

BASES DE DATOS RELACIONALES

**ERNESTO GALVIS LISTA
ALEXANDER BUSTAMANTE MARTÍNEZ**

Universidad del Magdalena

2022

Un enfoque aplicado y
orientado a resultados
de aprendizaje

Contenido

Capítulo I Datos, tablas, columnas y filas	7
1.1 Elementos básicos de una base de datos relacional	8
1.2 Uso básico de los datos almacenados en las tablas	11
1.3 Aprendizajes más importantes del capítulo I	17
1.4 Actividades de aplicación para evidenciar lo aprendido	18
Capítulo 2 Consultas básicas y filtrado de filas	21
2.1 Consultas básicas sobre una única tabla.....	23
2.2 Exclusión de filas duplicadas y ordenamiento de resultados	29
2.3 Consultas con filtrado de filas	34
2.4 Aprendizajes más importantes del Capítulo 2.....	44
2.5 Actividades de aplicación para evidenciar lo aprendido	45
Capítulo 3 Funciones y agrupamiento.....	47
3.1 Funciones escalares.....	51
3.2 Funciones de agregación	57
3.3 Agrupamiento.....	65
3.4 Filtrado de datos agrupados.....	72
3.5 Aprendizajes más importantes del Capítulo 3	76
3.6 Actividades de aplicación para evidenciar lo aprendido	76
Capítulo 4 Subconsultas.....	79
4.1 Subconsultas autónomas.....	80
4.2 Subconsultas correlacionadas	98
4.3 Aprendizajes más importantes del Capítulo 4	103
4.4 Actividades de aplicación para evidenciar lo aprendido	103
Capítulo 5 Combinaciones	105
5.1 Combinación interna	108
5.2 Combinación externa	121
5.3 Combinación cruzada	127
5.4 Aprendizajes más importantes del Capítulo 5.....	129
5.5 Actividades de aplicación para evidenciar lo aprendido	129

Capítulo 6 Operaciones de conjunto y funciones de ventana	133
6.1 Operaciones de conjunto	134
6.2 Funciones de ventana	143
6.3 Aprendizajes más importantes del Capítulo 6	153
6.4 Actividades de aplicación para evidenciar lo aprendido	153
Capítulo 7 Otras operaciones de consulta	155
7.1 Expresión CASE.....	156
7.2 LIMIT y OFFSET.....	157
7.3 Pivoteo de filas con CROSSTAB.....	159
7.4 Aprendizajes más importantes del Capítulo 7.....	162
7.5 Actividades de aplicación para evidenciar lo aprendido	162
Capítulo 8 Diseño conceptual	165
8.1 Análisis de necesidades	166
8.2 Niveles de diseño	166
8.3 Entidades y atributos	168
8.4 Relaciones entre entidades	171
8.5 Aprendizajes más importantes del Capítulo 8.....	181
8.6 Actividades de aplicación para evidenciar lo aprendido	182
Capítulo 9 Diseño lógico	185
9.1 Modelo conceptual a modelo relacional	187
9.2 Evolución del modelo relacional	194
9.3 Aprendizajes más importantes del Capítulo 9	201
9.4 Actividades de aplicación para evidenciar lo aprendido	202
Capítulo 10 Normalización	204
10.1 Tablas sin normalizar	205
10.2 Primera forma normal.....	207
10.3 Segunda forma normal	209
10.4 Tercera forma normal.....	212
10.5 Aprendizajes más importantes del Capítulo 10.....	213
10.6 Actividades de aplicación para evidenciar lo aprendido	214
Capítulo 11 Implementación del diseño lógico	215
11.1 Creación, eliminación y modificación de tablas.....	216
11.2 Inserción, eliminación y actualización de filas	223

11.3 Creación de índices	237
11.4 Aprendizajes más importantes del Capítulo 11.....	244
11.5 Actividades de aplicación para evidenciar lo aprendido.....	244
Capítulo 12 Vistas y objetos de programación procedural	247
12.1 Vistas	248
12.2 Funciones	254
12.3 Disparadores	266
12.4 Procedimientos almacenados	272
12.5 Aprendizajes más importantes del Capítulo 12.....	277
12.6 Actividades de aplicación para evidenciar lo aprendido	278

Capítulo I

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 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 utilizar los datos almacenados en una base de datos, definiendo algunos elementos de análisis que permiten construir consultas básicas 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 utiliza alguna de sus funcionalidades. Por ejemplo, cuando se usa 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 y se ponen a disposición para ser consultados, actualizados, eliminados, administrados y controlados. También debe mencionarse que existe software construido para administrar y operar las bases de datos, los cuales se conocen 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, específicamente las bases de datos relacionales.

I.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 los artistas 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, presentada en la Tabla I-1 y la tabla Canciones, presentada en la Tabla I-2. Estas tablas tienen los datos iniciales de la colección de canciones, compuesta por 12 canciones interpretadas por 5 artistas.

Cada fila de la tabla Artistas representa a un solista o a un grupo del cual se tiene canciones en la colección. Para cada artista se registra el nombre, el tipo, es decir 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.

Tabla I-1 Tabla Artistas

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

Tabla I-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

Con los datos de la tabla **Artistas** puede determinarse que el artista *Carlos Vives* es un *Solistা* 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 nos 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 tenga relación.

En las tablas se registran únicamente los datos que se requieren almacenar, procesar y administrar. Estos son solamente algunos datos de los que podrían identificarse. Por ejemplo, en la tabla **Artistas** 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 I-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.

Figura 1-1 Datos a 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

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.

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**. 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* de acuerdo con lo que está registrado en la tabla **Artistas**.

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. Esta necesidad puede expresarse como una pregunta que espera ser respondida con los datos disponibles en las tablas. Un ejemplo 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 I-3.

Tabla I-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 básico puede definirse la forma en que se obtendrá la respuesta a las necesidades planteadas en las preguntas con base en 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 I-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 pregunta indica que se requiere seleccionar únicamente dos columnas de la tabla Artistas y que no es necesario aplicar filtros o condiciones porque se necesitan los datos de todos los artistas registrados en la colección. En la Tabla I-4 se señala específicamente el conjunto de datos de la tabla Artistas que debería generarse como respuesta a la pregunta.

Tabla I-4 Conjunto de datos a incluir en la respuesta

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

Las respuestas a las necesidades de datos normalmente se presentan en forma de una nueva tabla que se genera a partir de los 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 I-5.

Tabla I-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.

El análisis realizado para dar respuesta a esta pregunta también indica que se requiere seleccionar únicamente dos columnas, pero, en este caso, son columnas 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 necesitan todas las filas de la tabla. En la Tabla I-6 se señala específicamente el conjunto de datos de la tabla Canciones que debería generarse como respuesta.

Tabla I-6 Conjunto de datos a 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 pregunta tiene dos columnas y doce filas, tal y como se muestra en la Tabla I-7.

Tabla I-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

¿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 para la segunda pregunta, se requieren datos ubicados únicamente en la tabla Canciones. En este caso deben mostrarse los datos registrados en tres columnas de la tabla, pero tiene un elemento diferente y es la necesidad de utilizar los datos de la columna duración para determinar las filas que deben mostrarse. Con esta acción de filtrado se excluyen las filas para las cuales la condición no se cumple. En la Tabla I-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 I-8 Conjunto de datos a incluir en la respuesta

Canciones					
identificador	título	duración	género	artista	álbum
I0001	La tierra del olvido	4:25	Vallenato	50001	La tierra del olvido
I0002	Ojos así	3:57	Pop	50003	¿Dónde están los ladrones?
I0003	Mi gente	3:05	Urbano Latino	50005	Sencillo
I0004	Ambiente	4:08	Urbano Latino	50005	Vibras
I0005	Cali pachanguero	4:51	Salsa	50002	No hay quinto malo
I0006	La creciente	3:04	Vallenato	50004	El binomio de oro
I0007	Sueños de conquista	4:02	Vallenato	50004	Por lo alto
I0009	Carito	3:39	Pop	50001	Déjame entrar
I0011	Una aventura	5:16	Salsa	50002	Cielo de tambores
I0012	Ginza	4:39	Urbano Latino	50005	Sencillo
I0013	Octavo día	4:32	Pop	50003	¿Dónde están los ladrones
I0014	Quiero verte sonreír	3:18	Pop	50001	Déjame entrar

En este caso, la tabla resultante que da respuesta a la pregunta tiene tres columnas y siete filas, tal y como se muestra en la Tabla I-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 I-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 este caso 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 a incluir en la respuesta. Esta condición 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 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. En amarillo se resaltan los valores que cumplen parcialmente la condición y, por ende, esas filas no hacen parte de la respuesta.

La condición compuesta no es lo único diferente. También 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. Esto no quiere decir que se modifica la estructura de las tablas o que los datos almacenados en dichas tablas 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 I-10 Conjunto de datos a 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. En la Tabla I-11 puede observar la tabla resultante, la cual está compuesta de cinco columnas y una fila.

Tabla I-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 I

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 almacena 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 éstas, y, finalmente, si es necesario realizar operaciones para derivar nuevos datos que permitan satisfacer plenamente la necesidad.

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 en los siguientes literales.
 - 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.

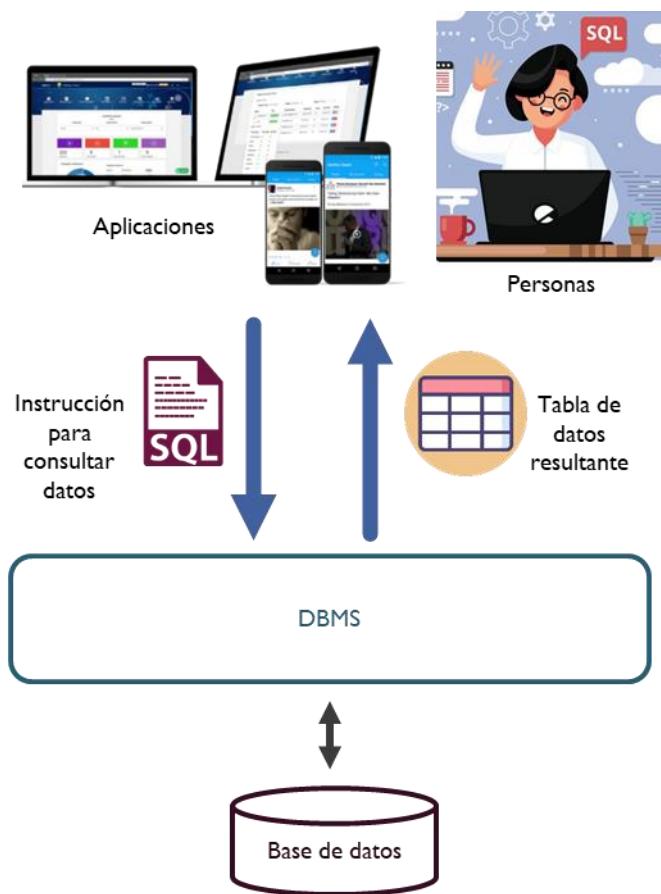
Las bases de datos relacionales se construyen para que los usuarios, personas o aplicaciones software, puedan consultar lo que está almacenado en ellas de forma sencilla, rápida y precisa. Para esto existe el Lenguaje Estructurado de Consulta o SQL (acrónimo de la expresión en inglés Structured Query Language) el cual es un lenguaje de dominio específico que fue creado con base en lo planteado en el modelo relacional de bases de datos propuesto en 1970 por Edgar Frank Codd, específicamente con la aplicación del álgebra y el cálculo relacional. Luego 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, obteniendo 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 se enfocan exclusivamente al ámbito de las bases de datos relacionales y su carácter declarativo a alto nivel, contrario a un lenguaje procedimental, hace que la programación sea muy cercana a una especificación de lo que debe hacerse sin que sea necesario entrar en detalles sobre cómo debe hacerse. 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 ejecutarlas.

Por ejemplo, una instrucción en SQL para realizar una consulta podría solamente especificar los datos, es decir, las columnas requeridas, la tabla en dónde 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, por ejemplo, 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 sub-lenguajes de acuerdo con la función general que cumplen. El primero reúne a las operaciones de creación de los objetos de la base de datos y se conoce como el Lenguaje de Definición de Datos o DDL (acrónimo de la expresión en inglés *Data Definition Language*). El segundo, denominado Lenguaje de Manipulación de Datos o DML (acrónimo 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 (acrónimo 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 adicionándole 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 al 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*.

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 la renombrada columna *álbum original* se modificó el contenido para registrar el identificador del álbum en el que fue incluida la canción al 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, en la Tabla 2-3 se presenta la nueva tabla *Géneros* y en la Tabla 2-4 la nueva tabla *Álbumes*.

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	Solistा		1
50002	Niche	1979	NULL	Grupo		2
50003	Shakira	1990	NULL	Solistা		3
50004	Binomio de Oro de América	1976	NULL	Grupo		1
50005	J Balvin	2006	NULL	Solistা		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

Teniendo la claridad sobre los elementos que componen la nueva versión de la base de datos, se iniciará el aprendizaje del SQL utilizando los cuatro elementos de análisis planteados en el Capítulo 1 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 debe expresarse lo que debe realizar el DBMS sin necesidad de definir y programar la forma en que debe realizarse. Por consiguiente, para este caso en concreto debe indicársele 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. El resultado de esta consulta se presenta en 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	Solistा	1
50002	Niche	1979	NULL	Grupo	2
50003	Shakira	1990	NULL	Solistা	3
50004	Binomio de Oro de América	1976	NULL	Grupo	1
50005	J Balvin	2006	NULL	Solistা	4

En el Script 2-1 puede verse que una consulta SQL tiene una estructura y una sintaxis sencilla. 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 contienen esas columnas. Al hacer una equivalencia 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*, equivale 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 Postgres. En otros DBMS se utilizan otros símbolos para indicar que el nombre de una columna incluye espacios en blanco, como es el caso de SQL Server que 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. 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. Usar el carácter * tiene varias ventajas. Una es que permite reducir significativamente el tamaño código. También hace que la misma consulta funcione independientemente de que se agreguen o eliminen columnas de la tabla. Sin embargo, también tiene 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 declara de forma explícita todas las columnas de la tabla consultada, se tiene la opción de cambiar el orden en que aparecerán las columnas en la tabla resultante. Para esto, deben ubicarse en el orden deseado dentro de la lista separada por comas después de la palabra **SELECT**. El Script 2-4 ejemplifica esto.

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 muestra 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 a 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 son tres las columnas a mostrar, tal y como se presenta en la Figura 2-2.

Figura 2-2 Columnas requeridas para responder la pregunta

The diagram illustrates the selection of columns from a database table. It features two labels above a table: 'Columna que no debe incluirse en la tabla resultante' with a red arrow pointing to the 'identificador' column, and 'Columnas a mostrar en la tabla resultante' with three blue arrows pointing to the 'título', 'fecha de lanzamiento', and 'sello discográfico' columns.

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

Específicamente se requieren las columnas `título`, `fecha de lanzamiento` y `sello discográfico` de la tabla `Álbumes`. No se requieren filtros ni operaciones sobre los datos para conformar la tabla resultante.

Con esto puede proponerse la consulta SQL del Script 2-5 en la que se incluyen solamente los nombres de las tres columnas requeridas luego de la palabra `SELECT`. Estos nombres se presentan separados por coma. En la Tabla 2-7 se 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 la de asignarle un nombre alternativo o un alias a las columnas que se van a mostrar. Podría ser, por ejemplo, que quiera mejorarse la presentación del resultado expuesto en la Tabla 2-7.

En lugar de obtener una columna llamada `título` se podría utilizar la palabra `álbum`, en lugar de `fecha de lanzamiento` el encabezado `lanzamiento` y en lugar de `sello discográfico` 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, de forma 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`. La segunda, de forma implícita, en la que, 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 el resultado mostrado en la Tabla 2-9. La tabla resultante 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

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, generándose el resultado mostrado en 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 mostrará 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

Columnas con los datos a mostrar en la tabla resultante

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

La consulta SQL para responder la pregunta se presenta en el Script 2-9, con la cual se obtiene el resultado presentado en la Tabla 2-11.

Script 2-9

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

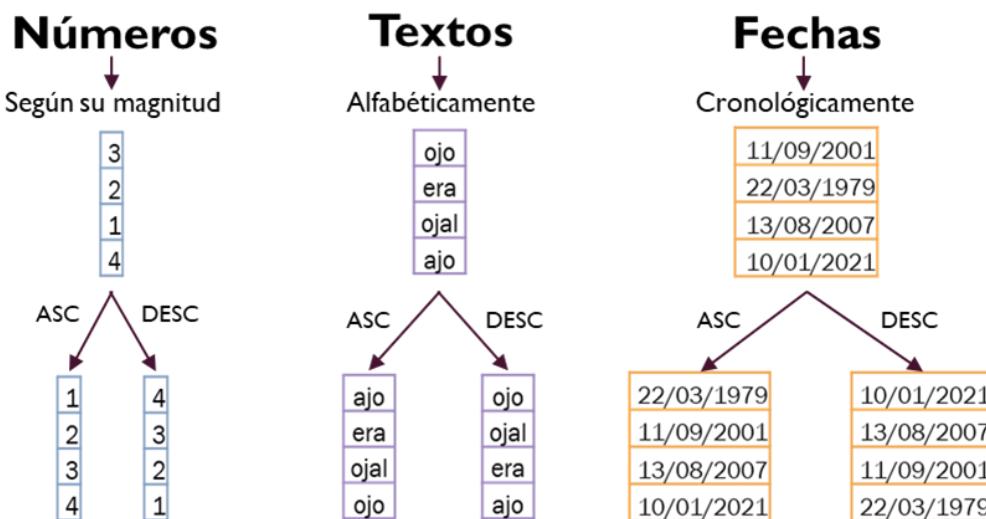
Tabla 2-11 Resultado de la consulta del Script 2-9

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 obtenga de la ejecución.

El SQL incluye una instrucción que permite indicarle al DBMS el criterio de ordenamiento a utilizar, es decir, la columna o las columnas que debe tomar como base para esta operación, y el tipo de ordenamiento, es decir, 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
```

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 utilizan como criterios de ordenamiento. 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.

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

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 se utiliza el nombre de la columna `título` tal y como está en la tabla `Álbumes` en lugar del alias `álbum`.

Script 2-11

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

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, descendenteamente, por los años de actividad

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.

Figura 2-6 Columnas requeridas para responder la pregunta

identificador	nombre	año de lanzamiento	año de retiro	tipo	género principal
50001	Carlos Vives	1986	NULL	Solist	1
50002	Niche	1979	NULL	Grupo	2

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 $2021 - \text{"año de lanzamiento"}$ se genera una nueva columna en la tabla resultante cuyo nombre es años de actividad al 2021.

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

```

La consulta permite obtener el conjunto de datos resultante que se presenta en la Tabla 2-13. Además de estar ordenados alfabéticamente de manera ascendente según la columna tipo, es decir, primero los de tipo Grupo y luego los de tipo Solista. Posteriormente, 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” puesto que tiene “42” años.

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	Solistा	1986	35
Shakira	Solistা	1990	31
J Balvin	Solistা	2006	15

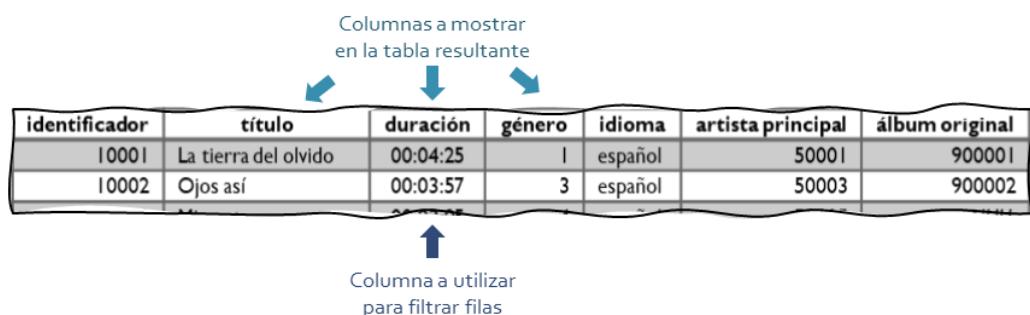
2.3 Consultas con filtrado de filas

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.

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'
  
```

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.

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[.nnnnnnn]`. 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.

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

Los operadores relacionales, como el operador igual que (`=`), mayor que (`>`), 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 es cuando evalúa las expresiones. El resultado de la ejecución del Script 2-13 es el conjunto de datos presentado en la Tabla 2-15.

Tabla 2-15 Resultado de la ejecución del Script 2-13

título	duración	género
La tierra del olvido	00:04:25	1
Ambiente	00:4:08	4
Cali pachanguero	00:4:51	2
Sueños de conquista	00:4:02	1
Ginza	00:4:39	4
Octavo día	00:4:32	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`, el cual permite 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 esta 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 presenta una solución alternativa y la Tabla 2-16 presenta la tabla resultante, que es igual en ambas consultas.

Figura 2-8 Columnas requeridas para resolver la pregunta.

Columnas a mostrar en la tabla resultante					
identificador	nombre	año de lanzamiento	año de retiro	tipo	género principal
50001	Carlos Vives	1986	NULL	Solistा	1
50002	Niche	1979	NULL	Grupo	2

↑
Columna a utilizar para filtrar filas

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
```

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?

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*.

Figura 2-9 Columnas requeridas para responder la pregunta

identificador	nombre	año de lanzamiento	año de retiro	tipo	género principal
50001	Carlos Vives	1986	NULL	Solista	1
50002	Niche	1979	NULL	Grupo	2

Al estar trabajando con una sola tabla, resolver la pregunta implica definir cómo referirse a los géneros “vallenato” y “pop” pues estos no están almacenados en la tabla *Artistas*. En dicha tabla, en lugar de estas cadenas de caracteres se 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 requeridos. Como se muestra en la Tabla 2-17, el género “Vallenato” corresponde al identificador “1” y el género “Pop” corresponde 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 la cual se obtiene como resultado el conjunto de datos presentado en 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`. Este se utiliza para comprobar que un valor hace parte de una lista de valores. Para 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 1 en la tabla `Géneros`). Si el valor de la columna `género` coincide 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

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

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”. El operador `IN` 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 anterior 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 1 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. Figura 2-11 se 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

Columnas a mostrar en la tabla resultante

identificador	título	duración	género	idioma	artista principal	álbum original
I0001	La tierra del olvido	00:04:25	1	español	50001	900001
I0002	Ojos así	00:03:57	3	español	50003	900002

Columnas a utilizar para filtrar filas

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

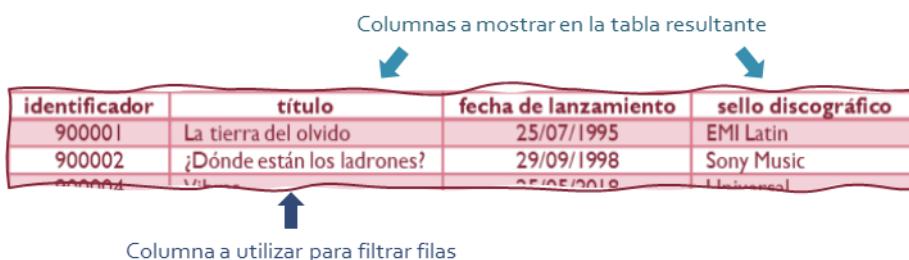
título	duración
Ojos así	00:03:57
La creciente	00:03:04
Carito	00:03:39
Quiero verte sonreír	00:03:18
Ojos así	00:03:57

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 Columnas requeridas para responder la pregunta



Para 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 % representa una cadena cualquiera (0 o más caracteres) y el _ 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 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.

Script 2-21

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

Tabla 2-20 Resultado de la ejecución del Script 2-21

título	sello discográfico
El binomio de oro	Codiscos

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 en el que una tabla contenga valores nulos porque no existen los datos o se desconocen al 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 este significa lo mismo que el valor cero (0), 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 0 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 0 para representar que no tiene hijos mientras que *NULL* representaría 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*.

Figura 2-13 Columnas requeridas para responder la pregunta

Columnas a mostrar en la tabla resultante

↑
Columna a utilizar para filtrar filas

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
900003	V/V	25/05/2010	Universal

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 utilizan 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 agregaría 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.

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?

Al igual que la pregunta resuelta con la consulta del Script 2-22, en esta pregunta 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'.

Figura 2-14 Columnas requeridas para responder la pregunta

Columnas a mostrar en la tabla resultante			
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
900003	V/V	29/09/2010	Universal

↑
Columna a utilizar para filtrar filas

La respuesta a esta pregunta puede obtenerse con la consulta del Script 2-23. Con esta consulta se obtendrá como resultado la Tabla 2-21, mostrando los cinco álbumes que fueron lanzados antes de la fecha indicada en la condición.

Script 2-23

```
SELECT nombre,
       "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*”.

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 sub-lenguajes: DDL, DML y DCL.
- DML es el lenguaje utilizado para especificar las consultas de acceso a los datos y como resultado de una consulta en este lenguaje siempre se obtiene una estructura en forma de tabla como respuesta.
- La palabra reservada `FROM` indica desde cuál tabla provienen los datos a 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

- 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.
- Construir las consultas SQL y mostrar el conjunto de datos resultante para dar respuesta a las cinco preguntas propuestas en la actividad I.
- ¿Cuál es la pregunta que se está supliendo con la siguiente consulta?

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

- 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?
- ¿Cuál es la pregunta se suple 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
```

- ¿Cuál es la pregunta o necesidad de datos que se suple 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 género vallenato, salsa o pop que hayan sido 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. 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. 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 utilizan combinaciones de los diversos operadores disponibles.

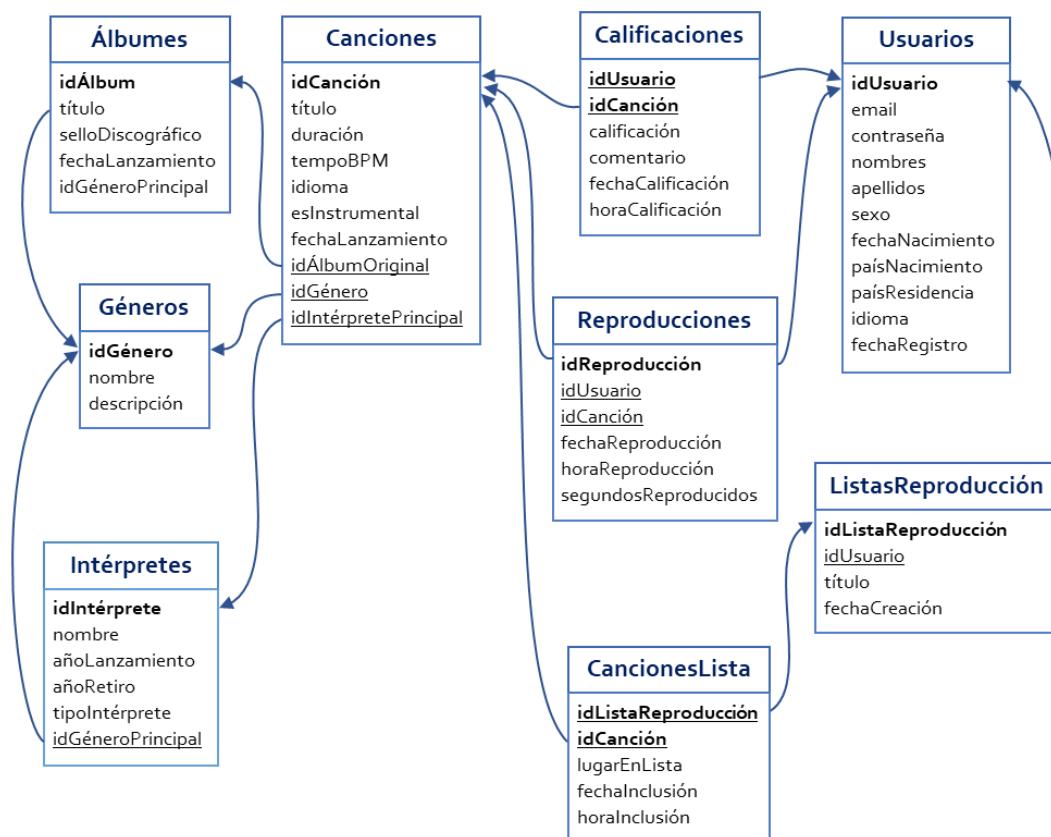
Las posibilidades que ofrece el SQL van mucho más allá de lo expresado en el párrafo anterior. Hasta este momento apenas se ha tenido una pequeña degustación que deja una agradable sensación sobre la potencia y sencillez del lenguaje. Solamente hay que decir lo que se quiere y el DBMS lo entregará en forma de una tabla resultante.

Para avanzar en el aprendizaje se continuará utilizando el caso de la colección de canciones. No obstante, en este capítulo se introduce 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, procedemos a explicar esta nueva versión de la base de datos de la plataforma “*Mis Canciones*”, la cual se resumen 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 distingue 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 a cada fila de manera única. Debe recordarse que estos valores se mantendrán distintos aun cuando se agreguen nuevas filas.

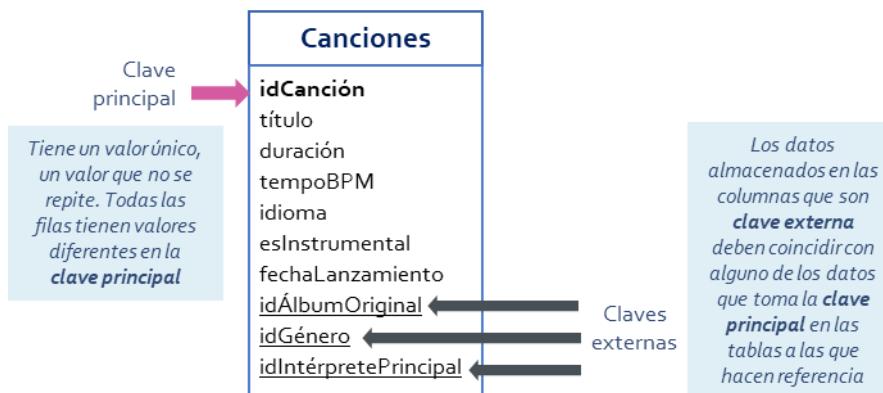
Figura 3-1 Versión 3 de la base de datos de la plataforma “*Mis Canciones*”



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, lo cual queda registrado en la tabla “Calificaciones”, y al incluirlas en alguna de sus listas de reproducción, lo cual queda registrado en las tablas “ListasReproducción” y “CancionesLista”.

Un usuario puede reproducir muchas veces la misma canción. En unas ocasiones puede reproducirla completa y en otras solamente algunos segundos. En todos los casos se registra la cantidad de segundos que reprodujo la canción.

Una canción solo puede tener una calificación por cada usuario. Si el usuario asigna una calificación y luego cambia su opinión, en la tabla calificaciones se modificará 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.

Un usuario puede crear todas las listas de reproducción que desee. En cada lista de reproducción puede agregar múltiples canciones y es posible que tenga listas de reproducción sin canciones, 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 la otra para registrar 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 y ahora se llama “*Intérpretes*”. Adicionalmente es necesario precisar que son pocas las columnas que permiten el registro de valores nulos (*NULL*). Específicamente están las columnas “*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. Con este se simplifica la escritura de las consultas porque se evita el uso de 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 un dato almacenado en una fila de una tabla. Con una función de agregación puede calcularse un valor a partir de operaciones aplicadas sobre todos los datos almacenados en una columna para un conjunto de filas. Con las operaciones de agrupamiento pueden definirse 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. Con esto será posible construir consultas que generen, por ejemplo, resúmenes de datos comúnmente utilizados en reportes para apoyar la toma de decisiones.

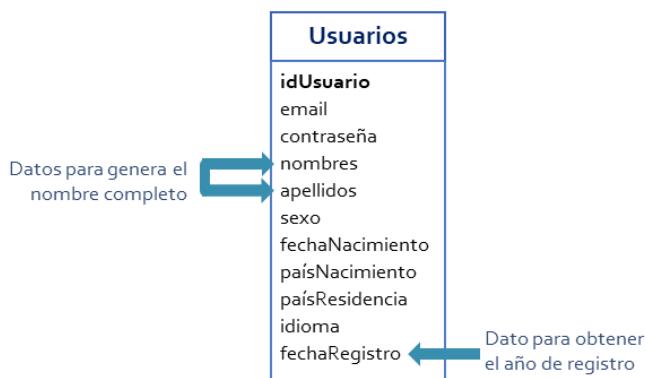
3.1 Funciones escalares

Los datos almacenados en las tablas son la base para obtener las respuestas y suprir 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 trabajarla 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`. Sin embargo, en esa tabla no existe una columna con el nombre completo y tampoco una columna con el año de registro, pero si 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`. En la concatenación hay que tomar la precaución de incluir un espacio entre las dos cadenas. 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

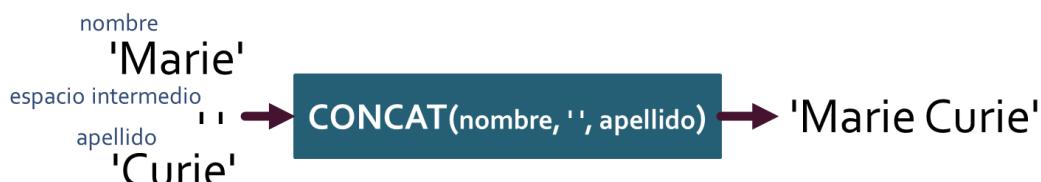
```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       EXTRACT(YEAR FROM fecharegistro) AS añoRegistro
  FROM Usuarios
 ORDER BY añoRegistro DESC,
          nombreCompleto
```

En el Script 3-1 se observan nuevos elementos dentro de una consulta SQL. Estos nuevos elementos son dos funciones escalares que se identifican con los nombres **CONCAT** y **EXTRACT**. La función **CONCAT** se utiliza para conformar una cadena de texto con los nombres y los apellidos de los usuarios. La función **EXTRACT** 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 funciones que se implementan en cualquier lenguaje de programación estructurada. Estas funciones reciben un dato de un tipo particular, realizan operaciones con este dato y como resultado retornan un nuevo dato. Los DBMS tienen implementadas y disponibles una cantidad significativa de funciones que ayudan a realizar 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 es una idea equivocada porque la operación si se aplica a todas las filas de la tabla, pero fila por fila.

La función **EXTRACT** 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 el funcionamiento de esta función.

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 ofrece la misma funcionalidad. Esta función podría utilizarse de la forma en que se presenta en el Script 3-3.

Script 3-3

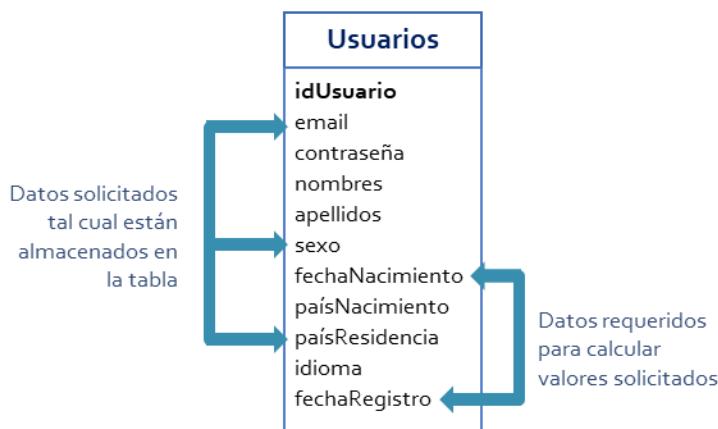
```
SELECT CONCAT(nombres, ' ', apellidos) AS nombreCompleto,
       DATE_PART('year', fechaRegistro) AS añoRegistro
  FROM Usuarios
 ORDER BY añoregistro DESC,
          nombreCompleto
```

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 email, el sexo, la edad actual, la edad al 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. Además de solicitar datos almacenados en la tabla, como el email, el sexo y el país de residencia, se están 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



Para obtener 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, es decir la fecha en el momento de ejecución de 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 tales fechas. Si solamente se proporciona una fecha como entrada, la función **AGE** realiza el cálculo con relación a la fecha actual, es decir, la fecha al 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.

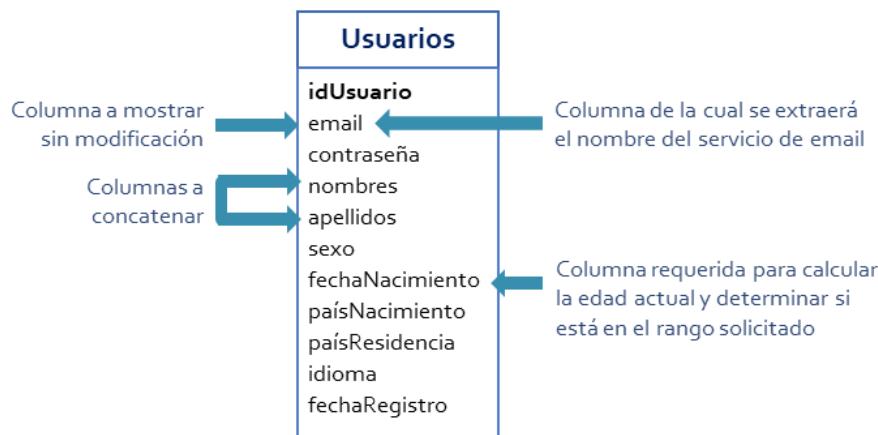
Un grupo de funciones tiene amplia utilización son las que realizan operaciones sobre cadenas de texto. Además de la función **CONCAT**, utilizada para anexar dos o más cadenas, existen funciones como la función **UPPER**, que permite poner un texto en

mayúsculas, o la función `LOWER` para ponerlo 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 email y el nombre de dominio del servicio de email 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 responder la pregunta.

Figura 3-8 Columnas requeridas para responder la pregunta.



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`. La edad actual no será incluida en la respuesta, solo se utilizará para filtrar las filas.

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

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 se presenta en 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 a extraer. En la Figura 3-9 se ilustra el funcionamiento de esta función.

Figura 3-9 Conceptualización del funcionamiento de la función escalar **SUBSTRING**

Para obtener la posición desde dónde debe iniciar la extracción de 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 a extraer se utiliza una expresión en la que se invoca a la función escalar **CHAR_LENGTH**.

Para resolver las tres preguntas anteriores se utilizaron 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, para identificar las funciones escalares estándar o de implementación propia que podrían servir para realizar el procesamiento de datos.

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

La obtención de 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 permite obtener 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 significar 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 facilita la generación de resúmenes que permitan construir rápidamente interpretaciones generales para sustentar la toma de decisiones y la ejecución de alguna actividad. En muchos casos es muy significativo conocer datos agregados, como el promedio de ingresos de los habitantes de una ciudad, para tomar decisiones 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 las funciones de agregación 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 mas no pude	4
10	"Sin medir distancias"	4
19	"La creciente"	4
20	"Olvídala"	4

El resultado presentado en la Tabla 3-2, en el 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 4 canciones del género 4. Pero, no siempre es sencillo llegar a resultados de este tipo y es en este escenario 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 a 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, existe la columna `idÁlbumOriginal` y la columna `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. Esto 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	<i>Null</i>

Al utilizar la función **COUNT** con el parámetro asterisco (*), tal y como se muestra en el Script 3-9, el DBMS contará las filas y genera como resultado el número 6, tal y como se muestra en la Tabla 3-5. Este resultado 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** pidiendo que cuente las filas, pero tomando únicamente la columna álbum, tal y como se presenta en el Script 3-10, generando el resultado presentado en 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

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**. Al utilizar **DISTINCT** 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

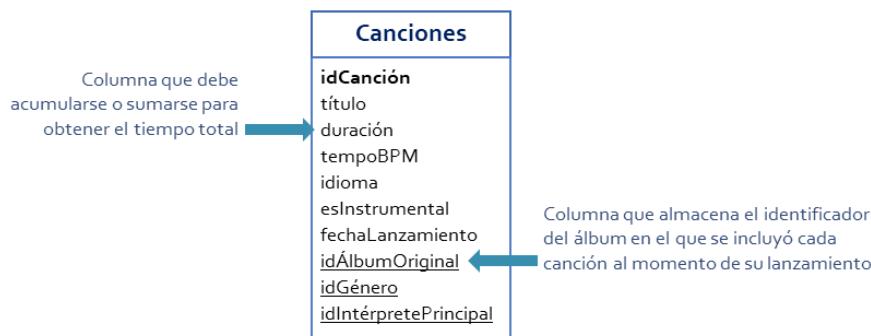
cantidadÁlbumes
4

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. Para 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 la pregunta.

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**. Esta función recibe como parámetro la columna o expresión que contiene los valores a 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 álbum con identificador “15”- La duración total será la suma de la duración 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

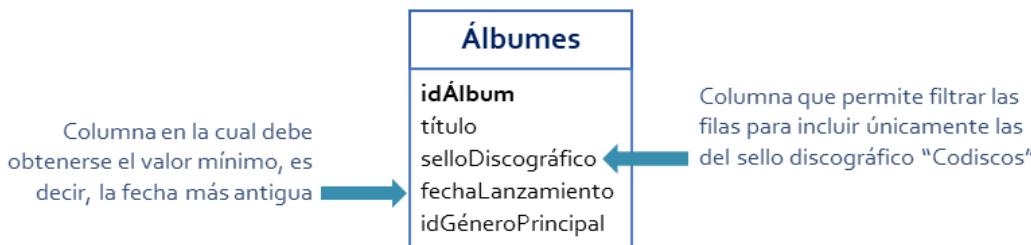
En varios casos se pide identificar cuáles son los valores extremos en una lista de valores, es decir, el máximo y el mínimo. Para esto, SQL proporciona la función **MAX** para el máximo y la función **MIN** para el mínimo.

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. La consulta que permite responder este interrogante se presenta en el Script 3-13.

Figura 3-13 Columnas requeridas para responder la pregunta



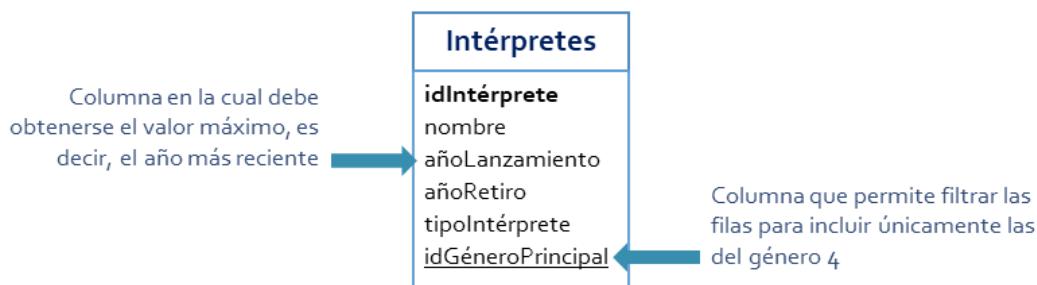
Script 3-13

```
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 resuman 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** para 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 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. Al ejecutarla se obtendrá que la calificación promedio para la canción en el período especificado es 2.

Script 3-15

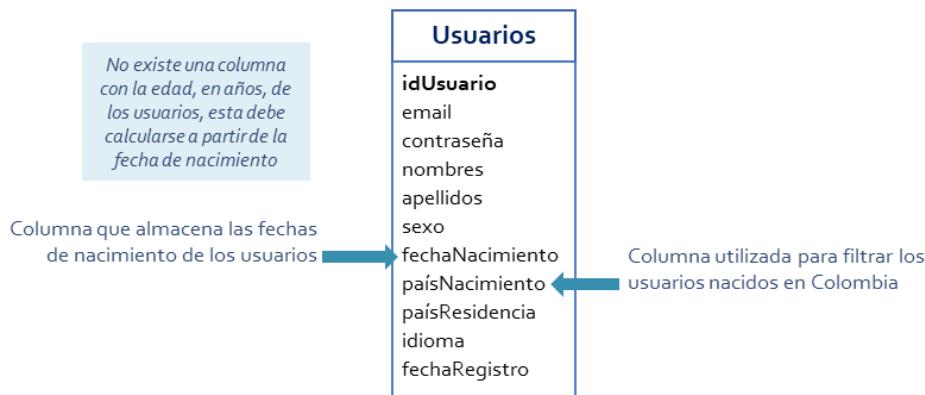
```
SELECT AVG(calificación) AS calificaciónPromedioCanción5Año2020
FROM Calificaciones
WHERE idCanción = 5 AND
    (fechaCalificación BETWEEN '01/01/2020' AND '31/12/2020')
```

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 a 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 `paísNacimiento` de la tabla `Usuarios` sea igual a “Colombia”. En segundo lugar, en relación con los datos a resumir con la función `AVG`, es decir, las edades de los usuarios, debe identificarse que en la tabla `Usuarios` no hay algunas columnas que almacene estos valores, pero hay una columna con la fecha de nacimiento. En la Figura 3-16 se identifican las 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 resultado obtenido de 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**, y es ese número entero que representa los años de un usuario 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 *¡Error! No se encuentra el origen de la referencia.*

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 su aplicación se realiza sobre todas las filas de una tabla o sobre las filas que cumplan las expresiones condicionales especificadas en la cláusula **WHERE** y generan 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 las operaciones de agrupamiento se propone la siguiente pregunta.

¿Cuántos intérpretes hay por cada tipo?

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. Se presentan solamente cuatro de las columnas que posee dicha tabla y todas las filas.

Tabla 3-10 Datos de la tabla Intérpretes

nombre	año lanzamiento	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 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 tiene el texto “Solista” y las demás tienen el texto “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 resolverla 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”, con la cual 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 que deba construirse una consulta por cada tipo, lo cual en este caso 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 `idgénero principal` 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	idgénero principal	
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 obtendría 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 5, y otro con el conteo de las filas que tienen el dato “Solista”, que en este caso también es 5. La consulta SQL para responder la pregunta sería la que se presenta en el Script 3-18, con la cual se genera el resultado presentado en 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. Después de especificar el criterio de agrupamiento se procede a aplicar la función de agregación sobre cada grupo. La especificación de las columnas o expresiones que se toman como base para crear los grupos se realiza 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 álbum. 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
```

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

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 operar en filas individuales. Cada grupo está representado por una única fila en el resultado final de la consulta. Esto 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.

¿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.

Figura 3-18 Columnas requeridas para responder la pregunta



La consulta que implementa lo descrito en el análisis se presenta en el Script 3-20. Con esta consulta se obtiene el resultado que se presenta en la Tabla 3-15.

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 resolverla debe trabajarse únicamente con las canciones que su identificador sea alguno de los valores de la lista, es decir, entre 1 y 10. Esto obliga a que deba realizarse un filtrado de filas para trabajar únicamente con las canciones que tienen esos identificadores.

También se indica que debe hallarse un promedio de calificación por canción, lo cual implica la creación de grupos para cada identificador de canción. En la Figura 3-19 se identifican 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, en la cual 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 observa la utilización 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”. También se aprecia que la peor valorada es la identificada con el número “3” la cual tiene un valor de “2.000”.

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

En la cláusula `GROUP BY` pueden 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.

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 requiere 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, los datos deben presentarse ordenados por la cantidad de reproducciones.

En el Script 3-22 se presenta 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. Sin embargo, 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. Pero, 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 más 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 la cláusula **HAVING** se especifican los criterios que deben cumplir todos los grupos que serán mostrados en el resultado final. Solo los grupos para los que la evaluación de los predicados de la cláusula **HAVING** es *verdadera* son incluidos en el resultado. 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. Este se define en la cláusula **SELECT** con la expresión *cantidad de canciones* y se invoca en la cláusula **ORDER BY**. Sin embargo, en la cláusula **HAVING** no se utiliza. La razón de esto es el 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 al orden en el cual aparecen en la consulta. El orden de procesamiento de las cláusulas es el siguiente.

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, lo cual se realiza en 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 si existe cuando 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 5 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.

Pr

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 `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 escenario se propone la siguiente pregunta.

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

Considerar únicamente las canciones del género 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, eliminado 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,
       id idioma,
       AVG(duración) AS "Duración promedio"
FROM Canciones
WHERE esInstrumental = '0'
GROUP BY idGénero,
       id 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 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,
       id idioma,
       AVG(duración) AS "Duración promedio"
FROM Canciones
WHERE esInstrumental = '0'
      AND idGénero IN (1, 3)
GROUP BY idGénero,
       id idioma
```

Tabla 3-18 Resultado de la ejecución del Script 3-26

género	idioma	Duración promedio
1	español	00:03:55
1	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 utilizando 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 la fase `SELECT`.

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 suplir 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 suplir 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 interprete 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 siguiente 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 la 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 menos reproducciones.
9. Explique cuál es la necesidad de datos que se busca suplir 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, permitiendo 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. También puede suceder 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 consulta. 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 pueden denominarse 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.

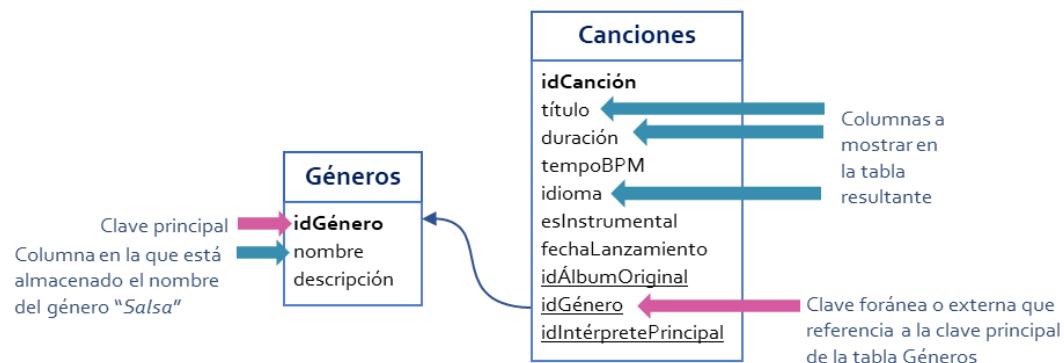
4.1 Subconsultas autónomas

Una subconsulta autónoma, auto contenida 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 dicha tabla. 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. Primero, buscar el identificador del género “Salsa” en la tabla Géneros y luego 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, generando como resultado la Tabla 4-1, es decir una tabla de una fila y una columna `idGénero` con el valor “3”. Una vez obtenido el valor del `idGénero`, “3” para este caso, puede utilizarse en otra consulta que permita obtener las canciones que pertenecen a este 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 responde 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 enfoque porque el código es altamente dependiente de los datos. Esta consulta podría calificarse como “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 pregunta 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 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

```

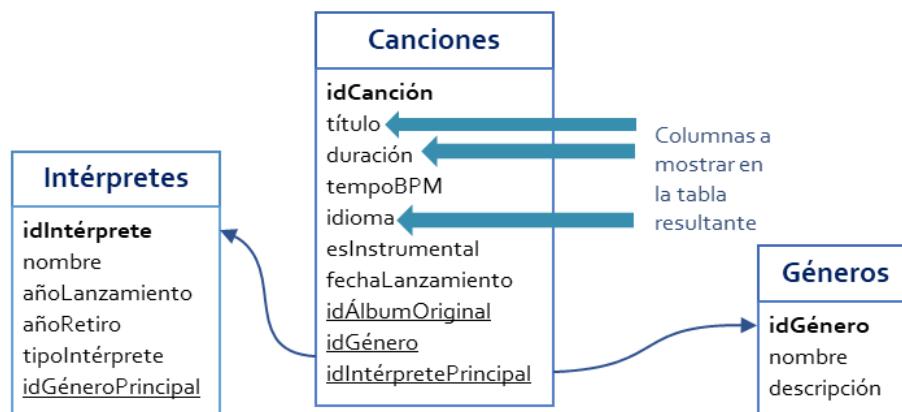
SELECT título,
       duración,
       idioma
  FROM Canciones
 WHERE idGénero = (SELECT idGénero
                      FROM Géneros
                     WHERE nombre = 'Salsa')
    
```

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 a 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, tal y como se presenta 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')
```

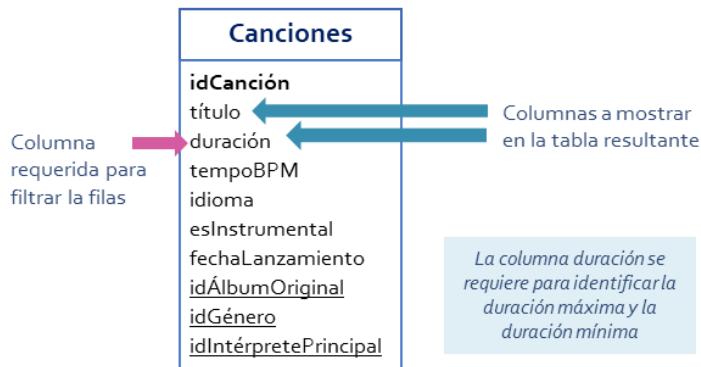
Tabla 4-3 Resultado de la ejecución del Script 4-5

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 trabajado hasta el momento, como es el caso de las funciones de agregación. Para ilustrar esto se propone abordar la siguiente pregunta.

¿Cuál es el título y la duración de las canciones mas “largas” y mas “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

Figura 4-3 Tablas y columnas requeridas para responder la pregunta

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.

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, como 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 Script 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)
```

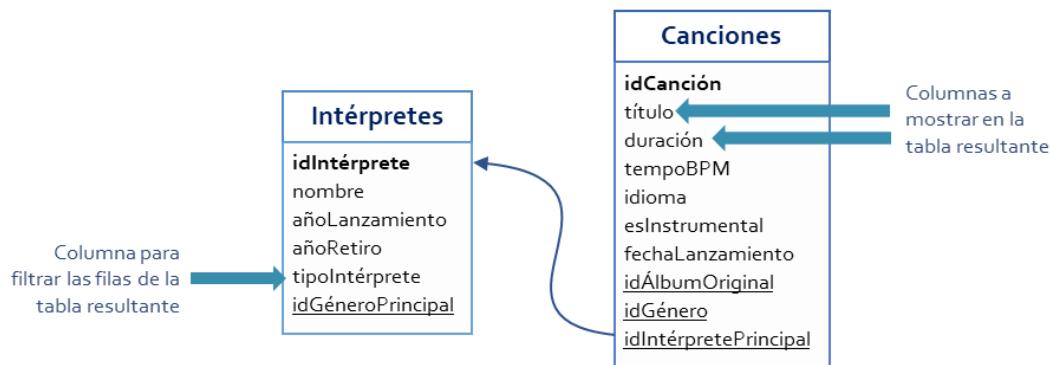
Hasta este punto se han utilizado subconsultas para obtener un escalar que es utilizado en una expresión condicional. En tal sentido, todas las consultas internas generan una tabla compuesta por una fila y una columna. No obstante, hay otros casos en los cuales se requieren subconsultas que generen más 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

Dese el inicio se ha argumentado que antes de responder cualquier necesidad de datos es recomendable un análisis para determinar lo qué 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 las canciones, 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 consulta principal y la consulta 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

título	duración
Amarte mas no pude	00:04:49
Cali es sabrosura	00:03:14
Ginza	00:02:51
Hips dont 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

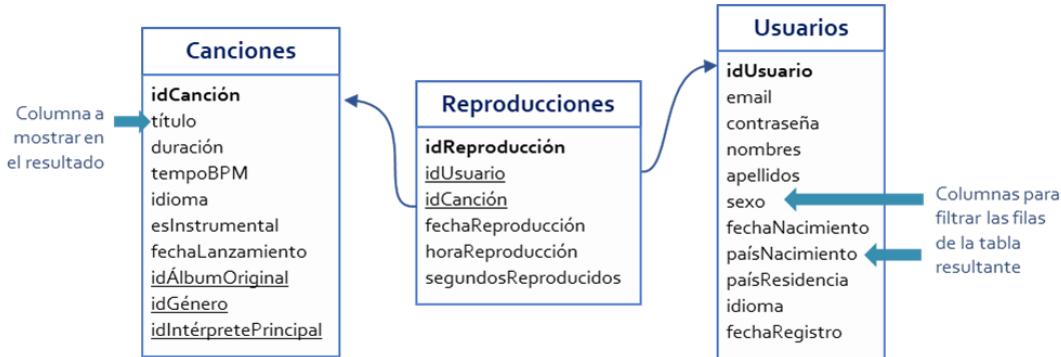
```
SELECT título,
       duración,
       idIntérpretePrincipal
  FROM Canciones
 WHERE idIntérpretePrincipal IN (SELECT idIntérprete
                                    FROM Intérpretes
                                   WHERE tipoIntérprete = 'Solista')
 ORDER BY título
```

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 una 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, identificar los *Usuarios* de sexo femenino nacidos en Colombia. Esto puede obtenerse 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 la consulta del Script 4-16, en la cual se utiliza la consulta del Script 4-15 como subconsulta para generar los *idUsuario* requeridos por la consulta externa para obtener 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 la consulta del Script 4-16. En este sentido, la consulta completa para responder la pregunta es la presentada 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

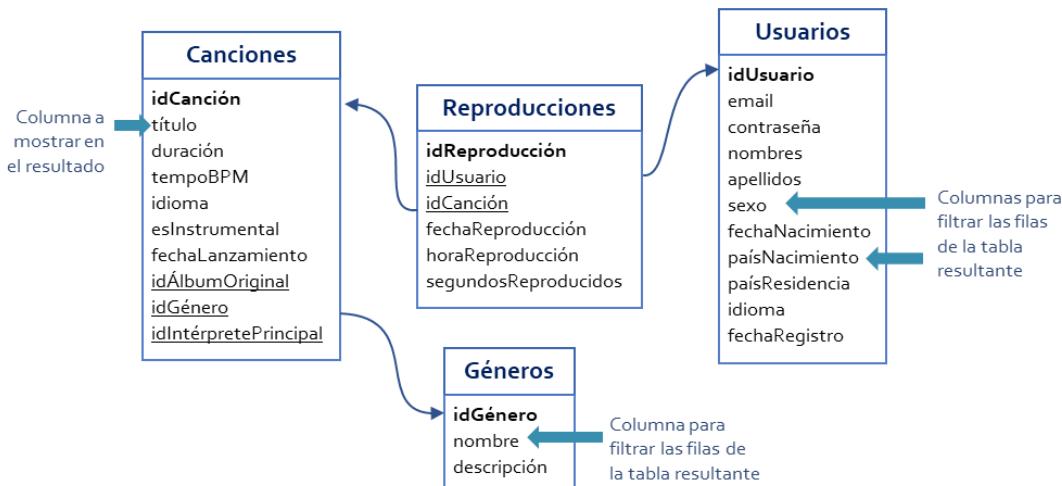
```
SELECT título
FROM Canciones
WHERE idCanción IN (SELECT idCanción
                     FROM Reproducciones
                     WHERE idUsuario IN (SELECT idUsuario
                                          FROM Usuarios
                                          WHERE sexo = 'F' AND
            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

En una consulta pueden utilizarse subconsultas que obtienen un valor y subconsultas que obtienen una lista de valores. Para esto puede plantearse una variación al caso resuelto 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 requeridas para resolver la pregunta.

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 y el resultado obtenido con su ejecución se observa 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

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

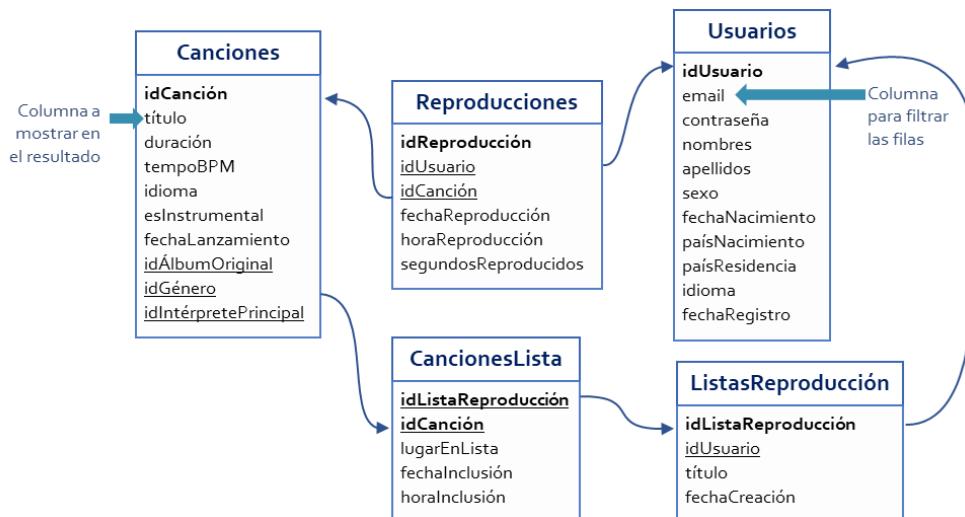
En algunos casos el nivel de anidamiento puede ser alto, generando una estructura de subconsultas que podría llegar a ser confusa a primera vista. Sin embargo, este tipo de consultas son fáciles de entender si se aplica la noción de descomposición de los problemas y se van obteniendo resultados parciales 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 email *alex@gmail.com*, pero que dicho usuario nunca las 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



Para la primera condición se requiere 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 una subconsulta para obtener 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. Si se requiere verificar que un valor está contenido en una lista de valores puede utilizarse el operador = en conjunto con la palabra ANY. Cuando se requiere verificar 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.

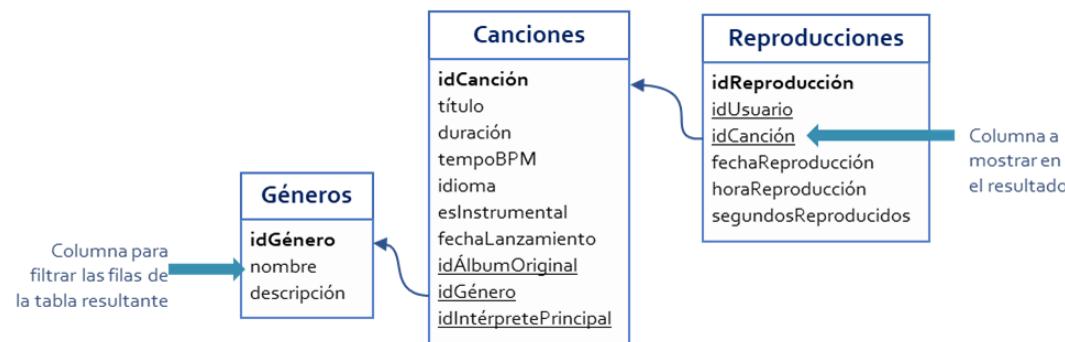
En este sentido, la consulta del Script 4-19 podría modificarse reemplazando la expresión IN con la expresión = ANY (es igual a alguno) y la expresión NOT IN con la expresión <> ALL (es diferente de todos), obteniendo el mismo resultado. Los constructos ANY, SOME y ALL abren muchas posibilidades de comparación 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 esa cantidad 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, puede construirse una consulta para obtener las cantidades de reproducciones que han tenido las canciones del género ‘Salsa’, tal y como se presenta en el Script 4-21.

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 4-21, 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**.

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

```

El uso dado a las subconsultas hasta ahora ha sido para obtener un valor escalar o una lista de valores, no obstante, el uso de las subconsultas 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.

¿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



Esto puede 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 obtenido de la ejecución de esta consulta se presenta en la Tabla 4-6.

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

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**. Para encontrar el valor máximo de una columna de una tabla 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.

Para completar la solución a esta pregunta debe recordarse que el resultado de una consulta es tratado por el SQL como una tabla, una tabla derivada. Puede pensarse como si se creara una tabla temporal que contiene los datos resultantes de la consulta. En este sentido, la consulta del Script 4-22 puede incluirse como una subconsulta en la cláusula **FROM** asignándole un alias y la consulta principal la puede utilizar de la misma forma en que se utiliza una tabla que existe físicamente en la base de datos.

En el Script 4-23 se muestra como se construye la consulta principal utilizando el Script 4-22 como una consulta interna o subconsulta en la cláusula **FROM** con el alias **CancionesPorlista**. 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 10 canciones?

Para resolver esta pregunta se hace necesario calcular la cantidad de canciones de cada lista y luego filtrar aquellas en las que la cantidad obtenida sea mayor a 10. En la Figura 4-11 se presenta un diagrama con la tabla requerida para resolver la pregunta.

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 `10`, 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. Esto 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 utilizar la cláusula `WHERE`, como se muestra en el Script 4-26.

Script 4-26

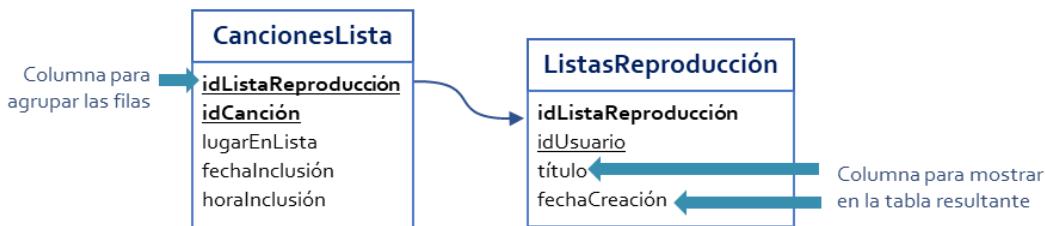
```
SELECT COUNT(*) AS cantidadListas
  FROM (SELECT idListaReproducción,
              COUNT(idCanción) AS cantidadCanciones
        FROM Cancioneslista
       GROUP BY idListaReproducción) AS ListasConCantidadCanciones
 WHERE cantidadCanciones > 10
```

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. 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 para resolver la pregunta.

Figura 4-12 Tablas y columnas requeridas para responder la pregunta



En este sentido, el Script 4-23 se utilizará como una subconsulta para obtener el valor a utilizar dentro de una expresión condicional para filtrar las filas de una consulta que permita obtener los identificadores de las listas de reproducción que tengan esa cantidad de canciones. 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 utilizado 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 dependen 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, puede plantearse la posibilidad de utilizar una subconsulta que permita obtener la duración máxima que tienen las canciones de un género en particular y así filtrar las filas de la tabla Canciones que tenga 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 valor de cada `idGénero` y así registrar las soluciones parciales. 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 diferentes 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. 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

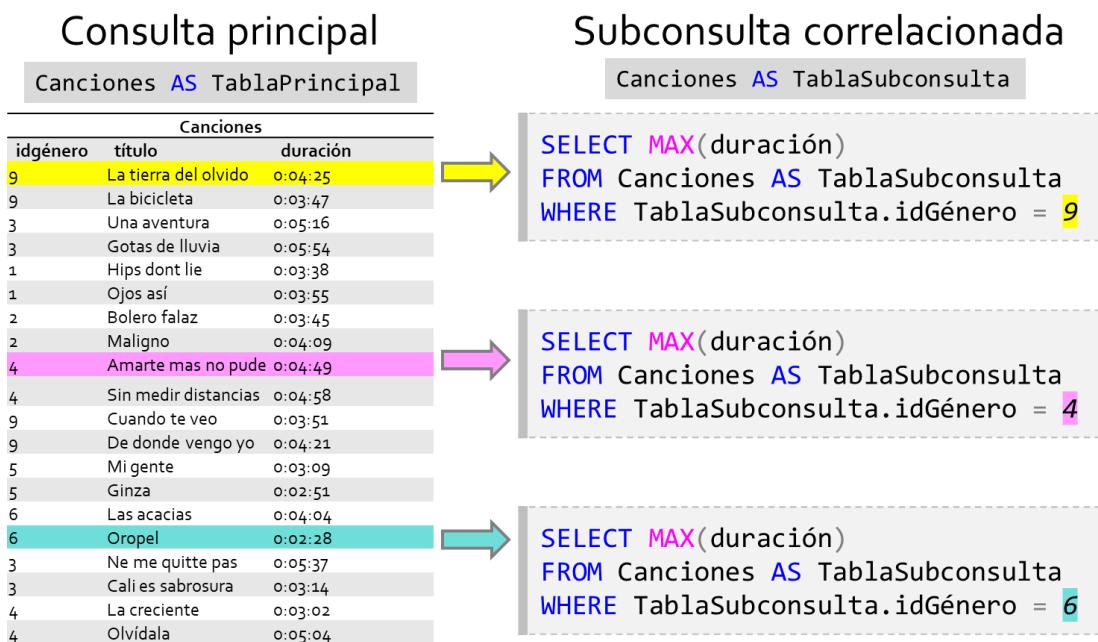
```
SELECT idGénero,
       título,
       duración
  FROM Canciones AS TablaPrincipal
 WHERE duración = (SELECT MAX(duración)
                     FROM Canciones AS TablaSubconsulta
                    WHERE TablaSubconsulta.idGénero = TablaPrincipal.idGénero)
```

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` identificada con 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 el asignar el alias no implica que se esté cambiando el nombre de la tabla. También debe estar claro que el utilizar la misma tabla en la subconsulta y asignarle otro alias no significa que se modifique la tabla o que se esté haciendo referencia 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.

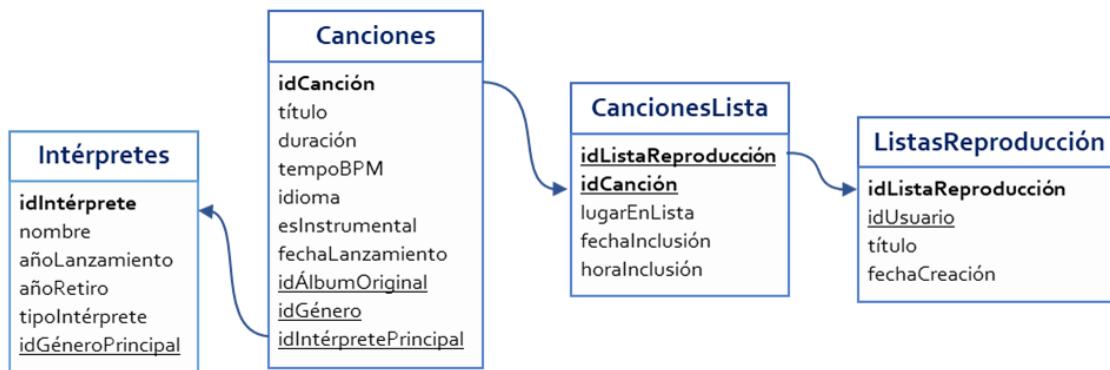
¿Cuál son los títulos de las canciones y los títulos de las listas de reproducción de las canciones interpretadas por “Carlos Vives” que han sido agregadas a listas de reproducción?

La respuesta esperada es una tabla con dos columnas. En la primera columna se presentará el título de la canción y en la segunda el título de la lista de reproducción en la cual está agregada dicha canción. Si una canción interpretada por “Carlos Vives” está agregada a varias listas de reproducción entonces deberá aparecer en varias filas de la tabla resultante. De igual forma, el título de una lista de reproducción aparecerá varias veces si en dicha lista hay varias canciones interpretadas por “Carlos Vives”.

En la tabla `CancionesLista` están almacenados los identificadores de las listas de reproducción y los identificadores de las canciones que pertenecen a las listas. En esta tabla no están los títulos de las canciones ni los títulos de las listas de reproducción.

Por consiguiente, se hace 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 “Carlos Vives”.

Esto 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 identificador de la canción interpretada por “Carlos Vives” que está contenida en dicha lista de reproducción. Ahora solamente falta presentar los títulos de las listas de reproducción y los títulos de las canciones en lugar de presentar los identificadores.

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'))
  
```

Para llegar a esta solución puede pensarse inicialmente 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, puede pensarse la forma de obtener el título de una lista de reproducción a partir del identificador como se muestra en el Script 4-32 para la lista de reproducción con `idListaReproducción` igual a 3.

Script 4-32

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

Script 4-33

```
SELECT título  
FROM ListasReproducción  
WHERE idListaReproducción = 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.

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 al 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 devolver 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 cuyo 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. Dándole la posibilidad de que la consulta secundaria utilice valores de la consulta principal.

4.4 Actividades de aplicación para evidenciar lo aprendido

- I. 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 período.

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 suplir 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 del género, la cantidad de canciones 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 cuales tipos de operaciones de combinación de tablas deben utilizarse y son las más adecuados en diferentes casos de uso

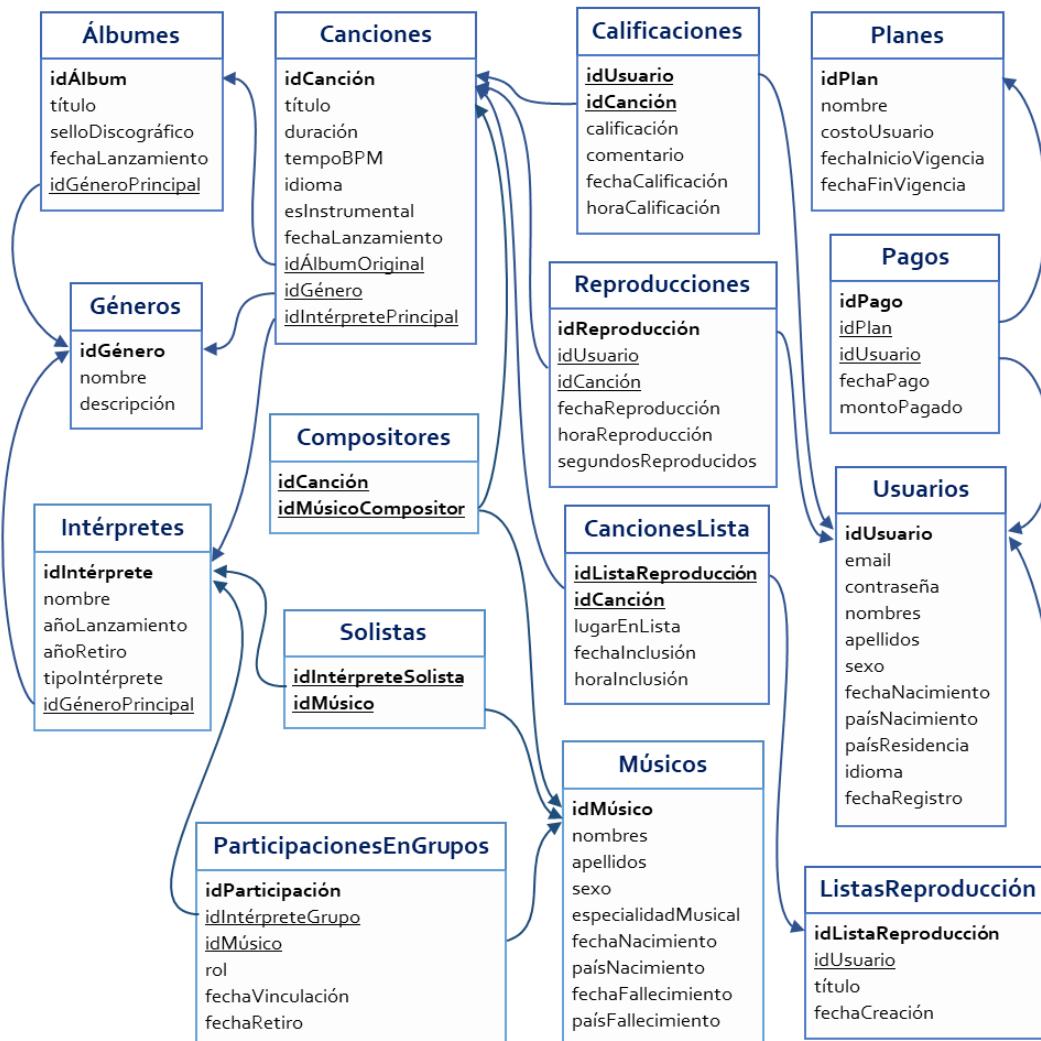
En las consultas para dar respuesta a las necesidades de datos normalmente deben utilizarse varias tablas. En ciertos casos, todos los datos requeridos están en una sola tabla, pero se hace necesario utilizar datos de otras tablas para realizar ciertas operaciones, como puede ser el filtrado de filas 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 necesario incluir en la consulta algunas operaciones que permitan el uso de esos datos.

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 generar 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 de 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 esta versión 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.

Figura 5-1 Versión 4 de la base de datos de la plataforma “Mis Canciones”



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 las 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 al momento de registrar un pago de un usuario, este valor puede cambiar en el tiempo.

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

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. Un músico puede ser un intérprete solista o puede hacer parte de un grupo. Además, un músico puede ser 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.

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 eso 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, al mismo tiempo podría ser solista y además podría hacer parte de una agrupación. Es importante que la base de datos permita almacenar todos estos detalles para facilitarle a los usuarios la búsqueda de canciones.

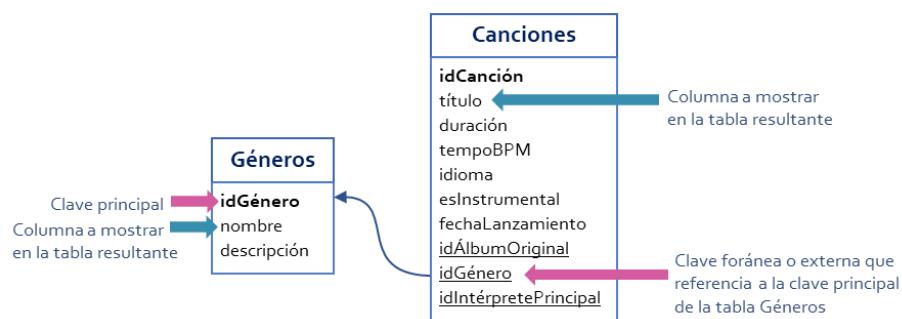
5.1 Combinación interna

Los datos en una base de datos se almacenan en varias tablas y es común que las necesidades que deben suplirse demanden, asimismo, la obtención y presentación de datos de varias tablas. Con las subconsultas se abrió la posibilidad de utilizar datos de varias 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 las necesidades de datos. En este caso, los datos que se requieren para generar el resultado están almacenados en dos tablas diferentes, la tabla Canciones y la tabla Géneros. En la Figura 5-2 se presenta un diagrama con las tablas de la base de datos requeridas para resolver la pregunta.

Figura 5-2 Tablas requeridas para responder la pregunta



El título de las canciones está almacenado en la tabla Canciones y el nombre del género en la tabla Géneros. En la tabla Canciones se tiene la columna idGénero con un número entero que corresponde al identificador del género de la canción según lo almacenado en la tabla Géneros. Por lo tanto, para dar respuesta a la pregunta debe utilizarse alguna operación que permite combinar los datos de las dos tablas de forma tal que la tabla resultante tenga dos columnas, la primera con el título de la canción y la segunda con el nombre del 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 en el producto cartesiano son las tablas, entendiendo que cada tabla 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, es decir, dos filas.

Figura 5-3 Ejemplo de un producto cartesiano entre dos tablas

Tabla **Canciones**

<i>idCanción</i>	<i>título</i>	<i>idGénero</i>
1004	Azul	3
1005	Cielo	4

Tabla **Géneros**

<i>idGénero</i>	<i>nombre</i>
3	Pop
4	Rock

Canciones* producto cartesiano *Géneros

(Canciones X Géneros)

<i>idCanción</i>	<i>título</i>	<i>idGénero</i>	<i>idGénero</i>	<i>nombre</i>
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 el ejemplo de 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, la columna *idGénero*. En este caso, las filas que tienen sentido son aquellas en las cuales coinciden los datos almacenados en las dos tablas, como se sucede en la primera y la última fila, 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, correspondiente 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.

Script 5-1

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

En el Script 5-1 se observa que en la cláusula `FROM` se indican las tablas a combinar separadas por una coma. Esto 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` de las dos tablas. 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 utilizan una sola tabla.

El nombre de las columnas se presenta en conjunto con el nombre de la tabla a la que pertenece porque las dos tablas tienen columnas con exactamente el mismo nombre y el SQL exige que no existan ambigüedades causadas por la coincidencia en el nombre de 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. De requerirse todas las columnas de las tablas participantes en la combinación, puede utilizarse el símbolo `*`, de la misma forma en que se utiliza 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`, proveniente de la tabla `Géneros`, lo cual 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.

Tabla 5-1

título	género
Amarte mas 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 dont 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. Allí se introdujo una forma diferente, más precisa y organizada de especificar las combinaciones. Se introdujo la palabra `JOIN` con sus diferentes variaciones y la palabra `ON` para establecer las expresiones condicionales a 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 es 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 una combinación interna. En otras palabras, si en lugar de utilizar la expresión `INNER JOIN` se utiliza únicamente 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 utilizar únicamente la palabra `JOIN` o de especificar las tablas a 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

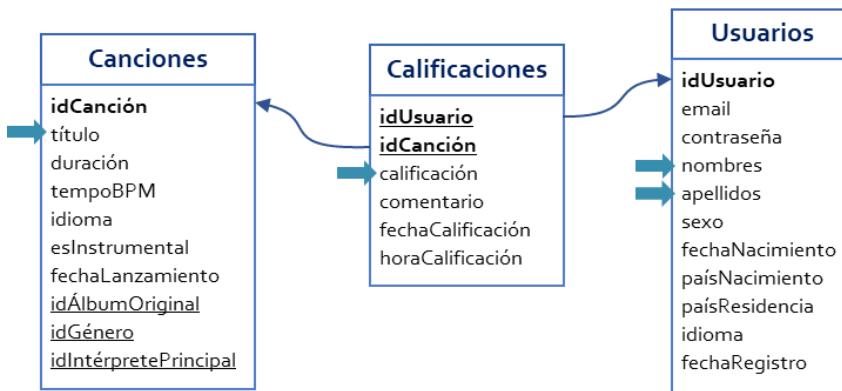
```
SELECT título,
       (SELECT nombre
        FROM Géneros
        WHERE Géneros.idGénero = Canciones.idGénero) AS género
  FROM Canciones
 ORDER BY título
```

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.

¿Cuál es título de la canción, los nombres y apellidos del usuario y la calificación asignada por cada usuario a cada una de las canciones?

Los datos necesarios para responder la pregunta están almacenados en tres tablas. Los títulos de las canciones están en la tabla Canciones, los nombres y apellidos de los usuarios están en la tabla Usuarios y las calificaciones asignadas por los usuarios a las canciones están en la tabla 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, es posible especificar dentro de la cláusula `FROM` una combinación de dos tablas con la expresión `INNER JOIN` y con la palabra `ON` para definir las condiciones de combinación, y ese resultado combinarlo 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. La tabla Canciones tiene como clave principal la columna `idCanción` y la tabla Calificaciones también tiene la columna `idCanción` la cual es una clave foránea que referencia a la columna `idCanción` 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 coincide con el valor almacenado en la columna `idCanción` de alguna fila de la tabla Calificaciones. El Script 5-4 muestra una consulta SQL con esta combinación.

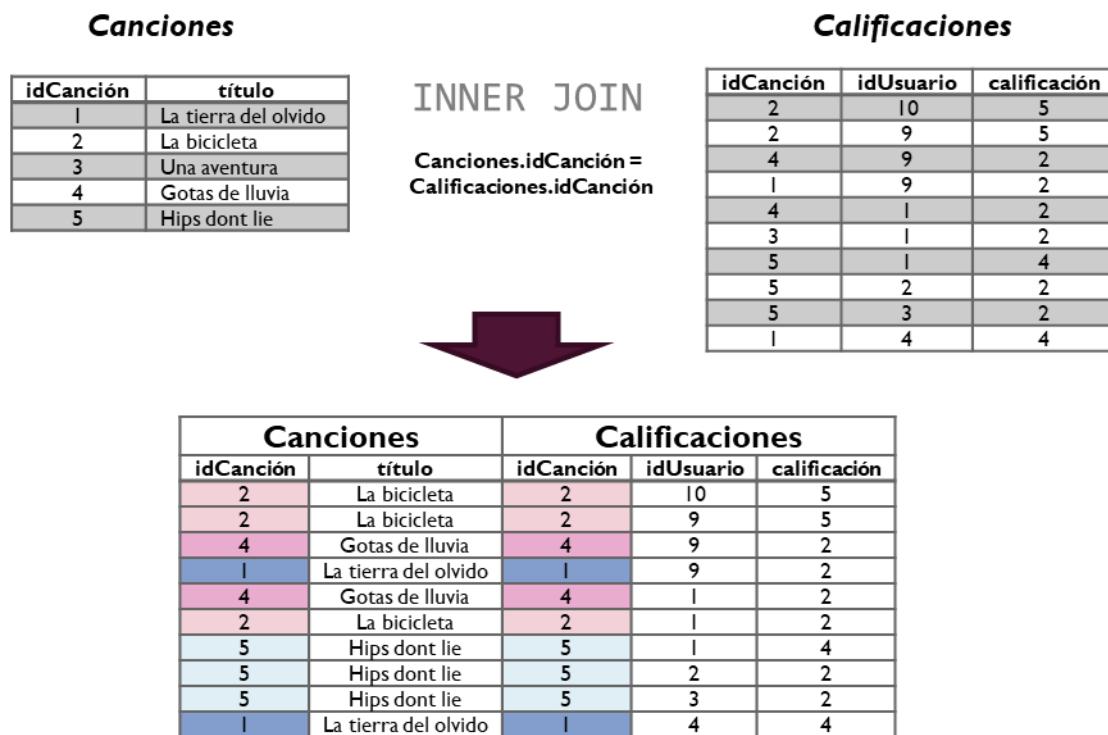
Script 5-4

```
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 combinación con relación a las columnas comunes, que en este caso son las columnas idCanción de ambas tablas.

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 almacenados 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.

Figura 5-5 Combinación interna de filas de las tablas Canciones y Calificaciones



Si la tabla Canciones tuviese solamente esas dos columnas y esas cinco filas, y la tabla Calificaciones tuviese solamente esas tres columnas y esas diez filas, el resultado de la combinación interna sería una tabla cinco columnas, dos provenientes de Canciones y tres provenientes de Calificaciones, y diez filas, las mismas diez filas 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 tanto, el DBMS no encuentra filas coincidentes y queda por fuera del resultado.

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

En la tabla Usuarios existe la columna idUsuario, la cual está definida como clave principal. Po su parte, en la tabla Calificaciones también exoste una columna llamada idUsuario la cuál es una clave foránea que referencia a la columna idUsuario, clave principal de la tabla Usuarios. 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 combinación.

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 concatenando los nombres y los apellidos de los usuarios en una sola cadena de texto, la cual será una columna derivada llamada nombreCompleto, tal y como se presenta en la consulta del Script 5-6.

Script 5-6

```

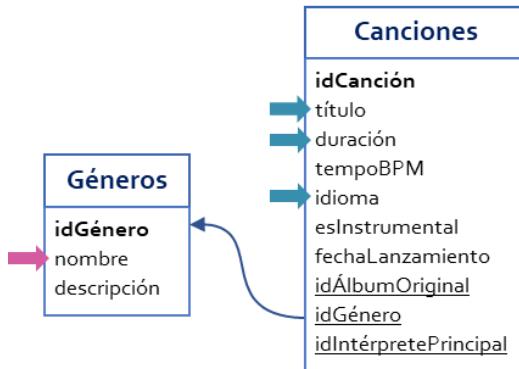
SELECT CONCAT(Usuarios.nombres, ' ', Usuarios.apellidos) AS nombreCompleto
      Canciones.título,
      Calificaciones.calificación
FROM Canciones
  INNER JOIN Calificaciones
    ON Canciones.idCanción = Calificaciones.idCanción
  INNER JOIN Usuarios
    ON Usuarios.idUsuario = Calificaciones.idUsuario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

En una consulta SQL pueden combinarse todas las tablas que sean necesarias para obtener el resultado esperado. 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, la 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'
```

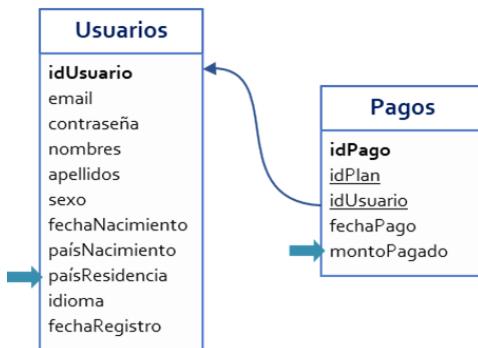
La combinación interna se realiza utilizando como criterio la igualdad en los datos de la columna `idGénero` de la tabla `Canciones` y la columna `idGénero` de la tabla `Géneros`. Nuevamente se utiliza la relación clave principal - clave foránea para conformar las filas combinadas en la tabla resultante. 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 pueden ser la aplicación de funciones escalares, funciones de agregación y 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. Los datos de los pagos realizados por los usuarios están almacenados en la tabla `Pagos`, en la cual se tiene la columna `idUser` como clave foránea que referencia a la columna `idUser` de la tabla `Usuarios` y la columna `montoPagado` que representa la cantidad de dinero de cada transacción realizada por cada usuario. 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 de combinación las columnas que establecen la relación entre las dos tablas al ser clave principal y 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 utilice la función de agregación `SUM` sobre los datos de la columna `montoPagado` de la tabla `Pagos`.

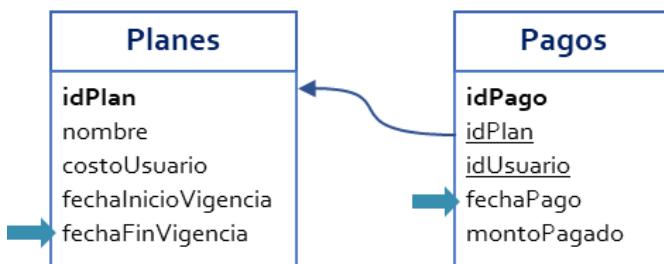
Para realizar el conteo de los usuarios se utiliza la función de agregación `COUNT` sobre los datos de la columna `idUser` de la tabla `Usuarios` en conjunto con la palabra `DISTINCT`. Esto es necesario porque un usuario puede tener registrados varios pagos y su identificador aparecerá tantas veces como 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.

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 `=`. Aunque este es el uso más frecuente no es la única forma de realizar las combinaciones. En la cláusula `ON` 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 busca identificar los pagos realizados por los usuarios en alguna fecha posterior a la fecha en la que el plan en el que estaban afiliados perdiera vigencia. Los datos requeridos para resolver esta pregunta 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 fechalinicioVigencia 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, en la cual se almacena 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, en la cual se utiliza 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 igual 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, para identificar si hay alguna fila en la tabla Pagos que tenga una fecha de pago anterior a la fecha 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

```

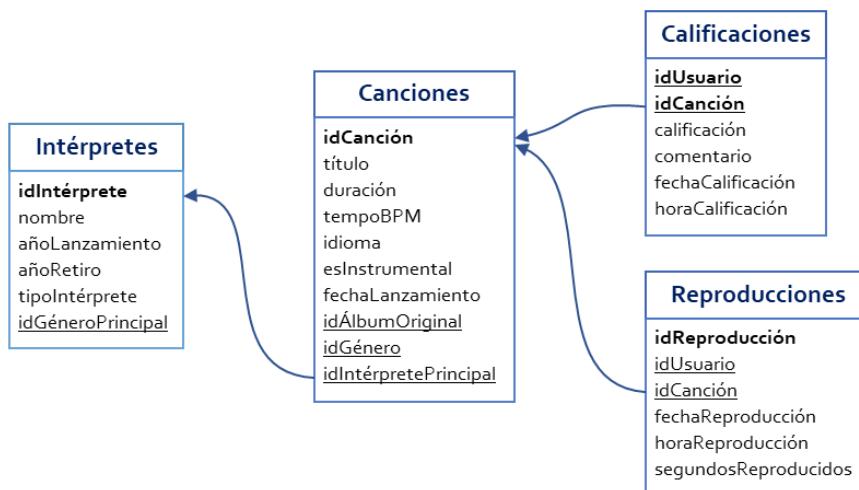
SELECT *
FROM Pagos
INNER JOIN Planes ON Pagos.idPlan = Planes.idPlan AND
                    (Pagos.fechaPago < Planes.fechaInicioVigencia OR
                     Pagos.fechaPago > Planes.fechaFinVigencia)
  
```

En la mayoría de los casos se combinan tablas de la base de datos. Sin embargo, hay casos que demandan la combinación entre 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 las canciones del intérprete y la calificación promedio de todas las canciones de intérprete. En tal sentido, para responder la pregunta se requieren datos de las 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. Con esa combinación 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 las canciones de cada intérprete. 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 esta combinación.

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 convierte en una subconsulta que hace las veces de una tabla en la cláusula `FROM`. Las tablas generadas por ambas subconsultas son combinadas utilizando `INNER JOIN` y estableciendo como condición de combinación la columna `idIntérprete`. Esta consulta puede dar el resultado, pero tiene una limitación. En caso de que algún intérprete tenga que canciones que no hayan sido calificadas o que no hayan sido reproducidas, los datos de ese intérprete 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.

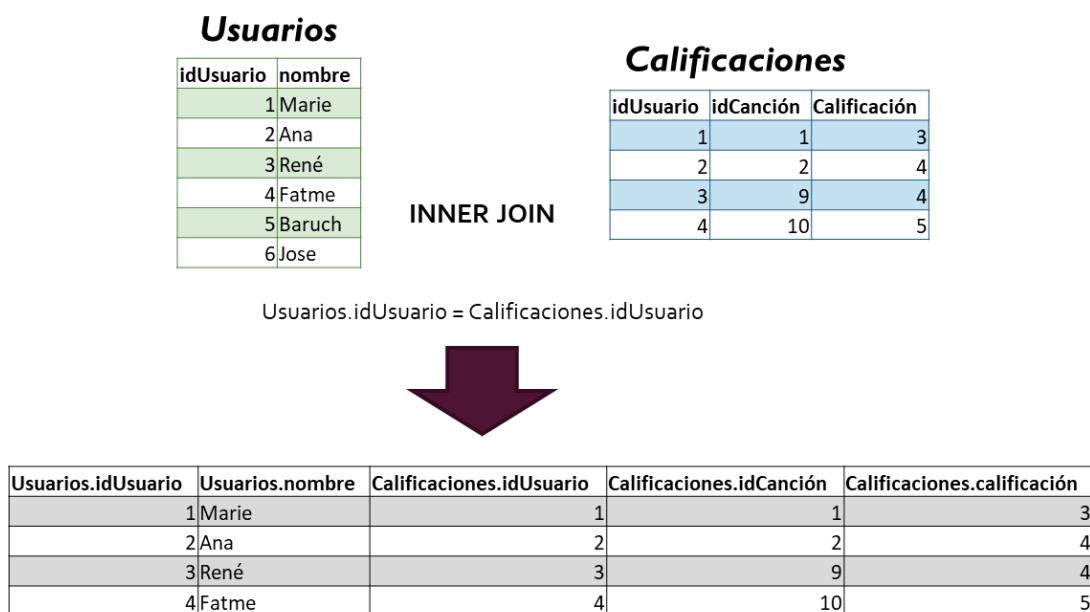
5.2 Combinación externa

En algunos 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 fundamenta 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 la tabla `Usuarios` y la tabla `Calificaciones`. Puede utilizarse una combinación interna entre las tablas o `INNER JOIN` que utilice la columna `idUser` la cual es clave principal de la tabla `Usuarios` y es clave foránea en la tabla `Calificaciones` como criterio para determinar las filas que se incluirán en la combinación. En la Figura 5-10 se ejemplifica la combinación con unas versiones simplificadas de las tablas.

Figura 5-10 Ejemplo de combinación interna



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 la tabla `Usuarios` y la tabla `Calificaciones` se nota que el usuario con

identificador “6” y nombre “José” no ha calificado canciones, o lo que es lo mismo, su identificador aparece en la tabla Usuarios, pero no en la tabla Calificaciones. Por esta razón es que el usuario “José”, al igual que el usuario “Baruch” no aparecen en la tabla resultante de la combinación y tampoco aparecería en el resultado que se obtendría si se ejecuta la consulta del Script 5-18, en la cual se utiliza la función COUNT para establecer la cantidad de canciones que ha calificado cada usuario.

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
```

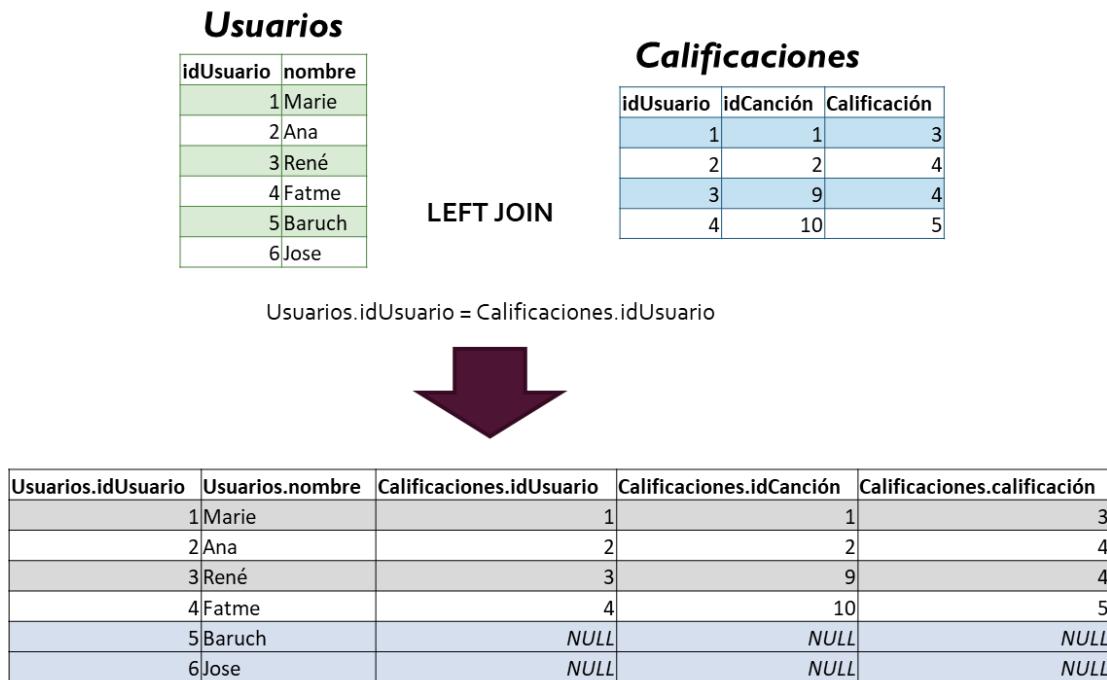
En el Script 5-18 se observa que la condición de la combinación está planteada con la igualdad de los datos de IdUsuario en ambas tablas, causando que los usuarios que no han calificado canciones no se incluyan en la tabla resultante. Sin embargo, al volver a la pregunta que motiva la creación de esta consulta podría inferirse que en el resultado se espera que aparezcan todos los usuarios “*registrados en la plataforma*” y no solamente los usuarios 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 o OUTER JOINS, las cuales tiene 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 filas de la tabla ubicada en la parte izquierda de la combinación que no cumplen la condición y debe asignar 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 cumplen 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 se incluyen las filas de la tabla Usuarios que no tienen coincidencia con alguna fila de la tabla 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 presentada en la Figura 5-11, incluye dos filas adicionales. La fila del usuario “Baruch” y la fila del usuario “Jose”, que son los usuarios que aún no han calificado canciones. Como no hay filas coincidentes en la tabla Calificaciones, el DBMS pone en NULL los datos de las columnas idUsuario, idCanción y calificación de la tabla de la derecha, es decir la tabla Calificaciones.

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. Este valor cero se obtiene 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.

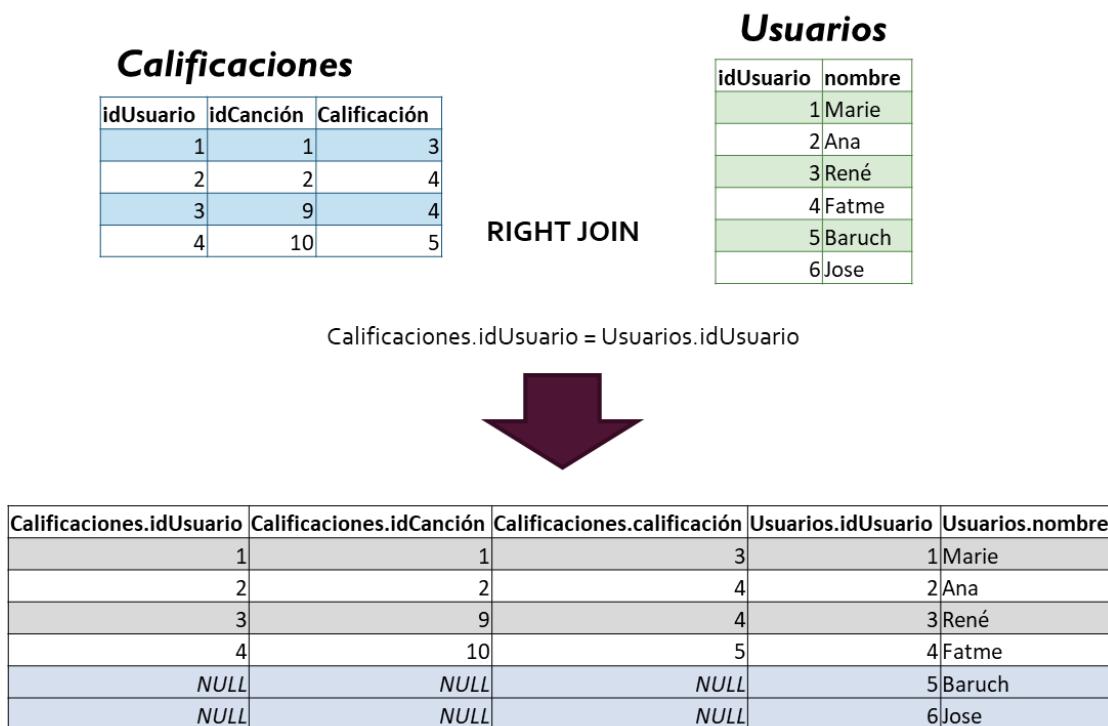
Script 5-16

```

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
  
```

Así como existe la combinación izquierda *LEFT OUTER JOIN* también existe la combinación externa derecha *RIGHT OUTER JOIN* o simplemente *RIGHT JOIN*. Esta combinación realiza la misma operación que *LEFT JOIN*, pero en el otro sentido. Es decir, incluye todas las filas de la tabla ubicada en la parte derecha de la combinación que no cumplen la condición y asigna valores nulos a las columnas correspondientes a la tabla ubicada en la parte izquierda de la combinación, 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.

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
  
```

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 que 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 están en la tabla CancionesLista.

Teniendo claro lo anterior, debemos comenzar con la combinación de las tablas. Primero combinemos la tabla Canciones y Reproducciones, como lo que queremos 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.idReproducciones
  FROM Canciones AS C
 INNER JOIN Reproducciones AS R
    ON C.idCanción = R.idCanción
```

Habiendo combinado la tabla Canciones con la tabla Reproducciones poder continuar a combinar con la tabla CancionesLista, ¿Qué tipo de JOIN utilizamos? Al revisar con detenimiento la pregunta nos percataremos que la respuesta es obvia, se nos está pidiendo las canciones que han sido reproducidas pero que no están en una lista, es decir, que están en la tabla Reproducciones, pero no están en la tabla CancionesLista. Si ubicamos la tabla Reproducciones a la izquierda del JOIN y la tabla CancionesLista a la derecha del JOIN, sería 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
```

Al realizar el LEFT JOIN tendremos todas las canciones que estén en la tabla Reproducciones incluidas las que estén en una lista y las que no. Por lo que el siguiente paso es quedarnos solo con las que no están en la tabla *CancionesLista*.

Al realizar la combinación, recordemos 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 son llenadas con NULL. Por consiguiente, como resultado del LEFT JOIN tendremos todas las canciones que han sido reproducidas, pero no agregadas a listas con NULL en las columnas correspondientes a la tabla *CancionesLista*. Planteado lo anterior, para mantener únicamente las que no han sido agregadas a una lista, preguntamos en el WHERE que el identificador de la lista de reproducción sea nulo.

También agregamos un DISTINCT porque al momento de realizar la combinación entre Canciones y Reproducciones, el hecho de que una canción puede haber sido reproducida más de una vez, nos llevará a tener más de una fila por la misma canción. El DISTINCT nos permitirá quedarnos con una sola fila por cada canción.

¿Cuál es el nombre de todas las canciones y de todos los álbumes presentes en la plataforma?

Esta pregunta nos pone en un contexto similar a los anteriores, pero diferente en sus detalles. En la tabla Álbumes tenemos álbumes cuyas canciones aún no han sido agregadas a la plataforma y también tenemos canciones de las cuales no sabemos el álbum al que pertenece. Por consiguiente, necesitamos 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 requiere las filas de la parte izquierda del JOIN y de la parte derecha del JOIN.

Para cumplir con lo anterior, SQL proporciona el FULL OUTER JOIN, este nos permite mantener las filas de ambas tablas que intervienen en la combinación, cumplan o no la condición, llenando con NULL las columnas correspondientes para las filas que no cumplen con la condición para la combinación. El Script 5-23 muestra el SQL correspondiente para realizar el FULL OUTER JOIN entre estas dos tablas.

Script 5-20

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

Como resultado de esta consulta tendríamos la Tabla 5-2. En esta tabla las filas con rellenos amarillos representan esos casos donde o la canción no tiene asignado un

álbum (como el caso de “Calí es sabrosura”) o esos álbumes que no tienen asignada al menos una canción (como es el caso de “Déjame entrar” y “Pies descalzos”).

Tabla 5-2

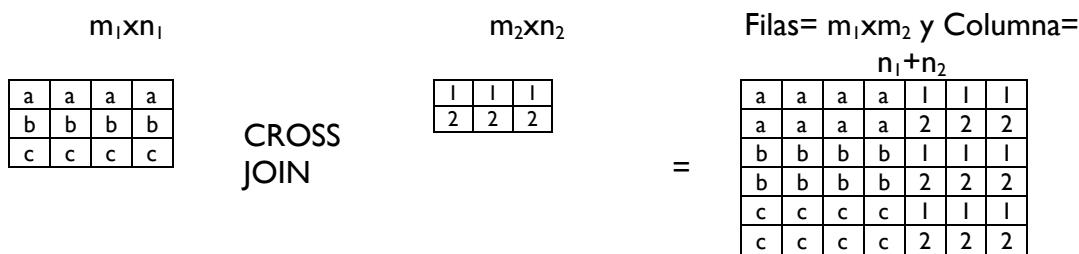
títuloCanción	títuloÁlbum
La tierra del olvido	La tierra del olvido
Vives	La bicicleta
Cielo de tambores	Una aventura
Huellas del pasado	Gotas de lluvia
Oral Fixation, Vol. 2	Hips dont lie
¿Dónde están los ladrones?	Ojos así
El Dorado	Bolero falaz
Caribe atómico	Maligno
Título de amor	Amarte mas no pude
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 para la combinación una condición y dependiendo del cumplimiento de esta se realiza la combinación. En el INNER JOIN se incluyen solo las filas de ambas tablas que cumplen la condición y en el OUTER se pueden mantener filas de las tablas que intervienen en la combinación, aunque estas no cumplan la condición de la combinación. Una tercera forma de combinación es la conocida como “producto cartesiano”, “producto cruz” o simplemente CROSS JOIN.

El producto cartesiano al combinar dos tablas 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. Si deseamos combinar con un producto cartesiano la tabla “ListasReproducción” y la tabla “Canciones”, se tomará cada una de las listas de reproducción y se combinará con cada una de las canciones sin tener en cuenta ninguna condición para la combinación. Si la tabla “ListasReproducción” tiene 20 filas y la tabla “Canciones” tiene 20 filas, al final tendremos 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 la tabla “ListasReproducción” (supongamos que tiene 4 columnas) como las de la tabla Canciones (supongamos, tiene 10), es decir, la nueva tabla tendrá 14 columnas. La Figura 5-13, muestra lo que acabamos de describir.

Figura 5-13



El Script 5-21 muestra una forma para especificar el producto cartesiano entre dos tablas y en el Script 5-22 se presenta una forma más explícita para realizar esta combinación haciendo uso 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 tenemos 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 siguiente paso sería quedarnos únicamente con las canciones que aún no han sido agregadas a la lista 2, esto lo podemos realizar con 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 podemos realizar combinaciones de tablas de tres tipos: INNER JOIN, OUTER JOIN y CROSS JOIN.
- El INNER JOIN lo podemos utilizar en los escenarios donde la combinación se hará entre las filas que cumpla una condición.
- El uso más común para realizar las combinaciones en los INNER JOIN es la igualdad de una clave foránea y una clave primaria, no obstante, no en la condición se puede incluir cualquier expresión de la cual se puede determinar su valor de verdad.
- El OUTER JOIN lo utilizamos cuando queremos mantener las filas que no cumplen la condición de combinación, si queremos mantener las filas de la izquierda usamos el LEFT OUTER JOIN; si queremos mantener las filas de la derecha el RIGHT OUTER JOIN; si queremos mantener ambas filas el FULL OUTER JOIN.
- El CROSS JOIN lo utilizamos cuando deseamos 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 soporta dos formas de escribir los JOIN, una siguiente la sintaxis de 1986 y otra siguiendo la sintaxis de 1992. La comunidad recomienda como una buena práctica utilizar la de 1992 porque incrementa la legibilidad hacer explicitar las operaciones que se realizan.

5.5 Actividades de aplicación para evidenciar lo aprendido

1. Escriba una consulta para obtener el título, la duración, el título del álbum original y el nombre del sello discográfico del álbum original de 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 las listas de reproducción, el número de canciones incluidas en cada lista, la duración total de las canciones incluidas y el email del usuario que la creó, de todas las listas que hayan sido creadas por usuarios residentes en Colombia.
6. Modifique el siguiente script para obtener el mismo resultado sin utilizar subconsultas y argumente cuál es la necesidad de datos que se está respondiendo.

```
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 y el nombre de los grupos en los que han participado aquellos músicos que han hecho parte de más de un grupo durante su carrera.
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” para cada intérprete. Si todas las canciones de un intérprete hacen parte de algún álbum, el nombre de ese intérprete 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 esa consulta tiene 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, intersección y 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 obtienen datos desde una o varias tablas y estos datos pueden presentarse de forma resumida, agrupada o transformada, utilizando funciones escalares y funciones de agregación. Aunque las operaciones presentadas son muy potentes y permiten resolver la mayoría de las necesidades de datos, el SQL incluye otras operaciones que brindan otras 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. En tal sentido, 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 las tablas 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 va a ejecutar la operación. La primera restricción 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?

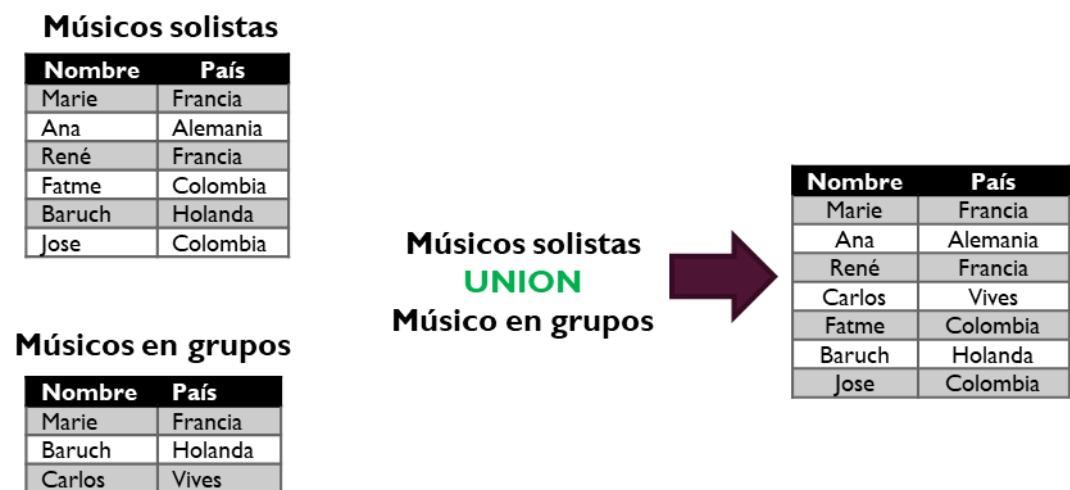
Para responder a esta pregunta puede identificarse que los datos requeridos están almacenados en las tablas `Usuarios` y `Músicos`, las cuales se presentan en la Figura 6-1. Específicamente es necesario obtener algunos de los datos de los usuarios y unir ese resultado con lo que se obtenga de consultar 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 como resultado de una consulta que devuelve el nombre y el país de músicos que son solistas con otra tabla obtenida como resultado de una consulta que devuelve el nombre y el país de músicos que participan en grupos, el resultado obtenido será una tabla con 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 como sería la operación unión de esas dos tablas de datos, en la cual se precisa que la tabla resultante tendría siete filas porque las filas que están repetidas, a saber, 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 las dos tablas. En este sentido, la solución puede abordarse dividiendo el problema en dos problemas 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 tendrá filas de la tabla [Usuarios](#) seguidas de filas de la tabla [Músicos](#). Del resultado 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](#). Este caso podría suceder si algún músico tiene un usuario registrado en la plataforma con los mismos datos con los que está registrado como músico. Si en algún escenario debe 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, agregar una columna para incluir una cadena de texto que diga si la fila proviene de la tabla [Usuarios](#) o de la tabla [Músicos](#). La nueva versión de la consulta con este elemento se muestra en el Script 6-4. Como hay la 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 proponemos resolver 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 contendrá los identificadores de los músicos que han sido solistas o que han participado en grupos. Esto puede obtenerse uniendo los identificadores de los

músicos que están almacenados en la tabla Solistas con los identificadores de músicos de la tabla ParticipacionesEnGrupos, lo cual se puede obtener con la consulta del Script 6-5. Aunque para efectos del resultado final no habría ninguna diferencia, en esta consulta se utiliza UNION ALL porque no es necesario eliminar los duplicados, dado que el resultado se utilizará como una subconsulta con el operador IN. La unión de las consultas puede utilizarse como subconsulta 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);
```

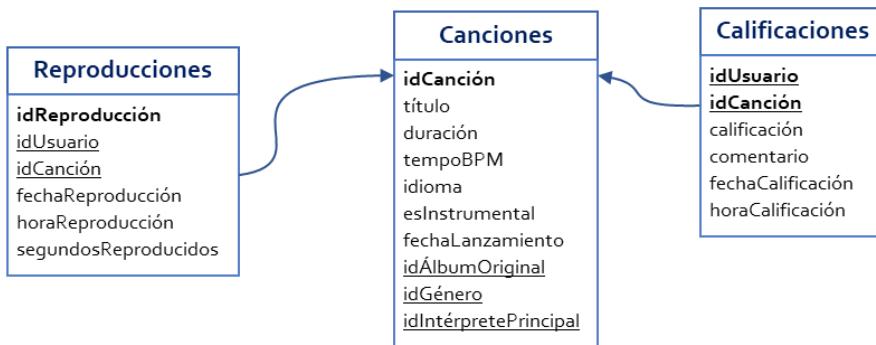
La operación UNION puede combinarse con las demás operaciones para responder preguntas como la que se plantea a continuación.

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

En esta pregunta se requieren datos almacenados en las tablas Canciones, específicamente el título de la canción, pero se necesitan 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, esto se presenta en 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)
    
```

De manera similar, el Script 6-8 permite obtener la canción con mejor promedio de calificación y 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 canciones con mejor valoración, tal como se muestra en el Script 6-9.

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)

```

Las operaciones de conjunto permiten 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 responderse se necesita realizar operaciones de filtrado basadas en los datos de las tablas Compositores, Solistas y ParticipacionesEnGrupos, las cuales se presentan en la Figura 6-5.

Figura 6-5



La respuesta a esta pregunta implica la obtención de los datos de los músicos, específicamente los identificadores, que estén almacenados en tres tablas. Esto puede lograrse utilizando tres subconsultas, pero también puede obtenerse utilizando la operación intersección.

En la operación 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, se obtiene como resultado una tabla de dos filas y dos columnas, con los datos de los músicos de nombre “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 identificado está solamente en dos tablas, no debe incluirse. La consulta que permite obtener este conjunto de identificadores se presenta en el Script 6-10.

Script 6-10

```

SELECT idMúsico
FROM Solistas
INTERSECT
SELECT idMúsicoCompositor
FROM Compositores
INTERSECT
SELECT idMúsico
FROM ParticipacionesEnGrupos ;

```

La consulta presentada en el Script 6-10 puede utilizar como una subconsulta que genera los datos requeridos para filtrar 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 6-11

```

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. El conjunto A será el de la izquierda y el conjunto B el de la derecha de la palabra **EXCEPT**. En la Figura 6-7 se presenta un ejemplo de lo descrito, se calcula la diferencia entre los músicos que son solistas y los músicos que han pertenecido a grupos. El conjunto de datos resultante contendrá únicamente los músicos solistas que nunca han estado en un grupo musical, que 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 pregunta se está pidiendo los nombres y los apellidos de los músicos cuyo identificador esté almacenado en la tabla Solistas pero que no esté 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 tres se trabajó con las funciones escalares y las funciones de agrupamiento. Las primeras están diseñadas para operar sobre un valor y las segundas para resumir un conjunto de valores. En el SQL existe 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. Sin embargo, 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 dato 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
```

El 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. La tabla resultante entonces

Figura 6-8 Ejemplo de procesamiento por subconjuntos o grupos de filas

The diagram shows a table of song data on the left and a summary table on the right. Arrows point from each group of rows in the main table to its corresponding average value in the summary table.

idCanción	título	idGénero	duración
5	Hips dont 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 mas 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 presentado en la columna duraciónPromedioPorGénero es el mismo.

Figura 6-9

idCanción	título	idGénero	duración	duraciónPromedioPorGénero
5	Hips dont 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 mas no pude	4	0:04:49	00:04:28.25
19	La creciente	4	0:03:02	00:04:28.25
20	Ovidala	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

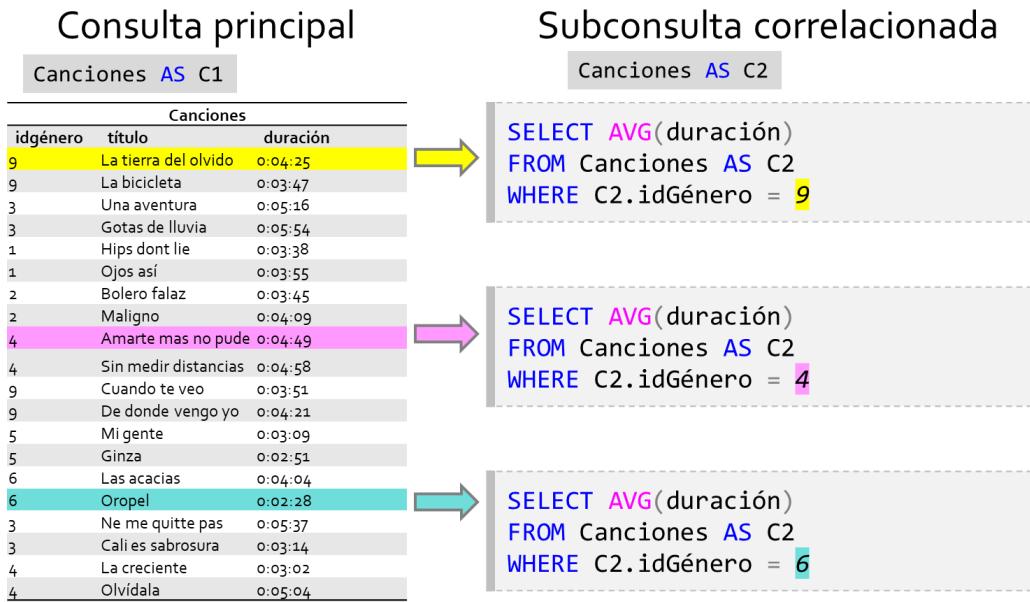
Una segunda alternativa para responder la pregunta es utilizar una subconsulta correlacionada, tal y como se presenta en el Script 6-17. La subconsulta correlacionada genera un único valor por cada fila de la tabla Canciones que tiene el alias C1, este valor se muestra en la tabla resultante como una 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



Una tercera alternativa para responder la pregunta se logra utilizando el mismo concepto general del procesamiento por grupos que se hace con la cláusula `GROUP BY`, pero en lugar de grupos, lo que se definen son particiones o ventas 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 grupos 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 la palabra **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, utilizando la cláusula **ORDER BY** seguida de los nombres de las columnas que se utilizarán 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 funciones de agregación que ya se habían utilizado y otras funciones que permiten generar 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 debe utilizarse 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 requeridas.

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 canciones que no han sido reproducidas y su identificador no está almacenado en la tabla Reproducciones. La tabla resultante se presenta ordenada descendente por la cantidad de reproducciones y da la sensación de que esa sería la respuesta esperada.

Sin embargo, aunque el listado que se obtiene está ordenado, cuando se utiliza el concepto escalafón o ranking, se espera que exista al menos una columna que indique la posición de cada canción en dicho escalafón. En otras palabras, se espera una columna con números naturales ordenados de menor a mayor, la cual representará la posición, el lugar que ocupa la canción en ese escalafón. Para obtener lo que se está solicitando, el SQL ofrece una conjunto de funciones que permiten generar este tipo de columnas de “posiciones” en un ranking, tal como se presenta 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

```

En la consulta del Script 6-20, construida a partir de la consulta presentada en el Script 6-19, se adicionaron tres columnas a la tabla resultante. Para agregar estas tres columnas se utilizan las funciones de ventana denominada `ROW_NUMBER`, `RANK` y `DENSE_RANK`.

En la Tabla 6-1 se presenta un ejemplo de una posible tabla resultante de ejecutar esta consulta, el cual permite comprender de manera más fácil el funcionamiento de estas tres funciones que, 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 observa las filas resaltadas pueden identificarse las diferencias en los números generados.

Tabla 6-1

Género	Título	Cantidad	fila	puesto	puestoDenso
Latina fusión	La bicicleta	51	1	1	1
Pop	Hips dont 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 mas 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

La función `ROW_NUMBER` enumera todas las filas, simplemente asigna un número entero que se va incrementando de uno en uno de acuerdo con el orden en que aparecen las filas. Este número entero representa la posición de la fila dentro de la tabla y es lo que se presenta en 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. Es por esto por lo que en la Tabla 6-1, hay dos filas que tienen el mismo valor en la columna `puesto`, específicamente las filas 10 y 11 tienen asignado el número 10 y las filas 13 y 14 tienen asignado el número 13. En otras palabras, si hay un empate entre dos filas, las dos filas tendrán asignada la misma posición. Hay que notar que la posición 11 no se asigna, sino que la siguiente posición del ranking será la posición 12, tal como se observa para la fila 12 de la tabla.

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, se observa que en la columna `puestoDenso` para las canciones de las filas 10 y 11 se asignó el valor 10, es decir que esas dos canciones están empatadas en la posición 10, y 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 aplicaran sobre ventanas, es decir, cuando se incluye la expresión `OVER`, no aparece la expresión `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 estas particiones. 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, el 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

```

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 con título “Hips dont lie” es la más reproducida y que la canción “Ojos así” ocupa el segundo lugar. En el caso del género “Salsa” se observa que la más reproducida es “Una aventura”, seguida por “Ne me quitte pas” y luego “Cali es sabrosura”.

Tabla 6-2 Ejemplo de resultado obtenido con el Script 6-21

Género	Título	Cantidad	DENSE_RANK
Pop	Hips dont 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 mas 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

Ya habiendo presentado las funciones de ventana, es tiempo para comentar que las funciones de agregaciones como vimos 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, perdida de detalle. Cuando creamos grupos de datos utilizando la cláusula GROUP BY, todos los cálculos que realicemos 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 el 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. Resolvamos la siguiente pregunta para hacer lo anterior más evidente.

¿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 dividir el valor resultante entre la cantidad total de reproducciones ($100 * \text{CRC} / \text{CRT}$). Hacer este proceso utilizando únicamente agrupamientos y subconsultas implicaría construir una consulta bastante extensa. 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, es decir, 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 si una canción 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 las columnas con posición coincidente tenga el mismo 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 la misma.
- Lo más sorprendente de las funciones de ventanas es que pueden ejecutarse operaciones de agregación o resumen sin perder el detalle, lo que si ocurre con las funciones de agregación y la operación de agrupamiento.

6.4 Actividades de aplicación para evidenciar lo aprendido

1. Escribir 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 por cada mes.
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 que 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.
6. Escriba una consulta que muestre la cantidad de usuarios que han reproducido las canciones por cada uno de los intérpretes. De manera que se obtenga 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 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 ejecutare 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 puede obtenerse con una consulta. En este sentido, este capítulo introduce la expresión [CASE](#), las expresiones [LIMIT](#) y [OFFSET](#), y las operaciones para generar columnas al pivotar datos de filas en nuevas columnas. En este capítulo 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, nos permite obtener un valor basado en una condición lógica. 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 la expresión 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. Resolvamos la siguiente pregunta para ejemplificar el uso de la expresión CASE.

¿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 preguntas que hemos respondido es que se nos pide derivar un escalar de tipo texto dependiendo de la cantidad de canciones que un usuario haya calificado. Esto es lo que podemos 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 este entre cuatro y ocho, debe obtenerse “entre cuatro y ocho canciones”. Cuando este entre uno y tres, entonces que escriba “entre una y tres canciones”. Si ninguna de las condiciones anteriores se cumple (ELSE), que escriba “Ninguna canción”. Lo que acabamos de expresar, en SQL lo podemos escribir como está en el Script 7-1.

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, la expresión CASE puede ser incluida en cualquier lugar donde se acepte un escalar, reformulemos 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 debemos 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. Lo pretendido lo podemos conseguir 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, su implementación puede variar de uno a otro. En el caso de POSTGRES existen las palabras reservadas LIMIT y OFFSET. La primera palabra, LIMIT, permite utilizar un número para indicar cuantas filas se desean obtener, por ejemplo, 1 para indicar que se desea que el conjunto de datos resultante este compuesto por una fila. La segunda palabra, OFFSET, se utiliza para indicar a partir de cuál fila queremos obtener 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 respondamos la siguiente pregunta.

¿Cuál es el título de las últimas cinco canciones reproducidas?

Para responder esta pregunta primero ordenaremos de forma descendente las reproducciones realizadas de las canciones utilizando como criterio de ordenamiento la fecha y hora de reproducción de cada canción, 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 hora de reproducción, solo tenemos que incluir la palabra reservada LIMIT seguida del número 5 para indicar al DMBS que solo nos interesan 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 utilizar el enfoque anterior tiene el siguiente inconveniente: si las canciones devueltas por la consulta, antes de alcanzar el LIMIT, en la quinta y sexta posición tienen la misma fecha y hora de reproducción solo una se devolverá. Por lo anterior, en algunos escenarios no se puede asegurar que siempre se obtendrán los mismos resultados (adquiere la consulta un cierto grado de no determinismo)

porque en una próxima ejecución la canción ubicada en la fila seis, ahora puede estar en la cinco y la cinco en la seis.

También podemos 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. Ejemplifiquemos con la siguiente pregunta.

¿Cuál es el nombre de la primera canción que se reprodujo después de las diez primeras?

La pregunta actual es similar a la respondida con el Script 7-5, con la diferencia que en esta se nos está pidiendo que omitamos las diez primeras filas y que mostremos solo la primera que sigue a esas diez. Puesto en otras palabras, mostrar la fila número once. Para saltar las diez primeras filas es solo necesario incluir la palabra reservada OFFSET seguida del valor 10, como se realizó 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. En algunas ocasiones es necesario convertir los valores de una columna en columnas de manera que mejore la compresión de los datos por parte de quien los visualiza. Utilicemos la siguiente pregunta como escenario para ejemplificar esto.

¿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 que entendemos lo que se nos están pidiendo, tener claro el estado al que queremos llegar para decir, ¡cumplimos 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 este resultado es

explotar las potencialidades de la expresión CASE, derivando 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 deseamos contar la reproducción, si es igual le pasamos a la función COUNT ese id de reproducción. La consulta resultante sería la del Script 7-6.

Tabla 7-1

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 utilizando la función CROSSTAB que está incluida dentro de la extensión de POSTGRES denominada “tablefunc”. La 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 la tenemos en el Script 7-7. Al ejecutarla obtenemos 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 vemos que los años aparecen en la columna “Años” y cantidades para esos años en la columna “Cantidad”. Como lo que queremos es que los valores distintos de la columna “Año” se conviertan en columna, es precisamente

este 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. Esto se presenta en el Script 7-10.

Tabla 7-2

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;CREATE extension tablefunc;
```

Con estos dos parámetros POSTGRES tomará la primera consulta y la dejará intacta y la segunda, de los años, los valores distintos definirán las nuevas columnas y los valores de la tercera columna los asigna 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 central
    '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);
```

Finalmente, los dos enfoques anteriores nos permiten obtener el resultado deseado, tal como se presenta en la Tabla 7-3, en la que podemos ver fácilmente que durante el 2017 la canción con identificador “5” tuvo cero reproducciones.

Tabla 7-3

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

7.4 Aprendizajes más importantes del Capítulo 7

- La expresión CASE nos permite incluir lógica condicional en la determinación de los valores escalares. Esto abre un gran abanico de posibilidades puesto que podemos incluirla en cualquier lugar donde una consulta acepte un escalar.
- Con la palabra reservada LIMIT podemos especificar cuál es la cantidad precisa de filas que deseamos mostrar de una consulta.
- El operador OFFSET permite especificar a partir de qué posición deseamos empezar a mostrar los resultados.
- Podemos reorganizar las filas y columnas de una tabla pivoteándolas. Esto puede ser conseguido utilizando la expresión CASE o la función CROSSTAB.

7.5 Actividades de aplicación para evidenciar lo aprendido

1. Escribir 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 por cada mes.
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 de la consulta.

```

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 manera que se obtenga 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 suplidas 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 de una base de datos relacional. Por el contrario, esta actividad demanda la identificación y análisis de las necesidades de almacenamiento de datos, la especificación de las características que debe exhibir la solución a implementar y la especificación de 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 suplirse con ese nuevo vehículo e identifica las características que debe tener, 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 le permita realizar pruebas simuladas. Finalmente, se crean unos primeros vehículos y, si todo va bien, empieza la producción masiva.

Diseñar una base de datos tiene cierta similaridad con este enfoque, aunque con la base de datos no se busca que la solución creada sea replicada muchas veces, frecuentemente si se busca que la solución creada sea usada por muchos usuarios de manera simultánea.

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 inefficiencia en el almacenamiento y la recuperación de datos. Esto se traduce muchas veces en sistemas software con demoras excesivas y datos incorrectos, si lo vemos desde un punto de vista técnico, y pérdida de clientes y dinero, si se extrapolara 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, pero que no lo haga un profesional de la informática es, además de poco presentable, peligroso.

8.2 Niveles de diseño

En el diseño de las bases de datos como en el de otros elementos software solemos hablar de tres niveles de diseño: el conceptual, el lógico y el físico. El primero, centrado 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 centra en los detalles de implementación en un DBMS.

No deben percibirse estos tres niveles aislados unos de otros y no deben realizarse con rigidez secuencial, como muestra la . entre cada uno de estos tres niveles hay interacción. En este capítulo nos centraremos en el diseño conceptual.

Figura 8-1



Para iniciar con el modelado conceptual de una base de datos, es condición ineludible identificar cuáles son los hechos que se están intentando modelar. Para poder 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, 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 exemplificar el modelado conceptual seguiremos utilizando el caso de la plataforma de canciones, pero en lugar de presentar las diferentes tablas que hemos utilizado hasta el capítulo anterior lo que haremos será presentar cuáles fueron las decisiones de diseño que fueron llevando a que las diferentes tablas fueran obteniendo la forma que se presenta. Recordemos que, dentro del dominio musical, hemos venido trabajando 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 presentados de forma incremental de forma que se aprecie cómo de manera iterativa podemos ir evolucionando la base de datos a través de la incorporación de nuevos requisitos, corrección de errores o introducción de mejoras. Los tres, acciones muy comunes en el ámbito profesional.

Para exemplificar el proceso del modelado conceptual usaremos la plataforma “Mis Canciones” para lo cual iniciaremos con 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 los artistas pueden ser solistas o grupos. También es importante tener presente 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 queremos modelar. En la descripción anterior tenemos que las entidades a la que se está haciendo alusión son las entidades “Canción” y “Artista”. Puesto en otras palabras, en la base de datos que vamos a modelar tendremos un conjunto de artistas y un conjunto de canciones y cada una de estas entidades tiene unos atributos que son importante para el contexto particular que se está trabajando. Miremos el caso de la entidad “Artista”.

En la plataforma “Mis Canciones” queremos poder almacenar datos de cada uno de los artistas, saber su nombre, la cantidad de años de vida artística, si es un grupo o un solista y el género principal que interpreta.

Del texto anterior identificamos que “Artista” representa una entidad que posee atributos como el nombre y la cantidad de años de vida artística. Una forma para identificar las entidades es poner atención en los sustantivos (artistas) o 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 los podemos identificar prestando atención donde el sustantivo o el sintagma nominal es una propiedad, calidad, identificador o característica de una de estas entidades como el nombre del artista, donde “nombre” es un atributo de la entidad “Artista”. Los atributos suelen responder a la pregunta: ¿qué datos queremos mantener de una entidad? En este caso, ¿qué datos queremos mantener de la entidad “Artista”?

En la figura 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 artista. Es importante señalar que los atributos que seleccionemos para representar la abstracción de una entidad dependen de los datos que tengan sentido para el dominio que se está trabajando. Para el caso de la plataforma “Mis Canciones” son irrelevantes los datos relacionados con la apariencia física del artista.

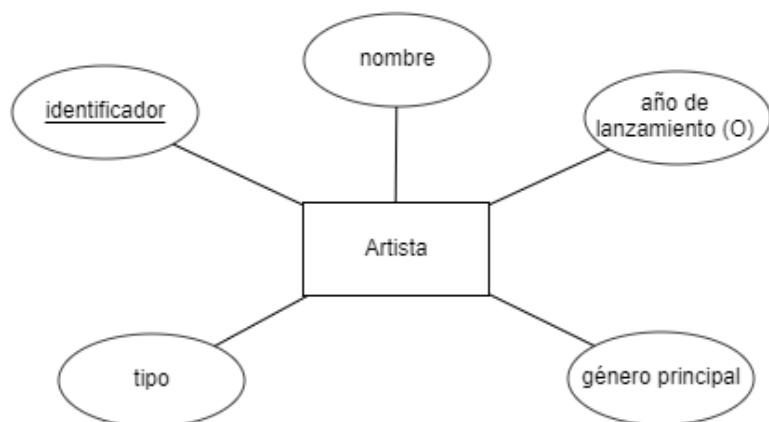
Una vez identificada la entidad que deseamos modelar y los atributos que esta posee, podemos utilizar un diagrama que nos permita representar de manera visual el modelo que empezamos a elaborar. Para realizar el modelado conceptual de una base de datos han surgido diferentes notaciones, en este libro utilizaremos la propuesta por Peter Chen, notación conocida como la 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 MIT. El desarrollo de su idea lo publicó en el año 1976 y lo tituló "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



Figura 8-3



Las entidades agrupan una colección de “objetos”, es decir la entidad “Artista” representa una abstracción que contiene los atributos que se consideran relevantes para representar un artista dentro de la plataforma “Mis Canciones”. Por lo tanto, la entidad “Artista” representa un molde que permitirá representar artistas concretos cuando los atributos tomen un valor específico. Por ejemplo, podrá representar un

artista que tenga por identificador “50001”, como nombre “Carlos Vives”, año de lanzamiento “1986”, tipo “solista” y género principal “Vallenato”. Así podemos utilizar este “molde” para representar tantos artistas como deseemos.

En el diagrama, uno de los atributos (el identificador) está subrayado, esto se debe a que una entidad debe garantizar que hay al menos una forma en que cada uno de los objetos que representa pueda ser identificados de manera única. El nombre no lo podemos utilizar para este fin porque puede haber más de un artista con el mismo nombre, también puede haber más de un artista con el mismo año de lanzamiento y así por los demás atributos, por lo cual, que el atributo “identificador” esté subrayado significa que por cada artista toma un valor distinto.

El atributo o grupo de atributos que hemos seleccionado para identificar de forma indistinta cada objeto recibe el nombre de llave principal de la entidad. Una entidad puede tener más de un atributo que permita identificarlo de forma indistinta; todos los atributos que satisfacen esta condición se llaman llaves candidatas. Supongamos que de los artistas también queremos almacenar el número de pasaporte, entonces el identificador y el número del pasaporte sería 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 vemos que uno de los atributos es representado incluyendo una letra “O” entre paréntesis luego del nombre, esto se hace para indicar que este atributo es **opcional**, o lo que es lo mismo en la plataforma “Mis Canciones”, queremos tener almacenados a los artistas, aunque se desconozca el “año de lanzamiento”.

Con el atributo “año de lanzamiento” sucede algo particular, si se observa nuevamente el requisito nos percataremos que lo que realmente se quiere saber es la cantidad de años de vida artística que tiene el artista y nosotros estamos almacenando es la fecha de lanzamiento. Esto lo hacemos debido a que los años de vida artística es un valor calculado y cambiante, es decir, hoy “Carlos Vives” tendrá 34 años de vida artística, pero el próximo año 35.

A razón de lo anterior, es recomendable, si es posible, utilizar atributos de los cuales se puedan calcular lo requerido, en este caso, partiendo del “año de lanzamiento” es posible calcular los años de vida artística. Los atributos cuyos valores se basan en los valores de otros atributos se conocen como atributos derivados. Otro ejemplo de un atributo derivado es la edad, 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 deriva se puede eliminar o modificar, en este caso, el atributo derivado debe mostrarse en el modelo de datos para evitar esta posible pérdida de información.

8.4 Relaciones entre entidades

En la plataforma “Mis Canciones” también queremos tener almacenadas canciones y de cada canción el título, el artista que la interpreta, la duración y el álbum a que pertenece.

Una buena estrategia al momento de modelar en un diagrama entidad relación los requisitos planteados es traducir el requisito en restricciones o reglas semánticas, por ejemplo, lo anterior lo podemos traducir en:

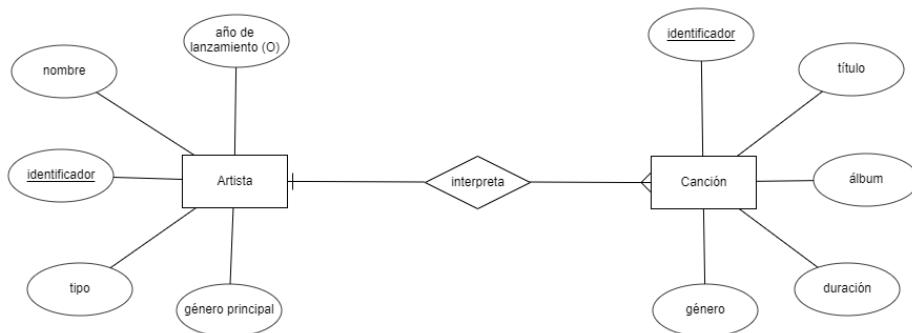
- Las canciones deben tener un identificador, un título, el nombre del álbum al que pertenece, la duración de la canción, el género en el que es clasificado y el artista que la interpreta.
- Un artista puede interpretar más de una canción.
- Una canción tendrá un artista principal que la interprete. Aunque una canción también puede ser interpretada por más de un artista en este contexto consideraremos únicamente el intérprete principal, es decir, cada canción tendrá solo un intérprete principal.

La primera restricción semántica solo nos está indicando que tendremos una entidad de nombre “Canción” y unos atributos asociados a ella, esto lo vemos en la parte derecha de la Figura 8-4. En cuanto a las otras dos restricciones planteadas vemos que se desea saber quién es el artista que interpreta una canción y cuáles canciones son las interpretadas por un artista. Una opción sería añadir un atributo del nombre del “artista” a la entidad “Canción”, pero nosotros ya tenemos una entidad “Artista” que tiene sus propios atributos, por lo tanto, lo que podemos hacer es establecer una relación entre estas dos entidades, a la relación le ponemos el nombre “interpreta”, es decir 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

	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

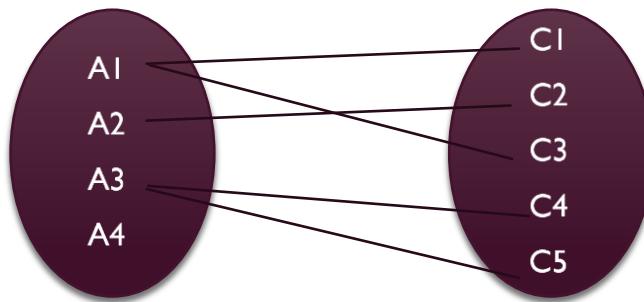
Figura 8-4



Otro aspecto que es importante acotar ahora que tenemos claro que entre ambas entidades podemos plantear una relación es cuántos objetos de cada entidad pueden intervenir en la relación, esto se conoce como cardinalidad e impone restricciones estructurales en el diseño. Como planteamos 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 “uno a muchos” en otro dominio sería una madre que puede tener cero o varios hijos, pero un hijo tiene una sola madre.

Para comprender como operan las relaciones de uno a muchos miremos la Figura 8-5. En esta vemos un diagrama sagital que nos representa dos conjuntos, el de la izquierda representa a los artistas y tiene cuatro elementos el artista “A1”, el artista “A2”, el artista “A3” y el artista “A4”. El conjunto de la derecha que representa las canciones tiene cinco elementos. En el mismo diagrama vemos que a un elemento del conjunto artistas, le pueden corresponder 0 o varias canciones. Mientras que a una canción solo le puede corresponder 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, se puedan involucrar más de dos entidades y también puede haber relaciones recursivas que involucren solo una entidad.

Figura 8-5



Cómo las entidades interactúan o se asocian entre sí, podemos pensar en las relaciones como si fueran verbos. Por ejemplo, en la frase “un artista interpreta canciones”, el verbo interpretar nos estará indicando cuál es la relación entre la

entidad “Artista” y la entidad “Canción”. En otro dominio, la frase “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. Las relaciones se muestran, por lo general, como rombos como vimos en la Figura 8-4.

Para cerrar con esta representación de la entidad canciones vemos que la llave principal es el identificador. Las llaves pueden ser naturales, es decir pueden provenir de dominio que se desea modelar, por ejemplo, si utilizamos el código estudiantil para representar la llave principal de una entidad “estudiante”. Pero, también pueden ser llaves artificiales o subrogadas, es decir, las añadimos a la entidad para facilitar y garantizar la unicidad de cada instancia de esa entidad, pero no existe 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, podríamos clasificar un mismo concepto como cualquiera o todos estos. El diseño es en cierto nivel subjetivo y depende de cómo se nos pida o como queramos 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. Analicemos el siguiente escenario.

En la plataforma “Mis Canciones” también queremos saber de cada álbum, además del título, el sello discográfico que lo produjo y la fecha de lanzamiento. De igual forma de los géneros también queremos saber el nombre y una breve descripción de cada uno. Adicionalmente, sería útil de cada artista saber el año en que se retiró

Al analizar la descripción anterior, hay un cambio sustancial con respecto al manejo que le estábamos dando a los conceptos de álbum y género, los estábamos trabajando como atributos de las entidades porque era lo que tenía sentido para nuestro diseño. Pero, ahora se presentan unas exigencias adicionales que hacen que nuestro diseño tenga que evolucionar. Ahora se nos indica que cada uno de estos conceptos tienen sus propios atributos, es decir, ahora 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”. Expresemos las restricciones semánticas que nos imponen estos requisitos.

- 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 género mientras que en un género 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.

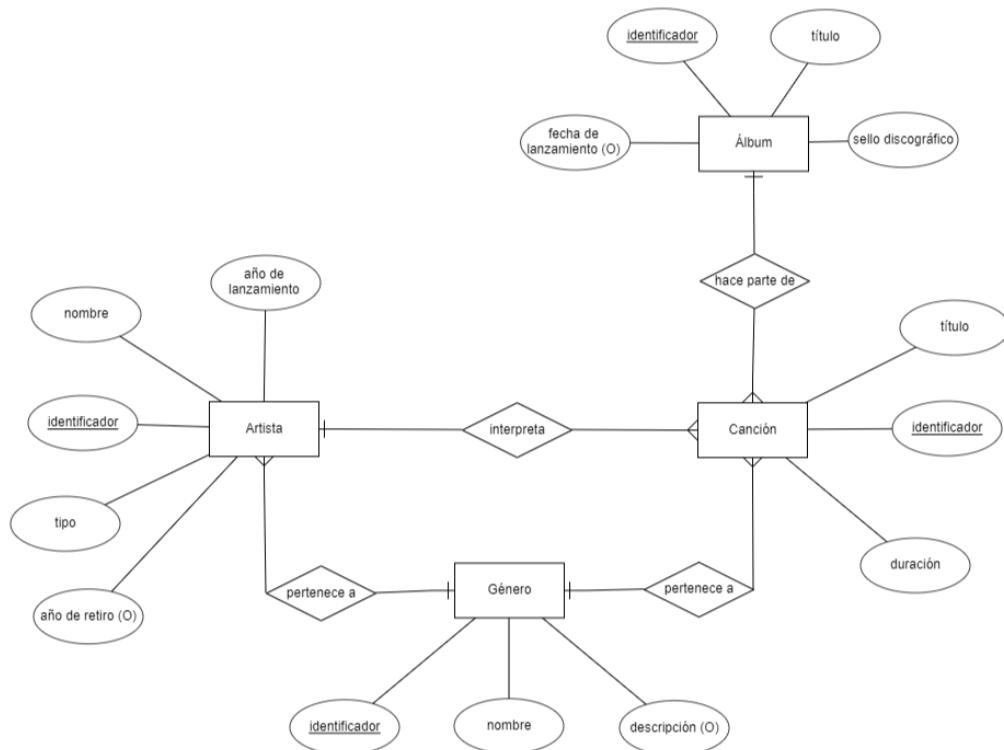
Las relaciones aquí descritas entre las entidades son todas de uno a muchos y se resumen en la Tabla 8-2. En

Figura 8-6 se presenta el diagrama ER que satisface los nuevos requisitos planteados.

Tabla 8-2

Relación	Entidades participantes	Cardinalidad
Hace parte de	Canción - Álbum	I:N
Pertenece	Género - Canción	I:N
Pertenece	Género - Artista	I:N

Figura 8-6



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 en se añadió el “año de retiro” del artista el cual es opcional.

Finalmente, debemos destacar que las entidades con que trabajamos pueden ser tangibles, o lo que es lo mismo, tienen una correspondiente representación material en el mundo natural o abstractas o conceptual, es decir, son ideas definidas por el hombre. En la Tabla 8-3 se aprecia un ejemplo de cada grupo.

Tabla 8-3

Entidades concretas	Entidades abstractas
Artista	Género Canción Álbum

No todas las relaciones que tenemos entre las entidades tienen que ser de uno a muchos como las que hemos tenido hasta este punto. Miremos la siguiente descripción.

“Mis Canciones” también debe almacenar datos de los usuarios (nombre, email y la contraseña). Además, queremos saber las canciones que reproducen, en qué momento lo hacen y por cuánto tiempo.

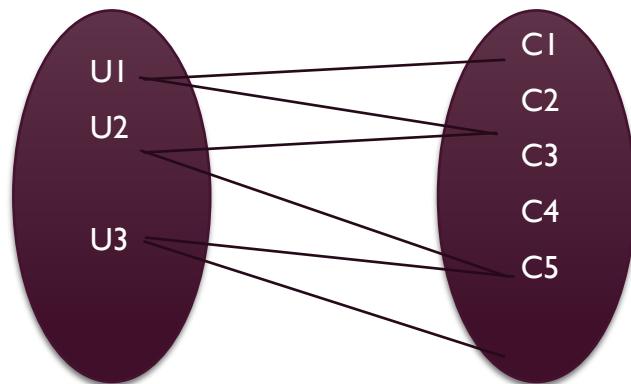
Como hemos realizado con las descripciones anteriores lo primero es convertir la descripción anterior en restricciones semánticas que queremos modelar. Estas son las siguientes:

- Tendremos usuarios de los que deseamos tener un identificador, correo electrónico, la contraseña, el nombre (primer nombre y segundo nombre), el sexo, la fecha de nacimiento, país de nacimiento, país de residencia, idioma y fecha de registro.
- Un usuario puede reproducir muchas canciones, inclusive muchas veces la misma canción. De igual manera una misma canción puede ser reproducida por muchos usuarios.

Esta última restricción nos está planteando un escenario diferente al que veníamos modelando hasta este punto con las relaciones de uno a muchos. Ahora se nos está diciendo que un objeto de la entidad “Canción” puede ser reproducido por muchos objetos de la entidad “Usuario” y que un objeto de la entidad “Usuario” puede reproducir muchos objetos de la entidad “Canción”. Este tipo de relación se conoce como “muchos a muchos”. Con la Figura 8-7 podemos ilustrar lo que sucede en una

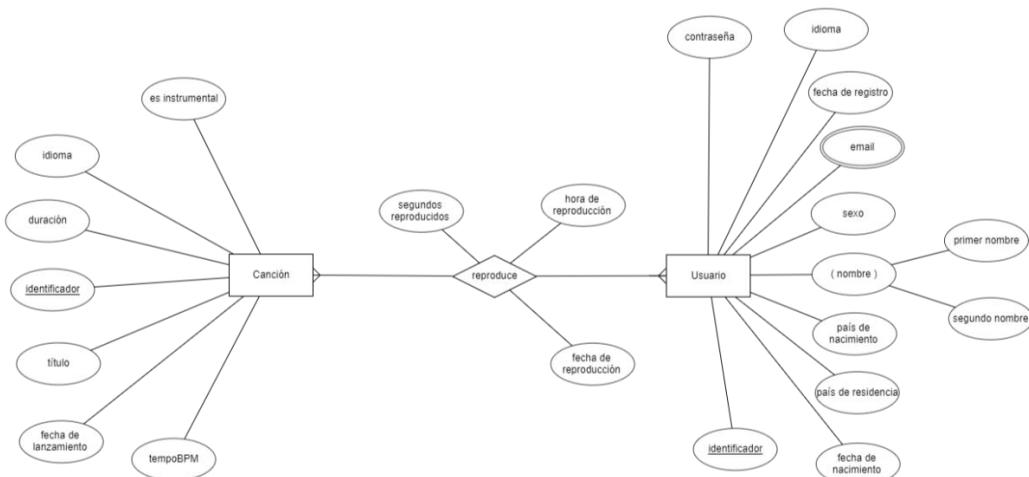
relación de muchos a muchos. En este tipo de relación cada uno de los ejemplos de un conjunto, por ejemplo, usuarios, puede relacionarse con uno o muchos elementos de otro conjunto, en este caso canciones. Lo inverso también se cumple, un elemento del conjunto que representa a las canciones puede estar relacionado con uno o muchos elementos del conjunto usuarios. En la Figura 8-8 podemos apreciar cómo se modelaría utilizando el diagrama ER.

Figura 8-7



En el diagrama también podemos observar que el atributo nombre es un atributo compuesto por 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 8-8



Además de ser simple o compuesto, un atributo también puede contener un solo valor o varios valores, como el caso el caso de correo electrónico (*email* en la Figura 8-8). Al decir que es un atributo multivalorado estamos indicando 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. Una alternativa a la presentada pudo haber sido representar los correos como una entidad separada de la entidad “Usuario”. No obstante, como se verá en el capítulo 9, ambas formas producen el mismo resultado al ser traducidas al modelo relacional.

Para finalizar, centremos la atención en la relación “reproduce” ya que hasta este punto ninguna de nuestras relaciones había tenido atributos. En este caso se le han asignado tres atributos que con “segundos reproducidos”, “hora de reproducción” y “fecha de reproducción”. ¿Por qué se ha hecho de esta forma? Es una pregunta natural.

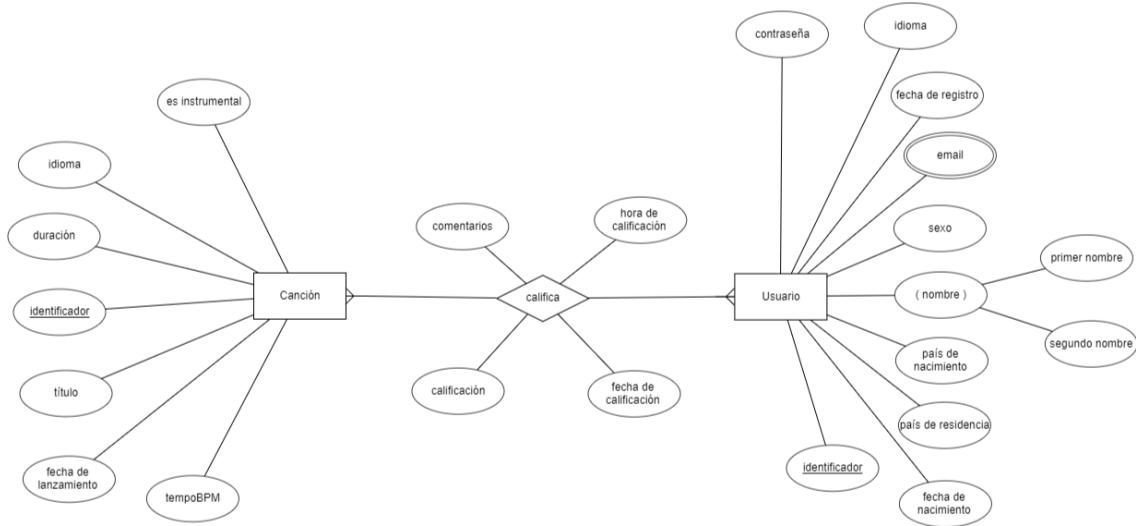
La respuesta es fácil de entender, los tres atributos mencionados solo tienen sentido cuando un usuario hace una reproducción, no para el usuario per se o la canción de manera aislada, solo cuando se configura una reproducción, puesto en otras palabras, la hora de reproducción tiene sentido cuando un usuario UI reproduce una canción C2. Miremos otro ejemplo de lo anterior.

En “Mis Canciones” también queremos que los usuarios califiquen las canciones. De las calificaciones queremos 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 como este nuevo requisito es modelado. También observamos que el atributo “comentario” es opcional, cuando se realiza una calificación de una canción se puede dejar un comentario o no.

Figura 8-9



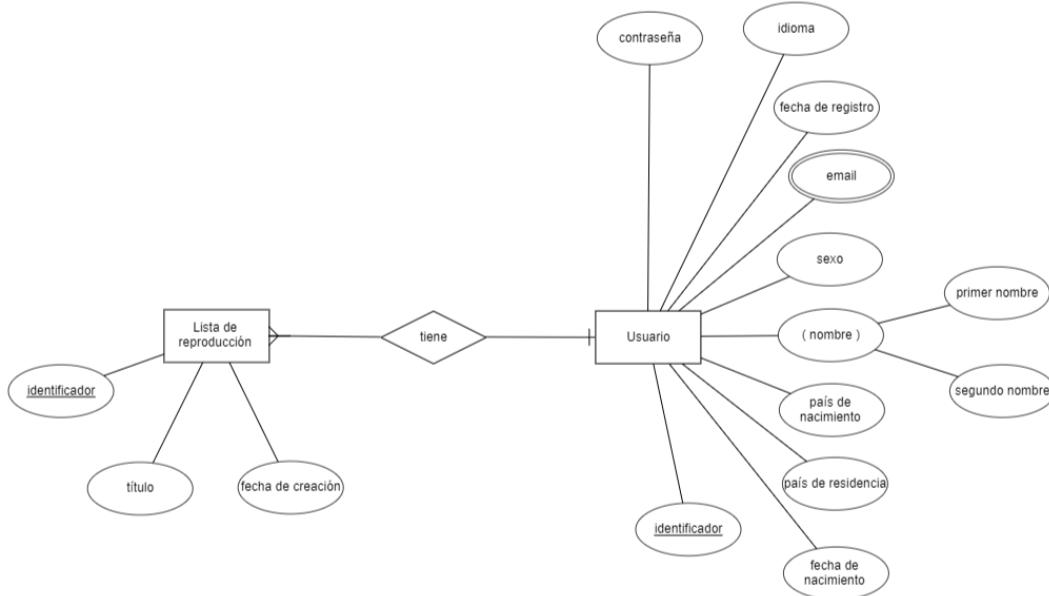
En “Mis Canciones” los usuarios pueden guardar sus listas de reproducción las cuales puede tener un nombre y una fecha de creación

Para modelar el requisito anterior tenemos 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.

De las restricciones anteriores extraemos que la lista de reproducción es una entidad con tres atributos, el identificador que nos sirve de llave principal, el título y la fecha de creación. Mientras que entre la entidad lista de reproducción y la entidad usuarios se establece de uno a muchos. La cardinalidad la hemos definido 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



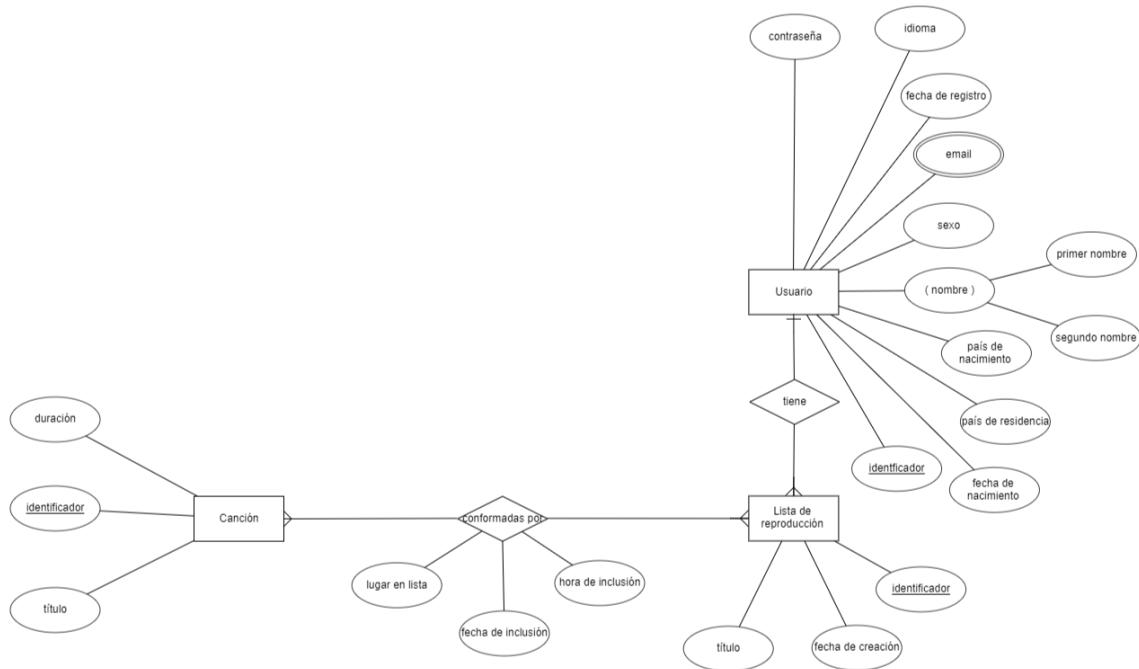
Para finalizar con esta primera versión del modelo abordemos este requisito.

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 tenemos claro que la relación que existe entre las entidades listas de reproducción y canciones 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 cual 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 tiene relevancia y se configura cuando se da la relación por lo tanto 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



La versión completa de esta primera parte del diagrama la encontramos 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 ésta última se ajusta más al dominio conceptual, es decir, quién interpreta la canción.

Tabla 8-4

Entidades	Intérprete	Canción	Género	Lista de reproducción	Álbum	Usuario
Atributos	identificador	identificador	identificador	identificador	identificador	identificador
	nombre	título	nombre	título	título	email
	tipo	duración			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

Figura 8-12

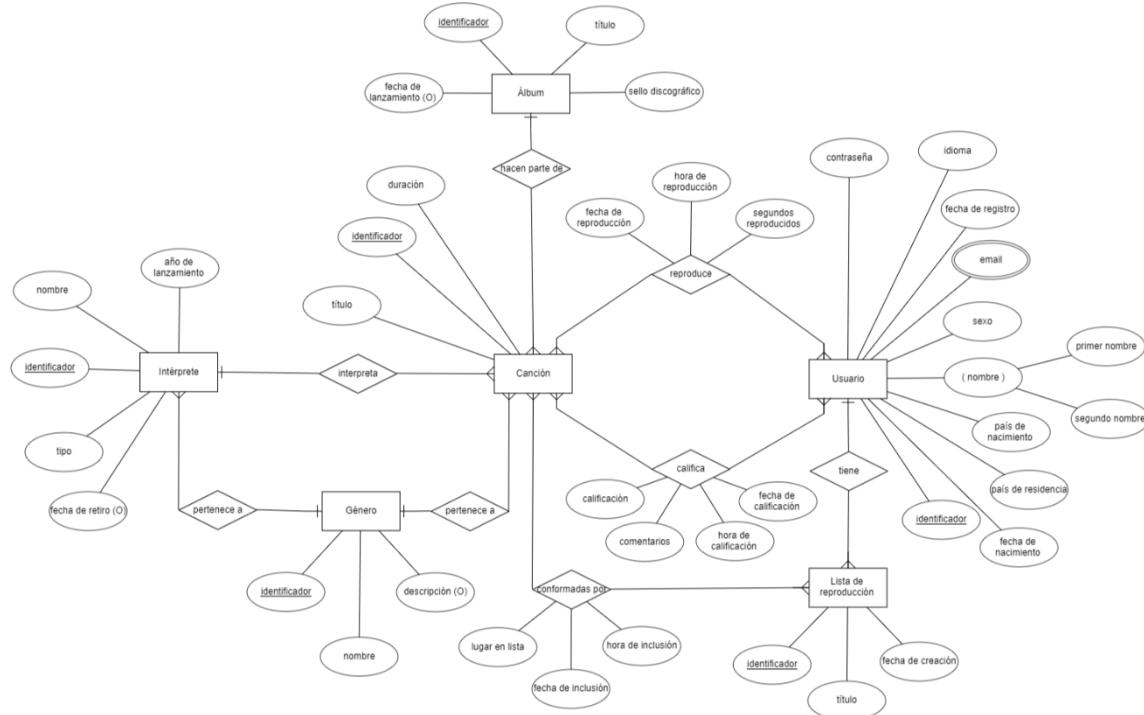


Tabla 8-5

Relación	Entidades participantes	Cardinalidad
interpreta	Intérprete – Canción	I:N
hace parte de	Canción - Álbum	I:N
pertenece	Género - Canción	I:N
pertenecer	Género – Intérprete	I:N
reproduce	Usuario – Canción	M:N
califica	Usuario – Canción	M:N
tiene	Usuario – Lista de reproducción	I:N
conformada por	Lista de reproducción - Canción	M:N

8.5 Aprendizajes más importantes del Capítulo 8

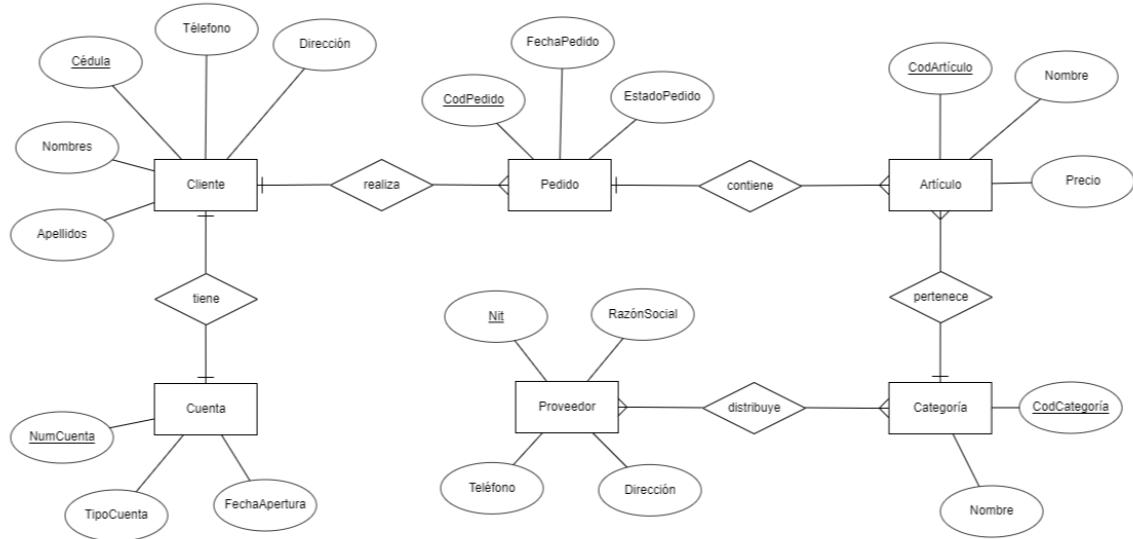
- El diagrama entidad relación permite obtener una representación visual de una base de datos en un nivel conceptual.
- Los elementos de un diagrama entidad relación son tres: las entidades, las relaciones y los atributos de cada uno de estos.

- Las entidades serán cualquier objeto sobre el cuál se está recogiendo datos, ejemplo: persona, cosa, proceso, hecho, etc. Estas pueden ser tangibles o intangibles, ejemplo. proveedor (tangible), venta (intangible).
- Las relaciones serán las acciones que se realicen o las asociaciones que existan entre las entidades, 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 si debe ser similar o indicar el mismo tipo de relación, ejemplo: gobernar → gobierna o dirige.
- Los atributos serán las características que describen a la entidad o la relación, ejemplo: códigos, nombres, fechas, direcciones y estados.
- Las relaciones tienen una cardinalidad que nos ayuda a saber la cantidad de objetos que pueden intervenir en estas.

8.6 Actividades de aplicación para evidenciar lo aprendido

- I. Describir las restricciones semánticas que implementa el diagrama ER de la Figura 8-13.

Figura 8-13



2. Cómo mejoraría la representación propuesta en la Figura 8-13. Justifique cada una de las mejoras planteadas e impleméntelas en un nuevo diagrama.

3. Teniendo en cuenta las siguientes restricciones semánticas, cree el modelo entidad-relación 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. Seleccionar un sitio web, por ejemplo, Facebook, e imaginar y diseñar como sería un modelo entidad relación que soporte algunas de sus funcionalidades.
5. Consultar la ley 1273 de 2009 e identificar 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, cumplimos con el propósito de definir cómo las diferentes entidades identificadas en los requisitos de la base de datos a construir se deben relacionar entre sí para hacer posible la satisfacción de las necesidades de datos. Sin embargo, el hecho 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 utilizaremos 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 mencionamos en el capítulo I, en el modelo relacional propuesto por Edgard Frank Codd. Este modelo está soportado en la “teoría de conjuntos” y en la “lógica de predicado”. La primera, nos permite manipular las tablas y sus elementos de forma que las podamos unir y combinar; la segunda, la lógica de predicado, nos permite 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. Es importante no confundir el concepto de relaciones del modelo entidad relación con el concepto relaciones del modelo relacional. En este último, para facilitar la compresión podemos utilizar el 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. Dentro de las principales ventajas del modelo relacional están su simplicidad (todo los objetos son tablas) y que tiene un sustento matemático que lo enviste de formalidad (teoría de conjuntos y lógica de predicado).

En la Tabla 9-1 vemos una relación de nombre “Artistas” con 5 atributos que son el “identificador”, el “nombre”, el “año de lanzamiento”, el “tipo” y el “género principal”. La relación “Artistas” contiene 5 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 utilizaremos los conceptos de tabla, fila y columna por simplicidad y por coherencia con la forma como nos hemos expresado hasta este punto.

Tabla 9-1

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 nos permite organizar todos los datos en tablas y operar sobre estas para insertar, actualizar, eliminar y consultar datos. En este capítulo, primero nos centraremos en cómo llegar a tener un modelo lógico partiendo del modelo entidad relación; luego, en cómo introducir algunas mejoras destinadas a mejorar la semántica del modelo.

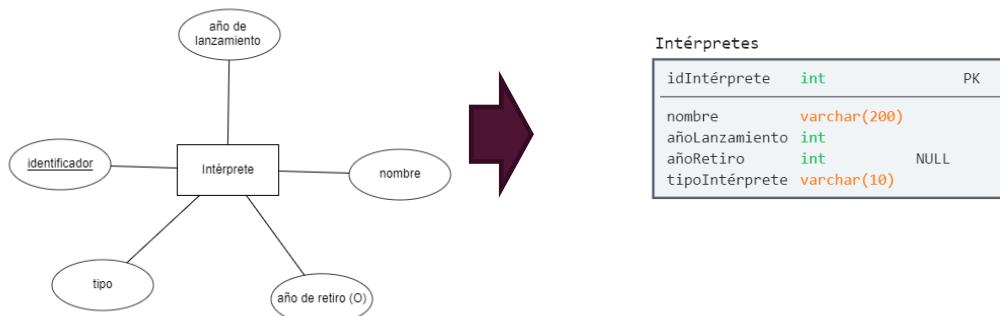
9.1 Modelo conceptual a modelo relacional

El primer pensamiento que aflora cuando surge la necesidad de transformar el modelo entidad relación al modelo relacional es si existe un procedimiento que tenga como entrada el modelo conceptual y que, al seguirlo, obtengamos como salida el modelo relacional. La respuesta a esta pregunta es sí. Existe un conjunto sencillo de reglas que nos indican las equivalencias que debemos hacer entre cada uno de los elementos del modelo entidad relación y los elementos 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 entidad relación obtenido al final del capítulo 8 y plasmado en la Figura 8-12. Lo primero que hacemos 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. La llave principal de la entidad “Intérprete” que como vemos en la figura es el atributo “identificador” será ahora la llave principal (PK, del término en inglés, Primary Key) 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 presenta en el lado izquierda la entidad que toma como entrada el procedimiento y en el lado derecho la tabla resultante.

Figura 9-1



Para nombrar las tablas hemos decidido por convención utilizar el plural del sustantivo que da nombre a la entidad. También hemos decidido utilizar 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”, la primera letra de todas las palabras se escribe en mayúscula. En ninguna de las dos se deja espacio entre las palabras.

Otro elemento importante al momento de realizar la transformación del modelo entidad relación 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, el “añoLanzamiento” y el “añoRetiro” deben ser números. Esta restricción en el dominio de valores de cada columna la podemos representar con el tipo de datos, en el caso del “nombre” y “tipointérprete” hemos indicado que debe ser una cadena de texto (VARCHAR) y para el “idIntérprete”, “añoLanzamiento” y “añoRetiro” un número entero (INT).

En la Figura 9-1 también se observa que los atributos opcionales de la entidad, como el “año de retiro”, son marcados con la palabra NULL en la nueva tabla; esta palabra nos indica que para estas columnas no es necesario proporcionar un valor al momento de la inserción de una nueva fila o que podemos utilizar el descriptor NULL al momento de operar con esta columna.

La descripción que acabamos de hacer por cada columna es importante tenerla presente al momento de entender en su total extensión un modelo lógico, por lo que es clave tener la buena práctica de ir creando un diccionario de datos paralelamente al modelo relacional que haga las veces de metadatos sobre nuestros datos o puesto en otros términos “datos sobre nuestros datos”, orientados a mejorar no solo nuestra comprensión, sino la 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 6 entidades que tenemos en nuestro modelo entidad relación, por lo que, al realizar este paso, obtendremos la versión del modelo presentada en la Figura 9-2.

Tabla 9-2

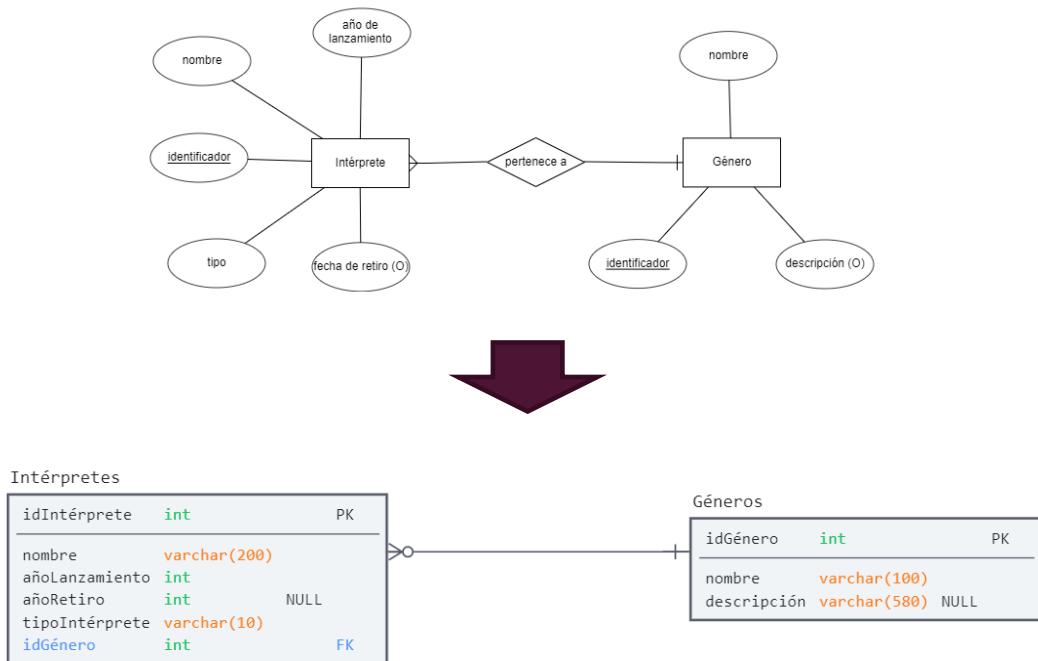
Nombre	PK/FK	Acepta NULL	Descripción	Dominio
idIntérprete	PK	No	Un número independiente de dominio que representa el identificador de cada intérprete.	Un número entero positivo.
nombre		No	Representa el nombre artístico de cada intérprete	Una cadena de texto.
añoLanzamiento		No	Un número entero de cuatro 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 artista. 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

Álbumes	Canciones	Usuarios																																																																		
<table border="1"> <thead> <tr> <th>idÁlbum</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>título</td><td>varchar(100)</td><td></td></tr> <tr> <td>selloDiscográfico</td><td>varchar(100)</td><td></td></tr> <tr> <td>fechaLanzamiento</td><td>date</td><td></td></tr> </tbody> </table>	idÁlbum	int	PK	título	varchar(100)		selloDiscográfico	varchar(100)		fechaLanzamiento	date		<table border="1"> <thead> <tr> <th>idCanción</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>título</td><td>varchar(100)</td><td></td></tr> <tr> <td>duración</td><td>time</td><td></td></tr> <tr> <td>tempoBPM</td><td>int</td><td></td></tr> <tr> <td>idioma</td><td>varchar(30)</td><td></td></tr> <tr> <td>esInstrumental</td><td>bit</td><td></td></tr> <tr> <td>fechaLanzamiento</td><td>date</td><td></td></tr> </tbody> </table>	idCanción	int	PK	título	varchar(100)		duración	time		tempoBPM	int		idioma	varchar(30)		esInstrumental	bit		fechaLanzamiento	date		<table border="1"> <thead> <tr> <th>idUser</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>email</td><td>varchar(254)</td><td></td></tr> <tr> <td>contraseña</td><td>varchar(128)</td><td></td></tr> <tr> <td>nombres</td><td>varchar(100)</td><td></td></tr> <tr> <td>apellidos</td><td>varchar(100)</td><td></td></tr> <tr> <td>sexo</td><td>varchar(30)</td><td></td></tr> <tr> <td>fechaNacimiento</td><td>date</td><td></td></tr> <tr> <td>paísNacimiento</td><td>varchar(100)</td><td></td></tr> <tr> <td>paísResidencia</td><td>varchar(100)</td><td></td></tr> <tr> <td>idioma</td><td>varchar(30)</td><td></td></tr> <tr> <td>fechaRegistro</td><td>date</td><td></td></tr> </tbody> </table>	idUser	int	PK	email	varchar(254)		contraseña	varchar(128)		nombres	varchar(100)		apellidos	varchar(100)		sexo	varchar(30)		fechaNacimiento	date		paísNacimiento	varchar(100)		paísResidencia	varchar(100)		idioma	varchar(30)		fechaRegistro	date	
idÁlbum	int	PK																																																																		
título	varchar(100)																																																																			
selloDiscográfico	varchar(100)																																																																			
fechaLanzamiento	date																																																																			
idCanción	int	PK																																																																		
título	varchar(100)																																																																			
duración	time																																																																			
tempoBPM	int																																																																			
idioma	varchar(30)																																																																			
esInstrumental	bit																																																																			
fechaLanzamiento	date																																																																			
idUser	int	PK																																																																		
email	varchar(254)																																																																			
contraseña	varchar(128)																																																																			
nombres	varchar(100)																																																																			
apellidos	varchar(100)																																																																			
sexo	varchar(30)																																																																			
fechaNacimiento	date																																																																			
paísNacimiento	varchar(100)																																																																			
paísResidencia	varchar(100)																																																																			
idioma	varchar(30)																																																																			
fechaRegistro	date																																																																			
Intérpretes	ListasReproducción	Géneros																																																																		
<table border="1"> <thead> <tr> <th>idIntérprete</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>nombre</td><td>varchar(200)</td><td></td></tr> <tr> <td>añoLanzamiento</td><td>int</td><td></td></tr> <tr> <td>añoRetiro</td><td>int</td><td>NULL</td></tr> <tr> <td>tipoIntérprete</td><td>varchar(10)</td><td></td></tr> </tbody> </table>	idIntérprete	int	PK	nombre	varchar(200)		añoLanzamiento	int		añoRetiro	int	NULL	tipoIntérprete	varchar(10)		<table border="1"> <thead> <tr> <th>idListaReproducción</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>título</td><td>varchar(50)</td><td></td></tr> <tr> <td>fechaCreación</td><td>date</td><td></td></tr> </tbody> </table>	idListaReproducción	int	PK	título	varchar(50)		fechaCreación	date		<table border="1"> <thead> <tr> <th>idGénero</th><th>int</th><th>PK</th></tr> </thead> <tbody> <tr> <td>nombre</td><td>varchar(100)</td><td></td></tr> <tr> <td>descripción</td><td>varchar(580)</td><td>NULL</td></tr> </tbody> </table>	idGénero	int	PK	nombre	varchar(100)		descripción	varchar(580)	NULL																																	
idIntérprete	int	PK																																																																		
nombre	varchar(200)																																																																			
añoLanzamiento	int																																																																			
añoRetiro	int	NULL																																																																		
tipoIntérprete	varchar(10)																																																																			
idListaReproducción	int	PK																																																																		
título	varchar(50)																																																																			
fechaCreación	date																																																																			
idGénero	int	PK																																																																		
nombre	varchar(100)																																																																			
descripción	varchar(580)	NULL																																																																		

Una vez hemos creado todas las tablas que tienen como equivalente una entidad en el modelo entidad relación surge la pregunta: ¿qué se debe hacer ahora con las relaciones? La respuesta como vimos en el listado de las reglas es: depende del tipo de relación. Para las relaciones uno a muchos como la que existe entre la entidad “Género” y la entidad “Intérprete” se procede de la siguiente forma: 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 entidad con 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.

Figura 9-3

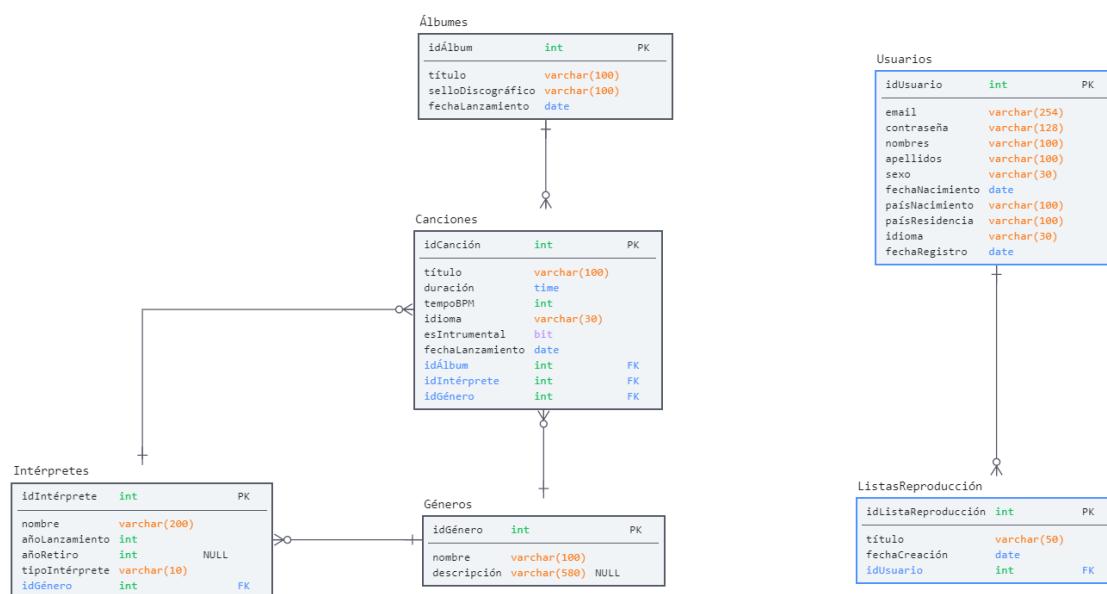


El paso recién descrito lo repetimos por cada una de las relaciones de uno a muchos que tenemos en nuestro diagrama relacional, los que nos permitirá obtener una nueva versión del modelo lógico, la cual es presentada en la Figura 9-4. Sobre esta nueva versión se debe resaltar como la tabla “Canciones” ahora tiene 3 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, si no es NULL debe existir previamente en la columna “idÁlbum” de la tabla “Álbumes”.

Figura 9-4



Siguiendo nuestra recomendación, agregamos la descripción de la tabla “Canciones” al diccionario de datos como se observa en la Tabla 9-3.

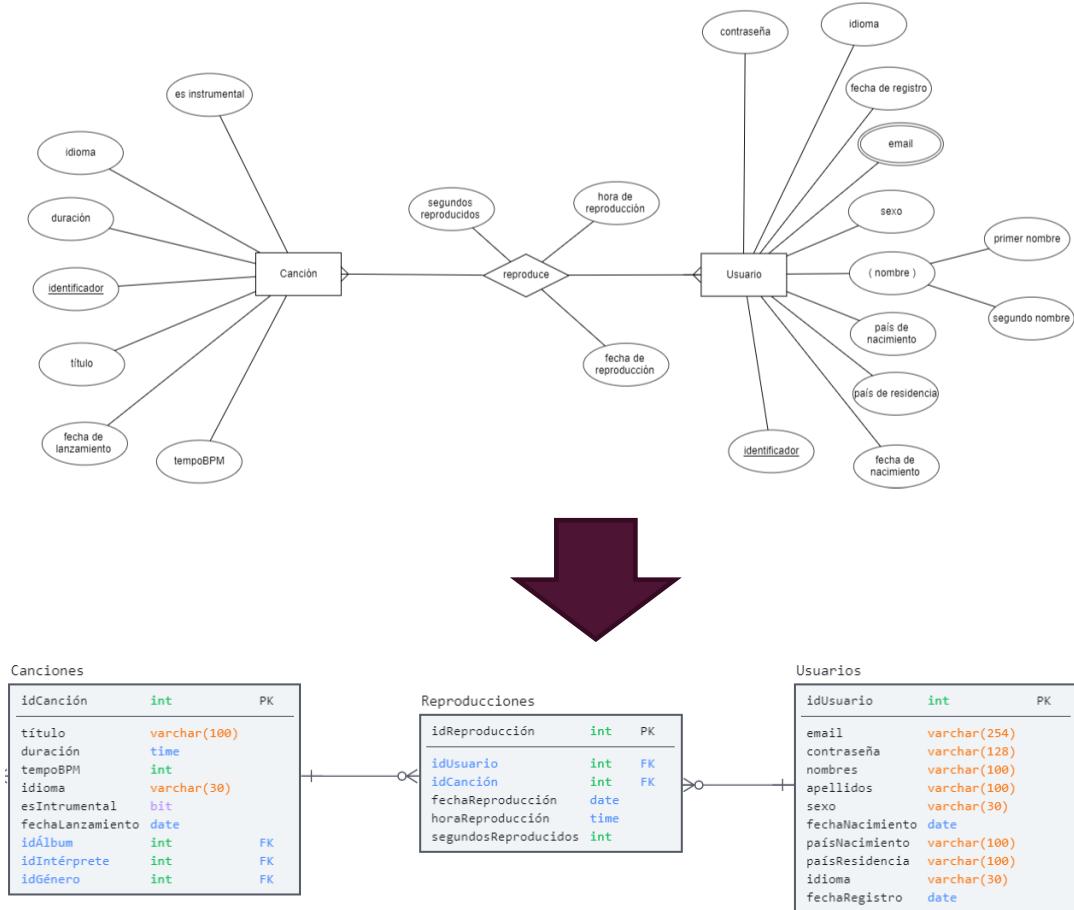
Tabla 9-3

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 e inglés.	Una cadena de texto
esInstrumental		No	Un carácter que indica si la canción es instrumental. El “1” indica que es instrumental, el “0” 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 y esta nueva tabla tendrá como llaves foráneas la llave principal de las entidades que intervienen en la relación, además de todos los atributos que la entidad posee por se. Tomemos como ejemplo la relación “reproduce” que hay entre la entidad “Usuario” y la entidad “Canción”, la cual se entiende que los usuarios reproducen canciones, por lo que la nueva tabla creada tendrá el nombre “Reproducciones”. Esta tabla 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 tenemos que tomar una decisión, ¿cuál es la llave principal de la nueva tabla? Podemos decidir que es la combinación de “idCanción” e “idUsuario”, a saber, una llave principal compuesta o superclave. Pero, en este caso tendríamos que no se puede repetir una misma 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 optar por incluir otra columna como la fecha y la hora a la llave principal, pero hemos optado por incluir una nueva columna de nombre “idReproducción” que permita identificar de forma inequívoca cada relación y a su

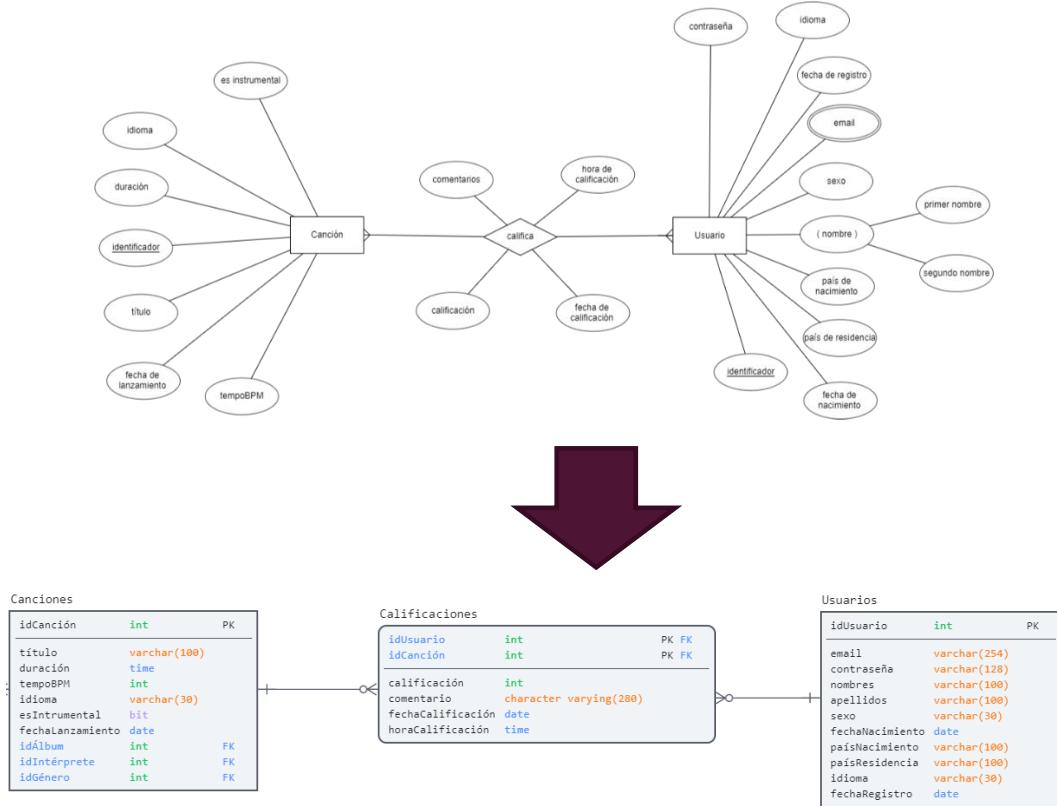
vez, tener más de una reproducción de la misma canción por el mismo usuario. En la Figura 9-5 se muestra el resultado de esta transformación.

Figura 9-5



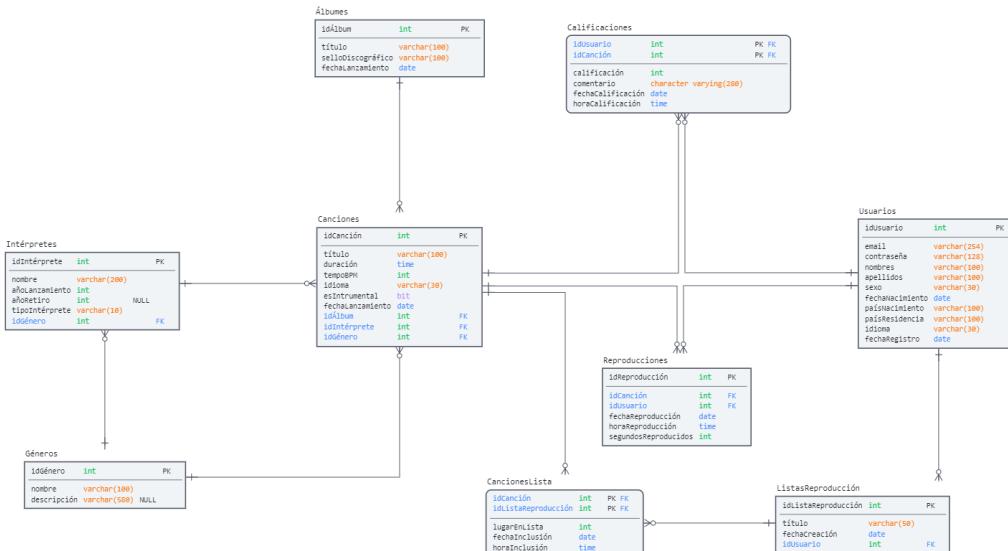
No siempre es necesario añadir una nueva columna para definir la llave principal de una tabla producto de una relación de muchos a muchos, miremos el caso de la relación “califica” entre “Usuario” y “Canción”. En este caso hemos definido una nueva tabla “Calificaciones” que contiene como llaves foráneas el identificador de la canción y el identificador de los usuarios. En este caso la combinación de ambas llaves foráneas es la llave principal de la tabla, es decir, una llave compuesta. En este contexto nos es útil porque nosotros queremos 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, estos son, “fechaReproducción”, “horaReproducción” y “segundosReproducidos”. El resultado de la transformación se aprecia en la Figura 9-6.

Figura 9-6



Luego de seguir cada uno de los pasos del procedimiento explicado se obtiene el modelo relacional equivalente al modelo entidad relación presentado en la Figura 8-12 . El modelo resultante es el mostrado en la Figura 9-7.

Figura 9-7



9.2 Evolución del modelo relacional

En este punto tenemos una versión del modelo relacional que representa la base de datos que queremos 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 que el cambio no debe ser visto con miedo sino como una señal que nuestro modelo debe evolucionar para adaptarse a lo que el entorno organizativo está exigiendo. Miremos ahora el siguiente requisito.

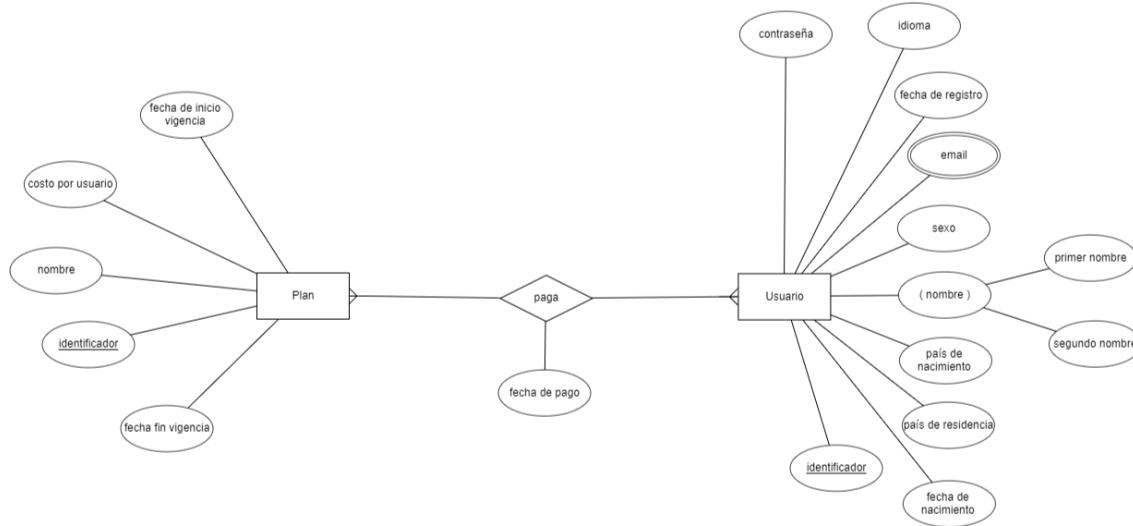
En la plataforma “Mis Canciones” existen unos planes a los cuales el usuario puede suscribirse, por tanto, se desea llevar el registro de los pagos que realiza el cliente de cada plan.

Analicemos el requisito planteado y extraigamos las reglas semánticas que guiarán nuestro 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 el pago y el monto pagado.

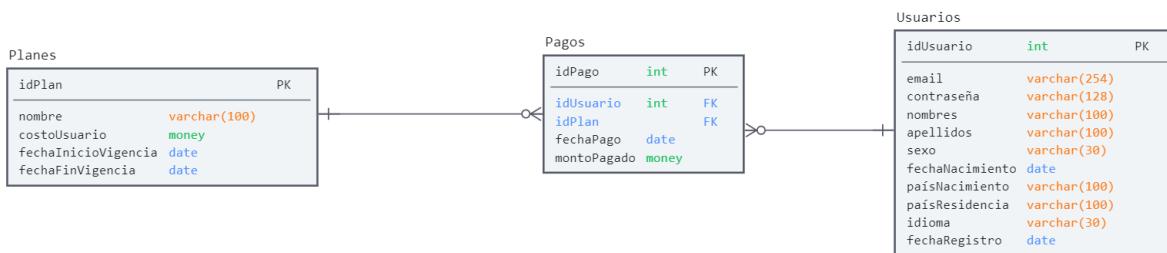
Una alternativa para modelar lo que acabamos de describir es plantear una relación de muchos a muchos entre una entidad “Plan” y la entidad “Usuario”. El nombre de esta relación sería “paga” en donde un usuario realiza uno o muchos pagos de un plan y un plan es pagado por uno o muchos usuarios. En la Figura 9-8 se observa lo descrito, en la relación “paga” hemos asignado el atributo “fecha de pago” debido a que este tiene únicamente sentido cuando se realiza un pago.

Figura 9-8



La trasformación de lo modelado en la Figura 9-8 al modelo relacional plasmado en la Figura 9-9 lo hacemos 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 la entidad “Usuario” y la entidad “Plan”, se convierte en una nueva tabla donde las llaves principales de estas dos entidades se convierten en llaves foráneas. Se define una nueva columna como llave principal, el cual será un número entero que identifique cada pago. En la nueva tabla “Pagos” también hemos añadido la columna “fechaPago” puesto que el atributo “fecha de pago” pertenece a la relación “paga”. Finalmente, también hemos añadido la columna “montoPagado” a la nueva tabla “Pagos” debido a que el valor de la columna “costoUsuario” puede cambiar y si cambia no podríamos saber cuánto pago un usuario por su plan, salvo que el pago haya sido posterior al último cambio del costo por usuario del plan.

Figura 9-9



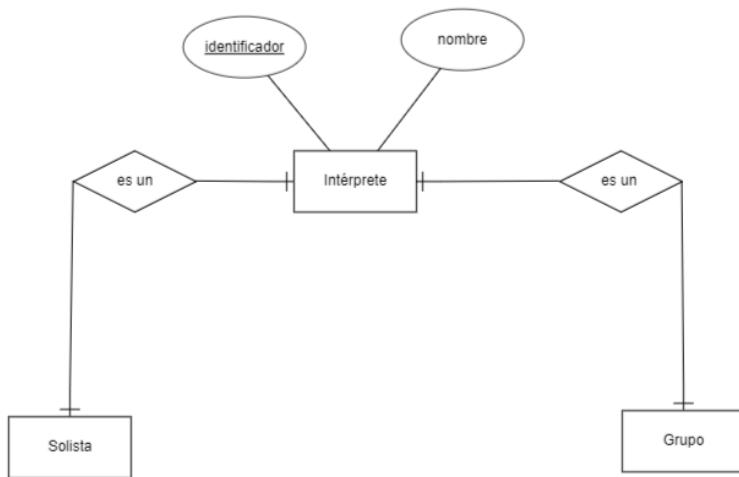
Analicemos ahora los requisitos planteados a continuación.

En la plataforma “Mis Canciones” queremos 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 periodo.

Los intérpretes de las canciones los tenemos de dos tipos “solistas” o “grupos”, hasta ahora los hemos manejado en una misma entidad debido a que todos los atributos que almacenamos son comunes tanto para “solistas” como para los “grupos”, pero con el requisito antes descrito se demanda almacenar para los grupos, 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 este concepto 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 nos permite que a partir de aquellas entidades que incluyen la ocurrencia de uno o más subgrupos, por ejemplo, de la entidad “Intérprete” incluye la ocurrencia del subgrupo “Solista” y el subgrupo “Grupo”, se puedan desprender tantas entidades como subgrupos haya. Siguiendo con el caso de los intérpretes, tenemos que de la entidad “Interprete” se desprenden dos nuevas entidades, a saber, “Solista” y “Grupo”. Estas nuevas entidades son también del tipo de la entidad de la que se desprendieron, puesto en otros términos, un “Solista” es un “Intérprete”. Las razones de hacer lo anterior son básicamente que: cada uno de estos subgrupos como nos plantea el requisito recién presentado puede tener atributos que lo distinguen de otros subgrupos; agrega información semántica al diseño. En la Figura 9-10 se observa cómo se puede plantear esta relación.

Figura 9-10



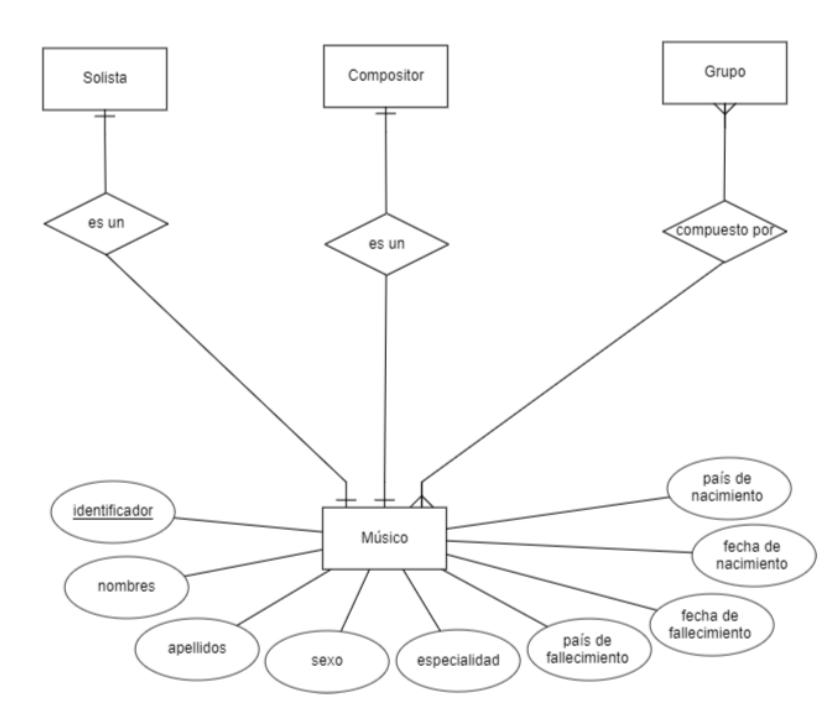
Las entidades entonces pueden tener distintas subclases (entidades hijas) y las subclases pueden tener distintas superclases (entidades padres). En la Figura 9-10 la entidad “Intérprete” es la superclase y las entidades “Solista” y “Grupo” son las subclases. Como mencionamos, cada miembro de una subclase también es miembro de la superclase. En otras palabras, un miembro en la subclase es también un miembro en 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 relaciones disjuntas. Pausemos por un instante y analicemos el siguiente requisito.

En la plataforma “Mis Canciones” también se desea almacenar los compositores de las canciones y mantener datos personales como el nombre real y fecha de nacimiento tanto de compositores, solistas y miembros de cada grupo

La descripción que se presenta, aunque se plantea como un requisito al analizarlo con detenimiento, podemos inferir que hay mínimamente dos contenidos en ella: el primero centrado en que se desea saber quiénes son los compositores de las canciones y el segundo en que se desea tener datos personales de solistas, compositores y miembro de grupo. Centrémonos inicialmente en el segundo aspecto, para darle solución podemos pensar en plantear tres entidades: una entidad “Solista”, una entidad “Compositor” y una entidad “Miembros de grupo” y en cada una de estas incluir los atributos personales que deseamos almacenar, es decir, atributos como

“nombres”, “apellidos”, “sexo”, etc. Para cada entidad, este enfoque, aunque intuitivo y útil, tiene algunos inconvenientes, por ejemplo, ¿qué sucede si una misma persona ha sido compositor, miembro de grupo y solista? Tendríamos que repetir datos y ya sabemos que repetir datos en las bases de datos relacionales con propósito transaccional son 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 si un grupo de entidades ubicadas en el mismo nivel conceptual tienen atributos comunes se puede realizar una abstracción que consolide todos los atributos comunes y se convierta en superclase o clase padre de las otras clases. Aplicado a nuestro requisito, nosotros podemos afirmar que tanto compositores, como solistas y los miembros de los grupos musicales son músicos y asignar a esta nueva entidad “Músico” los nuevos atributos. Lo descrito está representado en la Figura 9-11.

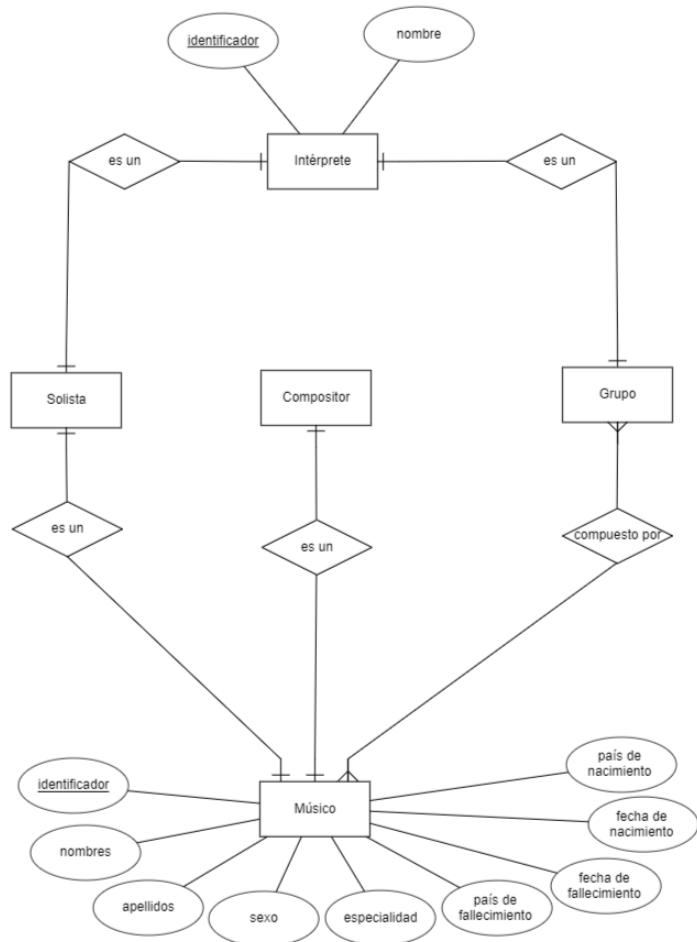
Figura 9-11



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 nos 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 hemos unificado la especialización que se realizó de los intérpretes

y la generalización que se hizo de compositores, solistas y miembros de grupos musicales.

Figura 9-12

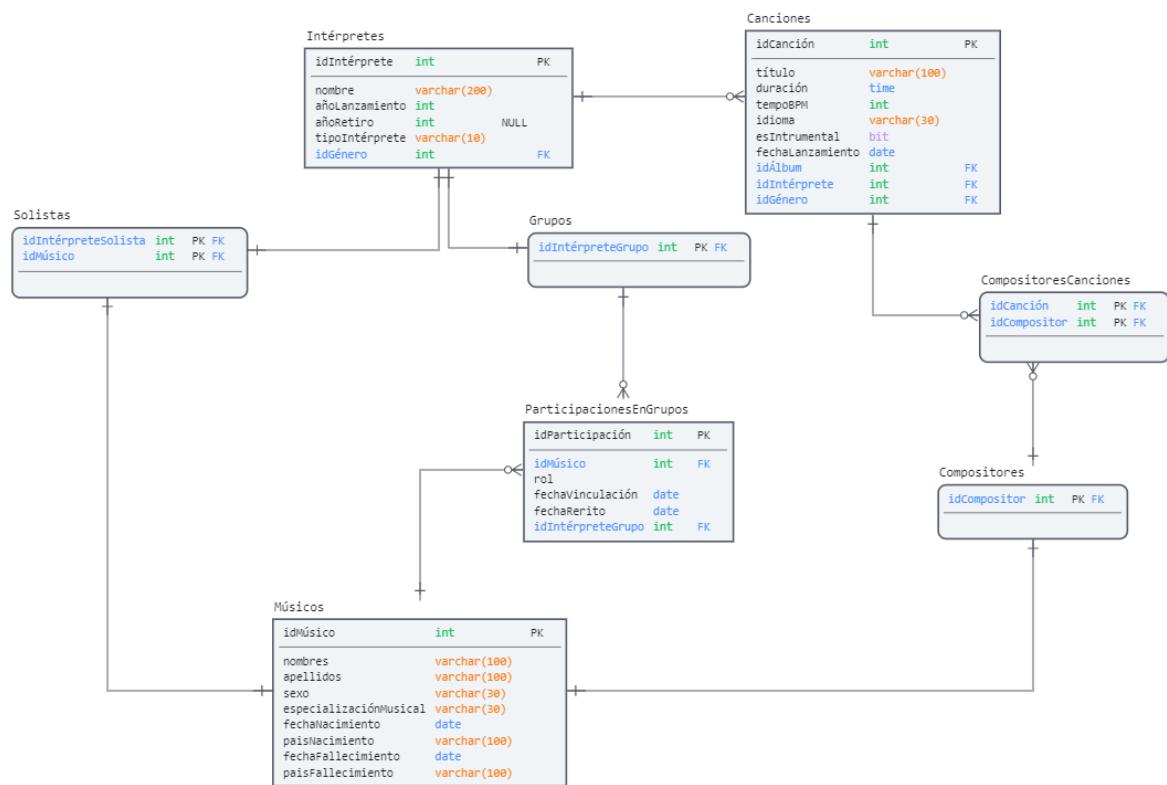


En la Figura 9-12 se presenta como los solistas tienen una relación de uno a uno con intérpretes, es decir, un solista es un intérprete y otra relación de uno a uno con “Músico”, un solista es un músico, por lo que un solista tendrá acceso tanto a los atributos de la entidad “Intérprete” como a los atributos de la entidad “Músico”. En el caso de los miembros de los grupos, lo modelamos teniendo presente que un músico puede pertenecer a muchos grupos durante su vida musical y 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, fecha de retiro y 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 el músico y el grupo.

No olvidemos que uno de los requisitos es saber cuáles son las canciones que ha creado cada compositor. Como ya tenemos una entidad “Compositor” especializada y lo que queremos modelar es que un compositor puede crear varias canciones y que una canción puede tener varios compositores, lo podemos representar como una relación de muchos a muchos entre las entidades “Canción” y “Compositor”.

Lo traducción de los nuevos requisitos modelados al modelo relacional los tenemos en la Figura 9-13. Para hacer esto posible hemos 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 vemos en la tabla “Solistas”; en el caso de “Grupo” y “Músico” como existe una relación de muchos a muchos hemos creado una tabla “ParticipacionesEnGrupos” donde se almacenan los atributos de la relación. En el mismo sentido ocurre con la relación entre la entidad “Compositor” y “Canción” que al ser de muchos a muchos provoca la creación de una nueva tabla que hemos nombrado “CompositoresCanciones” que tiene como llave principal la combinación de las llaves principales de la tabla “Canciones” y de la tabla “Compositores”.

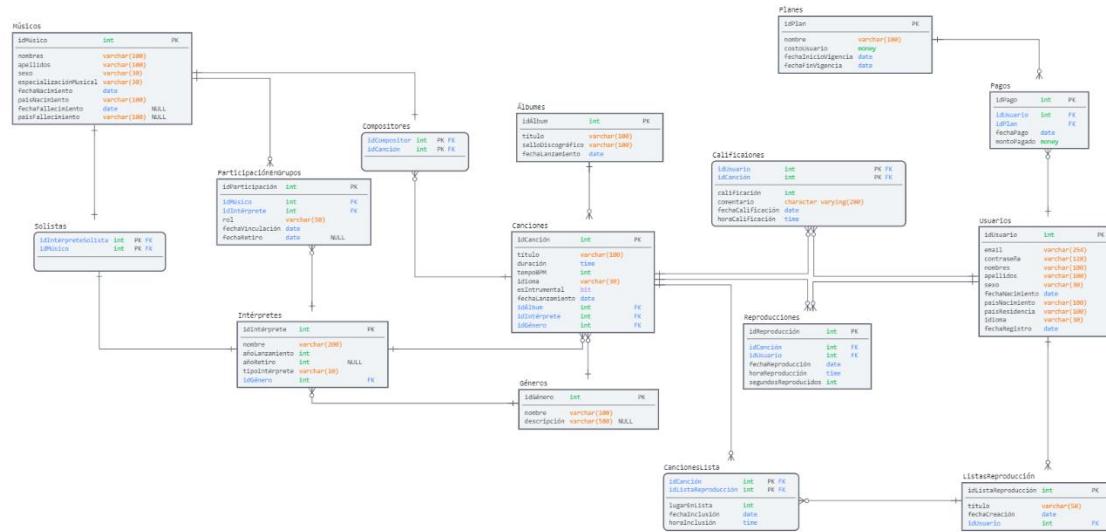
Figura 9-13



Para concluir, en la Figura 9-14 presentamos la versión que da respuesta a todos los requisitos que hemos presentado hasta este punto. En esta versión “final” también se han realizado unas mejoras, recordemos que el diseño y construcción de bases de datos es un proceso de mejora continua; los cambios que hemos incorporado son:

- Renombramos algunas columnas para aumentar el valor semántico y la legibilidad del modelo, por ejemplo, en la tabla “Solistas” hemos renombrado la llave foránea que proviene de la tabla “Intérpretes” de “idIntérprete” a “idIntérpreteSolista”.
- Hemos definido que algunas columnas acepten valores nulos como es el caso de las columnas “fechaFallecimiento” y “paísFallecimiento” de la tabla “Músicos”.
- Hemos removido 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



9.3 Aprendizajes más importantes del Capítulo 9

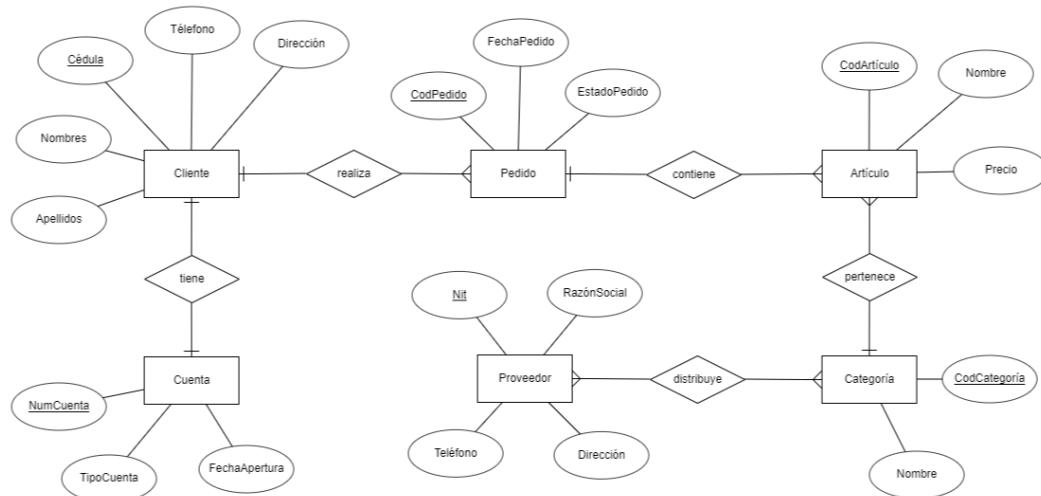
- Existe un conjunto claro de reglas que nos permite convertir fácilmente un modelo entidad relación al modelo relacional.
- A nivel lógico, es el modelo relacional en el que están soportadas las bases de datos relacionales. Este modelo trae consigo ventajas como la simplicidad y la formalidad.

- Es importante, además del diagrama del modelo relacional, ir creando un diccionario de datos que sirva de metadatos para nuestros datos, puesto en otros términos, es importante ir creando artefactos que faciliten la comprensión del modelo creado.
- Cuando seleccionamos el enfoque de especialización, intentamos resaltar las diferencias entre entidades definiendo una o más subclases de una entidad superclase. Si seleccionamos el enfoque de generalización, intentamos identificar características comunes entre entidades para definir una entidad generalizadora.

9.4 Actividades de aplicación para evidenciar lo aprendido

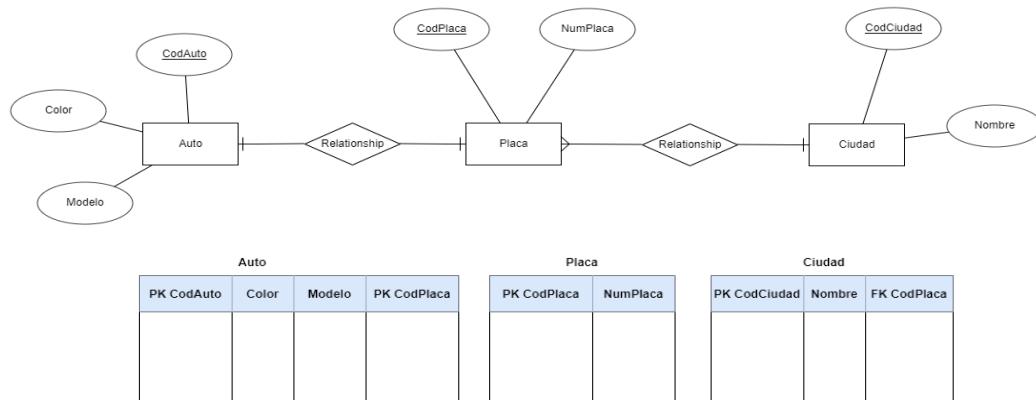
1. Aplicar el procedimiento para convertir un modelo entidad relación en relacional al modelo presentado en la Figura 9-15

Figura 9-15



2. Hemos argumentado que se han removido las tablas “Grupos” y “Compositores” del diagrama de la Figura 9-13 porque no hemos añadido ningún atributo que justifique su inclusión. Proponga un atributo que haga necesaria la inclusión de estas dos tablas.
3. Indique si la conversión realizada en la Figura 9-16 es correcta o no. En caso de que no lo sea, plantee la solución correcta.

Figura 9-16



4. Añadir una generalización o especialización al modelo entidad relación definido en el punto 3 del capítulo 8. Luego realice la conversión del modelo entidad relación resultante al modelo relacional.
5. Defina qué son los datos públicos, los datos privados, los datos semiprivados y 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 asegurar que los esquemas relacionales diseñados no tienen elementos que pueden generar un bajo rendimiento en el acceso a los mismos. 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, relaciones entre los datos y restricciones sobre los datos, 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 el producto que se está construyendo se ajusta a unos estándares. Una de las técnicas que se utilizan en el diseño de las bases de datos es la normalización.

La normalización se fundamenta en la verificación de la dependencia entre los atributos de las tablas, la cual se denomina dependencia funcional. Con la normalización 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. Con esto se espera que la base de datos tenga 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

En la Tabla 10-1 se representa un conjunto de datos compuesto por 7 columnas que almacenan el identificador, el título, la duración, el género y el intérprete un grupo de canciones. De los intérpretes también se tiene el tipo y el año de lanzamiento a la vida artística de cada uno de los intérpretes. Por ejemplo, la fila número 4 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”.

Tabla 10-1

Canciones						
identificador	título	duración	género	nombre	tipo de intérprete	año de lanzamiento
10001	La tierra del olvido	4:25	Vallenato	Carlos vives	Solistा	1986
10002	Ojos así	3:57	Pop	Shakira	Solistा	1990
10003	Mi gente	3:05	Urbano Latino	J. Balvin	Solistा	2006
10004	Ambiente	4:08	Urbano Latino	J. Balvin	Solistा	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	Solistা	1986
10011	Una aventura	5:16	Salsa	Niche	Grupo	1979
10012	Ginza	4:39	Urbano Latino	J. Balvin	Solistা	2006
10013	Octavo día	4:32	Pop	Shakira	Solistা	1990
10014	Quiero verte	3:18	Pop	Carlos vives	Solistা	1986

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 canciones? En este primer escenario se estaría 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.

¿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 son tres. Si alguna fila no se modifica, se tendrán valores diferentes para un mismo hecho. ¡Inconsistencia!

¿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! Pero, 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 de normalización nos permite evaluar una tabla respecto de una serie de reglas para comprobar si estamos 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 cumplan las formas normales.

El concepto 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*”. En este se presentó el concepto de tablas normalizadas para referirse a aquellas que no tienen grupos repetidos. 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 Tabla 10-1, el valor “*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, mostraremos cómo la normalización puede utilizarse como una técnica de validación para comprobar la estructura de las tablas. Para esto, presentaremos 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 primera forma normal se partirá de un análisis de la Tabla 10-2 para identificar si hay algún problema redundancia o pérdida de integridad. La tabla presentada tiene por objetivo registrar las calificaciones que han realizado los usuarios a las diferentes canciones, para lo cual tiene asociada 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 nombre del intérprete (intérprete) de la canción, la calificación (“calificación”) que el usuario le otorga a la canción, el nombre y 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. Lo anterior, impone la restricción que un usuario puede calificar diferentes canciones, pero no la misma canción dos veces.

Tabla 10-2

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@mai.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@mai.com
14	8	Ginza	7	J. Balvin	1	Piedad Bonett	bonett@gmail.com piedad@gmail.com

En la Tabla 10-2 empezaremos por verificar si cumple con la IFN, la que plantea que una tabla está en primera forma normal si el dominio de todas las columnas es “atómico”, es decir, son unidades indivisibles. Recordemos que cuando estábamos modelando la entidad “Usuario” definimos que el atributo “correos” podía aceptar múltiples valores, o lo que es lo mismo, que un usuario puede tener varios correos electrónicos. También definimos 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 imcuplir la IFN.

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 se observa que la columna *email*, permite almacenar diferentes direcciones de correo electrónico en la misma columna. Continuando con el mismo usuario “*Marie Curie*” vemos que tiene registrado dos correos electrónicos que son “*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. Supongamos que deseamos enviar un correo electrónico a todos los usuarios con la siguiente estructura “Señor (a) <primer apellido>”, para hacer esto posible tendríamos que extraer el apellido de cada usuario, lo que implicaría un procesamiento del texto y puede desencadenar en inconvenientes si consideramos que hay apellidos tanto compuestos como simples, mientras que si tenemos almacenado el valor del primer nombre y el 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. Mientras que, 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, para el caso de “*usuario*”, una columna para el primer nombre y otra columna para el 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 quien pertenece cada uno de los valores de la columna de múltiples valores y una columna que permite 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 las columnas correo electrónico e 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 hemos cumplimos con la primera forma normal, es decir, cada intersección fila-columna contiene un valor único del dominio aplicable.

Tabla 10-3

Calificaciones							
idcanción	idUser	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

Correos	
idUser	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 evaluemos si el conjunto de tablas resultantes se encuentra en 2FN. La regla de la segunda forma normal plantea que para estar en segunda forma normal 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 el concepto de dependencia funcional, por lo que explicaremos brevemente en que consiste la dependencia funcional entre columnas de una tabla.

Para dos columnas A y B de una tabla Z, decimos 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 podemos obtener el valor de B.

Miremos lo explicado con un ejemplo de la Tabla 10-3 “Calificaciones”. En esta tabla podemos afirmar que el valor de la columna “idCanción” determina funcionalmente el valor de la columna “título”, el idCanción número “14” siempre tendrá el valor de “Ginza” en el título, por tanto “idCanción” \rightarrow “título”. También podemos apreciar que el valor de la columna “idUser” determina funcionalmente el valor de las columnas “primerNombre” y “primerApellido”; Finalmente, también observamos que el “idIntérprete” determina funcionalmente el valor de la columna “intérprete”.

Es importante tener claro que la dependencia funcional se debe evaluar teniendo presente 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” podríamos 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” el cual siempre tendrá por primer apellido “Curie” y por identificador del usuario “6”, el sentido de la columna nos indica que puede haber más de un usuario con el nombre “Marie”, por lo que no podemos 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 evaluemos nuestra tabla “Calificaciones” (idCanción, idUsuario, título, idIntérprete, intérprete, calificación, primerNombre, segundoNombre) contra la 2FN. Esta tabla como podemos observar tiene una llave principal compuesta, las partes de la llave son el identificador de la canción y el identificador del usuario. Recordemos que la 2FN nos 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. Verifiquemos 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, a saber, solo con tener el valor de un identificador de la canción se puede saber cuál es su título y su intérprete. Por lo que no cumple la segunda forma normal. La columna calificación si 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 tiene una dependencia funcional del identificador del usuario, es decir, solo de una parte de la llave, por lo que no cumple la segunda forma normal.

Para solucionar este inconveniente lo que debemos hacer es 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 las determina funcionalmente. Es decir, crearemos dos nuevas tablas, una tabla “Canciones” que contendrá el identificador de la canción junto con el título, el identificador del intérprete y el nombre del intérprete; otra tabla “Usuarios” que contendrá 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

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

Usuarios		
idUser	PrimerNombre	SegundoNombre
6	Marie	Curie
8	Piedad	Bonett
5	Baruch	Spinoza

Tabla 10-7

Correos	
idUser	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

Calificaciones		
idcanción	idUser	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” tiene 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”. La tabla “Correos” se mantiene igual. Todas estas tablas están ahora tanto en primera forma normal como en segunda.

10.4 Tercera forma normal

La normalización que hemos realizado hasta este punto para dejar las tablas en 1FN y 2FN nos 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. Esto lo conseguimos poniendo en una nueva tabla los atributos parcialmente dependientes de la tabla, junto con una copia de su determinante. Aunque este procedimiento nos dejó con tablas que tienen menos redundancia que las de 1FN, aún pueden sufrir efectos secundarios al momento de realizar las tareas de actualización. Por ejemplo, si queremos actualizar el nombre de un intérprete, como “J. Balvin” (idIntérprete 7), tendremos que actualizar tres filas en la tabla “Canciones”.

Es en este contexto en donde es útil la tercera forma normal. Esta nos indica que no deben existir dependencias transitivas de la llave principal, o puesto en términos más claros, ningún atributo que no sea parte de la llave principal puede depender funcionalmente de otro atributo que tampoco es miembro de la llave principal.

Analicemos las dependencias funcionales de la tabla “Canciones”. El título de la canción, el identificador del intérprete y el nombre del intérprete (transitivamente) dependen funcionalmente de identificador de la canción (la llave principal). Sin embargo, podemos observar que 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 hablamos de dependencia transitiva, porque $A \rightarrow B$ y $B \rightarrow C$, por tanto, $A \rightarrow C$. puesto en otras palabras quien determina realmente en un primer grado el nombre del intérprete es el identificador del intérprete no el de la canción, pero como el identificador del intérprete depende funcionalmente del identificador de la canción, entonces el identificador de la canción determina transitivamente el nombre del intérprete. Por consiguiente, tenemos 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 está incluido el atributo “nombre” que hace referencia al nombre del intérprete y el identificador del intérprete que es quien determina al nombre.

En este punto se puede afirmar que ya tenemos nuestras tablas en 3FN. Las tablas en 3FN están normalmente lo suficientemente bien estructuradas para evitar los problemas asociados con la redundancia de datos. Sin embargo, debemos recordar que existen otras formas normales (la FNBC, la 4FN y la 5FN); pero, no las

desarrollaremos 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

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

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 nuestras 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 5FN que fueron propuestas posteriormente.
- La primera forma normal se centra que en los valores aceptados en las columnas tengan un dominio atómico. Podemos decir que nos permite manejar atributos compuesto y multivaluados de forma eficiente.
- La segunda forma normal 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 la llave principal.
- La tercera forma normal 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 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) Comprobar si el modelo relacional creado por usted en el punto 4 del capítulo 9 está en 3FN, si no lo está, llevarlo a la tercera forma normal.
- 2) Modificar la tabla “Reproducciones” del modelo presentado en la Figura 9-14 para que viole las tres formas normales: 1FN, 2FN y 3FN.
- 3) Llevar la siguiente tabla que representan las capacitaciones presenciales ofrecidas por una institución de enseñanza a la 3FN. Cada curso es de una única sesión, se ofrecen 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) Consultar en qué consiste la auditoria de bases de datos y qué aspectos de la auditoria considera importantes tener presente al momento de diseñar una base de datos.

Capítulo III

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 y sin la existencia física de las tablas no pueden ni almacenarse datos ni mucho menos, consultarse los datos. Dicho de otra manera, el diseño lógico especifica la estructura de las tablas que debemos crear, pero luego es necesario realizar la creación de 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 este lenguaje 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 las tablas, en términos generales, es bastante sencilla. Sin embargo, existen algunos detalles que si no son presentados de forma correcta pueden generar problemas, llevando a inconsistencias en los datos, sobreesfuerzo en la gestión de estas estructuras y problemas de rendimiento.

Con el DDL pueden crearse, modificarse y eliminarse las tablas de una base de datos. Las tablas son el tipo de objeto más importante que puede gestionarse con el DDL, pero, no el único. Existen otros tipos de objetos como los índices, las vistas, las funciones, los procedimientos almacenados, que también pueden ser gestionados con este sublenguaje. En el capítulo se abordará la implementación de índices, los cuales son objetos que permiten mejorar el rendimiento del DBMS al 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 abordar 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 hemos 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.

III.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 a 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 a crear, 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 conforman la tabla. A continuación, con la creación de la tabla “Géneros”, mostrada en la Figura III-1, se busca ejemplificar lo dicho hasta este punto.

Para crear esta tabla se une la palabra [CREATE](#), seguida por la palabra [TABLE](#) y el nombre de la tabla a crear, en este caso, Géneros. Luego se especifica cada uno de los nombres de las columnas que conforman la tabla, para la tabla Géneros son tres (idGénero, nombre, descripción). Cada columna debe tener un tipo de datos; En la figura se aprecia que 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 III-1.

Figura 11-1

Géneros		
		:
idGénero	int	PK
nombre	varchar(100)	
descripción	varchar(500)	NULL

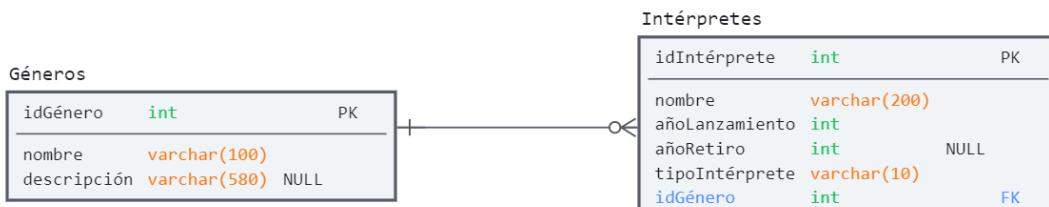
Script 11-1

```
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”; es menester recordar que una tabla solo puede tener una llave principal. También se indican 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**.

La creación de la tabla **Géneros** representaba un escenario bastante sencillo debido a que no tiene ninguna llave foránea. En la Figura 11-2 se muestra la relación entre la tabla **Intérpretes** y la tabla **Géneros**, en esta se puede apreciar una tabla con una llave foránea, la tabla **Intérpretes**. En primera instancia puede procederse con la creación de la tabla **Intérpretes** de forma similar a se hizo con la tabla **Géneros** como se muestra en el Script 11-2.

Figura 11-2



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
);
```

Sin embargo, en el proceso de creación de la tabla `Intépretes` 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, eliminar la tabla y corregir el error volviendo a crear la tabla con todas sus columnas y especificaciones; el segundo, 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, en este caso la tabla `Intérpretes`. La instrucción completa es como se muestra en el Script 11-3.

Script 11-3

```
DROP TABLE Intérpretes;
```

Una vez eliminada la tabla `Intérpretes`, se puede crear nuevamente, pero, esta vez no olvidando incluir la relación entre las dos tablas. Para hacer esto, como se ve en el Script 11-4, además de crear la columna, se debe especificar que hay una llave foránea, esto se hace con las palabras reservadas `FOREIGN KEY`, seguidas por cuál es la columna que será la llave foránea, 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, la tabla `Géneros`) y luego la columna de esa tabla a la que se está haciendo referencia (la columna `idGénero`).

Script 11-4

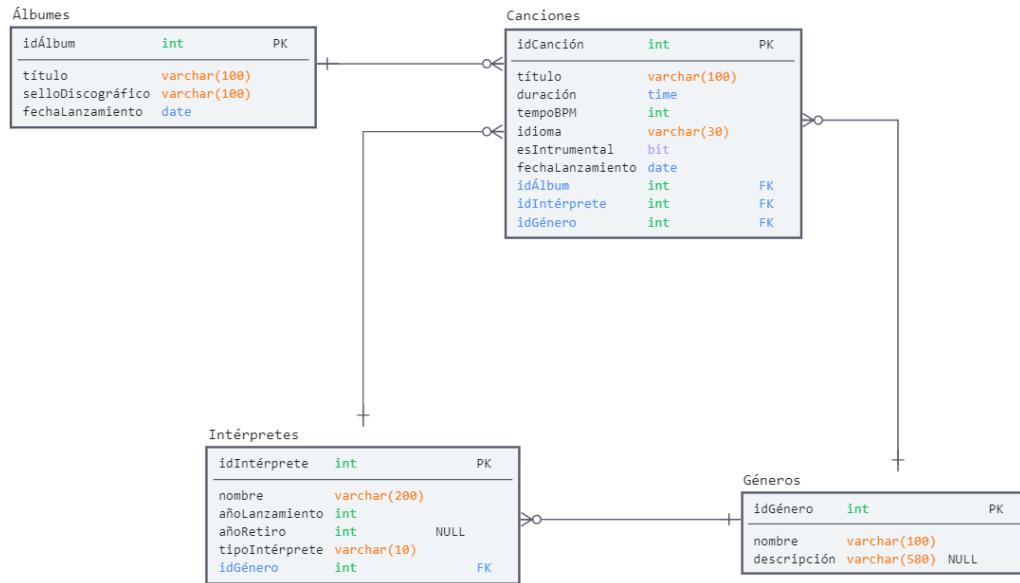
```
CREATE TABLE Intérpretes
(idIntérprete      INT PRIMARY KEY,
nombre            VARCHAR(200) NOT NULL,
añoLanzamiento   VARCHAR(200) NOT NULL,
añoRetiro         INT,
tipoIntérprete    VARCHAR(10) NOT NULL,
idGénero          INT NOT NULL,
FOREIGN KEY(idGénero) REFERENCES Géneros(idGénero)
);
```

La primera alternativa, la de eliminar y crear nuevamente la tabla `Intérpretes`, aunque permite conseguir el objetivo buscado: la tabla `Intérpretes` con todas sus columnas y la llave foránea deseada, puede resultar no plausible. Razón por lo cual existe otra alternativa y es 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 lo que se hace es indicar cuál es el cambio que se quiere hacer en la tabla: `ADD`, (añadir, en inglés) una llave foránea.

Script 11-5

```
ALTER TABLE Intérpretes
ADD FOREIGN KEY (idGénero) REFERENCES Géneros(idGénero);
```

La cantidad de llaves foráneas que podemos 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

En la Figura 11-3 se introducen dos nuevas tablas, la tabla **Álbumes** y la tabla **Canciones**. Por una parte, la tabla **Álbumes** que posee 4 columnas y ninguna de ellas es llave foránea. La tabla **Canciones** que posee 10 columnas, de las cuales 3 son llaves foráneas. El identificador del álbum (idÁlbum), que proviene de la tabla **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 crear primero la tabla **Álbumes** con todos 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
);
```

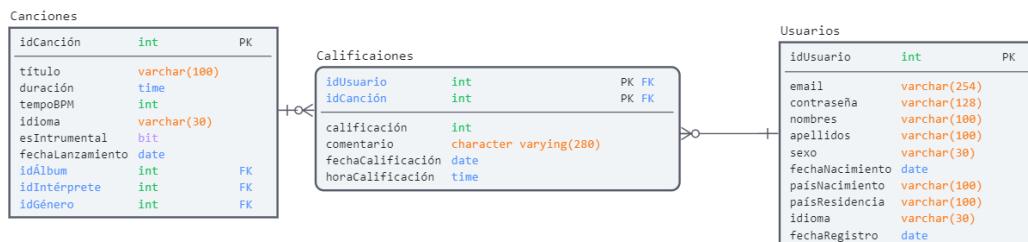
Con la tabla Álbumes lista ahora se puede crear la tabla Canciones sin ningún problema con las instrucciones mostradas en el Script 11-7. En esta se ve que se especifican todos los elementos: columnas, tipos de datos, llave principal, llaves foráneas y si aceptan o no el descriptor NULL.

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)
);
```

Una novedad que se incorporó en la definición de la tabla Canciones es definir 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 queremos que se ingrese por defecto, en este caso ‘0’. Ahora es momento de mirar la creación de la siguiente parte de la base de datos, presentada en la Figura 11-4.

Figura 11-4



De las tres tablas presentes en la Figura 11-4 solo dos son nuevas: la tabla Calificaciones y la tabla Usuarios. La tabla Canciones se acabó de crear en el Script 11-7. Debido a que la tabla Calificaciones tiene una llave foránea proveniente de la tabla Usuarios se procederá primero a crear la tabla Usuarios, para esto utilizamos 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, el uso de la palabra **UNIQUE**, el de la palabra **CHECK** y el de la palabra **CONSTRAINT**, todas estas palabras permiten incorporar restricciones. Estas restricciones ejercen como reglas que son comprobadas al momento de realizar una inserción o una actualización de los datos.

Anteriormente ya se trabajó con otras restricciones, estas son las de llave principal, las de llave foránea y la de valores nulos. Cuando se crean las tablas podemos especificar al nivel de columnas, unas reglas que aplican a los valores insertados en una columna en particular o que relacionan 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.

Tabla 11-1

Restricción	Descripción
PRIMARY KEY	Identifica de forma única una fila y no permite valores repetidos.
FOREING 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 asegurarse que los valores de una columna son únicos entre todas las filas de la tabla.

Especificamente las restricciones que se han añadido a la tabla **Usuarios** son:

- Utilizado la palabra **UNIQUE** para especificar que la dirección de correo electrónico no se puede repetir.
- Empleado la palabra **CHECK** para indicar que la columna **sexo** solo puede tomar dos valores, bien sea “M” o bien “F”.

- Utilizado la palabra **CONSTRAINT** para indicar una regla que debe cumplir cada fila, en nuestro caso, es que el valor de la columna `fechaNacimiento` sea menor que el de la columna `fechaRetiro`.

Al igual que con los demás componentes que definimos cuando creamos una tabla, las restricciones también se pueden definir después de que la tabla ha sido creada. Para esto se hace uso de las palabras reservadas **ADD** y **CONSTRAINT**. En el Script 11-9 se presenta cómo añadir una restricción que comprueba que el valor de la columna `contraseña` es diferente al de la columna `email`. Se ha utilizado la función **MD5** porque la contraseña ha sido guardada en este formato. De modo similar, puede eliminarse una restricción como se presenta en el Script 11-10.

Script 11-9

```
ALTER TABLE Usuarios  
ADD CONSTRAINT contraseña_validad 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`, su creación se consigue con el 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)  
);
```

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`. Aunque el Script 11-11 se ejecuta correctamente, allí se cometieron algunos errores que deben corregidos. El primero de ellos fue nombrar mal la tabla, se le ha puesto el nombre de `Valoraciones`. Por lo que se debe corregir este error para que quede con el nombre correcto, `Calificaciones`. El Script 11-12 muestra como conseguirlo.

Script 11-12

```
ALTER TABLE Valoraciones
RENAME TO Calificaciones;
```

De forma análoga sucedió con la columna valoración. La corrección del nombre de la columna 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, quiere decir esto que no recibe el descriptor NULL, por lo que se debe eliminar esta restricción. La eliminación de la restricción 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 en que van sucediendo los hechos de los cuales se guardan datos; actualizar los datos existentes, bien sea por la necesidad de reflejar cambios o porque deban corregirse errores de registro; borrar las filas que no se requieran y que no deben seguir ocupando espacio de almacenamiento.

Al ejecutar las operaciones de inserción, eliminación y actualización es indispensable tener control sobre cuatro aspectos en la tabla a la que estamos 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.

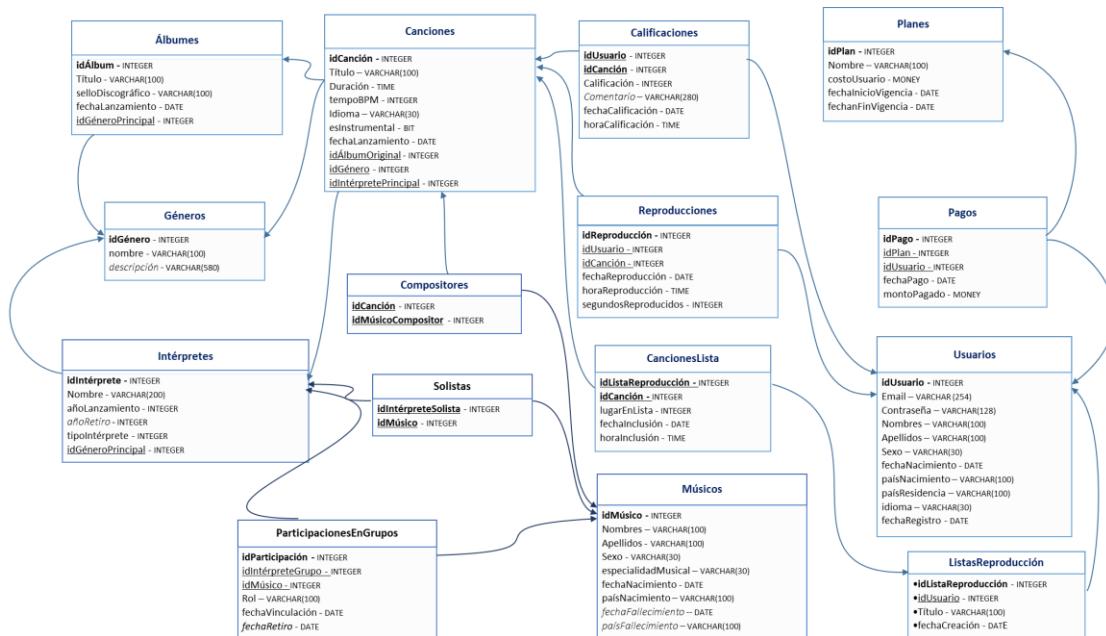
- Estructura de la clave principal. Es preciso saber cuál o cuáles columnas son parte de la clave principal para cumplir la restricción de unicidad, es decir, que los valores no pueden repetirse en diferentes filas de la tabla.
- Existencia de claves externas o foráneas. Debe saberse a cuál o cuáles 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 tienen en el diagrama de la Figura 11-5 . En este diagrama las claves foráneas de cada tabla se representan subrayando la columna y con una flecha se nos 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 que son “idÁlbumOriginal” que proviene de la tabla “Álbumes”, “idGénero” que proviene de la tabla “Géneros” e “idIntérpretePrincipal” que proviene de la tabla “Intérpretes”.

Figura 11-5



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, la fecha de lanzamiento de tipo DATE y el título de la canción de tipo VARCHAR(200). Cada tipo impone restricciones con respecto del formato en que deben ser insertados los datos y los valores máximos y mínimos aceptados. Por ejemplo, el VARCHAR(200) está indicando que esa columna almacena un texto, que al momento de ingresarla 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 los tipos de datos soportados en POSTGRESQL.

Tabla 11-2

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. Especificar el orden en que se suministraran los datos es opcional, si no se especifica el orden, 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 3), y ninguna de sus columnas representa una clave externa o foránea; el tipo de dato del identificador del género es entero, el del nombre y la descripción son de tipo cadenas de texto; la única columna que acepta el descriptor NULL es descripción. La clave principal es la columna idGénero por lo que el valor de esta columna para la nueva fila no debe existir previamente en la tabla. El género que se va a ingresar tendrá el identificador “14”, el nombre “Rap” y la descripción “Este género surgió como un movimiento social...” y se hace 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 caso del

nombre y de la descripción, en ambos casos se usan las comillas simples. Como se indicó previamente, el orden en que los valores de cada columna son suministrados puede ser modificado, siempre que se indique el orden en que se van a suministrar, tal como se muestra 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 el mismo script se aprecia cómo se utilizó también 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 nuestra responsabilidad saber cuáles valores hemos insertado en la llave principal de la tabla para no repetirlo, puesto que si se repite se obtendrá un mensaje de error indicando que se está violando la restricción de unicidad de la llave principal. Existe una alternativa que se puede usar al momento de crear una tabla y así no tener que ocuparse de garantizar la unicidad: la alternativa consiste en indicar que el tipo de datos (realmente es un pseudo-tipo) va a ser autoincremental (**SERIAL**), de esta forma el DBMS se encargará que en cada inserción el valor sea igual al último valor que se intentó insertar más uno. Es un pseudo-tipo porque el tipo real es entero, solo que tiene el comportamiento descrito. En el Script 11-18 se muestra como 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 al 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, hace referencia a valores que deben estar presentes en las tablas de la cual proviene la clave foránea antes de poder existir en la tabla `Canciones`. Este es el caso de las columnas `idIntérpretePrincipal` que hace referencia a la columna `idIntérprete` de la tabla `Intérpretes`, la columna `idGénero` que hace referencia a la columna `idGénero` de la tabla `Géneros` y al “`idÁlbumOriginal`” que hace referencia a la columna “`idAlbum`” de la tabla “`Álbumes`”. Para el caso de las dos primeras columnas, no aceptan el descriptor `NULL`, por lo que el valor que se suministre en estas columnas al momento de insertar una canción debe existir en la tabla a la cual hace referencia. En el caso de “`idÁlbumOriginal`” que, si acepta el descriptor `NULL` se tienen dos opciones, o suministrar un valor que exista en la tabla a la cual hace referencia o ingresar el descriptor `NULL`.

Aclarado lo anterior, ahora nuestra atención se debe centrar 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 los hemos explicados y los tipos `BIT`, `DATE` y `TIME`. El tipo `BIT` indica que solo puede tomar dos valores “0” o “1”. El tipo `DATE` nos dice que solo puede aceptar fechas en el formato preestablecido según la configuración regional, es decir, “`dd/mm/yyyy`”, “`mm/dd/yyyy`”, etc. El tipo `TIME` permite ingresar el tiempo en formato horas, minutos, segundos y milisegundos, 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 momento de especificar los valores a insertar en la tabla también se pueden utilizar subconsultas autónomas que devuelvan el valor que deseamos 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 es realizado 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 podemos ingresar dos reproducciones que se 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 deseamos realizar la inserción, en este caso, la tabla “Reproducciones”. La clave primaria es el identificador de la reproducción (idReproducción), luego tiene dos claves foráneas, una que hace referencia a la canción reproducida y la otra al usuario que la reprodujo; finalmente los tipos de datos que maneja son INTEGER, TIME, y DATE.

Una primera forma en la que podemos proceder es utilizar dos sentencias `INSERT`, una por cada reproducción. En el Script 11-22 se aprecia como insertar la primera reproducción, en esta se observa que utiliza el identificador “498” y que 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, pero la fecha de reproducción es de tipo DATE y la hora de tipo TIME, por lo cual 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`. Sin embargo, SQL permite realizar más de una inserción separando cada una de las filas a insertar 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.bonnell@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 a insertar operan como una transacción, o lo que es lo mismo, o se insertan las dos filas o no se inserta ninguna. El concepto de transacción es de vital importancia en el manejo de las operaciones realizadas por los DBMS.

¿Cómo puedo insertar en una tabla todas las canciones interpretadas en español?

Para realizar lo anterior 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 que 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, esto se realiza con el Script 11-24.

Script 11-24

```

SELECT C.idCanción,
C.título AS títuloCanción,
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 la nueva tabla “CancionEnEspañol” 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, por lo que el Script 11-25 nos permite conseguir lo descrito.

Script 11-25

```
SELECT C.idCanción,
       C.título AS títuloCanción,
       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 títuloCanción,
       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 nuestra 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 eliminan. Para eliminar las filas de una tabla, SQL proporciona dos sentencias: `DELETE` y `TRUNCATE`. La sentencia `DELETE` permite borrar las filas de una tabla basados 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” y cuáles son las filas en particular que se desean eliminar, en este caso todas aquellas donde el identificador del usuario sea igual a “6”. Lo anterior se puede realizar como está planteado en el Script 11-27.

Script 11-27

```
DELETE FROM Reproducciones
WHERE idUsuario = 6
```

La condición de la cláusula **DELETE**, como se planteó, es opcional, si no se incluyen todas las filas de la tabla son borradas. Por consiguiente, al realizar operaciones de eliminación se debe 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 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 a borrar, en este caso se hace uso del operador **LIKE** para indicar que son aquellas canciones cuyo título termina en la vocal “a” y una subconsulta autónoma para indicar que deben ser del género “salsa”. Lo descrito por palabras esta expresado en SQL en 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 y 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 )
```

Pero, pese a que la consulta anterior no tiene ningún error de sintaxis ni lógico, el DBMS puede que no deje realizar la acción que se desea si las filas que se desean eliminar están siendo referenciadas en otras tablas, es decir, si el valor del identificador de la canción que se desea eliminar aparece referenciado en otras tablas. En este caso el “idCanción” de la tabla “Canciones” aparece en las tablas “Calificaciones”, “Reproducciones”, “CancionesLista” y “Compositores”. Por lo que para eliminar las filas de la tabla “Canciones”, se debe eliminar antes las referencias que hacen de estas las tablas mencionadas. En los Script 11-31, Script 11-32, Script 11-33 y Script 11-34 se eliminan las filas que hacen referencia a los identificadores de las canciones que deseamos eliminar. Una vez no hay referencias a las canciones que se desean 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 II-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 II-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)))
```

Script II-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)))
```

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 a eliminar, por consiguiente, esta sentencia siempre elimina todas las filas de la tabla.

Borrar todas las filas de la tabla canciones

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;
```

Al ejecutar la anterior sentencia se obtiene 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, se tiene 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;
```

Con la combinación **TRUNCATE CASCADE** es necesario ser cuidadoso puesto que puede eliminar todas las filas de una tabla sin tener intención de hacerlo. Cuando se desea eliminar todas las filas de una tabla **TRUNCATE** y **DELETE** tienen diferencia en cuanto al rendimiento, la primera ofrece un mejor rendimiento que la segunda debido a que los registros que hace de las eliminaciones son menos detallados que los que realiza el **DELETE**. Adicional a esta diferencia relacionada con el rendimiento, también hay una diferencia funcional, 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 a insertar en esa tabla tendría el mínimo valor que se acepta (1 para el caso de los enteros). **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**. Debe especificarse el nombre de la tabla, los cambios a realizar en las columnas de cada fila de tabla y finalmente el predicado que permite determinar cuáles son las filas a actualizar.

La canción con id 2 debe aparecer como instrumental

Para realizar el cambio se utiliza la sentencia `UPDATE` seguida de la tabla en la que se desea realizar el cambio, en este caso “Canciones”. Luego cual es el cambio a realizar, en este caso 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';
```

Al ejecutar el script se haría el cambio sobre todas las filas de la tabla canciones, a razón de esto, es necesario especificar cuál es la canción a modificar, en este caso la canción 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”. Para realizar el cambio se hace uso de una subconsulta autónoma que devuelve el identificador del plan familiar en la cláusula **SET**. Además, se cambia la fecha del pago a la fecha actual, para eso volvemos a aprovechar las funciones **NOW** y **CAST**. Finalmente, con la clave principal de la fila que deseamos modificar se tiene la garantía de que la operación afectara una sola fila, la deseada. 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 de tablas. Combinando la tabla “Calificaciones” con la tabla “Canciones” y con la tabla “Género”. Para estos escenarios se puede incluir la cláusula **FROM** para incluir el **INNER JOIN** entre la tabla “Canciones” y la tabla “Géneros”. Luego en la cláusula **WHERE** se especifica que entre la tabla “Calificaciones” y la tabla “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 de “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 este usuario particular. La consulta que permite determinar el lugar de cada canción dentro del escalafón del usuario es la presentada en el Script 11-43.

Script 11-43

```
SELECT idCanción, COUNT(Reproducciones),
DENSE_RANK() OVER(ORDER BY COUNT(Reproducciones) DESC ) ranking
FROM reproducciones
WHERE idUsuario = 1
GROUP BY idCanción
```

Esta consulta se puede embeber en una sentencia `UPDATE` que permita determinar el lugar que ocupa la canción dentro de esta lista y si el lugar ocupado es el uno o el dos cambiar el comentario actual a “Me gusta esta canción” tal cual se presenta en el Script 11-44.

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 y es que, aunque en el diseño lógico y conceptual de las bases de datos relacionales se toman decisiones importantes, en la fase de implementación también se toman decisiones 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.

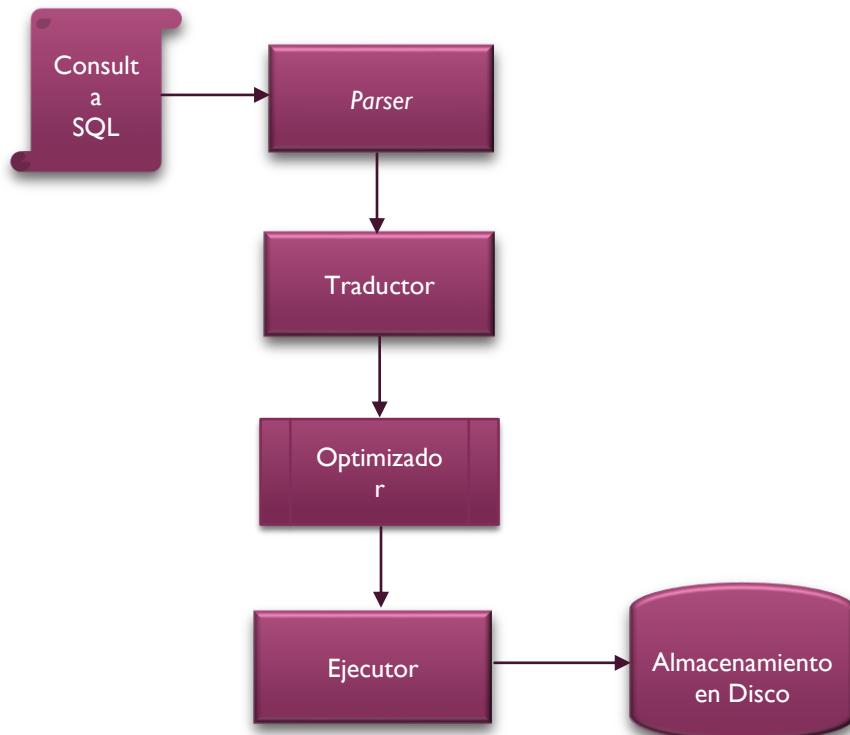
Como se ha mencionado más de un par de ocasiones a lo largo de este libro, SQL es un lenguaje declarativo, por consiguiente, no se puede definir cómo los datos almacenados en las tablas son accedidos, esta responsabilidad es dejada totalmente al DBMS que en cada consulta debe determinar: cuál es la mejor forma para acceder a los datos. Para cumplir con este propósito, los DBMS tienen un componente, llamado

el planificador/optimizador, que debe decidir entre todas las alternativas posibles para responder a una consulta cuál es la mejor.

A groso modo se puede afirmar que una consulta SQL antes de ser ejecutada transita por un conjunto de fases y en cada una de estas fases está a disposición de diferentes componentes del DBMS como se muestra en la Figura 11-6. De estas cuatro fases, la tercera fase, de planificación/optimización, es la que puede ser afectada por medidas administrativas y de diseño que se tomen al momento de definir y configurar las tablas. Una de estas medidas es la inclusión de índices. Muchas veces las consultas pueden demorar en su ejecución más tiempo del que consideramos “normal”, en este escenario se puede optar por reescribir la consulta o utilizar un índice apropiado para acelerar el acceso a los datos subyacentes. En este punto 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.
- Que el acceso de 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 acceder a los datos puede tornarse ineficiente según los datos que se estén intentando acceder.

Figura 11-6



Los índices de una tabla operan de forma similar a cómo funcionan los índices y las tablas de contenido de los libros. Los índices de los libros permiten acceder de forma

rápida a la página del libro donde está el contenido deseado, de forma que no se pierda tiempo revisando todo el libro u hojeando todas las páginas del libro. En las tablas de las bases de datos, se crean índices para saber dónde están las filas de la tabla, estos índices se crean tomando como referencia a las columnas. Entonces, un índice es una estructura de datos que permite un acceso más rápido a los datos de la tabla subyacente, para que se puedan encontrar las filas específicas rápidamente. Aquí, "rápidamente" significa más rápido que escanear toda la tabla subyacente y analizar cada fila.

Un índice se puede construir sobre una sola columna o en múltiples columnas a la vez; Además, un índice 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 costo de mantener organizado y balanceado el árbol.

Con lo dicho se puede pensar: si los índices mejoran el rendimiento al momento de acceder a los datos, usemos índices en todas las columnas y en todas las combinaciones posibles. Aunque este razonamiento puede ser intuitivo no es correcto, 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 considerar al contemplar el uso de índices, si es importante considerar que:

- Los índices tienen un costo de mantenimiento. Al ser gestionados como arboles 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, lo que significa que se tiene almacenado un archivo de datos para la tabla y uno para cada índice que se cree en la tabla. 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. Con todo esto debemos evaluar cuándo los índices son útiles para mejorar nuestras 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 el índice se ve cómo se mantiene una referencia a cada uno de las filas de la tabla en función del valor que tiene en la

columna título. Este ejemplo muy básico permite ver 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 que a recuperar. Es evidente que entre más filas haya en la tabla de la derecha resultará más útil el índice.

Figura 11-7

INDICE		DATOS				
		idCanción	título	intérprete	calificación	usuario
Ginza		14	Ginza	J. Balvin	3	Marie Curie
Una aventura		3	Una aventura	Niche	4	Piedad Bonett
Bolero falaz		2	Una aventura	Niche	2	Marie Curie
		7	Bolero falaz	Aterciopelados	5	Piedad Bonett
		4	Una Aventura	Niche	2	Baruch Spinoza
		8	Ginza	J. Balvin	4	Baruch Spinoza
		9	Bolero falaz	Aterciopelados	4	Marie Curie
		7	Bolero falaz	Aterciopelados	3	Marie Curie
		14	Ginza	J. Balvin	1	Piedad Bonett

Los diferentes DBMS cuentan con instrucciones que permiten dilucidar el razonamiento hecho por el componente planificador/optimizador para elegir entre las diferentes alternativas al momento de ejecutar una consulta. En el caso de PostgreSQL esta sentencia es `EXPLAIN`. Esta es una herramienta poderosa para entender cómo el DBMS interpreta las consultas y determinar cómo “ayudar” a que éste realice de forma más rápida el acceso a datos. Es importante tener claro que la optimización de las consultas es un tema más complejo de lo que en este capítulo se presenta; muchas veces requiere largas sesiones de pruebas, por lo que no es el objetivo abarcar toda la complejidad que trae consigo la optimización de consultas en las bases de datos relacionales, sino solo un entendimiento básico de los 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 se muestra el plan de ejecución que crea el DBMS y en la Tabla 11-3 el resumen de las estadísticas del mismo.

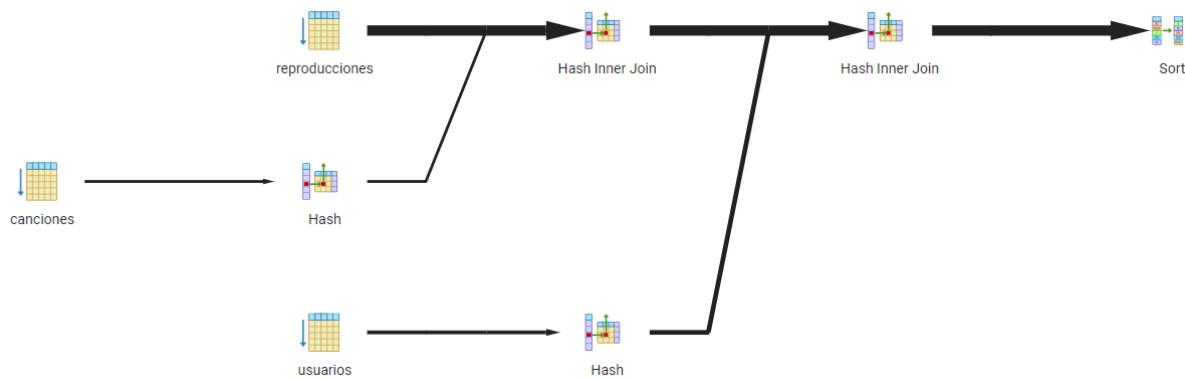
Script 11-45

```
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 8 nodos (los nodos son las unidades encargadas de las tareas de procesamiento de datos) y que en el equipo en que se ejecuto la instrucción empleó 1561.638 ms. En la figura aparecen tres nodos (Canciones, Reproducciones y Usuarios) que si única función es leer las tablas de manera secuencial. El modo secuencial es el modo por defecto para acceder a los datos. En un escaneo secuencial, 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 tenemos dos combinaciones (*hash inner join*) como su nombre lo indica realizan la tarea de combinar las tablas usando una función hash, es por esto que las tablas “Canciones” y “Usuarios” son pasadas previamente por una función hash. Finalmente, se tiene un nodo de ordenamiento (*sort*) que es el encargo de entregar los datos ordenados tal cual se solicitan en la consulta.

Figura 11-8



Al mirar la tabla de las estadísticas con detenimiento se observa cómo el mayor costo de ejecución se emplea en el ordenamiento, por lo que puede surgir la duda si al emplear un índice que permite recuperar los datos basados en la fecha de reproducción, mejoraría el tiempo de ejecución de la consulta.

Tabla 11-3

Nodo	Tiempo acumulado
Sort	1523.258 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 fecha de reproducción.

Script 11-46

```
CREATE INDEX idx_reproducción_date
ON Reproducciones( fechaReproducción );
```

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 1162.713 ms, es decir, un 24 % menos que sin usar el índice. En el plan de ejecución se aprecia que ahora se está utilizando un escaneo de índice para acceder a los datos de las tablas.

Figura 11-9

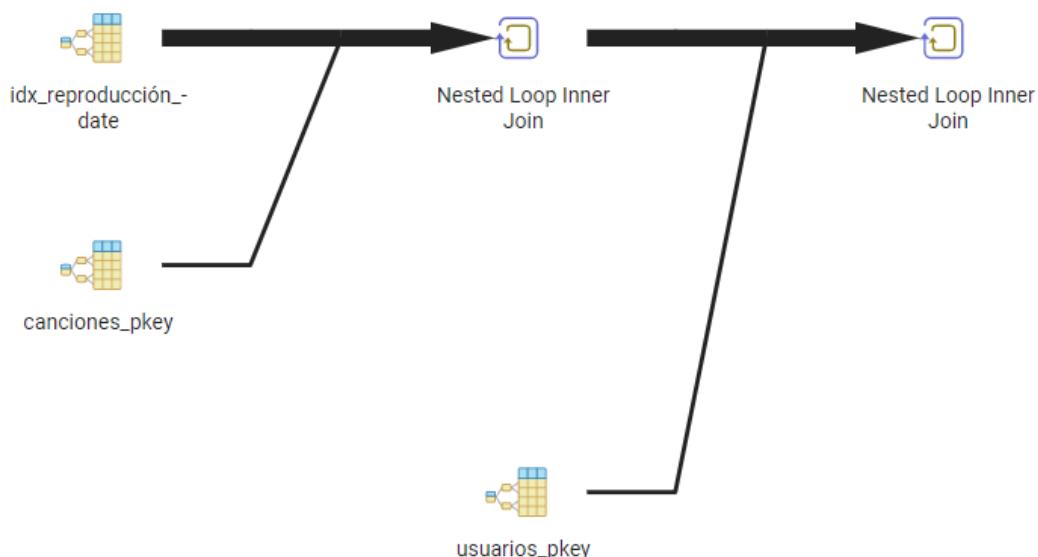


Tabla 11-4

Nodo	Tiempo acumulado
Nested loop inner join (Reproducción y Canciones)	1162.713 ms
Nested loop inner join (Reproducción 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

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”,

es simplemente plantear una condición con 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, al crear índices de varias columnas, siempre se debe colocar primero las columnas más selectivas. Puesto que, si las primeras columnas son las más selectivas, el método de acceso al índice será el más económico. Para incluir un índice compuesto por las columnas “idUsuario” y “fechaReproducción” es muy simple, solo se deben indicar cuáles son las columnas que hacen parte del índice compuesto, como se muestra en el Script 11-48.

Script 11-48

```
CREATE INDEX idx_reproducción_usuario_date
ON Reproducciones( idUsuario, fechaReproducción );
```

Finalmente, para eliminar un índice se debe usar las palabras reservadas `DROP INDEX` acompañadas por el nombre del índice a borrar. En el Script 11-49 se presenta el caso si se desease borrar 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 desean 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 en mayor rendimiento, necesitan menos espacio en disco y generan menos 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 estas estructuras.
- El sublenguaje DML no se agota con la sentencia `SELECT`, también proporciona las sentencias `INSERT`, `UPDATE` y `DELETE` para la inserción, actualización y 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 insercciones o no se realiza ninguna.
- Las sentencias `UPDATE` y `DELETE` pueden realizarse o sobre todas las filas de una tabla o sobre un subconjunto si se utiliza la cláusula `WHERE`.
- Al momento de realizar inserciones y actualizaciones se debe 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 momento de realizar eliminaciones se debe ser cuidadoso con el uso de `DELETE FROM` y `TRUNCATE` puesto que eliminan 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 eliminar las filas de las tablas relacionadas.
- Los índices son una herramienta que permiten mejorar el rendimiento al 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. Usar el sublenguaje DDL para crear las tablas del modelo lógico resultante del punto número uno del capítulo anterior.

2. Seleccionar un intérprete musical, por ejemplo “The Beatles”, e insertar las canciones de dos álbumes de ese interprete en la base de datos. Para eso realizar las inserciones necesarias en las tablas Álbumes, Géneros, Canciones, Intérpretes, ParticipaciónEnGrupos o Solistas, Compositores y Músicos.
3. Proponer una operación de actualización y otra de inserción que se puedan resolver utilizando subconsultas correlacionadas. Luego escribir el código SQL necesario para su implementación.
4. Guardar 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. Eliminar 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 para las canciones con la peor valoración promedio el nuevo valor para el comentario sea “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. Crear un índice sobre la tabla “Canciones”, luego utilizar el comando **EXPLAIN** para conocer el plan de ejecución creado para una consulta sobre dicha tabla que utilice el índice recién creado. Si, al utilizar el comando **EXPLAIN** sobre la consulta creada, se observa en el plan de ejecución que no se utiliza el índice, insertar más filas hasta que se utilice el índice o modificar el índice para que sea útil para el planificador/optimizador.
9. Revise la ley 1581 de 1992 del estado de Colombia e identifique cómo regula esta ley 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 que 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 procedural en

particular, por consiguiente, han evolucionado para hacer posible la implementación de subprogramas utilizando un enfoque imperativo. Han evolucionado porque 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 creó el PL/PGSQL, en ambos casos, PL significa Lenguaje Procedimental (del término en inglés, *Procedural Language*). Con estos lenguajes podemos 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 hemos 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 son subprogramas que pueden tomar un conjunto de parámetros que son proporcionados por quien los invoca y realizan una serie de operaciones que le permiten cumplir con una tarea. Tanto los procedimientos como las funciones pueden modificar los datos que se le pasan como argumento; pero, solo las funciones tienen que devolver datos a quien las invoca, mientras que un procedimiento no necesariamente lo hace. Es por esto por lo que por regla general se utilizan procedimientos a menos que se necesite un valor de retorno.

En este capítulo nos enfocaremos en mostrar la utilidad que tienen los subprogramas en el contexto de las bases de datos relacionales. Para cumplir con este fin utilizaremos el lenguaje PL/PGSQL que como ya mencionamos, es el lenguaje construido por POSTGRESQL para permitir la programación procedural. También, veremos 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. Es importante dejar claro que, aunque utilizaremos el lenguaje creado por POSTGRESQL, la fundamentación conceptual y elementos que en este capítulo se utilizan son equivalentes entre los diferentes DBMS.

12.1 Vistas

Frecuentemente nos surgen necesidades como reutilizar una consulta que hemos creado, agregar una capa de abstracción para dar a acceso a nuestros datos o simplemente queremos “guardar” el resultado de una consulta para no tener que escribirla nuevamente cuando la misma necesidad de datos aparezca o bien una necesidad de datos basada en los resultados de una consulta previa. En situaciones como las mencionadas, SQL nos proporciona un objeto llamado vistas (VIEW, por el término en inglés).

Una vista la podemos definir como una tabla virtual que está basada en el resultado de una consulta SELECT. Decimos que es 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, no 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. 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 que se tenga. Pero, puede surgir la pregunta, por qué tener una tabla virtual a partir de una consulta, si ya tenemos las tablas y una vista no nos permite responder preguntas que no podamos responder ya con una sentencia SELECT sobre las tablas. Los beneficios de las vistas son muchos, a continuación, listamos solo algunos centrados en la seguridad, el rendimiento y la productividad:

- Las vistas nos permiten simplificar consultas complejas haciendo uso de la modularidad y reuso 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 podemos hacer esta descomposición porque con ellas podemos resolver una necesidad de datos y utilizarla cuantas veces sea necesario en lugar de escribir el código SQL completo una y otra vez, disminuyendo así la cantidad de código SQL que necesitamos escribir.
- Como consecuencia de lo anterior, el descomponer el problema en partes haciendo uso de las vistas nos permite que sea más fácil probar y depurar nuestras sentencias SQL. Algunas veces las consultas resultan ser tan extensas que se complica su lectura y comprensión.
- Adicionalmente, con las vistas podemos proporcionar un control de acceso sobre los datos presentes en nuestras bases de datos, puesto en otros términos, podemos implementar tanto una autorización a nivel de fila, dejando de lado las filas que no cumplen con un determinado predicado, así como también, proporcionar autorización a nivel de columnas no incluyendo las columnas que no se desean 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. Teniendo clara toda la argumentación que hemos esbozado a favor de las vistas miremos cómo funcionan en la práctica. Supongamos que como administradores o usuarios de la base de datos de la plataforma “Mis Canciones” queremos 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 necesidad podemos hacer uso del Script 12-1 donde se utilizan funciones de agregación, combinaciones de tablas, agrupamiento y subconsultas.

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
```

Imaginemos ahora que esta es una necesidad de datos que emerge con cierta frecuencia, tener que escribir una y otra vez la misma consulta resulta agotador y aburridor. Pocas cosas son tan aburridoras como “perder” el tiempo haciendo siempre lo mismo. En este contexto son útiles las vistas; las vistas nos permiten darle nombre a una consulta y poder acceder luego a ella utilizando solo el nombre que le hemos dado. Para crear una vista usando la consulta del Script 12-1 solo tenemos que incluir las palabras reservadas CREATE y VIEW como se muestra en el Script 12-2.

Script 12-2

```

CREATE VIEW ReproduccionesYCalificacionesPorInterprete
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

```

Con el Script 12-2 hemos asociado el nombre “ReproduccionesYCalificacionesPorInterprete” a la consulta del Script 12-1, por lo que ahora podemos tratarla como una tabla. Es pertinente recordar que la podemos tratar como una tabla al momento de acceder a ella, pero no es una tabla, es una vista. Podemos realizar las mismas operaciones que hacíamos 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.

Script 12-3

```

SELECT *
FROM ReproduccionesYCalificacionesPorInterprete

```

También seleccionar unas columnas en específico, filtrar las filas y demás operaciones que hemos presentado en este libro. En el Script 12-4 se presenta cómo utilizamos la vista recién creada para seleccionar el nombre de los interpretes que tienen una calificación promedio al menos de “3.0” entre todas sus canciones.

Script 12-4

```
SELECT nombre
FROM ReproduccionesYCalificacionesPorInterprete
WHERE calificación >= 3.0
```

Las vistas también pueden ser modificadas, pero no de la misma forma como lo hacíamos con las tablas. Para modificar las vistas el estándar SQL nos provee de la palabra reservada REPLACE. Es clave tener presente que al reemplazar la definición de una vista con la palabra clave REPLACE, la lista de columnas devueltas por la consulta debe ser idéntica antes y después del reemplazo, incluido el tipo, el nombre y el orden de la columna. En el Script 12-5 se aprecia cómo se puede modificar la vista “ReproduccionesYCalificacionesPorInterprete”.

Script 12-5

```
CREATE OR REPLACE VIEW ReproduccionesYCalificacionesPorInterprete
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 nos abre a la posibilidad de poder modificar la lógica de la vista, no nos permite modificar las columnas que esta devuelve. En casos donde necesitemos modificar las columnas que devuelve la vista, se hace necesario borrar la vista y crearla nuevamente. Las vistas la podemos borrar haciendo uso de la palabra reservada DROP como se presenta en el Script 12-6.

Script 12-6

```
DROP VIEW ReproduccionesYCalificacionesPorInterprete;
```

Para concluir con el tema de las vistas es necesario mencionar que el enfoque presentado de las vistas es el tradicional, a saber, vistas que son tablas virtuales, es decir, que no se almacenan físicamente y que solo reciben operaciones de consulta. Con la evolución de los DBMS se ha visto la necesidad de que las vistas se envistan con otras propiedades que en principio eran propias de las tablas: que se guarden y que se le puedan aplicar otras operaciones del lenguaje DML. En cuanto a la primera propiedad, que las vistas se guarden, para ello existen las vistas materializadas (o vistas donde el resultado de la consulta es guardado físicamente). Para crear una vista materializada solo se necesita emplear luego de la palabra VIEW, la palabra MATERIALIZED como mostramos en el Script 12-7.

Script 12-7

```
CREATE MATERIALIZED VIEW ReproduccionesYCalificacionesPorInterprete
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 que las vistas materializadas sí existan físicamente en nuestras bases de datos, pero que a su vez dependen de tablas u otras vistas que son utilizadas en las consultas SQL, les impone una nueva necesidad que no tiene las vistas tradicionales, esta es:

¿cómo se actualizan los datos de las vistas materializadas? La respuesta es que no se actualizan automáticamente, es necesario indicarle al DBMS que deseamos que la vista se sincronice con las tablas de las cuales depende, lo dicho lo conseguimos con la palabra reservada REFRESH tal como se muestra en el Script 12-8.

Script 12-8

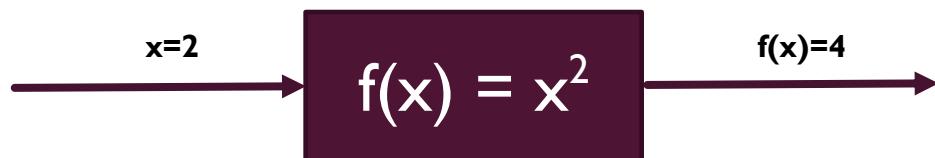
```
REFRESH MATERIALIZED VIEW ReproduccionesYCalificacionesPorInterprete;
```

12.2 Funciones

Hemos trabajado con diferentes tipos de funciones a lo largo de este libro: las funciones escalares, las funciones de agregación y las funciones de ventana. Todas estas funciones nos las proporciona el DBMS y las podemos utilizar siempre que deseemos sin preocuparnos por su implementación; pero, pese a la amplia gama de funciones que se nos proporciona, es imposible que se satisfagan a priori todas las necesidades que podamos tener, por lo que frecuentemente nos veremos en la necesidad de tener que crear nuestras propias funciones; este tipo de funciones que son de nuestra autoría se conocen como funciones definidas por el usuario.

Las funciones definidas por el usuario son subprogramas que podemos crear para encapsular comportamientos, estas funciones pueden recibir parámetros y retornar un valor. Nosotros estamos acostumbrados al uso de funciones desde las matemáticas, donde frecuentemente hablamos de unas variables independientes (argumentos); una definición de la función (operaciones) y un resultado de aplicar la función (salida). En la Figura 12-1 vemos el ejemplo de una función $f(x) = x^2$, esta función recibe un argumento y realiza una operación con este argumento, en este caso, esa operación es elevarlo al cuadrado. Teniendo claro esto, cuando el argumento tiene el valor de dos, la salida será cuatro.

Figura 12-1



El ejemplo anterior es muy sencillo, pero nos permite abstraer la idea del objetivo de las funciones y sus partes básicas, pero ahora llevémoslo al terreno de la programación imperativa procedural 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”?

Para responder a esta necesidad de datos encontramos que en la tabla “Usuarios” tenemos el insumo necesario para resolverla: la fecha de nacimiento de los usuarios y podemos calcular con la función NOW, la fecha del día de hoy. Con estas dos fechas podemos 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, podemos encapsular este comportamiento en una función de nombre *edad*.

Para crear una función debemos utilizar las palabras reservadas CREATE FUNCTION seguida por el nombre de la función que deseamos crear, en este caso *edad*. En el Script 12-14 se presenta cómo crear la función. Además del nombre de la función, apreciamos que se recibe un argumento que hemos 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 nos indicará cuantos 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 viene la implementación de la función, que es el código que se ejecuta cada vez que la función es invocada y empieza luego de la palabra AS. En el cuerpo de la función observamos que se declaran dos variables: una “edad” de tipo entero y otra “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 la palabra DECLARE (declarar) y la palabra BEGIN. Luego vienen todas las operaciones que la función debe realizar y estas se escriben entre las palabras BEGIN (inicio) y END (fin).

En el Script 12-14 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 la variable edad se le asigna la cantidad de años que hay entre la fecha que se proporciona como argumento y la fecha almacenada en la variable “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 que hemos utilizado hasta el momento, el punto y coma especifica el final de cada instrucción, pero para definir los subprogramas necesitamos utilizar el punto y coma dentro del cuerpo del mismo subprograma, lo cual causaría cierta confusión al DBMS al momento de interpretar dicho script. Es por esto que debemos indicarle al DBMS que temporalmente no interprete el punto y coma como final de la instrucción que crea la función. Para esto, se utilizan las etiquetas de \$BODY\$ (Tanto de inicio como de fin), con las cuales le estamos diciendo que entre estas dos etiquetas podemos 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. Esto lo vemos cuando especificamos que el lenguaje que se está utilizando es PL/PGSQL y terminamos con “;”.

Una vez creada la función podemos invocarla siempre que lo necesitemos y siempre que le proporcionemos los parámetros apropiados (si los hay), es este caso una fecha. En el Script 12-10 se muestra que si pasamos el valor “08/08/1987” a la función “edad” obtendremos el valor de “33”. La forma más simple de invocar una función es hacerlo directamente en una sentencia SQL.

Script 12-10

```
SELECT edad('08/08/1987')
```

Recordemos que como la función “edad” devuelve un entero puede ser utilizada en cualquier lugar donde se acepte un escalar. Por consiguiente, podemos consultar la tabla “Usuarios” y proporcionarle como argumento la columna “fechaNacimiento”, como lo presentamos en el Script 12-11.

Script 12-11

```
SELECT nombres, edad(fechaNacimiento)
FROM Usuarios
```

Este primer ejemplo, aunque sencillo, nos ayuda a tener claras las generalidades necesarias para crear una función, ahora miremos 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 duró mientras que para los que no se han retirado la cantidad de años que lleva.

En la necesidad de datos planteada en la pregunta anterior se dice que se quiere mostrar el nombre de los intérpretes que lo tenemos con la columna “nombre” de la tabla “Intérpretes”, mientras que el estado se deriva dependiendo de si el intérprete está o no retirado, de la siguiente forma:

- Si el intérprete 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.

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” sabemos si un intérprete está retirado o no con el valor de la columna “retiro”, si esta columna tiene cualquier valor, es decir, si no se ha insertado el descriptor NULL, es porque el intérprete se retiró en el año que el valor de la columna indique. Si, por el contrario, la columna “retiro” tiene el descriptor NULL es porque no se ha retirado aún.

Para abordar esta pregunta podemos hacer uso de complejas expresiones CASE con expresiones anidadas que dificultan la legibilidad y son propensas a errores; o descomponer el problema en dos partes, siguiendo el siempre presente principio de divide y vencerás. Para hacerlo podemos pensar en 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 determine los años de vida artística que lleva o los años que llevaba cuando se retiró, preparando así el mensaje indicado según cada caso. Esta lógica que acabamos de describir la encapsulamos en la función “estado” del Script 12-12.

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';

```

Esta nueva función tiene un nuevo elemento, el uso de la estructura de control IF/ELSE, con esta estructura de control podemos determinar el flujo de ejecución del programa. La utilizamos 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”; posteriormente, el valor que esté almacenado en la variable “msg” es devuelto a quien invoque la función.

La función creada podemos invocarla 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;

```

Dentro del cuerpo la función amplía la variedad de elementos que se pueden utilizar, por ejemplo, podemos utilizar las sentencias SQL que hemos visto a lo largo de este libro. Analicemos 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 nos está planteando 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 lograr esto, lo primero que haremos será crear una función de nombre “Roles” que recibe el identificador del músico y nos devuelve una fila

compuesta por tres columnas donde la primera columna indicará si es solista, la segunda si es compositor y la tercera si ha pertenecido a un grupo.

La implementación es bastante sencilla como se muestra en el Script 12-14, en la declaración de la función hemos especificado que la función utiliza cuatro argumentos, uno de entrada (IN) que debe ser proporcionado por quien invoque la función; y tres argumentos de salida (OUT) que son devueltos por la función a quien la invoque. Existe un tercer tipo de argumento, que no utilizaremos, los argumentos 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 apreciamos que el tipo de valor retornado es de RECORD, o lo que es lo mismo una fila compuesta por los tres argumentos de salida que definimos.

Una vez declarada la función, la 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” es guardado el identificador del músico resultante de buscar el músico en la tabla “Solista”. Si el valor es encontrado la variable almacenará un entero que representará al músico, en caso contrario, almacenará el descriptor NULL que indicará que ese músico en particular no es solista. Esta misma operación se repite para la tabla “Compositores” y la tabla “ParticipacionesEnGrupo”.

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';
```

También es clave destacar que a los argumentos de salida (OUT) le hemos 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, o en otras palabras se le asigna TRUE si es solista y se le asigna FALSE si no es solista. Lo mismo se aplica para las variables “esCompositor” y “perteneceGrupo”. Como hemos utilizado argumentos de tipo OUT no debemos tener una expresión con RETURN puesto que este tipo de argumento son devueltos automáticamente cuando la función ha llegado a su fin.

Para invocar esta función podemos hacerlo como se presenta en el Script 12-15, en este caso obtendremos 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 lo podemos utilizar como argumento de la cláusula FROM como se muestra en el Script 12-16.

Script 12-16

```
SELECT * FROM Roles(1);
```

Finalmente, si deseamos acceder a solo un valor dentro del registro lo podemos hacer como se muestra en el Script 12-17.

Script 12-17

```
SELECT (Roles(1)).esSolista;
```

Esto lo podemos combinar con una consulta a la tabla “Musicos” 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
```

Hemos devuelto una fila; pero, ¿qué sucede si queremos devolver más de una fila, es decir, una tabla? Miremos la siguiente necesidad donde para resolverla hacemos uso de una función que devuelve una tabla.

¿Cuáles son las canciones que pertenecen a un género de nombre “X”?

Crear una consulta donde el conjunto de datos devuelto sea el que coincida con un valor específico de alguna de sus columnas, es algo que hemos creado muchas veces, por ejemplo, para saber cuáles son las canciones del género “salsa” creamos una consulta como la del Script 12-19. Esta consulta, aunque útil no satisface lo que deseamos alcanzar, deseamos que, si queremos cambiar y ahora obtener las canciones de género “Vallenato” no tengamos que escribir nuevamente la consulta, una función puede ayudarnos a cumplir con este objetivo.

Script 12-19

```
SELECT *
FROM Canciones
WHERE idGénero = (SELECT idGénero FROM Géneros WHERE nombre = 'salsa');
```

La función que necesitamos para cumplir con lo necesario sería una que reciba como argumento de entrada el nombre del género que se desea buscar y que nos devuelva un conjunto de datos o filas (SETOF) de la tabla “Canciones” como presentamos en el Script 12-20. Teniendo claro la declaración de la función, en la implementación solo tenemos que retornar el resultado de la consulta (RETURN QUERY) del Script 12-19, utilizando en lugar del valor específico del género “Salsa” el valor 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 podemos invocarla con diferentes valores del argumento “género”, este el caso del Script 12-21 que lo hace con el género “Vallenato” y el Script 12-22 que lo hace con el género “Salsa”

Script 12-21

```
SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato');
```

Script 12-22

```
SELECT * FROM CancionesPorGéneroEIntérprete('Salsa');
```

En este escenario hemos devuelto un conjunto de datos que tienen la misma estructura de la tabla “Canciones”, sin embargo, puede que el conjunto de datos que necesitamos devolver no tenga la estructura de ninguna de las tablas de nuestra base de datos, analicemos el siguiente escenario.

¿Cuál es el título de la canción y del álbum de aquellas canciones que pertenecen a un género de nombre “X”?

En esta necesidad se nos especifica que el conjunto de datos devuelto por la función tenga una estructura particular que no se ajusta a ninguna de las tablas que tenemos definidas en nuestra base de datos, por esto utilizaremos un conjunto de filas genéricas (SETOF RECORD); el otro cambio necesario fue 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 sería la creación de 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 hemos utilizado el objeto genérico RECORD por lo que al momento de invocarla debemos darle nombre y tipo a las columnas que deseamos obtener como se muestra en el Script 12-24.

Script 12-24

```

SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato') f(título
VARCHAR(100), álbum VARCHAR(100));

```

Realizar la invocación de la función de la forma anterior resultaría bastante tedioso, por lo que existen alternativas, como especificar en la definición cuál es la estructura de las filas como está en el Script 12-25. En este hemos definido como argumentos de salida el “título” y el “álbum”, ambos de tipo VARCHAR, para que den forma al conjunto de datos que vamos a obtener. Por lo demás, la implementación de la función no varía con respecto a lo que mostramos en el ejemplo anterior.

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, esto lo podemos ver en el Script 12-26.

Script 12-26

```
SELECT * FROM CancionesPorGéneroEIntérprete('Vallenato')
```

Para finalizar con las funciones revisemos 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.

La necesidad planteada nos pide explícitamente que el conjunto de datos devuelto varía de acuerdo con tres valores: un valor del sexo y un rango que son proporcionados cuando se invoca la función, razón por la cual en la declaración de la función se plantearon tres argumentos de entrada (IN): el sexo y el límite inferior y el límite superior del rango. Como se presenta en el Script 12-27, sin embargo, este script tiene nuevos elementos que detallaremos a continuación:

- En la declaración de los argumentos hemos utilizado la palabra DEFAULT para indicar que utilizaremos un valor por defecto en caso de que ningún valor se proporcione. Para el caso del argumento “sexoBuscado” el valor por defecto es “F”.
- Hemos definido que se debe retornar una tabla que tiene una estructura definida por cinco columnas que son “idUsuario”, “totalReproducciones”, “reproduccionesMañana”, “reproduccionesTarde” y “reproduccionesNoche”. Todas las columnas de tipo entero.
- También se incluyó la definición de variables que representa la fila de una de las tablas que tenemos en nuestra base de datos. Lo hemos hecho para una fila de la tabla “Usuarios” y una fila de la tabla “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 tenemos 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, el ciclo FOR IN LOOP. El uso que se le dio a este ciclo fue el de recorrer las filas

devueltas por una consulta a la tabla “Usuarios” y una consulta a la tabla “Reproducciones”. En cada ciclo se utilizan las filas que fueron previamente definidas con la expresión <Nombre de Tabla>%rowtype para acceder a cada fila devuelta por la consulta. El cuerpo del ciclo estará siempre delimitado entre LOOP y END LOOP.

- Con la expresión RETURN NEXT añadimos las filas a la tabla que definimos como conjunto de datos resultante. Es por esto que podemos asignar valores directamente a las columnas que definimos en la tabla y una vez se ejecute la expresión RETURN NEXT se añadirá una fila con los valores que tengan las variables que representan las columnas en ese momento.
- Con la expresión RETURN de devuelve la tabla con todas las filas que han sido añadidas y 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 concluimos nuestro acercamiento a las funciones recordando que existen otras estructuras de control y repetición que pueden ser utilizadas para crear subprogramas que faciliten nuestras tareas cuando se combinan con nuestra 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;
END LOOP;
END;

```

```

-- 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';

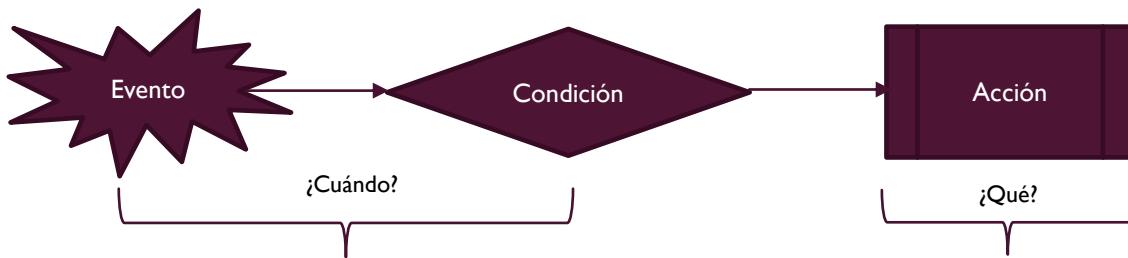
```

12.3 Disparadores

En la sección anterior hemos abordado el uso de las funciones. En esta sección nos aprovecharemos de los conceptos presentados sobre estas para gestionar eventos que se puedan dar dentro de una base de datos. Hay diferentes tipos de eventos que afectan a los objetos de una base de datos. Nosotros nos enfocaremos en los eventos del sublenguaje DML que afectan a las tablas, a saber, INSERT, UPDATE y DELETE.

Cuando este tipo de eventos ocurren es normal que se necesite gestionarlos para, por ejemplo, realizar una acción asociada a un evento, comprobar ciertas reglas antes de permitir que el evento realice cambios, entre otras. Una forma que tienen las bases de datos relacionales para gestionar los eventos son los disparadores (*Triggers*, por el término en inglés). En la práctica se suele utilizar los disparadores para hacer cumplir algunas restricciones de integridad referencial, para hacer cumplir restricciones complejas o para auditar cambios en los datos. Los disparadores están basados en el modelo Evento – Condición – Acción presentado en la Figura 12-2, donde el evento y las condiciones representan qué evento desencadenará la ejecución del disparador y la acción representa cuáles son las acciones que realizará el disparador una vez se ha activado.

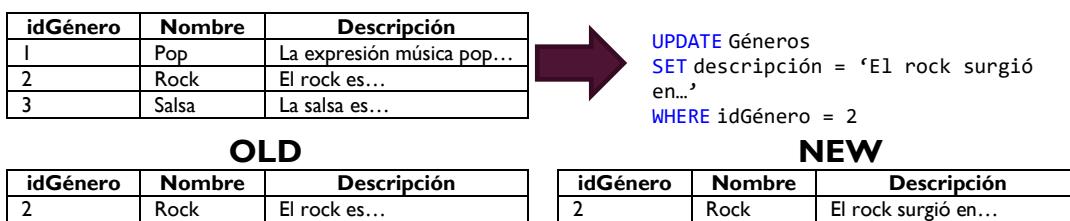
Figura 12-2



Básicamente hay dos tipos de disparadores: 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 disparadores 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 realiza una acción sobre una tabla que tiene el efecto de provocar que se dispare otro evento que tiene un disparador asociado.

En el contexto de ejecución de un disparador disponemos de dos variables que sus valores varían en función del evento desencadenante, estas son las variables OLD y NEW. Ambas representan el estado de la fila en la tabla antes o después del evento, los valores OLD y NEW son punteros que representan la fila completa. Para comprender mejor esto, considere una operación UPDATE; en este caso, la variable OLD contiene el valor de la fila con los valores ya presentes en la tabla, mientras que la variable NEW contiene los nuevos valores que tendrá la fila de la tabla después de la operación UPDATE. En la Figura 12-3 se muestra este ejemplo.

Figura 12-3



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, pero la variable OLD no tiene sentido que esté presente. En la Tabla 12-1 se muestra la relación entre los eventos y las dos variables.

Tabla 12-1

Evento	NEW	OLD
E.A. Galvis-Lista - A.A. Bustamante-Martínez		

INSERT	Presente	Ausente
UPDATE	Presente	Presente
DELETE	Ausente	Presente

Abordemos la creación de nuestro primer disparador en la siguiente situación.

¿Cómo podemos controlar que la cantidad máxima de reproducciones sean 100 por usuario?

En el contexto que nos pone la situación planteada es que debemos controlar que un usuario no puede tener más de 100 reproducciones o puesto en términos de filas y tablas, que en la tabla “Reproducciones” no puede haber más de 100 filas asociadas al mismo identificador del usuario. Esta es una situación que podemos manejar con los disparadores, podemos plantear 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. 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 la fila número 100 para el mismo usuario es ejercido por la función “f_controlReproducciones”.

Script 12-28

```
CREATE TRIGGER t_controlReproducciones
BEFORE INSERT on Reproducciones
FOR EACH ROW
EXECUTE PROCEDURE f_controlReproducciones();
```

El control ejercido por la función es bastante sencillo, lo primero que se hace como vemos en el Script 12-29 es determinar la cantidad de filas asociadas usuario que desea realizar la inserción en la tabla “Reproducciones” antes de realizar la nueva inserción, ese valor es guardado en la variable “cantidad” de tipo entero. Luego, se comprueba con la estructura de control IF/ELSE si el valor es menor que 100, de ser verdadero se indica que se puede continuar con la operación de inserción (RETURN NEW), en caso contrario se cancela la operación de inserción (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 realicemos 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 a insertar representa el número 100 o inferior, se guardará con éxito, de representar un número mayor que 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);

```

Abordemos ahora la siguiente necesidad.

¿Cómo podemos controlar que cuándo un usuario actualice su contraseña, la nueva contraseña no sea igual a la anterior?

Como comentamos con anterioridad cuando se va a actualizar una fila (UPDATE) tenemos disponibles las dos variables, una que referencia a los valores antes (OLD) del evento de actualización y otra a los valores después (AFTER) de la actualización. Recordemos también que ambas variables representan una fila de la tabla que está siendo afectada, en este caso la tabla “Usuarios”. Por lo que con ambas variables tenemos acceso a los valores que toman las columnas para esa fila en particular, es así como “OLD.contraseña” almacena el valor de la contraseña antes de ser actualizado y “NEW.contraseña” almacena el nuevo valor que se quiere dar a la contraseña. Por consiguiente, lo que tenemos que hacer es comprobar la igualdad de estos dos valores, si el son iguales no realizar la actualización y si son diferentes proseguir con ella. 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 intentemos realizar una actualización como la del Script 12-33 se ejecutará el disparador asociado a ella y por consiguiente la función que comprueba que el nuevo valor de la contraseña no puede ser igual al anterior.

Script 12-33

```
UPDATE Usuarios
SET contraseña = 'new password'
WHERE idUsuario = 1;
```

Culminemos el uso de los disparadores con la siguiente necesidad, en un contexto donde la eliminación en cascada no está habilitada.

¿Cómo podemos eliminar todas las filas asociadas a un usuario al momento de su eliminación?

En el capítulo siete mostramos que en algunos escenarios cuando queremos eliminar un registro de una tabla y hay registros en otras tablas que hacen referencia a este registro el DBMS arroja un mensaje que nos indica que antes de eliminar esa fila, debemos eliminar todas las filas de las otras tablas que hacen referencia a la que inicialmente queríamos eliminar. También vimos que hay dos formas de hacer esto, una eliminando las filas que hacen referencia a la que deseamos eliminar tabla por tabla; o utilizar la palabra reservada CASCADE para que haga estas eliminaciones por nosotros. 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 del uso de disparadores como mostraremos a continuación.

En el Script 12-34 creamos 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 vemos en el Script 12-35, lo que hace es eliminar todas las filas de las tablas “CancionesLista”, “ListasReproducción”, “Reproducciones”, “Calificaciones” y “Pagos” y una vez estas filas han sido borradas con RETURN OLD se indica que se puede continuar con la eliminación de la fila de la tabla “Usuario” que se está intentando eliminar. Es clave destacar que los disparadores operan integrados como una transacción al evento que los dispara, 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';
```

Lista la creación del disparador cada vez que intentemos eliminar una fila de la tabla “Usuarios” con una sentencia como la presentada en el Script 12-36 se eliminarán todas las filas que hacen referencia a esta fila en particular.

Script 12-36

```
DELETE FROM Usuarios WHERE idUsuario = 2;
```

Si en algún momento queremos eliminar un disparador lo podemos hacer con las palabras reservadas DROP TRIGGER, especificando en que tabla está el disparador que se desea eliminar como mostramos 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 planteamos que los subprogramas son de dos tipos funciones y procedimientos. Al inicio del capítulo nos centramos en las funciones, 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 si pasa con las funciones. 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 devolverá una única tupla con el valor final de los valores 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. Miremos un primer escenario en que los procedimientos pueden ser útiles.

¿Cómo podemos insertar un nuevo intérprete y crear el género al cual está asociado?

En esta necesidad se nos está solicitando que se realicen dos inserciones en una misma llamada y estas son:

- Que se inserte el nuevo género que se desea crear en la tabla “Géneros”.
- Luego, que se inserta el 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 que necesitamos 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 cada caso lo requiera.

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 'Intérprete % insertado', nombre;

END
$CODE$
LANGUAGE 'plpgsql';
```

Un procedimiento se invoca mediante la instrucción CALL. Los procedimientos no se pueden utilizar en instrucciones SQL regulares como SELECT, INSERT, UPDATE o DELETE. De hecho, si se invoca un procedimiento a través de una de las declaraciones anteriores, PostgreSQL abortará la ejecución e informará un error que especifica que debemos usar la declaración CALL. En el Script 12-39 se muestra cómo invocar el procedimiento “insertarIntérprete”.

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 el contenido del procedimiento o se deshacen los cambios que se haya podido realizar.

Hemos repetido más de una vez el concepto de transacciones a lo largo del libro, pero ya es momento de formalizar a qué nos referimos cuando hablamos de transacciones en el contexto de las bases de datos relacionales. El modelo relacional describe la unidad lógica de procesamiento de datos como la transacción; las transacciones se pueden definir como un conjunto de operaciones realizadas en secuencia y que operan como un mecanismo de bloqueo para garantizar la integridad de las transacciones.

Una transacción se ejecuta con éxito si todas las operaciones incluidas en la transacción se ejecutan correctamente. Si una operación en una transacción falla, el efecto de la operación ejecutada parcialmente en la transacción se puede deshacer. En la Figura 12-4 se muestra de modo gráfico una transacción compuesta por cuatro operaciones, a saber, dos INSERT, un DELETE y un UPDATE.

Figura 12-4



Otra diferencia importante entre los procedimientos y las funciones es que las funciones no pueden interactuar con el nivel de la transacción en la que se está ejecutando (por ejemplo, para guardar valores parcialmente) mientras que los procedimientos almacenados si pueden interactuar con la transacción en la que se están ejecutando y realizar cambios en la base de datos, aunque no se complete con éxito la ejecución de todo el procedimiento. Miremos cómo funciona asumiendo la siguiente necesidad.

¿Cómo podemos insertar un nuevo intérprete y crear el género al cual está asociado?

Pero, si falla la inserción del intérprete queremos que no se deshaga la inserción del género.

Interactuar con la transacción dentro de los procedimientos almacenados lo podemos hacer, por ejemplo, con la palabra COMMIT. Con esta palabra le damos al DBMS la instrucción de guardar todos los cambios que el procedimiento haya realizado hasta ese punto. Puesto en práctica lo anterior, sería como se muestra en el Script 12-40. En este se aprecia cómo guardamos los cambios una vez el género ha sido insertado, de forma que, si la inserción del intérprete falla, el cambio realizado en la tabla “Género” perdurará, no como en el 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 'Intérprete % 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 deseamos que se deshagan los cambios que se han realizado durante la ejecución del procedimiento, esto lo conseguimos con la palabra reservada ROLLBACK.

El Script 12-40 lo podemos modificar para deshacer de forma voluntaria los cambios cuando intérprete que deseamos insertar ya existe. Entonces el Script 12-41 realiza las siguientes interacciones con la transacción.

- Guarda los cambios si luego del género se comprueba que el intérprete que se está intentando insertar no existe aún en la tabla “Intérpretes”.
- Hacemos un ROLLBACK si luego de insertar el género, se comprueba que el intérprete que se desea insertar ya existe en la tabla “Intérpretes”, es decir, si el intérprete que se desea insertar 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 realizar 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 'Intérprete % 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 podemos realizar en nuestras bases de datos.

- Las vistas permiten definir una capa de abstracción sobre nuestros datos y son especialmente útiles cuando se busca uno de los siguientes atributos: modularidad, reuso y seguridad.
- Aunque el concepto de vistas implica de manera general el no almacenamiento de los datos en la vista, la evolución de los DBMS y 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 procedural podemos crear subprogramas que a su vez nos 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 tipos asocian un nombre a un conjunto de instrucciones que se ejecutarán cada vez que se invoque el subprograma. De igual forma ambos reciben un conjunto de argumentos que utilizan recibir datos que le permitan, 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 devolver valores 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 exitosa o se deshagan los cambios que hayan podido realizar.
- En los subprogramas podemos utilizar los diferentes constructos a los que estamos acostumbrados en los lenguajes imperativos, como las estructuras de control y las repetitiva.
- Con los disparadores podemos controlar eventos que afectan a las tablas de nuestra base de datos, de forma se pueda reaccionar ante acciones de inserción, modificación y eliminación.

12.6 Actividades de aplicación para evidenciar lo aprendido

- I. Proponer la creación de dos vistas que no sean almacenadas físicamente y dos que sí sean almacenadas físicamente.

2. Proponer tres funciones que pueden ser utilizadas para resolver necesidades de datos que puedan surgir en el contexto de la plataforma “Mis Canciones”.
3. Crear una tabla de nombre “Auditoria”, la tabla debe tener una estructura como la de la Tabla 12-2. En esta tabla 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?

Tabla 12-2

Tabla	Acción	Valores	Fecha	Hora