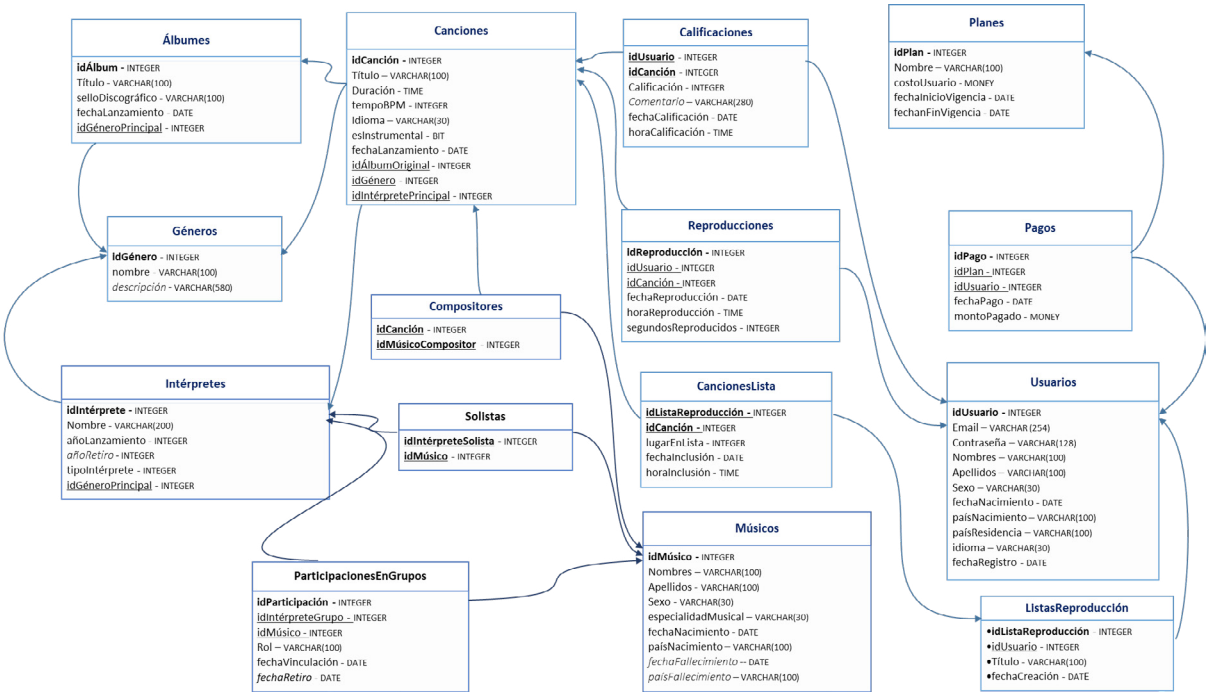
***Script* 11.14**

```
ALTER TABLE Calificaciones
ALTER COLUMN comentario DROP NOT NULL;
```

11.2. Inserción, eliminación y actualización de filas

El uso de las bases de datos relacionales no se limita a la ejecución de consultas en SQL para resolver preguntas y satisfacer necesidades. Por el contrario, mantener una base de datos completa y actualizada demanda acciones que permitan insertar nuevas filas en las tablas a medida que van sucediendo los hechos de los cuales se guarda datos; actualizar los datos existentes, bien sea por la necesidad de reflejar cambios o porque deban corregirse errores de registro, o borrar las filas que no se requieran y que no deben seguir ocupando espacio de almacenamiento.

Figura 11.5. Modelo lógico de la base de datos *«Mis Canciones»*



Al ejecutar las operaciones de inserción, eliminación y actualización es indispensable tener control sobre cuatro aspectos de la tabla a la que se está afectando: la estructura de la clave principal, la existencia de claves externas o foráneas, los tipos de datos de las columnas sobre las que se realizarán operaciones, y la posibilidad de almacenar valores nulos.

- Estructura de la clave principal: es preciso saber cuál o cuáles columnas son parte de la clave principal para cumplir la restricción de unicidad, es decir, que los valores no puedan repetirse en diferentes filas de la tabla.
- Existencia de claves externas o foráneas: debe saberse a cuál o cuáles columnas de otra tabla está haciendo referencia la columna definida como clave foránea para cumplir la restricción de integridad referencial, es decir, asegurar que los valores que se almacenan en esa columna existan previamente en la columna correspondiente de la tabla referenciada.
- Tipo de datos de las columnas: se requiere para asegurar que el tipo de dato de los valores que se van a almacenar es coherente con lo aceptado por cada columna.
- Valores nulos: debe saberse si es obligatorio que exista un valor almacenado o si es posible proporcionar el descriptor `NULL` para especificar que no se ingresará valor para esa columna.

Todos estos cuatro aspectos se contemplan en la Figura 11.5 . En este diagrama, las claves foráneas de cada tabla se representan subrayando la columna, y con una flecha se indica la tabla de la cual proviene la clave foránea. Por ejemplo, en el caso de la tabla `Canciones` se aprecia que tiene tres claves foráneas: `idÁlbumOriginal`, que se toma de la tabla `Álbumes`; `idGénero`, que procede de la tabla `Géneros`, e `idIntérpretePrincipal`, que se encuentra en la tabla `Intérpretes`. En el diagrama también se puede apreciar el tipo de dato que acepta cada columna. Tomando como referencia la misma tabla `Canciones`, se ha definido que el identificador de la canción es de tipo `INTEGER`, mientras que la fecha de lanzamiento es de tipo `DATE`, y el título de la canción, de tipo `VARCHAR(200)`. Cada tipo impone restricciones con respecto al formato en que deben ser insertados los datos y los valores máximos y mínimos aceptados. Por ejemplo, `VARCHAR(200)` indica que esa columna almacena un texto, que en el momento de ingresarlo debe estar escrito entre comillas simples (ej., 'Ojos así') y que la extensión máxima es una cadena de 200 caracteres.

Existen diversos tipos de datos, pero se puede hablar de tres grandes categorías: los que representan un número, los que representan una cadena de texto y los que representan fecha y hora. A nivel de detalle, cada DMBS nombra e implementa de manera particular cada tipo de dato. En el caso de PostgreSQL, en la documentación oficial se puede encontrar el detalle de todos los tipos soportados. En la Tabla 11.2 hay una muestra de algunos de ellos.

Tabla 11.2. Descripción de algunos tipos de datos de uso común en las bases de datos relacionales

Tipo de dato	Descripción	Ejemplo
`INTEGER`	Números enteros.	10, –50
`VARCHAR`	Cadena de caracteres o texto. Debe ingresarse entre comillas simples.	'Ojos así', 'Marie Curie'
`DATE`	Fecha en formato *«dd/mm/yyyy»*. Debe ingresarse entre comillas simples.	'08/08/1987', '23/04/2021'
`TIME`	Hora en formato *«hh:mm:ss.ms»*. Debe ingresarse entre comillas simples.	'00:00:00.00', '23:59:59.99'

Es evidente que una base de datos debe contener datos para que tenga un valor real. Por consiguiente, la primera inquietud que surge es: ¿cómo se pueden insertar filas en las tablas de una base de datos? Para realizar esta tarea hay diferentes alternativas que varían en cuanto a las facilidades que proporcionan y la cantidad de inserciones que permiten realizar.

La primera de estas alternativas y la más simple de todas es la sentencia `INSERT INTO VALUES`. La idea de esta sentencia es sencilla: para insertar datos en una tabla, se debe indicar en qué tabla se van a insertar los datos, cuál es el orden en que los datos se van a suministrar y luego suministrar los datos en ese orden. Sin embargo, especificar el orden en que se va a proveer la nueva información es opcional. Si no se hace esta claridad, se espera que los datos sean insertados en el mismo orden que tienen las columnas en la tabla. En el siguiente ejemplo se ilustra de mejor manera lo descrito.

¿Cómo agregar una fila a la tabla Géneros?

Ingresar una nueva fila a una tabla como `Géneros` resulta muy sencillo debido a que, como muestra la Figura 11.5, esta tabla tiene pocas columnas (solo tres), y ninguna de ellas representa una clave externa o foránea. El tipo de dato del identificador del género es entero, y el del nombre y la descripción son de cadenas de texto. Además, la única columna que acepta el descriptor `NULL` es `descripción`.

La clave principal es la columna `idGénero`, por lo que el valor de esta para la nueva fila no debe existir previamente en la tabla. El género que se va a ingresar tendrá el identificador *«14»*, con el nombre *«Rap»* y la descripción *«Este género surgió como un movimiento social...»*, tal como se muestra en el *Script* 11.15.

***Script* 11.15**

```
INSERT INTO Géneros (idGénero, nombre, descripción)
VALUES(14, 'Rap', 'Este género surgió como un movimiento social...');
```

En el *Script* 11.15 se aprecia que el identificador *«14»*, como es de tipo entero, se suministra sin necesidad de hacer uso de comillas, mientras que para el nombre y para la descripción sí se emplean dichos signos. Como se indicó previamente, el orden en que los valores de cada columna son suministrados puede ser modificado, siempre que se indique cómo, según se puede observar en el *Script* 11.16. Allí se aprecia que primero está la descripción, luego el nombre y, finalmente, el identificador.

***Script* 11.16**

```
INSERT INTO Géneros (descripción, nombre, idGénero)
VALUES('Este género surgió como un movimiento social de expresión de clases
oprimidas...', 'Rap', 14);
```

De igual manera, si los valores se van a suministrar en el mismo orden en el que aparecen en la definición de las tablas, no es necesario indicar el nombre de las columnas. En el *Script* 11.17 hay una muestra de ello. En este último también se aprecia cómo se utilizó el descriptor `NULL` para indicar que de momento no se va a añadir descripción del género.

***Script* 11.17**

```
INSERT INTO Géneros
VALUES(11, 'Rap', NULL);
```

Es necesario saber cuáles valores se han insertado en la llave principal de la tabla para no repetirlos; si esto ocurre, se obtendrá entonces un mensaje de error que indica que se está violando la restricción de unicidad de la llave principal. Existe una alternativa que se puede usar al crear una tabla y así no tener que ocuparse de garantizar la unicidad: consiste en indicar que el tipo de datos (realmente, un pseudotipo) va a ser autoincremental (`SERIAL`). De esta forma, el DBMS se encargará de que en cada inserción el valor sea igual al último valor que se intentó insertar más uno. En este caso se habla de un «pseudotipo» porque el tipo real es entero, solo que tiene el comportamiento descrito. En el *Script* 11.18 se muestra cómo definir la columna `idGénero` como autoincremental.

***Script* 11.18**

```
CREATE TABLE Géneros
(idGénero SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripción VARCHAR(500) NULL
);
```

Ahora se puede omitir el identificador del género en el momento de insertar una fila. También se puede indicar que no se va a proporcionar la descripción debido a que este acepta el descriptor `NULL`, tal cual se indica en el *Script* 11.19.

***Script* 11.19**

```
INSERT INTO Géneros (nombre)
VALUES('Rap');
```

¿Cómo agregar una fila en la tabla Canciones?

La tabla `Canciones`, a diferencia de la tabla `Géneros`, tiene claves foráneas, es decir, toma valores que deben estar presentes en otras tablas antes de poder existir en `Canciones`. Este es el caso de las columnas `idIntérpretePrincipal`, que hace referencia a la columna `idIntérprete` de la tabla `Intérpretes`; `idGénero`, la cual remite a la columna `idGénero` de la tabla `Géneros`, e `idÁlbumOriginal`, que invoca a la columna `idAlbum` de la tabla `Álbumes`. Las dos primeras columnas no aceptan el descriptor `NULL`, por lo que el valor que se suministre en ellas al insertar una canción debe existir en la tabla a las cuales hacen referencia. En cambio, con `idÁlbumOriginal`, que sí acepta `NULL`, se tienen dos opciones: suministrar un valor que exista en la tabla a la cual hace referencia o ingresar dicho descriptor.

Aclarado lo anterior, ahora se puede concentrar la atención en los tipos de datos. Al observar detenidamente la Figura 11.5 se aprecia que se utilizan cinco tipos de datos: `INTEGER` y `VARCHAR`, que ya se explicaron, y `BIT`, `DATE` y `TIME`. El tipo `BIT` indica que solo se pueden tomar dos valores: *«0»* o *«1»*; `DATE` quiere decir que solo acepta fechas en el formato

preestablecido según la configuración regional, es decir, «*dd/mm/yyyy*», «*mm/dd/yyyy*», etc., y por último TIME permite ingresar el tiempo en formato horas, minutos, segundos y milisegundos. Recuerde además que el valor debe ir entre comillas simples. En el *Script* 11.20 se especifica cómo insertar una nueva canción.

***Script* 11.20**

```
INSERT INTO Canciones
VALUES(21,'Fidelina','00:04:23', NULL, 'Español','0','25/07/1995', 1, 4, 1 )
```

Al especificar los valores por insertar en la tabla también se pueden utilizar subconsultas autónomas que devuelvan el dato que se desea insertar. Por ejemplo, en el *Script* 11.21 se ha reemplazado el «*1*», que corresponde al identificador del álbum de nombre «*La tierra del olvido*», por una subconsulta que devuelve el identificador de este álbum. Lo mismo se realiza con los valores que corresponden al identificador del género y del intérprete.

***Script* 11.21**

```
INSERT INTO Canciones
VALUES(21,'Fidelina', '00:04:23', NULL, 'Español', '0', '25/07/1995', (SELECT idalbum
FROM Álbumes
WHERE título = 'La tierra del olvido'), (SELECT idGénero
FROM Géneros
WHERE nombre = 'Vallenato'), (SELECT idIntérprete
FROM Intérpretes
WHERE nombre = 'Carlos Vives'))
```

¿Cómo se pueden ingresar dos reproducciones que han realizado dos usuarios de la canción «*Fidelina*»?

Lo primero, al igual que en los casos anteriores, es identificar las restricciones que impone la tabla en la que se desea insertar información; en este caso, la tabla Reproducciones. La clave primaria es el identificador de la reproducción (idReproducción), y luego se encuentran dos claves foráneas: una que hace referencia a la canción reproducida, y otra que remite al usuario que la reprodujo. Finalmente, los tipos de datos que se manejan son INTEGER, TIME y DATE.

Una forma inicial de proceder es utilizar dos sentencias INSERT, una por cada reproducción. En el *Script* 11.22, donde se aprecia cómo insertar la primera reproducción, se utiliza el identificador «*498*» y se recuperan tanto el identificador del usuario como el de la canción utilizando subconsultas autónomas. Luego, para ingresar la fecha y la hora de reproducción

se usa la función NOW(), que devuelve una marca de tiempo (TIMESTAMP) que representa el día y la hora actual. Sin embargo, dado que la fecha de reproducción es de tipo DATE, y la hora de tipo TIME, se convierte con la función CAST, lo que devuelve la función NOW() a DATE para insertar la fecha y a TIME para insertar la hora.

***Script* 11.22**

```
INSERT INTO Reproducciones
VALUES(498, (SELECT idUsuario
FROM Usuarios
WHERE email = 'panteismo@gmail.com'), (SELECT idCanción
FROM Canciones
WHERE título = 'Fidelina'), CAST(NOW() AS DATE), CAST(NOW() AS TIME), 189),
```

Luego se continuaría con la inserción de la segunda reproducción usando una nueva sentencia INSERT. No obstante, SQL permite realizar más de una inserción separando cada una de las filas que se ingresarán por una coma, como se muestra en el *Script* 11.23.

***Script* 11.23**

```
INSERT INTO Reproducciones
VALUES(498, (SELECT idUsuario
FROM Usuarios
WHERE email = 'panteismo@gmail.com'), (SELECT idCanción
FROM Canciones
WHERE título = 'Fidelina'), CAST (NOW() AS DATE), CAST (NOW() AS TIME), 189),
(499, (SELECT idUsuario
FROM Usuarios
WHERE email = 'piedad.bonnett@gmail.com'), (SELECT idCanción
FROM Canciones
WHERE título = 'Fidelina'), CAST (NOW() AS DATE), CAST (NOW() AS TIME), 50)
```

Antes de pasar al siguiente ejemplo, es clave tener presente que las dos filas que se insertan operan como una transacción. Es decir, o se agregan las dos filas o no se suma ninguna. El concepto de transacción es de vital importancia en el manejo de las operaciones realizadas por los DBMS.

¿Cómo insertar en una tabla todas las canciones interpretadas en español?

Para resolver esta solicitud hay dos formas. La primera permite, en una misma transacción, crear una tabla con las columnas devueltas por una consulta y luego insertar las filas devueltas por la consulta en la tabla recién creada. La segunda realiza la inserción de lo devuelto por la

consulta en una tabla preexistente. Inicialmente, se abordará la primera forma, para lo cual se debe crear la consulta que devuelve las filas que se desean ingresar mediante el *Script* 11.24.

***Script* 11.24**

```
SELECT C.idCanción,
       C.título AS tituloCanción,
       A.idAlbum,
       A.título,
       I.idIntérprete,
       I.nombre,
       I.tipoIntérprete,
FROM Canciones C
       INNER JOIN Álbumes A ON C.idÁlbumOriginal = A.idAlbum
       INNER JOIN Intérpretes I ON I.idIntérprete = C.idIntérpretePrincipal
WHERE C.idioma = 'español';
```

Teniendo la consulta, ahora se utiliza la sentencia `SELECT INTO` para crear la tabla `CancionEnEspañol` e insertar los datos en ella. Es importante dejar claro que esta nueva tabla tendrá las mismas columnas que las devueltas por la consulta: mismo número de columnas, mismos nombres y tipos de datos. La sentencia `SELECT INTO` no puede utilizarse para insertar datos en una tabla existente. En términos de sintaxis, simplemente es necesario agregar la palabra reservada `INTO` seguida del nombre de la tabla justo antes de la cláusula `FROM` de la consulta `SELECT` que desea usar para producir el conjunto de resultados. El *Script* 11.25 permite conseguir lo descrito.

***Script* 11.25**

```
SELECT C.idCanción,
       C.título AS tituloCanción,
       A.idAlbum,
       A.título,
       I.idIntérprete,
       I.nombre,
       I.tipoIntérprete,
INTO CancionEnEspañol
FROM Canciones C
       INNER JOIN Álbumes A ON C.idÁlbumOriginal = A.idAlbum
       INNER JOIN Intérpretes I ON I.idIntérprete = C.idIntérpretePrincipal
WHERE C.idioma = 'español';
```

Si la tabla en la cual se quiere insertar el resultado de la consulta ya existe, se puede utilizar la sentencia `INSERT INTO`, pero en el lugar donde deberían ir los valores se especifica la consulta `SELECT` que los proporciona, tal como se muestra en el *Script* 11.26.

***Script* 11.26**

```
INSERT INTO InfoCancionesEnEspañol
SELECT C.idCanción,
       C.título AS tituloCanción,
       A.idAlbum,
       A.título,
       I.idIntérprete,
       I.nombre,
       I.tipoIntérprete,
       CAST(NOW() AS DATE) AS fechaCreación,
       CAST(NOW() AS TIME) AS HoraCreación
FROM Canciones C
       INNER JOIN Álbumes A ON C.idÁlbumOriginal = A.idAlbum
       INNER JOIN Intérpretes I ON I.idIntérprete = C.idIntérpretePrincipal
WHERE C.idioma = 'español';
```

Algunas veces se necesita eliminar una o varias filas de las tablas de una base de datos; por ejemplo, si se cometió un error durante la inserción de la fila o porque el usuario titular de los datos que se tienen almacenados ha solicitado que se eliminen. Para eliminar las filas de una tabla, SQL proporciona dos sentencias: `DELETE` y `TRUNCATE`. La primera permite borrar las filas de una tabla con base en el filtro de un predicado que es opcional (cuando no aparece el predicado, se borran todas las filas de la tabla).

Deben borrarse todas las reproducciones realizadas por el usuario con identificador «*6*».

Para proceder, solo se debe indicar cuál es la tabla de la que se desea eliminar las filas; en este caso, la tabla `Reproducciones`. Asimismo, es necesario especificar cuáles son las filas en particular que se van a borrar, es decir, todas aquellas donde el identificador del usuario sea igual a «*6*». El *Script* 11.27 plantea cómo obtener este resultado.

***Script* 11.27**

```
DELETE FROM Reproducciones
WHERE idUsuario = 6
```

La condición de la cláusula `DELETE`, como se señaló, es opcional. Si no se incluye, todas las filas de la tabla son borradas. Por consiguiente, al realizar operaciones de eliminación conviene ser cuidadoso y asegurarse de que el efecto producido es el deseado. La eliminación planteada en el *Script* 11.27 también se puede realizar de la forma presentada en el *Script* 11.28, usando subconsultas. En esta oportunidad, en lugar de proporcionar el identificador «*6*» directamente, se reemplaza por una subconsulta que devuelve cuál es el identificador del usuario con nombres «*Marie*» y apellidos «*Curie*».

***Script* 11.28**

```
DELETE FROM Reproducciones
WHERE idUsuario = (SELECT idUsuario
                          FROM Usuarios
                    WHERE nombres='Marie' and apellidos='Curie')
```

Borrar todas las canciones del género «*Salsa*» que terminan en la vocal «*a*».

Para realizar lo deseado basta con expresar las condiciones que deben cumplir las filas por borrar. En este caso se hace uso del operador `LIKE` para indicar que son aquellas canciones cuyo título termina en la vocal «*a*», y mediante una subconsulta autónoma se precisa que deben ser del género «*Salsa*». Estas instrucciones se expresan en SQL tal como lo hace el *Script* 11.29.

***Script* 11.29**

```
DELETE FROM Canciones
WHERE título LIKE '%a' AND idGenero = (SELECT idGénero FROM Géneros WHERE
nombre='Salsa');
```

Borrar las canciones con mayor duración de cada género.

Dentro de la sentencia `DELETE` también se puede hacer uso de subconsultas correlacionadas. El *Script* 11.30 tiene una subconsulta correlacionada que en cada ejecución calcula cuál es la máxima duración entre las canciones que pertenecen al mismo género de la canción que está siendo referenciada cuando se invoca la subconsulta. Así, si la duración de esta canción que está siendo referenciada coincide con la duración máxima calculada, se intenta eliminarla de la tabla `Canciones`.

***Script* 11.30**

```
DELETE FROM Canciones CE
WHERE duración = (SELECT MAX(duración) FROM Canciones CI WHERE CI.idGénero =
CE.idGénero )
```

Sin embargo, pese a que la consulta anterior no tiene ningún error de sintaxis ni lógico, puede que el DBMS no permita realizar la acción que se desea si las filas que se quiere eliminar están siendo referenciadas en otras tablas; en este caso, si el valor del identificador de la canción que se busca borrar se encuentra relacionado en otras tablas. Esto sucede,

por ejemplo, con el `idCanción` de la tabla `Canciones`, que aparece en `Calificaciones`, `Reproducciones`, `CancionesLista` y `Compositores`.

Así las cosas, para eliminar las filas de la tabla `Canciones` es necesario borrar antes las referencias que se hacen a ellas en las tablas mencionadas. En los *Scripts* 11.31, 11.32, 11.33 y 11.34 se indica esta instrucción. De este modo, una vez no hay referencias a las canciones que se van a eliminar, se puede ejecutar nuevamente el *Script* 11.30.

***Script* 11.31**

```
DELETE FROM Reproducciones R
WHERE idCanción =
(
SELECT idCanción
FROM Canciones
WHERE duración = (
SELECT MAX(duración)
FROM Canciones CI WHERE idGénero = (SELECT idGénero FROM Canciones WHERE idCanción =
R.idCanción)))
```

***Script* 11.32**

```
DELETE FROM Calificaciones C
WHERE idCanción =
(
SELECT idCanción
FROM Canciones
WHERE duración = (
SELECT MAX(duración)
FROM Canciones CI WHERE idGénero = (SELECT idGénero FROM Canciones WHERE idCanción =
C.idCanción)))
```

***Script* 11.33**

```
DELETE FROM Compositores C
WHERE idCanción =
(
SELECT idCanción
FROM Canciones
WHERE duración = (
SELECT MAX(duración)
FROM Canciones CI WHERE idGénero = (SELECT idGénero FROM Canciones WHERE idCanción =
C.idCanción)))
```

Como ya se mencionó, para eliminar las filas de una tabla también existe la sentencia `TRUNCATE`. A diferencia de la sentencia `SELECT`, esta no utiliza ningún predicado para especificar cuáles son las filas sobre las que actuará; por consiguiente, siempre elimina todas las filas de la tabla.

Borrar todas las filas de la tabla Canciones.

***Script* 11.34**

```
DELETE FROM CancionesLista C
WHERE idCanción =
(
SELECT idCanción
FROM Canciones
WHERE duración = (
SELECT MAX(duración)
FROM Canciones CI WHERE idGénero = (SELECT idGénero FROM Canciones WHERE idCanción =
C.idCanción)))
```

La sintaxis de la sentencia `TRUNCATE` para borrar todas las filas de una tabla es muy sencilla: a la palabra `TRUNCATE` le sigue la palabra reservada `TABLE` y luego el nombre de la tabla que se desea afectar; en este caso, `Canciones`. En el *Script* 11.35 se aprecia cómo se procedería para eliminar todas las filas de la tabla `Canciones`.

***Script* 11.35**

```
TRUNCATE TABLE Canciones;
```

No obstante, al ejecutar el *Script* 11.35 surge el mismo inconveniente que se obtuvo con la sentencia `DELETE` del *Script* 11.30, es decir, debido a que el identificador de las canciones es referenciado por otras tablas, no se puede realizar la operación. De igual forma, en este caso se tienen que eliminar primero las filas de las tablas que hacen referencia al identificador de la canción de la tabla `Canciones`, como se muestra en el *Script* 11.36.

***Script* 11.36**

```
TRUNCATE TABLE Calificaciones;
TRUNCATE TABLE Reproducciones;
TRUNCATE TABLE CancionesLista;
TRUNCATE TABLE Compositores;
```

Sin embargo, la sentencia `TRUNCATE` proporciona un «atajo» para eliminar todas las filas de las tablas que referencian a las columnas de la tabla de la que se desean eliminar las filas. Este «atajo» consiste en agregar únicamente la palabra reservada `CASCADE`, como se muestra en el *Script* 11.37.

***Script* 11.37**

```
TRUNCATE TABLE Canciones CASCADE;
```

De todos modos, la combinación `TRUNCATE CASCADE` requiere ser cuidadoso puesto que puede eliminar todas las filas de una tabla sin tener intención de hacerlo. En este sentido, cabe precisar que, cuando se trata de borrar todas las filas de una tabla, `TRUNCATE` tiene un mejor rendimiento que `DELETE` debido a que los registros que hace de las eliminaciones son menos detallados que los de esta última.

Adicional a la diferencia en el rendimiento, también hay una diferencia funcional entre las dos sentencias mencionadas. Cuando una de las columnas de la tabla de la que se desean eliminar todas las filas tiene un valor que es autoincremental, el `TRUNCATE` reinicia el conteo, por lo cual la siguiente fila por insertar en esa tabla tendría el mínimo valor que se acepta (1 para el caso de los enteros). En cambio, `DELETE` no opera de esta forma.

Para la actualización de las filas, SQL proporciona la sentencia `UPDATE`, que funciona de forma similar a la instrucción `DELETE`. En estos casos deben especificarse el nombre de la tabla, los cambios por realizar en las columnas de cada fila de tabla y, finalmente, el predicado que permite determinar cuáles son las filas por actualizar. A continuación, se desarrollará un ejemplo de uso de esta sentencia.

La canción con identificador «*2*» debe aparecer como instrumental.

Para realizar este cambio solicitado se utiliza la sentencia `UPDATE` seguida de la tabla en la que se desea realizar el cambio; en este caso, `Canciones`. Luego se indica cuál es el cambio por aplicar, es decir, cambiar el valor de la columna `esInstrumental`, que es de tipo `BIT`, a «*1*», tal cual se muestra en el *Script* 11.38.

***Script* 11.38**

```
UPDATE Canciones
SET esInstrumental = '1';
```

Ahora, debido a que al ejecutar el *script* se haría el cambio sobre todas las filas de la tabla `Canciones`, es necesario especificar cuál es la canción por modificar, que en esta oportunidad sería la que tiene el identificador «*2*». El *Script* 11.39 muestra cómo quedaría la instrucción.

***Script* 11.39**

```
UPDATE Canciones
SET esInstrumental = '1'
WHERE idCanción = 2;
```

Indicar que todas las canciones del género vallenato almacenadas hasta la fecha son instrumentales.

Para realizar esta operación se puede hacer uso de una subconsulta autónoma que devuelva cuál es el identificador del género vallenato y embeberla en la cláusula `WHERE`, como se muestra en el *Script* 11.40.

***Script* 11.40**

```
UPDATE Canciones
SET
    esInstrumental='1'
WHERE idGénero = (SELECT idGénero
                  FROM Géneros
                  WHERE nombre = 'Vallenato');
```

Cambiar el tipo de plan especificado en el pago realizado con identificador 155 por plan familiar.

El cambio solicitado es sobre la tabla `Pagos`, y para realizarlo es preciso recurrir a una subconsulta autónoma que devuelva el identificador del plan familiar en la cláusula `SET`. Además, se debe cambiar la fecha del pago a la fecha actual, aprovechando las funciones `NOW` y `CAST`. Finalmente, con la clave principal de la fila que se desea modificar se tiene la garantía de que la operación afectará solo a esa fila en concreto. El *Script* 11.41 realiza lo demandado.

***Script* 11.41**

```
UPDATE Pagos
SET idPlan = (SELECT idPlan FROM Planes WHERE nombre = 'Familiar'),
    fechaPago = CAST(NOW() AS DATE)
WHERE idPago = 155
```

Calificar con 5 todas las canciones del género vallenato reproducidas durante el año 2020.

Esta demanda se puede satisfacer utilizando combinaciones entre las tablas `Calificaciones`, `Canciones` y `Género`. Para este escenario es posible recurrir a la cláusula `FROM` para incluir el `INNER JOIN` entre las tablas `Canciones` y `Géneros`. Luego, en la cláusula `WHERE` se especifica que entre las tablas `Calificaciones` y `Canciones` también se desea que se haga un `INNER JOIN`.

Luego de combinar las tablas, se puede continuar con añadir las otras condiciones: que el año de la calificación sea igual a *«2020»* y que el nombre de género sea *«Vallenato»*. El *Script* 11.42 muestra de forma completa el conjunto de instrucciones.

***Script* 11.42**

```
UPDATE Calificaciones Cal
SET Calificación = 5
FROM Canciones C INNER JOIN Géneros G USING(idGénero)
WHERE C.idCanción = Cal.idCanción and EXTRACT (YEAR FROM Cal.fechaCalificación) = 2020
AND G.nombre = 'Vallenato'
```

Asignar el comentario «Me gusta esta canción» a las canciones que ocupan el lugar uno o dos dentro de las más reproducidas de cada usuario.

La solución a esta solicitud se puede dar utilizando una subconsulta correlacionada que se ejecute por cada calificación realizada por cada usuario y determine el lugar que ocupa esa canción en el escalafón de las más reproducidas de esa persona en particular. La consulta que identifica esto último se presenta en el *Script* 11.43.

La consulta del *Script* 11.43 se puede embeber en una sentencia `UPDATE` que permita determinar el lugar que ocupa la canción dentro de esta lista y, si este es el primero o el segundo, cambiar el comentario actual a «Me gusta esta canción», tal cual se presenta en el *Script* 11.44.

***Script* 11.43**

```
SELECT idCanción, COUNT(Reproducciones),
DENSE_RANK() OVER(ORDER BY COUNT(Reproducciones) DESC ) ranking
FROM reproducciones
WHERE idUsuario = 1
GROUP BY idCanción
```

***Script* 11.44**

```
UPDATE Calificaciones Ca
SET Comentario = 'Me gusta esta canción'
WHERE
(SELECT ranking FROM (SELECT idCanción, COUNT(Reproducciones),
DENSE_RANK() OVER(ORDER BY COUNT(Reproducciones) DESC ) ranking
FROM reproducciones
WHERE idUsuario = Ca.idUsuario
GROUP BY idCanción) RankingCanciones
WHERE idCanción = Ca.idCanción) BETWEEN 1 AND 2
```

11.3. Creación de índices

El modelo lógico es una fuente de información necesaria pero no suficiente para la óptima implementación de una base de datos relacional. De hecho, aunque en el diseño lógico y conceptual de las bases de datos relacionales se toman decisiones importantes, en la fase de implementación también se ejecutan cambios que son relevantes para que el funcionamiento de una base de datos sea eficiente. Por ejemplo, es posible que se tenga que considerar modificar el modelo lógico para lograr niveles aceptables de desempeño.

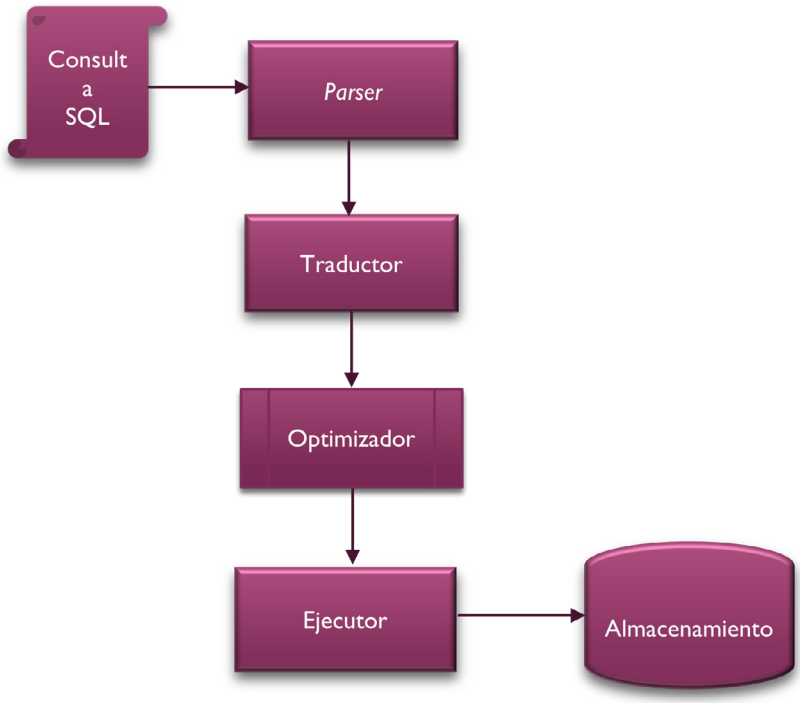
Como se ha mencionado en más de un par de ocasiones a lo largo de este libro, SQL es un lenguaje declarativo; por consiguiente, no se puede definir cómo se accede a los datos almacenados en las tablas. Esta responsabilidad se deja totalmente al DBMS, que en cada consulta debe determinar cuál es la mejor forma para tomar la información solicitada. Para cumplir con este propósito, los DBMS tienen un componente, llamado planificador/optimizador, que debe decidir entre todas las alternativas posibles para responder con la mejor a una consulta.

Grosso modo, se puede afirmar que una consulta SQL, antes de ser ejecutada, transita por un conjunto de cuatro fases, y en cada una de ellas está a disposición de diferentes componentes del DBMS, como se muestra en la Figura 11.6. De estas etapas, la tercera, de planificación/optimización, es la que puede ser afectada por medidas administrativas y de diseño que se tomen en el momento de definir y configurar las tablas, y una de estas medidas es la inclusión de índices.

Muchas veces las consultas pueden demorar en su ejecución más tiempo del que se considera «normal». En este escenario se puede optar por reescribir la consulta o utilizar un índice apropiado para acelerar el acceso a los datos subyacentes. Al respecto, es importante recordar que:

- Los datos de las bases de datos relacionales tienen su persistencia en disco, por lo que su acceso es más costoso que el realizado en memoria RAM.
- El acceso a los datos de una tabla se realiza de manera secuencial a través de un escaneo de los bloques del disco donde las filas de la tabla son almacenadas. Esta forma de trabajar puede tornarse ineficiente según los datos que se estén intentando acceder.

Figura 11.6. Componentes del DBMS para procesar las consultas



Los índices de una tabla operan de forma similar a como funcionan las tablas de contenido de los libros. Estas últimas permiten acceder rápidamente a la página donde se encuentra el tema deseado, de forma que no se pierda tiempo revisando todo el texto u hojeando todas las páginas. De igual modo, en las tablas de las bases de datos se crean índices para saber dónde están las filas de la tabla, tomando como referencia a las columnas. Entonces, un índice es una estructura que permite un acceso más expedito a los datos de la tabla subyacente, para que se puedan encontrar las filas específicas con mayor rapidez que si se escaneara toda la tabla subyacente y se analizara cada fila.

Un índice se puede construir sobre una sola columna o en múltiples columnas a la vez. Además, puede cubrir todos los datos de la tabla subyacente o solo indexar valores específicos; en ese último caso, el índice se conoce como «parcial». En PostgreSQL la implementación por defecto de los índices se hace utilizando un árbol balanceado (*B-Tree*, del inglés *Balanced Tree*). Saber esto permite inferir que el mantenimiento de los índices tiene un costo: el de mantener organizado y balanceado el árbol.

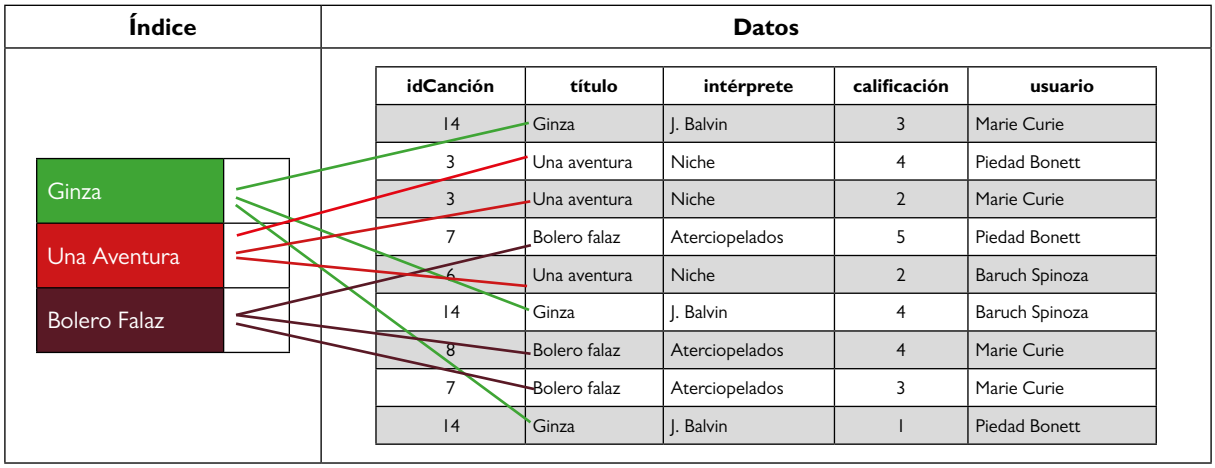
Con lo dicho se puede pensar: si los índices mejoran el rendimiento para acceder a los datos, convendría usarlos en todas las columnas y en todas las combinaciones posibles. Este razonamiento intuitivo, sin embargo, no es correcto, pues los índices, como ya se ha dicho, tienen varios costos asociados que deben ser considerados. Aunque no es propósito de este libro profundizar en todos los detalles administrativos que se deben tener en cuenta al contemplar el uso de índices, sí es importante pensar en que:

- Los índices tienen un costo de mantenimiento. Al ser gestionados como árboles cada vez que se inserta, modifica o elimina una fila de la tabla, el o los índices deben ser reorganizados, y las referencias a las filas, incluidas o modificadas.
- Los índices generalmente se almacenan en su propio espacio de disco, por lo que deben ser leídos en pasos diferentes al de lectura de los datos. Esto significa que se tienen almacenados un archivo de datos para la tabla y otro para cada índice que se cree. Un escaneo de índice siempre requiere dos accesos distintos al almacenamiento: uno para leer el índice y extraer la información de en qué parte de la tabla están las filas solicitadas, y otro para acceder al disco para buscar las filas señaladas por el índice.

A partir de esto, debe quedar claro que los DBMS evitan usar índices cuando no son útiles, que es cuando el doble acceso al almacenamiento mencionado anteriormente representa más desventajas que ventajas. A la luz de estas consideraciones, siempre es necesario evaluar cuándo los índices son pertinentes para mejorar las consultas.

En la Figura 11.7 se presenta gráficamente el ejemplo de un índice definido sobre la tabla de la derecha utilizando la columna `título`. En este caso se ve cómo se mantiene una referencia a cada una de las filas de la tabla en función del valor que tiene en la columna `título`. Esta representación muy básica evidencia el beneficio de los índices: si se desea recuperar todas las filas donde el título de la canción es igual a *«Ginza»*, solo se debe buscar en el índice la entrada correspondiente a *«Ginza»*, y la entrada en el índice dirá cuáles son las filas por recuperar. Es evidente que cuantas más filas haya en la tabla de la derecha, más útil resultará el índice.

Figura 11.7. Ejemplificación del funcionamiento de un índice



Los diferentes DBMS cuentan con instrucciones que permiten dilucidar el razonamiento hecho por el componente planificador/optimizador para elegir entre las diferentes alternativas posibles en el momento de ejecutar una consulta. En el caso de PostgreSQL, la sentencia

que cumple ese propósito es `EXPLAIN`. Esta es una herramienta poderosa para entender cómo el DBMS interpreta las consultas y determinar cómo «ayudar» a que este realice de forma más rápida el acceso a datos. Sin embargo, es importante tener claro que la optimización de las consultas es un tema más complejo de lo que en este capítulo se presenta. De hecho, muchas veces requiere largas sesiones de pruebas, por lo que no es el objetivo abarcar toda esta complejidad; solo se trata de aportar un entendimiento básico de cómo los índices pueden ser útiles en estos escenarios.

Para iniciar, el *Script* 11.45 muestra el uso de `EXPLAIN` para analizar una consulta. En la Figura 11.8, entretanto, se muestra el plan de ejecución que crea el DBMS, y en la Tabla 11.3 se resumen las estadísticas de este.

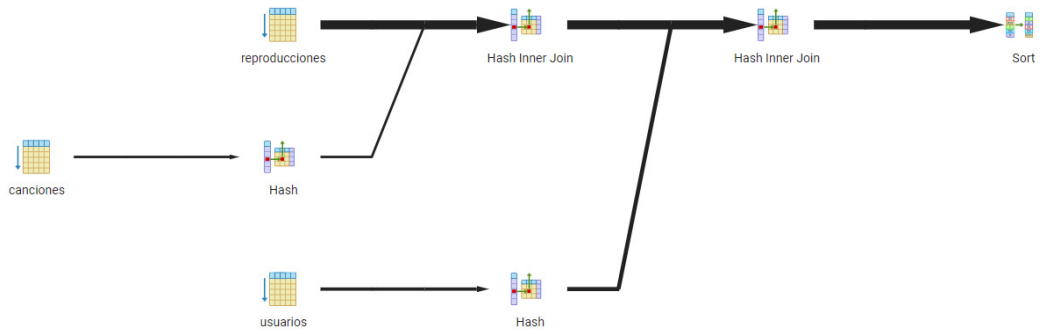
***Script* 11.45**

```
EXPLAIN
(FORMAT JSON, ANALYZE)
SELECT *
FROM Canciones JOIN Reproducciones USING(idCanción)
INNER JOIN Usuarios USING(idUsuario)
ORDER BY fechaReproducción
```

Del plan de ejecución creado es importante destacar que está compuesto por ocho nodos, que son las unidades encargadas de las tareas de procesamiento de datos, y que el equipo en que se ejecutó la instrucción empleó 1.561,638 ms. En la figura aparecen tres nodos (Canciones, Reproducciones y Usuarios) cuya única función es leer las tablas de manera secuencial, que es el modo por defecto para acceder a los datos. En un escaneo de esta clase, el ejecutor irá al comienzo de la tabla en el disco —por ejemplo, al comienzo del archivo correspondiente a una tabla— y leerá todos los datos un bloque tras otro en estricto orden.

También se observa que existen dos combinaciones (*hash inner join)* que realizan la tarea de combinar las tablas. En otras palabras, las tablas `Canciones` y `Usuarios` son pasadas previamente por una función *hash*. Finalmente, se tiene un nodo de ordenamiento *(sort)*, que se encarga de entregar los datos ordenados tal cual se solicitan en la consulta.

Figura 11.8. Plan de ejecución de la consuta del *Script* 11.45



Al evaluar la tabla de las estadísticas con detenimiento, se observa cómo el mayor costo de ejecución se emplea en el ordenamiento. Así pues, puede surgir la duda de si emplear un índice que permita recuperar los datos con base en la fecha de reproducción mejoraría el tiempo de ejecución de la consulta.

Tabla 11.3. Detalles del plan de ejecución del *Script* 11.45

Nodo	Tiempo acumulado
Sort	1.523,258 ms
Hash inner join (`Reproducciones` y `Usuarios`)	285,01 ms
Hash inner join (`Reproducciones` y `Canciones`)	170,408 ms
Acceso secuencial a `Reproducciones`	38,781 ms
Hash `Canciones`	0,013 ms
Acceso secuencial a `Canciones`	0,007 ms
Hash `Canciones`	0,017 ms
Acceso secuencial a `Canciones`	0,007 ms

En el *Script* 11.46 se presenta cómo crear (`CREATE`) un índice (`INDEX`) sobre la tabla `Canciones` utilizando la columna de fecha de reproducción.

***Script* 11.46**

```
CREATE INDEX idx_reproducción_date
ON Reproducciones( fechaReproducción );
```

Figura 11.9. Plan de ejecución luego de creación del índice del *Script* 11.46

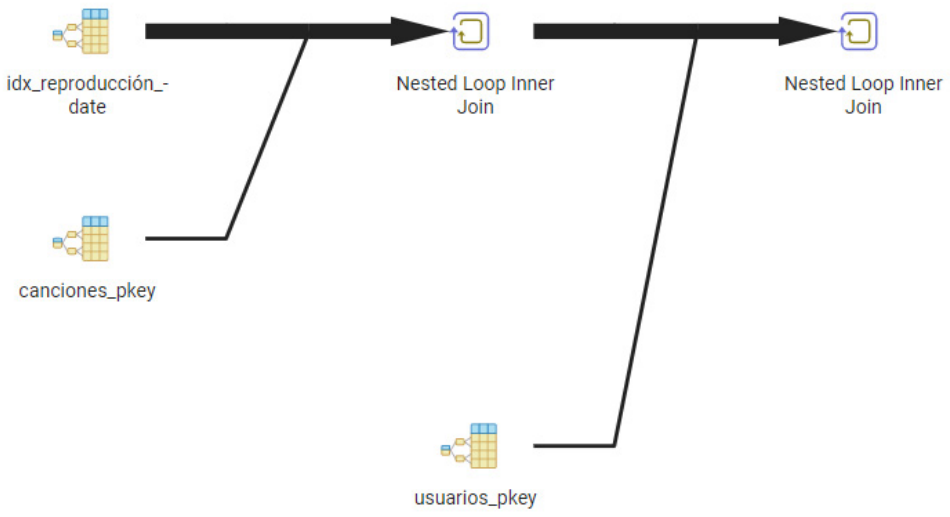


Tabla 11.4. Detalles del plan de ejecución del *Script* 11.46

Nodo	Tiempo acumulado
Nested loop inner join (`Reproducciones` y `Canciones`)	1.162,713 ms
Nested loop inner join (`Reproducciones` y `Usuarios`)	727,195 ms
Index scan (`Reproducciones`) usando `idx_reproducción_date`	241,965 ms
Index scan (`Canciones`) usando `canciones_pkey`	589,825 ms
Index scan (`Usuarios`) usando `usuarios_pkey`	0 ms

Luego de crear el índice y ejecutar nuevamente el *Script* 11.45, el plan de ejecución cambia (ver Figura 11.9). Ahora, como se aprecia en el resumen de la Tabla 11.4, el tiempo de ejecución es de 1.162,713 ms, es decir, un 24 % menos que sin usar el índice. En el plan de ejecución se evidencia que ahora se está utilizando un escaneo de índice para acceder a los datos de las tablas.

Como se planteó en un inicio, los índices pueden ser totales o parciales. Si por ejemplo se quiere que solo se indexen las reproducciones realizadas a partir del año 2019, basta con plantear una condición mediante la cláusula `WHERE`, como se aprecia en el *Script* 11.47.

***Script* 11.47**

```
CREATE INDEX idx_reproducción_date
ON Reproducciones( fechaReproducción )
WHERE EXTRACT(YEAR FROM fechaReproducción) >= 2019
```

El índice puede estar compuesto por más de una columna. Cuando sea así, es importante tener en cuenta que siempre se deben colocar primero las columnas más selectivas. De esta forma, el método de acceso al índice será el más económico. Por ejemplo, incluir un índice compuesto por las columnas `idUsuario` y `fechaReproducción` es muy simple: solo es necesario indicar cuáles son las columnas que hacen parte del índice compuesto, como se muestra en el sScript 11.48.

***Script* 11.48**

```
CREATE INDEX idx_reproducción_usuario_date
ON Reproducciones( idUsuario, fechaReproducción );
```

Finalmente, para eliminar un índice se deben usar las palabras reservadas `DROP INDEX`, acompañadas por el nombre del índice que se va a borrar. En el *Script* 11.49 se presenta el caso para el índice de nombre `idx_reproducción_usuario_date`.

***Script* 11.49**

```
DROP INDEX idx_reproducción_usuario_date;
```

Por último, se deben considerar las siguientes recomendaciones sobre cómo definir los índices que se desea crear:

- Los índices deben ser altamente selectivos, lo que básicamente significa que las consultas que llegan a ellos deben devolver un número bajo de filas.
- Evitar usar índices sobre tablas/columnas que son afectadas por muchas operaciones de inserción, modificación y eliminación.
- Definir índices que utilicen la menor cantidad de columnas posibles.
- Utilizar, preferiblemente, claves numéricas exactas como los números enteros puesto que ofrecen un mayor rendimiento, necesitan menos espacio en disco y generan menores gastos de mantenimiento.

11.4. Aprendizajes más importantes del capítulo 11

- En el nivel de implementación, las tablas son las estructuras básicas de almacenamiento, y con el sublenguaje DDL se puede hacer una gestión completa de ellas.
- El sublenguaje DML no se agota con la sentencia `SELECT`; también proporciona las sentencias `INSERT`, `UPDATE` y `DELETE` para la inserción, la actualización y la eliminación, respectivamente.
- La inserción de filas puede realizarse de diversas formas que van desde la más simple, proporcionando los valores de la fila, hasta más complejas donde se hace uso de subconsultas.
- Cuando se inserta más de una fila utilizando una sola sentencia `INSERT`, se toman todas las inserciones como una transacción, es decir, o se realizan todas las inserciones o no se realiza ninguna.
- Las sentencias `UPDATE` y `DELETE` pueden realizarse sobre todas las filas de una tabla o sobre un subconjunto si se utiliza la cláusula `WHERE`.
- Al realizar inserciones y actualizaciones se deben considerar detalles como el tipo de datos de las columnas de las tablas, las restricciones que imponen las claves foráneas y si las columnas aceptan el descriptor nulo.
- Al realizar eliminaciones es necesario ser cuidadoso con el uso de `DELETE FROM` y `TRUNCATE` puesto que borran todas las filas de una tabla.
- El uso de `TRUNCATE CASCADE` debe realizarse con cautela debido a que no solo elimina las filas de tabla indicada, sino que también puede borrar las filas de las tablas relacionadas.

- Los índices son una herramienta que permite mejorar el rendimiento en el momento de acceder a los datos de las tablas, pero su uso debe ser planificado y evaluado debido a que no siempre un índice es la mejor opción para acceder a los datos.
- SQL, al ser un lenguaje declarativo, deja en los DBMS la responsabilidad de elegir cuál es la mejor forma para acceder a los datos almacenados en disco.

11.5. Actividades de aplicación para evidenciar lo aprendido

1. Use el sublenguaje DDL para crear las tablas del modelo lógico resultante de la actividad 1 del capítulo anterior.
2. Seleccione un intérprete musical —por ejemplo, «*The Beatles*»— e inserte las canciones de dos albúmenes de ese intérprete en la base de datos. Para eso, realice las inserciones necesarias en las tablas `Álbumes`, `Géneros`, `Canciones`, `Intérpretes`, `ParticipaciónEnGrupos` o `Solistas`, `Compositores` y `Músicos`.
3. Proponga una operación de actualización y otra de inserción que se puedan resolver utilizando subconsultas correlacionadas. Luego escriba el código SQL necesario para su implementación.
4. Guarde en una tabla, de nombre `CancionesMásReproducidas`, el título de la canción, el nombre del género, el nombre del intérprete y la calificación promedio de todas las canciones que están en el primer lugar de las canciones reproducidas por cada usuario.
5. Elimine las reproducciones del género «*Pop*» realizadas durante el primer semestre del año 2020 que han durado menos de 30 segundos.
6. Actualice los comentarios de los usuarios sobre cada canción de manera que aquellas con la peor valoración promedio reciban como nuevo valor en el comentario «No ha gustado».
7. Identifique los errores lógicos o de sintaxis en el siguiente *script*. Explique cómo impide la correcta ejecución del *script* cada uno de los errores detectados.

```
UPDATE Calificaciones Ca
SET Comentario = 'Me gusta esta canción'
WHERE BETWEEN
(SELECT ranking
FROM (
SELECT idCanción, COUNT(Reproducciones),
DENSE_RANK() OVER(ORDER BY COUNT(Reproducciones) DESC ) ranking
FROM reproducciones
WHERE idUsuario = Ca.idUsuario
)
WHERE idCanción = Ca.idCanción) 2 AND 3
```

8. Cree un índice sobre la tabla `Canciones` y luego utilice el comando `EXPLAIN` para conocer el plan de ejecución generado para una consulta que requiera dicho índice. Si al utilizar `EXPLAIN` se observa que no se usa el índice, inserte más filas hasta que este sea empleado o modifique el índice para que sea útil para el planificador/optimizador.
9. Revise la Ley 1581 de 1992 de Colombia e identifique cómo regula la creación y la manipulación de las bases de datos.

Capítulo 12

Vistas y objetos de programación procedimental

Resultados de aprendizaje

– *Proporciona acceso personalizado a los datos a través del almacenamiento de consultas (vistas).*
– *Extiende las funcionalidades de la base de datos con objetos de programación como son las funciones, los procedimientos almacenados y los disparadores.*

Como se ha planteado hasta este capítulo, SQL es un lenguaje de programación declarativo que permite describir un conjunto de datos que se desean obtener, y el DBMS es quien determina cuál es la mejor forma de realizar las operaciones que se necesitan para obtener ese conjunto de datos. Este enfoque declarativo trae consigo los beneficios de una mayor abstracción y simplicidad, pero deja de lado las ventajas que ofrecen los lenguajes de programación imperativos (como C y Python), dentro de las cuales se encuentran la flexibilidad y la riqueza para permitir cálculos complejos, el uso de condicionales e iteraciones, y el manejo de errores.

El mundo de las bases de datos relacionales no es ajeno a los beneficios de los lenguajes de programa imperativos en general y a la programación procedimental en particular. Por consiguiente, han evolucionado para hacer posible la implementación de subprogramas utilizando un enfoque imperativo. Este ajuste obedece al hecho de que, en su enfoque clásico, el lenguaje SQL es completamente declarativo, por lo que los diferentes DBMS han optado por implementar su propio lenguaje imperativo para soportar la creación de subprogramas. Por ejemplo, Oracle creó el PL/SQL, y PostgreSQL, el PL/PGSQL (en ambos casos, «PL» significa lenguaje procedimental, en inglés: *Procedural Language*). Con estos lenguajes es posible agrupar varias operaciones, incluidas declaraciones SQL, en bloques de código

que se almacenan y ejecutan en el lado del servidor. Estos grupos de operaciones, como se ha mencionado con anterioridad, reciben el nombre de subprogramas y son básicamente de dos tipos: los procedimientos almacenados y las funciones.

Los procedimientos almacenados y las funciones pueden tomar un conjunto de parámetros que son proporcionados por quien los invoca y realizan una serie de operaciones que les permiten cumplir con una tarea. Ambos subprogramas son capaces de modificar los datos que se les proveen como argumento, pero solo las funciones tienen que devolver datos a quien las invoca, mientras que un procedimiento no necesariamente lo hace. Por tal razón, por regla general se utilizan procedimientos a menos que se necesite un valor de retorno.

Este capítulo se enfocará en mostrar la utilidad que tienen los subprogramas en el contexto de las bases de datos relacionales. Para cumplir con este fin, se utilizará el lenguaje PL/PGSQL, construido por PostgreSQL para permitir la programación procedimental. También se verá que los subprogramas pueden ser útiles en otras tareas de frecuente uso como el manejo de los eventos que afectan las tablas de la base de datos. En todo caso, es importante dejar claro que, aunque se recurrirá al lenguaje creado por PostgreSQL, la fundamentación conceptual y los elementos tratados en este capítulo son equivalentes entre los diferentes DBMS.

12.1. Vistas

Con frecuencia surgen necesidades como reutilizar una consulta creada, agregar una capa de abstracción para dar a acceso a los datos o, simplemente, se quiere «guardar» el resultado de una consulta para no tener que escribirla nuevamente cuando la misma necesidad de datos aparezca o bien se presente un requerimiento de datos basado en los resultados de una consulta previa. En situaciones como las mencionadas, SQL proporciona un objeto llamado vistas (`VIEW`, por el término en inglés).

Una vista se puede definir como una tabla virtual que está basada en el resultado de una consulta `SELECT`. Al respecto, cabe anotar que se habla de «una tabla» porque está constituida por filas y columnas, pero es «virtual» porque no existe físicamente en la base de datos. Las vistas, a diferencia de las tablas, solo aceptan operaciones de lectura; no de inserción, ni de modificación, ni de eliminación. Esto tiene sentido puesto que las vistas no están almacenando los datos; simplemente forman una capa de abstracción para leer los datos. Así pues, las vistas son una parte importante del modelado de bases de datos porque actúan como una interfaz.

Las vistas pueden ser creadas a partir de consultas sobre una, dos o más tablas, de acuerdo con las necesidades de datos. Ahora bien, puede surgir la pregunta: ¿por qué tener una tabla virtual a partir de una consulta si ya se cuenta con las tablas y una vista no responde preguntas que no se puedan resolver ya con una sentencia `SELECT` sobre las tablas? Sin embargo, los beneficios de las vistas son muchos. A continuación, se listan solo algunos centrados en la seguridad, el rendimiento y la productividad:

- Las vistas permiten simplificar consultas complejas haciendo uso de la modularidad y del reúso del código. La descomposición de los problemas es un principio central dentro de la disciplina de ingeniería del software y de la resolución de problemas utilizando pensamiento computacional. Con las vistas esta descomposición es posible porque resuelven una necesidad de datos y a su vez pueden ser utilizadas cuantas veces se requiera en lugar de escribir el código SQL completo una y otra vez, disminuyendo así la cantidad de código SQL que se debe plantear.
- Como consecuencia de lo anterior, descomponer el problema en partes haciendo uso de las vistas hace más fácil probar y depurar las sentencias SQL. Algunas veces las consultas resultan ser tan extensas que se complican su lectura y su comprensión.
- Adicionalmente, con las vistas se puede proporcionar un control de acceso sobre los datos presentes en las bases de datos. Dicho en otros términos, es posible implementar una autorización tanto a nivel de fila, dejando de lado las filas que no cumplen con un determinado predicado, así como a nivel de columnas, no incluyendo las columnas que no se desea mostrar.

Es clave tener presente que cada vez que se ejecuta una consulta `SELECT` usando una vista, los datos se reconstruyen, por lo que siempre están actualizados; no es una copia congelada almacenada en el momento en que se creó la vista. Así, una vez clara toda la argumentación esbozada a favor de las vistas, es momento de ver cómo funcionan en la práctica. Suponga que como administrador o usuario de la base de datos de la plataforma *«Mis Canciones»* se quiere satisfacer la siguiente necesidad de datos:

¿Cuál es la cantidad de reproducciones y la calificación promedio de las canciones escritas por cada compositor?

Para dar respuesta a esta pregunta, se puede hacer uso del *Script* 12.1, donde se utilizan funciones de agregación, combinaciones de tablas, agrupamiento y subconsultas.

Imagine ahora que esta es una necesidad de datos que emerge con cierta frecuencia, de manera que tener que escribir una y otra vez la misma consulta resultaría agotador y aburridor. Pocas cosas son tan tediosas como «perder» el tiempo haciendo siempre lo mismo. En este contexto son útiles las vistas, que permiten darle nombre a una consulta y acceder luego a ella a través de dicho nombre. Para crear una vista usando la consulta del *Script* 12.1, solo se deben incluir las palabras reservadas `CREATE` y `VIEW` como se muestra en el *Script* 12.2.

***Script* 12.1**

```
SELECT IR.nombre,
       IR.CantidadReproducciones,
       IC.Calificación
FROM(SELECT Intérpretes.idIntérprete,
       Intérpretes.nombre,
       COUNT(Reproducciones.idReproducción) AS CantidadReproducciones
  FROM Canciones
       INNER JOIN Intérpretes
       ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
       INNER JOIN Reproducciones
       ON reproducciones.idcanción = Canciones.idCanción
  GROUP BY Intérpretes.idIntérprete,
       Intérpretes.nombre) AS IR
  INNER JOIN(SELECT Intérpretes.idIntérprete,
                    Intérpretes.nombre,
                    AVG(Calificaciones.calificación) AS Calificación
       FROM Canciones
            INNER JOIN Intérpretes
            ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
            INNER JOIN Calificaciones
            ON Calificaciones.idcanción = Canciones.idCanción
       GROUP BY Intérpretes.idIntérprete,
                Intérpretes.nombre) AS IC
  ON IR.idIntérprete = IC.idIntérprete
```

Con el *Script* 12.2 se ha asociado el nombre `ReproduccionesYCalificacionsPorInterprete` a la consulta del *Script* 12.1, por lo que ahora puede ser tratada como una tabla. Sin embargo, es pertinente recordar que este tratamiento solo se da en el momento de acceder a la consulta, pero no es una tabla; es una vista. Asimismo, es posible realizar las mismas operaciones que se hacían con las tablas dentro de una sentencia `SELECT`; por ejemplo, seleccionar todos las columnas de la vista, como se muestra en el *Script* 12.3.

También se puede seleccionar unas columnas en específico, filtrar las filas y demás operaciones presentadas en este libro. En el *Script* 12.4, por ejemplo, se puede ver cómo se utiliza la vista recién creada para seleccionar el nombre de los intérpretes que tienen una calificación promedio al menos de «*3.0*» entre todas sus canciones.

Las vistas también pueden ser modificadas, pero no de la misma forma como se hace con las tablas. Para dicho fin, el estándar SQL provee la palabra reservada `REPLACE`. En el uso de este término para reemplazar la definición de una vista, es clave tener presente que la lista de columnas devueltas por la consulta debe ser idéntica antes y después del reemplazo, incluidos el tipo, el nombre y el orden de la columna. Por ejemplo, en el *Script* 12.5 se aprecia cómo se puede modificar la vista `ReproduccionesYCalificacionsPorInterprete`.

***Script* 12.2**

```
CREATE VIEW ReproduccionesYCalificacionsPorInterprete
AS
SELECT IR.nombre,
       IR.CantidadReproducciones,
       IC.Calificación
FROM(SELECT Intérpretes.idIntérprete,
            Intérpretes.nombre,
            COUNT(Reproducciones.idReproducción) AS CantidadReproducciones
    FROM Canciones
            INNER JOIN Intérpretes
            ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
            INNER JOIN Reproducciones
            ON reproducciones.idcanción = Canciones.idCanción
    GROUP BY Intérpretes.idIntérprete,
       Intérpretes.nombre) AS IR
  INNER JOIN(SELECT Intérpretes.idIntérprete,
                    Intérpretes.nombre,
                    AVG(Calificaciones.calificación) AS Calificación
             FROM Canciones
                  INNER JOIN Intérpretes
                  ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
                  INNER JOIN Calificaciones
                  ON Calificaciones.idcanción = Canciones.idCanción
             GROUP BY Intérpretes.idIntérprete,
                      Intérpretes.nombre) AS IC
    ON IR.idIntérprete = IC.idIntérprete
```

***Script* 12.3**

```
SELECT *
FROM ReproduccionesYCalificacionsPorInterprete
```

***Script* 12.4**

```
SELECT nombre
FROM ReproduccionesYCalificacionsPorInterprete
WHERE calificación >= 3.0
```

***Script* 12.5**

```
CREATE OR REPLACE VIEW ReproduccionesYCalificacionsPorInterprete
AS
SELECT IR.nombre,
       IR.CantidadReproducciones,
       IC.Calificación
FROM(SELECT Intérpretes.idIntérprete,
            Intérpretes.nombre,
            COUNT(Reproducciones.idReproducción) AS CantidadReproducciones
     FROM Canciones
          INNER JOIN Intérpretes
          ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
          INNER JOIN Reproducciones
          ON reproducciones.idcanción = Canciones.idCanción
     GROUP BY Intérpretes.idIntérprete,
              Intérpretes.nombre) AS IR
   INNER JOIN(SELECT Intérpretes.idIntérprete,
                     Intérpretes.nombre,
                     AVG(Calificaciones.calificación) AS Calificación
              FROM Canciones
                   INNER JOIN Intérpretes
                   ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
                   INNER JOIN Calificaciones
                   ON Calificaciones.idcanción = Canciones.idCanción
              GROUP BY Intérpretes.idIntérprete,
       Intérpretes.nombre) AS IC
     ON IR.idIntérprete = IC.idIntérprete
WHERE IC.Calificación >= 3.0;
```

Aunque la palabra reservada REPLACE abre la posibilidad de modificar la lógica de la vista, no permite cambiar las columnas que esta devuelve. En casos donde se requiera ese ajuste, se deberá borrar la vista y crearla nuevamente. Para eliminarlas, se usa la palabra reservada DROP, como se presenta en el *Script* 12.6.

***Script* 12.6**

```
DROP VIEW ReproduccionesYCalificacionsPorInterprete;
```

Para concluir con el tema de las vistas, es necesario mencionar que el enfoque presentado al respecto es el tradicional, es decir, vistas que son tablas virtuales, que no se almacenan físicamente y que solo reciben operaciones de consulta. Sin embargo, con la evolución de los DBMS se ha reconocido la necesidad de que las vistas adopten otras propiedades que en principio eran propias de las tablas: que se guarden y que se les puedan aplicar otras operaciones del lenguaje DML.

En cuanto a la propiedad de guardar, existen las vistas materializadas (o vistas donde el resultado de la consulta es guardado físicamente). Para crear una vista de este tipo solo se

necesita emplear, luego de `CREATE` y antes de `VIEW`, la palabra `MATERIALIZED`, como se muestra en el *Script* 12.7.

***Script* 12.7**

```
CREATE MATERIALIZED VIEW ReproduccionesYCalificacionsPorInterprete
AS
SELECT IR.nombre,
       IR.CantidadReproducciones,
       IC.Calificación
FROM(SELECT Intérpretes.idIntérprete,
            Intérpretes.nombre,
            COUNT(Reproducciones.idReproducción) AS CantidadReproducciones
     FROM Canciones
          INNER JOIN Intérpretes
          ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
          INNER JOIN Reproducciones
          ON reproducciones.idcanción = Canciones.idCanción
     GROUP BY Intérpretes.idIntérprete,
       Intérpretes.nombre) AS IR
   INNER JOIN(SELECT Intérpretes.idIntérprete,
                     Intérpretes.nombre,
                     AVG(Calificaciones.calificación) AS Calificación
                FROM Canciones
                     INNER JOIN Intérpretes
                     ON Canciones.idintérpreteprincipal = Intérpretes.idIntérprete
                     INNER JOIN Calificaciones
                     ON Calificaciones.idcanción = Canciones.idCanción
                GROUP BY Intérpretes.idIntérprete,
                         Intérpretes.nombre) AS IC
     ON IR.idIntérprete = IC.idIntérprete
WHERE IC.Calificación >= 3.0;
```

El hecho de que las vistas materializadas sí existan físicamente en la base de datos, pero a su vez dependan de tablas u otras vistas que son utilizadas en las consultas SQL plantea una nueva inquietud que no surge con las vistas tradicionales: ¿cómo se actualizan sus datos? La respuesta es que no lo hacen automáticamente; es necesario indicarle al DBMS que la vista se debe sincronizar con las tablas de las cuales depende. Para ese fin se utiliza la palabra reservada `REFRESH`, tal como se muestra en el *Script* 12.8.

***Script* 12.8**

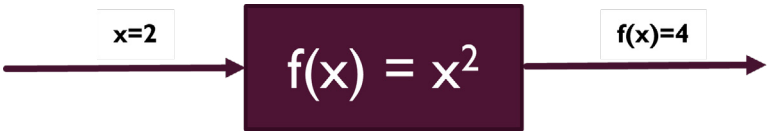
```
REFRESH MATERIALIZED VIEW ReproduccionesYCalificacionsPorInterprete;
```

12.2. Funciones

A lo largo de este libro se ha trabajado con diferentes tipos de funciones: las escalares, las de agregación y las de ventana. Todas ellas son proporcionadas por el DBMS y se pueden utilizar siempre que se desee sin preocuparse por su implementación. No obstante, pese a esta amplia gama de alternativas, es imposible que se satisfagan *a priori* todas las necesidades que pueden darse. Por esta razón, es frecuente que se deban crear funciones propias, es decir, definidas por el usuario.

Las funciones definidas por el usuario son subprogramas que se pueden crear para encapsular comportamientos y que pueden recibir parámetros y retornar un valor. Es habitual entender el uso de funciones desde las matemáticas, donde frecuentemente se proponen unas variables independientes (argumentos), una definición de la función (operaciones) y un resultado de aplicar la función (salida). En la Figura 12.1, por ejemplo, se representa el caso de una función $f(x) = x^2$, la cual recibe un argumento y realiza una operación con este: elevarlo al cuadrado. Teniendo claro esto, cuando el argumento tiene el valor de dos, la salida será cuatro.

Figura 12.1. Esquema general de una función



El ejemplo anterior es muy sencillo, pero permite abstraer la idea del objetivo de las funciones y sus partes básicas. Ahora estos conceptos pueden llevarse al terreno de la programación imperativa procedimental en las bases de datos relacionales para responder a la siguiente necesidad de datos:

¿Cuál es el nombre y la edad (en años) de todos los usuarios registrados en la plataforma «*Mis Canciones*»?

Uno de los insumos necesarios para responder a esta solicitud de datos se encuentra en la tabla `Usuarios`: la fecha de nacimiento de los usuarios. Además, mediante la función `NOW` se puede determinar la fecha del día de hoy. Así, al calcular la diferencia en años que existe entre estas dos fechas, es posible calcular la diferencia en años que hay entre ellas, y esta sería la edad del usuario. Teniendo clara la lógica de cómo calcular la edad de un usuario, se puede encapsular este comportamiento en una función de nombre edad.

Para crear una función se utilizan las palabras reservadas `CREATE FUNCTION`, seguidas por el nombre que se le desea dar; en este caso, «edad». En el *Script* 12.9 donde se presenta este proceso, se puede ver que, además del nombre de la función, se recibe un argumento

denominado `fechaNacimiento`, que es de tipo DATE. También, que el resultado (RETURNS) que dará la función luego de ser ejecutada será de tipo entero, es decir, la edad de la persona será un entero que indicará cuántos años tiene el usuario. Estos primeros elementos hacen parte de la declaración de la función (el nombre, la lista de argumentos y los tipos de valores retornados).

***Script* 12.9**

```
CREATE OR REPLACE FUNCTION edad(fechaNacimiento DATE)
RETURNS INT
AS
$BODY$
        DECLARE
        edad INT;
        hoy DATE;
        BEGIN
        hoy := NOW();
        edad := EXTRACT (YEAR FROM AGE(hoy, fechaNacimiento));
        return edad;
        END;
$BODY$
language 'plpgsql';
```

Luego de la declaración y de la palabra AS, se escribe cómo se implementará la función, es decir, el código que se ejecutará cada vez que esta se invoque. En este caso se observa que se declaran dos variables: edad, de tipo entero, y hoy, de tipo DATE, que almacenará la fecha del día que se invoque la función. Todas las variables se declaran en una sección especial de la implementación, entre las palabras DECLARE (declarar) y BEGIN. Después vienen todas las operaciones que la función debe realizar, las cuales se escriben entre las palabras BEGIN (inicio) y END (fin).

En el *Script* 12.9 se muestra que a la variable hoy se le asigna la fecha del momento en que se invoque la función NOW; en otras palabras, la fecha actual. Luego, a edad le corresponde la cantidad de años que hay entre la fecha que se proporciona como argumento y la almacenada en hoy. Finalmente, se indica que el valor devuelto por cada ejecución de la función será el que tenga la variable edad.

Para concluir, es necesario explicar el propósito de las dos etiquetas $BODY$. En todas las sentencias SQL utilizadas hasta el momento, el punto y coma especifica el final de cada instrucción, pero para definir los subprogramas se necesita utilizar dicho signo dentro del cuerpo del mismo subprograma, lo cual causaría cierta confusión al DBMS en el momento de interpretar el *script*. Por este motivo se requiere indicarle, mediante las etiquetas de $BODY$ (tanto de inicio como de fin), que temporalmente no interprete el punto y coma como final de la instrucción que crea la función. De este modo se le dice al software que entre estas dos etiquetas se puede utilizar el punto y coma sin que termine la instrucción que crea la función (es importante aclarar que, realmente, entre los dos signos pesos, $, puede incluirse cualquier

palabra). Luego del cierre de la etiqueta $BODY$ se vuelve a interpretar el punto y coma como delimitador del final de la instrucción, tal como se ve cuando se especifica que el lenguaje que se está utilizando es PL/PGSQL y se termina con «;».

Una vez creada la función, puede ser invocada siempre que se necesite y siempre que se le proporcionen los parámetros apropiados (si los hay); en este caso, una fecha. También cabe tener presente que la forma más simple de invocar una función es hacerlo directamente en una sentencia SQL. En el *Script* 12.10 se muestra que si se inserta el valor *«08/08/1987»* a la función edad, el resultado obtenido será el valor de *«33»*.

***Script* 12.10**

```
SELECT edad('08/08/1987')
```

Recuerde que como la función edad devuelve un entero, puede ser utilizada en cualquier lugar donde se acepte un escalar. Por consiguiente, es posible consultar la tabla `Usuarios` y proporcionarle como argumento la columna `fechaNacimiento`, como en el *Script* 12.11.

***Script* 12.11**

```
SELECT nombres, edad(fechaNacimiento)
FROM Usuarios
```

Este primer ejemplo, aunque sencillo, ayuda a tener claras las generalidades necesarias para crear una función. Ahora analice la siguiente necesidad planteada:

¿Cuál es el nombre y el estado de la vida artística de todos los intérpretes?
El estado de vida artística es: para los retirados, la cantidad de años que estuvieron activos; para los que no se han retirado, la cantidad de años que llevan activos.

En esta nueva solicitud se requieren el nombre de los intérpretes, que se encuentra en la columna nombre de la tabla `Intérpretes`, y el estado, el cual se determina de la siguiente forma:

- Si no se ha retirado aún, el valor del estado será un mensaje de texto que dirá: *«En ejercicio, tiene X años de vida artística»*, donde X será igual a la diferencia entre el año de la fecha actual y el año en que inició su vida artística.

- Si el intérprete ya se retiró, el valor del estado será un mensaje de texto que dirá *«Retirado luego de X años de vida artística»*, donde X será la diferencia entre el año en que se retiró y el año en que inició su vida artística.

***Script* 12.12**

```
CREATE OR REPLACE FUNCTION estado(lanzamiento INT, retiro INT)
RETURNS varchar(50)
AS
$BODY$
        DECLARE
        msg VARCHAR(50);
        años INTEGER;
        BEGIN
                IF retiro is null THEN
                     años := retiro - EXTRACT(YEAR FROM NOW());
                     msg:= CONCAT('En ejercicio, tiene ', años, ' años de vida
        artística');
                ELSE
                     años := retiro - lanzamiento;
                     msg:= CONCAT('Retirado luego de ', años, ' años de vida
        artística');
        END IF;
        return msg;
        END;
$BODY$
language 'plpgsql';
```

Teniendo claro lo anterior, es evidente que el valor del estado depende de si el intérprete ya se retiró o no se ha retirado. En la tabla `Intérpretes` esta condición se conoce según el valor de la columna `retiro`: si tiene cualquier valor, es decir, si no se ha insertado el descriptor `NULL`, es porque el intérprete se retiró en el año indicado allí; si, por el contrario, la columna tiene el descriptor `NULL`, es porque el artista no se ha retirado aún.

Para abordar esta pregunta se puede hacer uso de complejas expresiones `CASE` con expresiones anidadas que, sin embargo, dificultan la legibilidad y son propensas a errores. Por otra parte, es posible descomponer el problema en dos partes, siguiendo el siempre presente principio de divide y vencerás. Para hacerlo conviene crear una función que tome el año de lanzamiento y el año de retiro de un intérprete y determine si está retirado o no y, a partir de este cálculo, indique los años de vida artística que lleva o los años que llevaba cuando se retiró, preparando así el mensaje que corresponda según cada caso. Esta lógica se encapsula en la función `estado` del *Script* 12.12.

Esta función tiene un nuevo elemento: el uso de la estructura de control `IF`/`ELSE`, con la cual se puede determinar el flujo de ejecución del programa. La estructura se utiliza de la siguiente forma: se determina si (`IF`) el valor de la variable `retiro` pasada como argumento es `NULL`. Si este es el caso, se calcula la cantidad de años de vida artística que tiene hasta

la fecha en que se invoca la función. En caso contrario (`ELSE`), se calculan los años que el intérprete ejerció su actividad profesional. En cada caso, se construye un mensaje personalizado que es guardado en la variable `msg`, cuyo valor se devuelve posteriormente a quien invoque la función.

La función creada se puede invocar dentro de una sentencia `SELECT` para calcular el estado para cada uno de los intérpretes, como se muestra en el *Script* 12.13.

***Script* 12.13**

```
SELECT nombre, estado (añoLanzamiento, añoRetiro)
FROM Intérpretes;
```

La variedad de elementos que se pueden utilizar dentro del cuerpo de la función es amplia. Por ejemplo, es posible recurrir a las sentencias SQL que se han desarrollado a lo largo de este libro. Analice la siguiente necesidad:

¿Cuáles son los diferentes roles que tiene cada músico dentro de la plataforma «*Mis Canciones*»?
Devolver TRUE para los roles que desempeña o ha desempeñado; en caso contrario, devolver FALSE.

La siguiente necesidad plantea que se desea saber, por cada músico: (a) si es o ha sido solista; (b) si es compositor; (c) si hace parte o ha pertenecido a algún grupo musical. Para determinar esto, lo primero que se hará será crear una función de nombre `Roles` que reciba el identificador del músico y devuelva una fila compuesta por tres columnas: la primera indicará si es solista; la segunda, si es compositor, y la tercera, si ha pertenecido a un grupo. Como se muestra en el *Script* 12.14, en la declaración de la función se especifica que esta utiliza cuatro argumentos: uno de entrada (`IN`), que debe ser proporcionado por quien invoque la función, y tres de salida (`OUT`), que son devueltos por la función a quien la invoque.

En este punto es oportuno mencionar que existe un tercer tipo de argumento, que no se utilizará en este escenario: los de entrada y salida (`INOUT`). Estos, como su nombre lo sugiere, son una mezcla de los dos anteriores: el que invoque la función debe suministrarlo (`IN`), pero también puede ver todos los cambios que se realicen sobre este (`OUT`). Adicionalmente, en la declaración se aprecia que el tipo de valor retornado es `RECORD` o, lo que es lo mismo, una fila compuesta por los tres argumentos de salida definidos.

***Script* 12.14**

```
CREATE OR REPLACE FUNCTION Roles(IN idMúsicoBuscado INT, OUT esSolista BOOLEAN, OUT
esCompositor BOOLEAN, OUT perteneceGrupo BOOLEAN)
RETURNS record
AS
$BODY$
        DECLARE
        idSolista INT;
        idCompositor INT;
        idParticipaciónGrupo INT;
        BEGIN
             -- Búsqueda en la tabla solista
             SELECT idMúsico INTO idSolista
             FROM Solistas
             WHERE idMúsico = idMúsicoBuscado;
             esSolista := idSolista is NOT NULL;
             -- Búsqueda en la tabla Compositores
             SELECT idMúsicoCompositor INTO idCompositor
             FROM Compositores
             WHERE idMúsicoCompositor = idMúsicoBuscado;
             esCompositor := idCompositor is NOT NULL;
             -- Búsqueda en la tabla participación en grupos
             SELECT idMúsico INTO idParticipaciónGrupo
             FROM ParticipacionesEnGrupos
             WHERE idMúsico = idMúsicoBuscado;
             perteneceGrupo := idParticipaciónGrupo is NOT        NULL;
        END;
$BODY$
language 'plpgsql';
```

Una vez declarada la función, su implementación es extremadamente sencilla. Opera de la siguiente forma: busca el identificador del músico suministrado a la función (`idMúsicoBuscado`) en las tablas `Solistas`, `Compositores` y `ParticipacionesEnGrupo`. Por ejemplo, en la variable `idSolista` se guarda el identificador del músico resultante de la búsqueda de dicho intérprete en la tabla `Solista`. Si el valor es encontrado, la variable almacenará un entero que representará al músico; en caso contrario, le asignará el descriptor `NULL`, que indicará que ese músico en particular no es solista. Esta misma operación se repite para las tablas `Compositores` y `ParticipacionesEnGrupo`.

También es clave destacar que a los argumentos de salida (`OUT`) se les ha asignado el valor booleano correspondiente a la comprobación de si se obtuvo el descriptor `NULL` o no. Por ejemplo, a `esSolista` se le asigna `TRUE` si `idSolista` no tiene el descriptor `NULL`; en otras palabras, se le asigna `TRUE` si es solista, y `FALSE` si no lo es. Lo mismo se aplica para las variables `esCompositor` y `perteneceGrupo`. Asimismo, se debe tener en cuenta que, como se han utilizado argumentos de tipo `OUT`, no cabe usar una expresión con `RETURN` puesto que este tipo de argumentos son devueltos automáticamente cuando la función ha llegado a su fin.

Esta función se puede invocar tal como se presenta en el *Script* 12.15. En este caso se obtendrá un fila o registro de la siguiente forma: TRUE, TRUE, FALSE.

***Script* 12.15**

```
SELECT Roles(1);
```

Como lo devuelto por la función es un registro, se puede utilizar como argumento de la cláusula FROM, como se muestra en el *Script* 12.16.

***Script* 12.16**

```
SELECT * FROM Roles(1);
```

Finalmente, si se desea acceder a solo un valor dentro del registro, se puede hacer como lo indica el *Script* 12.17.

***Script* 12.17**

```
SELECT (Roles(1)).esSolista;
```

Estos procesos se pueden combinar con una consulta a la tabla Músicos para comprobar los roles que tiene cada uno de los músicos registrados en la plataforma *«Mis Canciones»*, como se muestra en el *Script* 12.18.

***Script* 12.18**

```
SELECT (Roles(idMúsico)).esSolista, (Roles(idMúsico)).esCompositor, (Roles(idMúsico)).
perteneceGrupo
FROM Músicos
```

De esta manera se ha devuelto una fila, pero ¿qué sucede si se quiere obtener más de una fila, es decir, una tabla? La siguiente necesidad permitirá explorar ese caso haciendo uso de una función que devuelve una tabla:

¿Cuáles son las canciones que pertenecen a un género de nombre «*X*»?

Para este requerimiento es preciso crear una consulta donde el conjunto de datos devuelto sea el que coincida con un valor específico de alguna de sus columnas, algo que en este

punto ya se ha creado muchas veces. Por ejemplo, para saber cuáles son las canciones del género «*Salsa*» se crea una consulta como la del *Script* 12.19.

Ahora bien, aunque la consulta del *Script* 12.19 es útil, no satisface lo que se propone, es decir, si se quiere cambiar y obtener ahora las canciones de género «*Vallenato*», no se debería escribir nuevamente la consulta. Una función puede ayudar a cumplir con este objetivo.

***Script* 12.19**

```
SELECT *
FROM Canciones
WHERE idGénero = (SELECT idGénero FROM Géneros WHERE nombre = 'Salsa');
```

La función necesaria para cumplir con lo solicitado sería una que reciba como argumento de entrada el nombre del género que se desea buscar y devuelva un conjunto de datos o filas (SETOF) de la tabla `Canciones`, como la del *Script* 12.20. Así, teniendo clara la declaración de la función, en la implementación solo hace falta retornar el resultado de la consulta (RETURN QUERY) del *Script* 12.19, utilizando, en lugar del valor específico del género «*Salsa*», el que se reciba en el argumento género.

***Script* 12.20**

```
CREATE OR REPLACE FUNCTION CancionesPorGéneroEIntérprete (IN género VARCHAR(20))
RETURNS SETOF Canciones
AS
$CUERPO$
BEGIN
        RETURN QUERY
        SELECT *
        FROM Canciones
        WHERE idGénero = (SELECT idGénero FROM Géneros WHERE nombre = género);
END
$CUERPO$
Language 'plpgsql';
```

Una vez creada la función, puede invocarse con diferentes valores del argumento género. Este el caso del *Script* 12.21, que lo hace con el género «*Vallenato*», y del *Script* 12.22, para el género «*Salsa*».

***Script* 12.21**

```
SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato') ;
```

***Script* 12.22**

```
SELECT * FROM CancionesPorGéneroEIntérprete('Salsa') ;
```

En este escenario se ha devuelto un conjunto de datos que tienen la misma estructura de la tabla `Canciones`. Sin embargo, puede que el conjunto de datos que se necesita no tenga la estructura de ninguna de las tablas de la base de datos. Analice el siguiente caso:

¿Cuál el título de la canción y del álbum de aquellas canciones que pertenecen a un género de nombre «*X*»?

En esta solicitud se especifica que el conjunto de datos devuelto por la función debe tener una estructura particular que no se ajusta a la de ninguna de las tablas definidas en la base de datos. Por esta razón se utilizará un conjunto de filas genéricas (`SETOF RECORD`). Asimismo, otro cambio necesario será modificar la consulta, que esta vez devuelve únicamente el título tanto de la canción como del álbum de todas las canciones de un género especificado. En el *Script* 12.23 se muestra cómo se crearía la función.

***Script* 12.23**

```
CREATE OR REPLACE FUNCTION CancionesPorGéneroEIntérprete (IN género VARCHAR(20))
RETURNS SETOF RECORD
AS
$CUERPO$
BEGIN
        RETURN QUERY
        SELECT Canciones.título, Álbumes.título
        FROM Canciones INNER JOIN Álbumes ON Canciones.idÁlbumOriginal = Álbumes.
        idAlbum
        WHERE idGénero = (SELECT idGénero FROM Géneros WHERE nombre = género);
END
$CUERPO$
Language 'plpgsql';
```

Para utilizar la función hay que considerar que, como se ha incluido el objeto genérico `RECORD`, en el momento de invocarla se les debe dar nombre y tipo a las columnas que se desea obtener, como se muestra en el *Script* 12.24.

No obstante, invocar la función de esta forma resultaría bastante tedioso, por lo que existen alternativas como especificar en la definición cuál es la estructura de las filas, conforme lo hace el *Script* 12.25. En este caso se han definido como argumentos de salida el `título` y el `álbum`,

ambos de tipo VARCHAR, para que den forma al conjunto de datos que se obtendrá. Por lo demás, la implementación de la función no varía con respecto a la del ejemplo anterior.

***Script* 12.24**

```
SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato') f(título
VARCHAR(100), álbum VARCHAR(100));
```

***Script* 12.25**

```
CREATE OR REPLACE FUNCTION CancionesPorGéneroEIntérprete (IN género VARCHAR(20), OUT
título VARCHAR(100), OUT Álbum VARCHAR(100))
RETURNS SETOF RECORD
AS
$CUERPO$
BEGIN
        RETURN QUERY
        SELECT Canciones.título, Álbumes.título
        FROM Canciones INNER JOIN Álbumes ON Canciones.idÁlbumOriginal = Álbumes.
        idAlbum
        WHERE idGénero = (SELECT idGénero FROM Géneros WHERE nombre = género);
END
$CUERPO$
Language 'plpgsql';
```

Teniendo ya una estructura, la invocación de la función resulta más sencilla, tal como se puede ver en el *Script* 12.26.

***Script* 12.26**

```
SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato')
```

Para finalizar con las funciones, revise la siguiente necesidad:

¿Cuál es la cantidad de reproducciones que han realizado los usuarios en cada parte del día (mañana, tarde o noche) según un sexo «*X*» y un rango de edad especificado?
Considérese mañana desde la primera hora del día hasta las ocho; tarde, desde las nueve hasta las dieciocho; y noche, desde las diecinueve hasta la última hora del día.

Esta solicitud indica explícitamente que el conjunto de datos devuelto varía de acuerdo con tres valores: un valor del sexo y otros dos que conforman un rango proporcionado cuando se invoca la función. Por esta razón, al declarar la función en el *Script* 12.27 se plantearon tres argumentos de entrada (`IN`): el sexo, el límite inferior del rango y el límite superior. Sin embargo, este *script* tiene nuevos elementos que se detallan a continuación:

- En la declaración de los argumentos se incluye la palabra `DEFAULT` para indicar que se utilizará un valor por defecto en caso de que ningún valor se proporcione. Para el argumento `sexoBuscado`, dicho valor es *«F»*.
- Se definió que se debe retornar una tabla conformada por cinco columnas: `idUsuario`, `totalReproducciones`, `reproduccionesMañana`, `reproduccionesTarde` y `reproduccionesNoche`. Todas ellas son de tipo entero.
- Se incluyó la definición de variables que representan la fila de una de las tablas de la base de datos; concretamente, una fila de `Usuarios` y otra de `Reproducciones`. Esto se consigue con la expresión `<Nombre de Tabla>%rowtype`. Por ejemplo, en el caso de la tabla `Reproducciones` sería `Reproducciones%rowtype`. De esta forma se cuenta con una variable que puede almacenar una fila con la estructura de la tabla `Canciones`.
- Se utilizó uno de los tipos de los ciclos soportados por el lenguaje: `FOR IN LOOP`, el cual permite recorrer las filas devueltas por consultas a las tablas `Usuarios` y `Reproducciones`. En cada caso se utilizan las filas que fueron previamente definidas con la expresión `<Nombre de Tabla>%rowtype` para acceder a la fila devuelta por la consulta. El cuerpo del ciclo estará siempre delimitado entre `LOOP` y `END LOOP`.
- Con la expresión `RETURN NEXT` se añadieron las filas a la tabla definida como conjunto de datos resultante. De este modo es posible asignar valores directamente a las columnas que conforman dicha tabla y, una vez se ejecute `RETURN NEXT`, añadir una fila con los valores que tengan las variables que representan las columnas en ese momento.
- Con la expresión `RETURN` se devuelve la tabla con todas las filas que han sido añadidas y se termina la ejecución de la función.
- Se añadieron comentarios de una línea utilizando el símbolo «-». Para añadir comentarios de más de una línea se debe empezar con «/*» y terminar con «*/».

Con la presentación de esta función concluye el acercamiento a las funciones, recordando que existen otras estructuras de control y repetición que pueden ser utilizadas para crear subprogramas que faciliten tareas cuando se combinan con imaginación y creatividad.

***Script* 12.27**

```
CREATE OR REPLACE FUNCTION ReporteUsuario (IN sexoBuscado VARCHAR(30) DEFAULT 'F', IN
limiteInferiorEdad INT DEFAULT 18, IN limiteSuperiorEdad INT DEFAULT 45 )
RETURNS TABLE (idUsuario INT, totalReproducciones INT, reproduccionesDía INT, reproduccionesTarde
INT, reproduccionesNoche INT)
AS
$CODE$
DECLARE
filaUsuario Usuarios%rowtype;
filaReproducción Reproducciones%rowtype;
cReproducciones INT := 0;
crDía INT := 0;
crTarde INT := 0;
crNoche INT := 0;
BEGIN
        -- recuperación de los usuarios
        FOR filaUsuario IN SELECT *
                                    FROM Usuarios
                                    WHERE (Usuarios.sexo = sexoBuscado) AND (edad(Usuarios.
        fechaNacimiento) BETWEEN limiteInferiorEdad AND limiteSuperiorEdad ) LOOP
        -- recuperación de las canciones para cada usuario
                    FOR filaReproducción IN SELECT *
                                    FROM Reproducciones
                                    WHERE Reproducciones.idUsuario = filaUsuario.idUsuario LOOP
                            -- contar la cantidad total de reproducción
                            cReproducciones := cReproducciones + 1;
                            -- contar la cantidad de reproducción según la época
                            CASE
                                    WHEN EXTRACT (hour FROM filaReproducción.horaReproducción)
        BETWEEN 0 AND 8 THEN crDía := crDía + 1;
                                    WHEN EXTRACT (hour FROM filaReproducción.horaReproducción)
        BETWEEN 9 AND 18 THEN crTarde := crTarde + 1;
                                    WHEN EXTRACT (hour FROM filaReproducción.horaReproducción)
        BETWEEN 19 AND 23 THEN crNoche := crNoche + 1;
                                END CASE;
                            END LOOP;
            -- guardar los valores en la tabla
            idUsuario := filaUsuario.idUsuario;
            totalReproducciones := cReproducciones;
            reproduccionesDía := crDía;
            reproduccionesTarde := crTarde;
            reproduccionesNoche := crNoche;
            RETURN NEXT;
            -- reiniciar los contadores
            cReproducciones := 0;
            crDía := 0;
            crTarde := 0;
            crNoche := 0;
      END LOOP;
      RETURN;
END
$CODE$
LANGUAGE 'plpgsql';
```

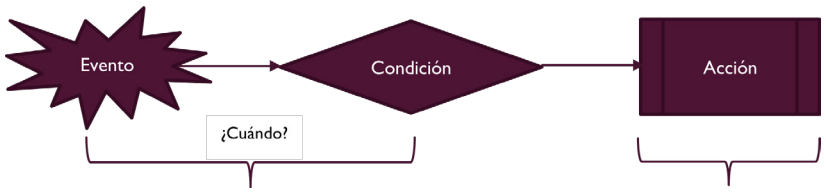
12.3. Disparadores

En la sección anterior se abordó el uso de las funciones, y en esta oportunidad se aprovecharán los conceptos presentados sobre ellas para gestionar eventos que se puedan dar dentro de una base de datos. Si bien hay diferentes tipos de sucesos que afectan a los objetos de estas, aquí se hará énfasis en los del sublenguaje DML que repercuten en las tablas, es decir, `INSERT`, `UPDATE` y `DELETE`.

Cuando este tipo de eventos ocurren, es normal que se necesite gestionarlos para, por ejemplo, realizar una acción asociada a ellos, comprobar ciertas reglas antes de permitir que estos realicen cambios, entre otras. Una forma que tienen las bases de datos relacionales para hacer dicha gestión son los disparadores (*triggers*, por el término en inglés).

En la práctica se suelen utilizar los disparadores para hacer cumplir algunas restricciones de integridad referencial, para hacer cumplir restricciones complejas o para auditar cambios en los datos. Estos elementos están basados en el modelo evento-condición-acción presentado en la Figura 12.2, donde el evento y las condiciones representan qué suceso desencadenará la ejecución del disparador, y la acción indica cuáles son las acciones que realizará el disparador una vez se ha activado.

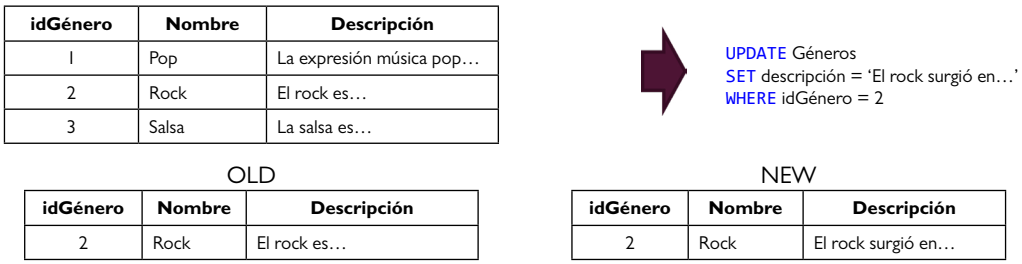
Figura 12.2. Esquema general del funcionamiento de un disparador



Básicamente, hay dos tipos de disparadores: a nivel de fila (`FOR EACH ROW`), que se ejecutan para cada fila de la tabla que se ve afectada por el evento desencadenante, y a nivel de instrucción (`FOR EACH SENTENCE`), que se ejecutan solo una vez, incluso si hay varias filas que se ven afectadas por el evento desencadenante. Los disparadores también pueden suceder en cadena cuando la acción del disparador sobre una tabla provoca que se dispare otro evento que tiene a su vez otro disparador asociado.

En el contexto de ejecución de un disparador, se dispone de dos variables cuyos valores cambian en función del evento desencadenante: `OLD` y `NEW`. Ellas representan el estado de la fila en la tabla antes o después del evento, respectivamente, y sus valores son punteros que representan la fila completa. Para comprender mejor esto, considere una operación `UPDATE`: en este caso, `OLD` contiene los valores de la fila ya presentes en la tabla, mientras que `NEW` comprende los nuevos valores que tendrá dicha fila después de la operación. En la Figura 12.3 se muestra este ejemplo.

Figura 12.3. Demostración del funcionamiento de las variables NEW y OLD



Como resulta obvio, el valor que tienen las variables OLD y NEW depende del evento que desencadene al disparador. Es así como en un evento `INSERT` solo tiene sentido la variable NEW, que representa los valores que se están insertando, mientras que OLD no. En la Tabla 12.1 se muestra la relación entre los eventos y las dos variables.

Tabla 12.1. Relación de los eventos `INSERT`, `UPDATE` y `DELETE` con las variables NEW y OLD

Evento	NEW	OLD
`INSERT`	Presente	Ausente
`UPDATE`	Presente	Presente
`DELETE`	Ausente	Presente

Con la siguiente situación se abordará la creación de un primer disparador:

¿Cómo controlar que la cantidad máxima de reproducciones sean 100 por usuario?

En esta situación planteada se debe controlar que un usuario no tenga más de 100 reproducciones o, en términos de filas y tablas, que en la tabla `Reproducciones` no haya más de 100 filas asociadas al mismo identificador del usuario. Este requisito se puede manejar con los disparadores, planteando que, antes (`BEFORE`) de insertar (`INSERT`) en (`ON`) la tabla `Reproducciones`, por cada fila (`FOR` EACH `ROW`) se compruebe si esa nueva fila a ingresar excedería las 100 reproducciones.

***Script* 12.28**

```
CREATE TRIGGER t_controlReproducciones
BEFORE INSERT on Reproducciones
FOR EACH ROW
EXECUTE PROCEDURE f_controlReproducciones();
```

En el *Script* 12.28 se muestra cómo se puede crear el disparador (`CREATE TRIGGER`). El control para que no se inserte la nueva fila si excede el número 100 para el mismo usuario es ejercido por la función `f_controlReproducciones`.

El control ejercido por la función es bastante sencillo: lo primero que hace, como se ve en el *Script* 12.29, es determinar la cantidad de filas asociadas al usuario antes de que este realice la inserción en la tabla `Reproducciones`. Ese valor es guardado en la variable `cantidad`, de tipo entero. Luego se comprueba, con la estructura de control `IF`/`ELSE`, si ese dato es menor que 100. De ser así, se indica que se puede continuar con la inserción (`RETURN NEW`); en caso contrario, se cancela dicha operación (`RETURN NULL`).

***Script* 12.29**

```
CREATE OR REPLACE FUNCTION f_controlReproducciones()
RETURNS trigger as
$$
DECLARE
cantidad INT := 0;
BEGIN
          SELECT COUNT(*) INTO cantidad
          FROM Reproducciones
          WHERE Reproducciones.idUsuario = NEW.idUsuario;
          IF cantidad < 100 THEN
              RETURN NEW;
              RAISE NOTICE 'insertando';
          END IF;
          RAISE NOTICE 'no insertando';
          RETURN NULL;
END;
$$
LANGUAGE 'plpgsql';
```

Ahora, cada vez que se realice una inserción sobre la tabla `Reproducciones`, como se muestra en el *Script* 12.30, se ejecutará la acción definida por el disparador. Si la fila por insertar representa el número 100 o inferior, se guardará con éxito; de corresponder a una cifra mayor a 100, no se realizará la inserción.

***Script* 12.30**

```
INSERT INTO Reproducciones
VALUES(500, (SELECT idUsuario
FROM Usuarios
WHERE email = 'panteismo@gmail.com'), (SELECT idCanción
FROM Canciones
WHERE título = 'La bicicleta'), CAST (NOW() AS DATE), CAST (NOW() AS TIME), 189);
```

Ahora se abordará la siguiente necesidad:

¿Cómo controlar que, cuando un usuario actualice su contraseña, la nueva contraseña no sea igual a la anterior?

Como se comentó, cuando se va a actualizar una fila (UPDATE), se encuentran disponibles las dos variables: una que referencia a los valores antes (OLD) de la actualización y otra que contiene los valores posteriores (AFTER) al evento. Recuerde también que ambas variables representan una fila de la tabla que está siendo afectada; en este caso, la tabla `Usuarios`. Por lo tanto, con ambas variables se tiene acceso a los valores que toman las columnas para esa fila en particular.

Así las cosas, `OLD.contraseña` almacena el valor de la contraseña antes de ser actualizado, y `NEW.constraseña`, el nuevo valor que se quiere dar a la contraseña. Por consiguiente, lo que se debe hacer es comprobar la igualdad de estos dos valores: si existe, no se realiza la actualización; si son diferentes, se prosigue con el evento. En el *Script* 12.31 está la función que manejará esta regla.

***Script* 12.31**

```
CREATE OR REPLACE FUNCTION f_modificarConstraseña()
RETURNS trigger as
$$
DECLARE
cantidad INT := 0;
BEGIN
        IF NEW.contraseña <> OLD.contraseña THEN
              RAISE NOTICE 'Cambiando';
              RETURN NEW;
        END IF;
        RAISE NOTICE 'no se puede cambiar';
        RETURN NULL;
END;
$$
LANGUAGE 'plpgsql';
```

La función del *Script* 12.31 está asociada al disparador definido en el *Script* 12.32, el cual es ejecutado antes (BEFORE) de cada actualización (UPDATE).

***Script* 12.32**

```
CREATE TRIGGER t_modificarConstraseña
BEFORE UPDATE on Usuarios
FOR EACH ROW
EXECUTE PROCEDURE f_modificarConstraseña();
```

Con el disparador creado, cada vez que se intente realizar una actualización como la del *Script* 12.33 se ejecutará la función que comprueba que el nuevo valor de la contraseña no sea igual al anterior.

***Script* 12.33**

```
UPDATE Usuarios
SET contraseña = 'new password'
WHERE idUsuario = 1;
```

La explicación sobre el uso de los disparadores se puede culminar con la siguiente necesidad, en un contexto donde la eliminación en cascada no está habilitada:

¿Cómo eliminar todas las filas asociadas a un usuario en el momento de su eliminación?

En el capítulo 7 se mostró que en algunos escenarios, cuando se pretende eliminar un registro de una tabla y hay registros en otras tablas que hacen referencia a este, el DBMS arroja un mensaje que indica que antes de borrar esa fila es preciso hacer lo propio con todas las filas de las otras tablas que hacen referencia a la que se va a quitar. También se mencionaron dos formas de lograr ese objetivo: una es eliminar las filas que hacen referencia a la que se desea eliminar tabla por tabla, y otra es utilizar la palabra reservada CASCADE para que haga esta tarea. Sin embargo, el uso de la eliminación en cascada, por ser en cierto modo peligroso, suele no estar habilitado por defecto en los DBMS, por lo que una forma de simular este comportamiento es a través de disparadores, como se verá a continuación.

En el *Script* 12.34 se crea un disparador que dice que antes de eliminar una fila de la tabla `Usuarios` se deben ejecutar las acciones encapsuladas en la función `f_eliminarUsuario`.

***Script* 12.34**

```
CREATE TRIGGER t_eliminarUsuario
BEFORE DELETE on Usuarios
FOR EACH ROW
EXECUTE PROCEDURE f_eliminarUsuario();
```

El cuerpo de la función es sencillo. Como se ve en el *Script* 12.35, lo que hace es eliminar todas las filas de las tablas `CancionesLista`, `ListasReproducción`, `Reproducciones`, `Calificaciones` y `Pagos`, y luego, con RETURN OLD indica que se puede continuar con la fila de la tabla `Usuarios` que se está intentando borrar. Es clave destacar que los disparadores operan integrados como una transacción al evento que los activa, por lo que, si alguna de las instrucciones contenidas en la función falla, se cancelará toda la operación.

***Script* 12.35**

```
CREATE OR REPLACE FUNCTION f_eliminarUsuario()
RETURNS trigger as
$$
BEGIN
        DELETE FROM CancionesLista
        WHERE (idListaReproducción IN (SELECT idListaReproducción FROM
        ListasReproducción WHERE idUsuario = OLD.idUsuario));
        RAISE NOTICE 'Se eliminaron las filas de la tabla CancionesLista';
        DELETE FROM ListasReproducción
        WHERE idUsuario = OLD.idUsuario;
        RAISE NOTICE 'Se eliminaron las filas de la tabla ListasReproducción';
        DELETE FROM Reproducciones
        WHERE idUsuario = OLD.idUsuario;
        RAISE NOTICE 'Se eliminaron las filas de la tabla Reproducciones';
        DELETE FROM Calificaciones
        WHERE idUsuario = OLD.idUsuario;
        RAISE NOTICE 'Se eliminaron las filas de la tabla Calificaciones';
        DELETE FROM Pagos
        WHERE idUsuario = OLD.idUsuario;
        RAISE NOTICE 'Se eliminaron las filas de la tabla Pagos';
        RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';
```

Una vez creado el disparador, cada vez que se intente eliminar una fila de la tabla `Usuarios` con una sentencia como la presentada en el *Script* 12.36 se borrarán todas las filas que hacen referencia a ella.

***Script* 12.36**

```
DELETE FROM Usuarios WHERE idUsuario = 2;
```

Si en algún momento se busca eliminar un disparador, se pueden emplear las palabras reservadas `DROP TRIGGER`, especificando en qué tabla está el disparador que se desea borrar, como se ve en el *Script* 12.37.

***Script* 12.37**

```
DROP TRIGGER t_eliminarUsuario ON Usuarios;
DROP FUNCTION f_eliminarUsuario;
```

12.4. Procedimientos almacenados

Al inicio del capítulo se afirmó que los subprogramas son de dos tipos: funciones y procedimientos. En un comienzo la atención se centró en las funciones, y ahora es el turno de los procedimientos.

Una primera gran diferencia que tienen los procedimientos con respecto a las funciones es que los primeros no tienen que retornar un valor, como sí pasa con las segundas. Traducido a los argumentos que utilizan, los procedimientos solo pueden ser de entrada (`IN`, el valor predeterminado) e `INOUT` (cuando un procedimiento proporciona una salida al llamador). No hay una dirección `OUT` explícita.

La única forma de devolver valores a la persona que llama es mediante parámetros `INOUT`. En este caso, se obtendrá una única tupla con el valor final de los datos especificados. Un procedimiento no tiene ningún valor de retorno, por lo que el uso de una instrucción `RETURN` con un valor que no sea `NULL` da como resultado un error de ejecución. Si el código necesita salir, entonces `RETURN NULL` es la declaración correcta para usar. De lo contrario, el código puede llegar al final sin tales declaraciones.

De forma análoga a los casos anteriores, los procedimientos se crean con la instrucción `CREATE PROCEDURE`, que también admite la cláusula `OR REPLACE`. Un procedimiento tiene un nombre, una lista de argumentos y un cuerpo de implementación. El siguiente escenario permitirá comprender en qué circunstancias los procedimientos pueden ser útiles:

¿Cómo insertar un nuevo intérprete y crear el género al cual está asociado?

En este caso se solicita realizar dos inserciones en una misma llamada:

- La del nuevo género que se desea crear en la tabla `Géneros`.
- La del nuevo intérprete en la tabla `Intérpretes`, asociado al género que acaba de crearse.

En el *Script* 12.38 se muestra que se crea un procedimiento de nombre `insertarIntérprete`, que recibe siete argumentos y que no retorna ningún valor. Posteriormente, en el cuerpo del procedimiento se realizan las dos inserciones solicitadas haciendo uso de simples sentencias `INSERT`: una para la tabla `Géneros` y otra para la tabla `Intérpretes`, proporcionándole a cada caso los argumentos que necesita según lo requieran.

Un procedimiento se invoca mediante la instrucción `CALL`, y no se puede utilizar en instrucciones SQL regulares como `SELECT`, `INSERT`, `UPDATE` o `DELETE`. De hecho, si se llama a un procedimiento a través de una de las declaraciones anteriores, PostgreSQL abortará la ejecución e informará de un error que especifica que se debe usar la declaración `CALL`. En el *Script* 12.39 se muestra cómo invocar el procedimiento `insertarIntérprete`.

***Script* 12.38**

```
CREATE OR REPLACE PROCEDURE
insertarIntérprete(idIntérprete INT, nombre VARCHAR(200), añoLanzamiento INT,
añoRetiro INT, tipoIntérprete VARCHAR(10), idGénero INT, nombreGénero VARCHAR(100))
AS
$CODE$
BEGIN
        RAISE NOTICE 'Insertando género %', nombreGénero;
              INSERT INTO Géneros(idGénero, nombre, descripción)
              VALUES(idGénero, nombreGénero, NULL);
        RAISE NOTICE 'Género % insertado', nombreGénero;
              RAISE NOTICE 'Insertando intérprete %', nombre;
              INSERT INTO Intérpretes
              VALUES (idIntérprete, nombre, añoLanzamiento, añoRetiro, tipoIntérprete,
        idGénero);
        RAISE NOTICE 'Intérprte % insertado', nombre;
END
$CODE$
LANGUAGE 'plpgsql';
```

***Script* 12.39**

```
CALL insertarIntérprete(25, 'Mr Black', 1990, NULL, 'Solista', 90, 'Champeta' );
```

Los procedimientos operan por defecto en su totalidad como una transacción, es decir, o se ejecuta correctamente todo su contenido o se deshacen los cambios que se hayan podido realizar. En este punto conviene entonces formalizar a qué se refiere el concepto de transacciones mencionado a lo largo del libro, en el contexto de las bases de datos relacionales.

El modelo relacional describe la unidad lógica de procesamiento de datos como la transacción, la cual se puede definir como un conjunto de operaciones realizadas en secuencia y que operan como un mecanismo de bloqueo para garantizar su integridad. Así pues, una transacción funciona con éxito si todas las operaciones incluidas en ella se ejecutan correctamente; si una operación en una transacción falla, el efecto de las operaciones ejecutadas parcialmente se puede deshacer. En la Figura 12.4 se muestra de modo gráfico una transacción compuesta por cuatro operaciones: dos `INSERT`, un `DELETE` y un `UPDATE`.

Figura 12.4. Una transacción con cuatro operaciones (`INSERT`, `INSERT`, `DELETE` y `UPDATE`) atómicas

Otra diferencia importante entre los procedimientos y las funciones es que estas últimas no pueden interactuar con el nivel de la transacción en la que se están ejecutando (por ejemplo, para guardar valores parcialmente), mientras que los procedimientos almacenados sí tienen esa posibilidad y son capaces de realizar cambios en la base de datos, aunque no se complete con éxito su ejecución. Este funcionamiento se puede comprender a la luz de la siguiente necesidad:

¿Cómo insertar un nuevo intérprete y crear el género al cual está asociado?
Si falla la inserción del intérprete, no se debería deshacer la inserción del género.

Para interactuar con la transacción dentro de los procedimientos almacenados se puede usar, por ejemplo, la palabra `COMMIT`. De esta forma el DBMS recibe la instrucción de guardar todos los cambios que el procedimiento haya realizado hasta ese punto. Puesto en práctica, sería como se muestra en el *Script* 12.40. En este se aprecia cómo se guardan los cambios una vez el género ha sido insertado, de forma que, si la inserción del intérprete falla, el ajuste realizado en la tabla `Género` perdurará, a diferencia del caso anterior, donde se perdía el cambio.

***Script* 12.40**

```
CREATE OR REPLACE PROCEDURE
insertarIntérprete(idIntérprete INT, nombre VARCHAR(200), añoLanzamiento INT,
añoRetiro INT, tipoIntérprete VARCHAR(10), idGénero INT, nombreGénero VARCHAR(100))
AS
$CODE$
BEGIN
        RAISE NOTICE 'Insertando género %', nombreGénero;
              INSERT INTO Géneros(idGénero, nombre, descripción)
              VALUES(idGénero, nombreGénero, NULL);
        RAISE NOTICE 'Género % insertado', nombreGénero;
              -- Guardar cambios
              COMMIT;
              RAISE NOTICE 'Insertando intérprete %', nombre;
              INSERT INTO Intérpretes
              VALUES (idIntérprete, nombre, añoLanzamiento, añoRetiro, tipoIntérprete,
        idGénero);
        RAISE NOTICE 'Intérprte % insertado', nombre;
END
$CODE$
LANGUAGE 'plpgsql';
```

Es importante resaltar que, si el procedimiento termina su ejecución sin contratiempo, realiza un `AUTOCOMMIT` y guardará todos los cambios que se realicen. También existe una forma de indicar que se desea deshacer las modificaciones hechas durante la ejecución del procedimiento, mediante la palabra reservada `ROLLBACK`. Por ejemplo, el *Script* 12.40 se puede ajustar para deshacer de forma voluntaria los cambios cuando el intérprete que se quiere insertar ya existe. Entonces el *Script* 12.41 realiza las siguientes interacciones con la transacción:

- Guarda los cambios si, luego de agregar el género, se comprueba que el intérprete que se está intentando insertar no existe aún en la tabla `Intérpretes`.
- Hace un `ROLLBACK` si, luego de insertar el género, se comprueba que el intérprete que se desea agregar ya existe en la tabla `Intérpretes`. Es decir, si este nuevo dato violaría la llave principal, el cambio realizado en la tabla `Géneros` se deshace.
- Si luego de insertar el género y realizar el `COMMIT` se logra insertar el intérprete y el procedimiento almacenado llega a su fin sin contratiempos, se realizará un `AUTOCOMMIT`.
- Si luego de insertar el género y realizar el `COMMIT` la inserción en la tabla `Intérpretes` falla, se realizará un `ROLLBACK` automático que afectará solo los cambios que se hayan realizado después del último `COMMIT`.

***Script* 12.41**

```
CREATE OR REPLACE PROCEDURE
insertarIntérprete(idIntérprete INT, nombre VARCHAR(200), añoLanzamiento INT,
añoRetiro INT, tipoIntérprete VARCHAR(10), idGénero INT, nombreGénero VARCHAR(100))
AS
$CODE$
DECLARE
intérprete Intérpretes%rowtype;
BEGIN
        RAISE NOTICE 'Insertando género %', nombreGénero;
        INSERT INTO Géneros(idGénero, nombre, descripción)
        VALUES(idGénero, nombreGénero, NULL);
        RAISE NOTICE 'Género % insertado', nombreGénero;
        SELECT * INTO intérprete
        FROM Intérpretes
        WHERE Intérpretes.idIntérprete = $1;
        IF intérprete.idIntérprete IS NULL THEN
           RAISE NOTICE 'COMMIT';
           COMMIT;
        ELSE
            RAISE NOTICE 'ROLLBACK';
            ROLLBACK;
        END IF;
        RAISE NOTICE 'Insertando intérprete %', nombre;
        INSERT INTO Intérpretes
        VALUES (idIntérprete, nombre, añoLanzamiento, añoRetiro, tipoIntérprete,
        idGénero);
        RAISE NOTICE 'Intérprte % insertado', nombre;
END
$CODE$
LANGUAGE 'plpgsql';
```

12.5. Aprendizajes más importantes del capítulo 12

- Contar en los DBMS con un lenguaje de programación procedimental aumenta las posibilidades del tipo de operaciones que se pueden realizar en las bases de datos.
- Las vistas permiten definir una capa de abstracción sobre los datos y son especialmente útiles cuando se busca uno de los siguientes atributos: modularidad, reúso y seguridad.
- Aunque el concepto de vistas implica de manera general no almacenar los datos, la evolución de los DBMS y la necesidad del entorno han llevado a que existan variaciones, como las vistas que se pueden almacenar y vistas donde las filas pueden ser cambiadas por instrucciones DML.
- Con el lenguaje de programación procedimental es posible crear subprogramas que a su vez proporcionan beneficios como la división de tareas, la reutilización de código y el manejo de la complejidad.

- Los subprogramas se dividen en dos tipos: las funciones o los procedimientos almacenados. Ambos asocian un nombre a un conjunto de instrucciones que se ejecutarán cada vez que se invoque el subprograma. De igual forma, los dos reciben un conjunto de argumentos que les permiten, por ejemplo, configurar su comportamiento.
- La función puede devolver datos de diferente tipo, bien sea un valor escalar, una fila o un conjunto de resultados (como todas las tuplas de una tabla).
- Los procedimientos almacenados no pueden devolver valores complejos ni de forma explícita. En PostgreSQL la única opción para este fin es utilizar parámetros del tipo `INOUT`.
- Otra gran diferencia es que las funciones no pueden interactuar con la transacción que los engloba, mientras que los procedimientos sí.
- Una transacción representa un conjunto de operaciones que deben ser ejecutadas como una sola unidad lógica donde todas sean exitosas o se deshagan los cambios que hayan podido realizar.
- En los subprogramas es posible utilizar los diferentes constructos habituales de los lenguajes imperativos, como las estructuras de control y las repetitivas.
- Con los disparadores se pueden controlar eventos que afectan a las tablas de la base de datos, de forma se pueda reaccionar ante acciones de inserción, modificación y eliminación.

12.6. Actividades de aplicación para evidenciar lo aprendido

1. Proponga dos vistas que no sean almacenadas físicamente y dos que sí lo sean.
2. Proponga tres funciones que puedan ser utilizadas para resolver posibles necesidades de datos en el contexto de la plataforma *«Mis Canciones»*.
3. Cree una tabla de nombre `Auditoría` que tenga una estructura como la siguiente:

Tabla	Acción	Valores	Fecha	Hora

En esta estructura las columnas representan:

- **Tabla:** ¿en qué tabla se realizó la operación?
- **Acción:** ¿qué acción se realizó sobre la tabla? ¿Una inserción, una modificación o una eliminación?
- **Valores:** ¿cuáles son los valores que han interactuado en la operación?
- **Fecha:** ¿el día en que se realizó la operación?
- **Hora:** ¿a qué hora se realizó la operación?

Glosario

Alias: Nombre alternativo para renombrar tablas y columnas, con la intención de evitar conflictos entre ellas en caso de tener el mismo nombre, simplificar referencias en un JOIN o, simplemente, facilitar y hacer más estética la lectura de la consulta; por ejemplo: «Año» en vez de «anio».

ALTER: Palabra reservada dentro del lenguaje DDL que se utiliza como comando para indicar algún tipo de modificación a la estructura o las propiedades de un elemento dentro de la base de datos, como las tablas, las columnas, las vistas, incluso la misma base de datos, etcétera.

AS: Palabra reservada para asignar un nombre a una columna, tabla, conjunto de datos devuelto por una consulta, entre otros.

ASC: Indicador de orden ascendente.

Atributo: Característica o propiedad representativa de una entidad, es decir, que la describe.

Base de datos relacional: Conjunto de tablas, denominadas relaciones, formadas por filas (registros) y columnas (campos o atributos). Cada tabla debe tener un nombre único, así como también deben ser únicos (dentro de ella) los nombres de cada campo o columna. Es necesario que cada registro cuente con un atributo único, denominado llave o clase, y demás atributos o columnas que caractericen la relación.

BETWEEN: Operador con el cual se permite determinar o comprobar si un valor o expresión se encuentra dentro de un intervalo o rango de valores de manera inclusiva, es decir, se incluyen los valores de inicio y fin (límites).

Cardinalidad: Número de entidades con la cual otra entidad se puede asociar mediante una relación.

CHECK: Cláusula que permite establecer restricciones de integridad en la declaración tanto de relaciones como de dominios.

Cláusula FROM: Permite determinar el origen de la consulta, es decir, el lugar desde donde se tomarán los datos: tabla, subconsulta, conjunto de datos, vistas, etc. Corresponde con la operación producto cartesiano del álgebra relacional. Dentro de la consulta SQL, es la primera operación por realizar, seguida del `WHERE` y, por último, el `SELECT`.

Cláusula SELECT: Se utiliza para recuperar información de la base de datos y proyectar las columnas o atributos que se deseen. De ahí que esta operación corresponda con la operación de proyección del álgebra relacional.

Cláusula WHERE: Permite establecer una o varias condiciones que la consulta en cuestión debe cumplir, limitando los resultados. Dentro del álgebra relacional se expresa como el predicado selección.

Columna: Dentro del contexto de las bases de datos relacionales, las columnas hacen referencia a los atributos del tipo de registro de la entidad que la tabla representa. Dentro de cada tabla, el nombre de cada columna debe ser único, y también es preciso especificar el tipo de datos.

Combinación cruzada: Operación expresada a través de la instrucción `CROSS JOIN` en la que se devuelve o genera una combinación entre cada fila de la primera tabla y cada fila de la segunda. Es una combinación o unión interna, sin condición de unión. También se conoce como unión cartesiana.

Combinación externa: Operación expresada a través de la instrucción `OUTER JOIN` (`LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, `FULL OUTER JOIN`), la cual actúa similar a la combinación interna (incluyendo las filas combinadas que cumplen la condición de unión), pero además permite incluir las filas que no cumplen la condición de combinación según sea el caso elegido.

Combinación interna: Operación expresada a través de la instrucción `INNER JOIN` en la que se devuelven únicamente aquellas filas o registros que tienen valores iguales o exactos en los dos campos (puede ser llave foránea en tabla de destino y llave primaria en tabla de origen) que se comparan para unir ambas tablas.

Combinaciones de tablas: Conocidas como operaciones `JOIN`. Un `JOIN` opera sobre dos tablas y produce como resultado una nueva tabla.

CONSTRAINT: Una `CONSTRAINT` o restricción es una cláusula implementada en las instrucciones `CREATE TABLE` y `ALTER TABLE` para crear o eliminar una regla o restricción. De esta forma, si existe un error durante la ejecución de la consulta a causa de dicha restricción o simplemente no se cumple con esta, la transacción será abortada.

CREATE: Comando perteneciente al lenguaje DDL que permite crear o definir estructuras como bases de datos, tablas, funciones, vistas, etcétera.

Datos: Unidad mínima de información, la cual representa simbólicamente una variable o atributo, sea cuantitativo o cualitativo. Puede ser numérica, alfabética, algorítmica, espacial, entre otros.

DBMS: Del inglés *Data Base Management System*. Hace referencia al SGBD o sistema gestor de bases de datos, y consiste en una colección de datos interrelacionados, normalmente denominados bases de datos, y los softwares o programas que permiten acceder a ellos. El propósito principal de estos sistemas es proporcionar una forma de almacenar y recuperar información.

DCL: Siglas del término en inglés *Data Control Languag*e, que se traduce como lenguaje de control de datos (LCD). Se utiliza para controlar el acceso a los datos contenidos en la base de datos mediante comandos, es decir, otorgar y revocar permisos, función exclusiva del administrador de dicha base.

DDL: Siglas del término en inglés *Data Definition Language*, que se traduce como lenguaje de definición de datos (LDD). Se utiliza para especificar o definir estructuras o propiedades de los datos, así como para modificarlas y eliminarlas.

DELETE: Comando del lenguaje de manipulación de datos (LMD o DML, según el término en inglés: *Data Manipulation Language*) utilizado para borrar tuplas de una relación, es decir, eliminar o borrar registros (filas) de tabla(s) o vista(s).

DESC: Indicador de orden descendente.

Diferencia: Operación de conjunto en la que se devuelven las filas de un conjunto A que no están en el conjunto B. Se realiza a través de la palabra reservada `EXCEPT`.

Diseño conceptual: Permite obtener una visión general detallada de los procesos por modelar. En esta instancia se traducen los requisitos de datos en entidades, las relaciones entre ellas y los atributos de las entidades y las relaciones. También es la fase en la que se busca describir «qué» atributos se quieren capturar en la base de datos y «cómo» agruparlos.

Diseño físico: Etapa que se centra en los detalles de implementación de una base de datos en un DBMS. Entre dichas características se encuentran la forma de organización de los archivos y las estructuras de almacenamiento interno.

Diseño lógico: Fase en la que se relaciona y traduce el diseño conceptual de alto nivel con el modelo de datos de implementación de datos del SGBD, también conocido como «Esquema de base de datos». El modelo más utilizado en esta etapa es el modelo relacional.

Disparadores: También denominados *triggers* (en inglés). Son instrucciones que el sistema ejecuta de manera automática como resultado de una modificación a la base de datos.

DISTINCT: Cláusula que permite eliminar u obviar registros o filas duplicadas en el resultado de una consulta. Esta palabra se añade después el comando `SELECT` y previamente al nombre de la primera columna o atributo por recuperar.

DML: Siglas del término en inglés *Data Manipulation Languag*e, que se traduce como lenguaje de manipulación de datos (LMD). Permite a los usuarios tener acceso a los datos almacenados y gestionarlos: insertar nuevos datos, recuperar información, modificar datos y borrarlos.

DROP: Comando utilizado para eliminar bases de datos, tablas, índices, columnas, vistas, funciones, procedimientos y demás elementos dentro de una base de datos. Hace parte del lenguaje DDL.

Entidad: Representación de un «objeto» del mundo real que puede distinguirse de otro «objeto» a través de sus características o atributos. Las entidades pueden ser tangibles, cuando tienen representación material en el mundo natural, e intangibles o abstractas, las cuales representan ideas definidas por el hombre.

EXPLAIN: Palabra reservada que se utiliza dentro de algunos DBMS, como PostgreSQL y MySQL, para obtener información sobre el plan de ejecución de una consulta. La ejecución de esta sentencia en una consulta devolverá una tabla con información sobre cada una de las tablas empleadas en ella, junto con una serie de columnas que proporcionarán información relevante sobre el plan de ejecución de esta.

Expresión CASE: Expresión que permite obtener un valor basado en una condición lógica. Al ser escalar, se puede usar en cualquier lugar donde se espere un valor.

Filas: También denominadas como «registros» o «tuplas», representan relaciones entre un conjunto de valores. Cada fila corresponde entonces a una entidad dentro de un grupo de entidades.

Formas normales: Serie de criterios que permiten determinar el grado de vulnerabilidad de una tabla en cuanto a posibles inconsistencias y errores lógicos de diseño, con el fin de mejorar la estructura de la base de datos. La acción de implementar una forma normal a una base de datos se conoce con el nombre de «normalizar». Dentro de las formas normales se encuentran la 1FN, la 2FN, la 3FN, la FNBC (versión más estricta de la 3FN), basadas en dependencias funcionales, y la 4FN y la 5FN, basadas en dependencias multivaloradas.

Función: Dentro de SQL, son instrucciones que permiten realizar tareas específicas de manera más rápida y fácil como devolver un valor calculado con los datos de una columna (funciones de agregación), devolver un solo valor basándose en el valor de entrada (funciones escalares) o trabajar sobre subconjuntos o ventanas de filas (funciones de ventana). También incluyen las funciones definidas por el usuario.

Funciones definidas por el usuario: Subprogramas que se pueden crear para encapsular comportamientos. Pueden recibir parámetros y retornar un valor.

Funciones de agrupamiento: También conocidas como «funciones de agregación», permiten tomar un conjunto de datos o valores de entrada y devolverlos como un solo valor.

Funciones de ventana: Funciones que operan sobre un grupo de filas a las que se hace referencia como ventana. En otras palabras, se arman grupos o ventanas de procesamiento por cada valor distinto que se encuentre en dicha columna y se aplica la función.

Funciones escalares: Funciones que aceptan argumentos (opcionalmente) y devuelven un único valor escalar, es decir, reciben un dato de un tipo particular, realizan operaciones con este y, como resultado, retornan un nuevo dato.

GROUP BY: Cláusula utilizada para realizar consultas que utilicen funciones de agregación y que deseen aplicarse no solo a un conjunto de tuplas, sino a un grupo de conjuntos de tuplas. Los atributos o columnas especificados en la cláusula `GROUP BY` formarán grupos, y las tuplas con el mismo valor en todos sus atributos pertenecerán a un mismo grupo.

HAVING: Cláusula utilizada para establecer una o más condiciones a los grupos resultantes de una consulta que implementa funciones de agregación, en vez de a las tuplas, como se hace con la cláusula `WHERE`.

Identificador: Denominado también como llave primaria o *Primary Key* (en inglés), es un atributo o combinación de atributos que debe ser única para cada instancia de un conjunto de entidades, es decir, debe ser único para cada fila de una tabla.

IN: Operador que permite comprobar si una expresión coincide con algún valor en una lista de valores. También ayuda a verificar la pertenencia a un conjunto de datos que sea el resultado de una consulta `SELECT`.

Índice: Estructura de datos que permite un acceso más rápido a los datos de la tabla subyacente, para que se puedan encontrar las filas específicas con mayor facilidad.

INSERT: Comando del lenguaje DML que permite la inserción de registros o filas en una tabla.

Intersección: Operación entre dos conjuntos en donde se obtienen solo las filas que son comunes en ellos.

IS: Palabra reservada para probar valores nulos (`NULL`). Se presenta de la forma `IS NULL` (para comprobar que el valor comparador sea nulo) o `IS NOT NULL` (para comprobar que el valor comparador no sea nulo).

LIKE: Operador utilizado para comparar patrones en una cadena de caracteres.

LIMIT: Cláusula utilizada para indicar la cantidad de registros o filas por retornar en una consulta.

Llave foránea: En inglés, *Foreign Key.* Es un atributo dentro de una tabla (destino) que corresponde a la llave primaria de otra tabla (origen).

Llave principal: En inglés, *Primary Key*. También conocida como identificador, es un atributo o combinación de atributos que debe ser única para cada instancia de un conjunto de entidades, es decir, debe ser único para cada fila de una tabla.

Modelo entidad-relación: Modelo lógico basado en objetos, el cual representa la percepción de la realidad mediante objetos llamados «entidades» y las relaciones entre ellos. Cada entidad se diferencia de otra a través de atributos.

Modelo relacional: Representa los datos y sus relaciones mediante una colección de tablas bidimensionales (filas y columnas), que contienen datos tomados de los dominios correspondientes. Cada tabla está conformada por un número definido y finito de columnas o campos, los cuales representan los atributos. Este modelo es un ejemplo de los modelos basados en registros.

Muchos a muchos: Dentro de una relación binaria entre las entidades A y B, una entidad en A está asociada con cualquier número de entidades en B (cero o más) y viceversa.

NULL: Valor que indica la ausencia de información sobre el valor de un campo o atributo.

OFFSET: Cláusula que permite especificar la cantidad de filas que se omitirán antes de que se recupere alguna fila de la consulta.

Operaciones de conjunto: Dado que las bases de datos relacionales se fundamentan en el álgebra relacional y la teoría de conjuntos, y teniendo en cuenta que una tabla es un conjunto de filas, estas mismas operaciones pueden realizarse en una base de datos. Las operaciones de conjunto se aplican comúnmente sobre los conjuntos de filas obtenidos por dos consultas. Es posible unir las filas de las dos tablas resultantes (unión), hallar la intersección de las filas (intersección) o identificar cuáles filas se obtuvieron en una consulta, pero no en la otra (diferencia).

Operador lógico: Permite comprobar la veracidad de varias condiciones, donde el resultado será `FALSE` o `TRUE`. Son `AND`, `OR` y `NOT`.

Operador relacional: Permite vincular un campo con un valor para que el DBMS compare cada registro (la columna o campo especificado) con el valor dado. Dentro de este tipo de operador se encuentran: `=`, `<>`, `>`, `<`, `>=` y `<=`.

ORDER BY: Cláusula que permite ordenar la forma en que las tuplas de una relación son presentadas, según uno o varios atributos. Por defecto, el orden de los elementos en una consulta es ascendente. Para indicar que debe ser descendente, se utiliza la palabra reservada `DESC`.

OVER: Cláusula que define una ventana o un grupo de filas establecido por el usuario en un conjunto de resultados de la consulta.

PARTITION BY: Cláusula que permite separar las tuplas por número de cuenta, con el fin de que para cada tupla solo se consideren aquellas de su partición. En ese sentido, se crea una ventana para cada tupla.

Primera forma normal: La 1FN es una forma normal dentro del proceso de normalización de una base de datos. Una relación se encuentra en 1FN cuando todos sus atributos son atómicos, es decir, los elementos de ese dominio son indivisibles.

Procedimientos: Conjunto de instrucciones que permiten recibir parámetros de entrada y devolver valores de salida, ejecutándose como resultado de un llamado de un programa. Pueden hacer cambios a la base de datos o simplemente realizar consultas de selección.

Relación: Elemento esencial de modelo relacional. Hace referencia a un conjunto de tuplas y se puede entender como sinónimo de tabla.

Segunda forma normal: La 2FN es una forma normal dentro del proceso de normalización de una base de datos. Una relación se encuentra en 2FN si y solo si cumple con la 1FN y, además, todos los atributos que no son clave o llave dependen completamente de la clave.

SQL: Siglas del término en inglés *Structured Query Language*; en español, lenguaje de consulta estructurado. Es el lenguaje estándar para las bases de datos relacionales.

Subconsulta: Expresión a manera de consulta dentro de la declaración de otra consulta con el fin de comprobar la pertenencia de conjuntos, comparar conjuntos y determinar cardinalidad de conjuntos.

Subconsulta autónoma: También denominada autocontenida o independiente, es aquella subconsulta que genera un valor o un conjunto de valores requeridos por la consulta principal para realizar alguna operación.

Subconsulta correlacionada: Subconsulta donde la consulta interna es dependiente de la consulta externa y no puede ser invocada de forma autónoma. En consecuencia, para las subconsultas correlacionadas, el DBMS evalúa la consulta interna una vez por cada fila de la consulta externa.

Tabla: Representación de relaciones dentro del modelo relacional, compuesta por filas o registros, que hacen referencia a una instancia de un conjunto de entidades, y columnas o campos, que corresponden a los atributos.

Tercera forma normal: La 3FN es una forma normal dentro del proceso de normalización de una base de datos. Una relación se encuentra en 3FN si y solo si cumple o se encuentra en 2FN y, además, todos los atributos que no son clave o llave dependen de forma no transitiva de la clave primaria.

Tipo de dato: Atributos que especifican el dominio o tipo de valor que un objeto puede contener. Pueden ser enteros, decimales, cadenas de texto, moneda, fechas, horas, binarios, entre otros. Los tipos de datos también dependerán del SGBS en el cual se trabaje.

***Transaction*:** En español, transacción. Son un conjunto de operaciones dentro de una base de datos que deben ejecutarse todas o ninguna. Es decir, si alguna operación no puede llevarse a cabo, la transacción se considera como no finalizada.

TRUNCATE: Comando que permite eliminar todos los registros o filas de una tabla sin eliminar la tabla.

Unión: Operación en la que se toman dos conjuntos con el mismo número de columnas y del mismo tipo y se devuelven como un solo conjunto unificado.

UNIQUE: Palabra reservada para expresar restricción de integridad en la que los atributos expresados en la creación de una tabla forman una clave candidata. Adicionalmente, también se utiliza como constructor en consultas de selección que contengan subconsultas para comprobar si las subconsultas tienen tuplas duplicadas en su resultado.

Uno a muchos: Dentro de una relación binaria entre las entidades A y B, una entidad en A esta asociada con cualquier número (cero o más) de entidades en B, pero una entidad en B puede estar asociada como máximo con una entidad en A.

Uno a uno: Dentro de una relación binaria entre las entidades A y B, una entidad en A está asociada con máximo una entidad en B (cero o una) y viceversa.

UPDATE: Comando del lenguaje DML que permite actualizar los valores de un campo(s) o atributo(s) teniendo en cuenta o no algún tipo de condición.

Vistas: Tablas virtuales que están basadas en el resultado de una consulta `SELECT`. Se consideran una tabla porque están constituidas por filas y columnas, pero el término «virtual» hace referencia a que no existe físicamente en la base de datos

Vistas materializadas: Vistas en las cuales no solo se almacena la consulta que las define, sino también su contenido. Aunque son más rápidas de leer, es probable que incluyan datos redundantes ya que su contenido puede deducirse de la definición de la vista y del resto del contenido de la base de datos.

Lo autores

ERNESTO GALVIS-LISTA. Profesor titular de la Facultad de Ingeniería de la Universidad del Magdalena; Investigador Senior en el Sistema Nacional de Ciencia, Tecnología e Innovación; ingeniero de sistemas y magíster en Informática de la Universidad Industrial de Santander; y doctor en Ingeniería de Sistemas y Computación de la Universidad Nacional de Colombia, sede Bogotá. Cuenta con más de 20 años de experiencia en la enseñanza de las bases de datos relacionales a nivel universitario y de postgrado, así como en otros tópicos del ámbito de la ingeniería de software.

ALEXANDER BUSTAMANTE-MARTÍNEZ. Profesor ocasional de la Facultad de Ingeniería de la Universidad del Magdalena; ingeniero de sistemas de la Universidad del Magdalena, magíster en Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander; y doctor en Informática de la Universidad Politécnica de Valencia (España). Tiene experiencia en la enseñanza de las bases de datos relacionales (transaccionales y analíticas) a nivel universitario y de postgrado.