INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES Y A SQL.

El modelo relacional.

El núcleo del modelo relacional es la relación. Un relación es un conjunto de columnas y filas reunidas en una estructura en forma de tabla que representa una entidad única formada por los datos relacionados.

Una entidad es una persona, lugar, cosa, evento o concepto sobre el cual los datos son recolectados. Cada relación comprende uno o más atributos (columnas). Los datos se almacenan en una relación (tabla) en tuplas (filas). Una tupla es un conjunto de datos cuyos valores hacen una instancia de cada atributo definido por esa relación. Cada tupla representa un registro de datos relacionados.

Los términos lógicos relación, atributo y tupla se usan principalmente para referirse al modelo relacional. SQL usa los términos físicos tabla, columna y fila para describir estos elementos.

Dos consideraciones importantes en el diseño e implementación de cualquier base de datos relacional son la normalización de los datos y la asociaciones de relaciones entre los distintos tipos de datos.

Normalización de datos.

La normalización es una técnica para producir un conjunto de relaciones que poseen un conjunto de ciertas propiedades que minimizan los datos redundantes y preservan la integridad de los datos almacenados. Se define por un conjunto de normas, que se conocen como formas normales, que proporcionan una directriz específica de cómo los datos son organizados para evitar anomalías que den lugar a inconsistencias y pérdida de los datos. Aunque, con el paso del tiempo se han agregado formas normales, las iniciales eran tres, y estas tres cubren los requerimientos de las bases de datos personales y empresariales. Solo analizaremos las tres primeras formas normales.

Elección de un identificador único.

Un identificador único es un atributo o conjunto de atributos que únicamente identifican cada fila de datos en una relación. El identificador único eventualmente se convertirá en la **clave principal** de la tabla creada en la base de datos física desde la relación de normalización, pero muchos usan los términos identificador único y clave principal de manera intercambiable. A cada identificador potencial único se le denomina candidato clave, y cuando hay varios candidatos, el diseñador elegirá el mejor, el cual es el menos probable de cambiar valores o el más simple y/o el más corto. En muchos casos, un solo atributo se identifica únicamente en los datos en cada tupla de la relación. Sin embargo, cuando ningún atributo que es único se encuentra, el diseñador busca varios atributos que puedan ser unidos (puestos juntos) para formar un identificador único. En los pocos casos donde ningún candidato clave razonable es encontrado, el diseñador debe inventar un identificador único denominado sustituto clave, frecuentemente con valores asignados al azar o secuencialmente cuando las tuplas sean agregadas a la relación.

Mientras no sea absolutamente necesario hasta la segunda forma normal, es habitual seleccionar un identificador único como primer paso en la normalización. Es más fácil de esa manera.

Primera forma normal 1FN.

La primera forma normal, que proporciona la fundación par ala segunda y la tercera forma, incluye las siguientes directrices:

- Cada atributo de una tupla contiene solo un valor.
- Cada tupla en una relación contiene el mismo número de atributos.
- Cada tupla es diferente, lo que significa que la combinación de los valores de todos los atributos de una tupla dada no puede ser como ninguna otra tupla en la misma relación.

Ejemplo de relación (tabla) que no cumple la 1FN.

NOMBRE_ARTISTA	NOMBRE_CD	AÑO_DERECHOSDEAUTOR
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue; Court and Spark	1971; 1974
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988
Jose Carreras; Placido Domingo; Luciano Pavarotti	Carreras Domingo Pavarotti in Concert	1990

Ejemplo de relaciones (tablas) que cumplen la 1FN.

Se normaliza la relación anterior atendiendo a la 1FN. El resultado es el siguiente:

ID_ARTISTA	NOMBRE_ARTISTA
10001	Jennifer Warnes
10002	Joni Mitchell
10003	William Ackerman
10004	Kitaro
10005	Bing Crosby
10006	Patsy Cline
10007	Jose Carreras
10008	Placido Domingo
10009	Luciano Pavarotti

ID_ARTISTA	ID_CD
10001	99301
10002	99302
10002	99303
10003	99304
10004	99305
10005	99306
10006	99307
10007	99308
10008	99308
10009	99308

ID_CD	NOMBRE_CD	AÑO_DERECHOSDEAUTOR
99301	Famous Blue Raincoat	1991
99302	Blue	1971
99303	Court and Spark	1974
99304	Past Light	1983
99305	Kojiki	1990
99306	That Christmas Feeling	1993
99307	Patsy Cline: 12 Greatest Hits	1988
99308	Carreras Domingo Pavarotti in Concert	1990

Fue necesario agregar los atributos `id_artista` e `id_cd`, se hizo esto porque no había otros candidatos a clave principal.

Segunda forma normal 2FN.

Para comprender la segunda forma normal, hemos de entender primero el concepto de dependencia funcional. Un atributo (B) es funcionalmente dependiente de otro atributo (A), si en cualquier momento no hay más que un valor del atributo B asociado con el valor dado del atributo A, es decir, A es un identificador único del atributo B.

La 2FN expone que una relación debe estar en la primera forma normal y que todos los atributos en la relación dependen del identificador único completo.

Tercera forma normal 3FN.

Al igual que la segunda forma normal, la tercera forma normal, depende de la relación del identificador único. Para cumplir las directrices de la tercera forma normal, una relación debe estar en la segunda forma normal y no debe haber

dependencias transitivas; esto significa que no debe existir una dependencia entre los atributos no clave de la tabla. Si un atributo no clave depende de otro atributo no clave, entonces ese atributo debe ser eliminado y trasladado a una nueva tabla.

IMPORTANTE: En el mundo teórico del diseño relacional, el objetivo es almacenar los datos de acuerdo con la reglas de normalización. Sin embargo, en el mundo real de aplicación de bases de datos, ocasionalmente se desnormalizan los datos, lo que significa que se violen deliberadamente las reglas de normalización, particularmente la segunda y la tercera forma normal. La desnormalización se usa principalmente para mejorar el rendimiento o reducir la complejidad en los casos en donde una estructura demasiado normalizada complica la implementación. Con todo, el objetivo de la normalización es asegurar la integridad de los datos; por lo tanto, la desnormalización se realizaría con sumo cuidado y como último recurso.

Relaciones.

Hasta ahora hemos explicado la relación (tabla) y la manera de normalizar los datos. Sin embargo, un componente importante de cualquier base de datos relacional es de qué forma esas relaciones se asocian entre sí. Esas asociaciones, o relaciones, se vinculan en forma significativa, lo que contribuye a garantizar la integridad de los datos de modo que una acción realizada en una relación no repercuta negativamente en los datos de otra relación.

Hay tres tipos principales de relaciones:

- Una a una: Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con al menos una tupla en la segunda relación, y una tupla en la segunda relación esté relacionada con al menos una tupla en la primera relación.

- Una a varias: Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con ninguna, una o más tuplas en la segunda relación, pero una tupla en la segunda relación esté relacionada con al menos una tupla en la primera relación.

- Varias a varias: Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con ninguna, una o más tuplas en la segunda relación, y una tupla en la segunda relación esté relacionada con ninguna, una o más tuplas en la primera relación.

IMPORTANTE: Las bases de datos relacionales sólo apoyan una relación una a varias directamente. Una relación varias a varias se implementa físicamente agregando una tercera relación entre la primera y la segunda para crear dos relaciones una a varias.

ID_ARTISTA	AGENCIA
10001	2305
10002	2306
10003	2306
10004	2345
10005	2367
10006	2049

ID_ARTISTA	NOMBRE_ARTISTA
10001	Jennifer Warnes
10002	Joni Mitchell
10003	William Ackerman
10004	Kitaro
10005	Bing Crosby
10006	Patsy Cline

ID_ARTISTA	ID_CD
10001	99301
10002	99302
10002	99303
10003	99304
10004	99305
10005	99306
10006	99307

ID_CD	NOMBRE_CD
99301	Famous Blue Raincoat
99302	Blue
99303	Court and Spark
99304	Past Light
99305	Kojiki
99306	That Christmas Feeling
99307	Patsy Cline: 12 Greatest Hits

Nomenclatura.

Tabla

Objeto de almacenamiento perteneciente a una base de datos. Es una estructura en forma de cuadrante donde se almacenan registros o filas de datos. Es un conjunto de filas. Cada tabla tiene un nombre único en la BD.

Campo

Cada uno de las columnas de un registro, donde se guardan los datos. Cada campo tiene un nombre único par ala tabla de la cual forma parte. Además, es de un tipo determinado.

Entidad

Algo de interés para el entorno del usuario de la base de datos. Los ejemplos incluyen clientes, localizaciones geográficas, etc.

Columna

Una parte de los datos individuales almacena en una tabla.

Fila

Un conjunto de columnas que justas describen totalmente una entidad o alguna acción que se realice sobre ella. También se denomina registro.

Conjunto de resultados

Nombre que define a una tabla temporal, normalmente el resultado de una consulta SQL.

Clave primaria

Una o más columnas que se pueden utilizar como un identificador único para cada fila de una tabla.

Clave ajena (externa)

Una o más columnas que se pueden utilizar juntas para identificar un afila única en otra tabla.

Clases de sentencias SQL.

El lenguaje SQL está divido en partes diferentes:

1. Sentencias SQL de manipulación del esquema de base de datos. Se utilizan para definir las estructuras de los datos almacenados en la base de datos.

2. Sentencias SQL de datos. Se utilizan para manipular las estructuras de los datos previamente definidos utilizando la primera.

3. Sentencias SQL de transacciones. Se utilizan para iniciar, finalizar y deshacer transacciones.

Aunque SQL se considera un sublenguaje debido a su naturaleza de no procesamiento (no es un lenguaje procedimental), aun así es un lenguaje completo que le permite crear y mantener objetos en una base de datos, asegurar esos objetos y manipular la información dentro de los objetos. Un método común usado

para categorizar las instrucciones SQL es dividirlas de acuerdo con las funciones que realizan. Basado en este método, SQL se separa en tres tipos de instrucciones:

Lenguaje de definición de datos (DDL, Data Definition Language): Las instrucciones DDL se usan para crear, modificar o borrar objetos en una base de datos como tablas, vistas, esquemas, dominios, activadores, y almacenar procedimientos. Las palabras clave en SQL más frecuentemente asociadas con las instrucciones DDL son CREATE, ALTER y DROP. Por ejemplo, se usa la instrucción CREATE TABLE para crear una tabla, la instrucción ALTER TABLE para modificar las características de una tabla, y la instrucción DROP TABLE para borrar la definición de la tabla de la base de datos.

Lenguaje de control de datos (DCL, Data Control Language): Las instrucciones DCL permiten controlar quién o qué (un usuario en una base de datos puede ser una persona o un programa de aplicación) tiene acceso a objetos específicos en la base de datos. Con DCL, puede otorgar o restringir el acceso usando las instrucciones GRANT o REVOKE, los dos comandos principales en DCL. Las instrucciones DCL también permiten controlar el tipo de acceso que cada usuario tiene a los objetos de una base de datos. Por ejemplo, puede determinar cuáles usuarios pueden ver un conjunto de datos específico y cuáles usuarios pueden manipular esos datos.

Lenguaje de manipulación de datos (DML, Data Manipulation Language): Las instrucciones DML se usan para recuperar, agregar, modificar o borrar datos almacenados en los objetos de una base de datos. Las palabras clave asociadas con las instrucciones DML son SELECT, INSERT, UPDATE y DELETE, las cuales representan los tipos de instrucciones que probablemente son más usadas. Por ejemplo, puede usar la instrucción SELECT para recuperar datos de una tabla y la instrucción INSERT para agregar datos a una tabla.

Ejemplo de sentencia de manipulación del esquema de base de datos:

```
CREATE TABLE corporation
      (corp_id SMALLINT,
      name VARCHAR(30),
      CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
); /* Crea una tabla. */
```

Ejemplos de sentencias de manipulación de datos:

```
INSERT INTO corporation (corp_id, name)
VALUES (27, 'Acme Paper Corporation'); /* Inserta una fila. */


SELECT name FROM corporation WHERE corp_id=27; /* Recupera datos de la tabla. */
```

Nota: La mayoría de las implementaciones de SQL tratan como comentario cualquier texto que figure entre las etiquetas /* y */.

SQL no es un lenguaje al uso procedimental; no será capaz de escribir aplicaciones completas. Es necesario integrar SQL con otro lenguaje de programación. Algunos proveedores de bases de datos ya hacen esto por usted, por ejemplo Oracle con su lenguaje PL/SQL o Microsoft con su lenguaje TransactSQL. Con estos lenguajes, las sentencias SQL de manipulación de datos forman parte del lenguaje, permitiéndole integrar directamente consultas de bases de datos con comandos de procedimientos. Sin embargo, si utiliza Java o C, por ejemplo, lenguajes no específicos de bases de datos, necesitará utilizar un conjunto de herramientas o utilidades para ejecutar las sentencias SQL de su código. Un par de ejemplos de conjuntos de herramientas de integración de SQL son JDBC, para Java o Pro*C para C.

CREAR E INTRODUCIR INFORMACIÓN EN UNA BASE DE DATOS.

Crearemos una base de datos sobre MySQL que nos servirá de referencia en lo que sigue. Para ello ingresaremos en nuestra consola de MySQL con el siguiente comando:

```
mysql -u root -p
```

Creamos la base de datos:

```
MariaDB [(none)]> CREATE DATABASE bank;
```

Le otorgamos todos los privilegios al usuario `user` para la base de datos `bank` que acabamos de crear.

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON bank.* TO 'user'@'%';
```

Por último, actualizamos los privilegios:

```
MariaDB [(none)]> FLUSH PRIVILEGES;
```

Podemos ahora ingresar con el usuario user a nuestra base de datos y mostrar las bases de datos que puede administrar con el comando `SHOW DATABASES`:

```
mysql -h host -u user -p
```

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| bank               |
| information_schema |
+--------------------+
```

El esquema de la base de datos se encuentra en el archivo `LearningSQLExample.sql`. Volvemos a la línea de comandos del sistema para cargar el esquema de la base de datos bank con el siguiente comando:

```
mysql -h host -u user -p bank < LearningSQLExample.sql
```

y comprobamos que hemos importado correctamente el esquema de la base de datos:

```
mysql -h host -u user -p
```

```
mysql> USE bank;
Database changed

mysql> SHOW TABLES;
+----------------+
| Tables_in_bank |
+----------------+
| account        |
| branch         |
| business       |
| customer       |
| officer        |
| product        |
| product_type   |
| transaction    |
| ...            |
+----------------+
```

Tipos de datos MySQL.

Datos de tipo carácter.

Los datos de tipo carácter pueden almacenarse como cadenas de caracteres de longitud fija o variable. La diferencia es que en las de longitud fija se introducen espacios a la derecha, mientras que en las de longitud variable no.

```
CHAR(20)    /* longitud fija */
VARCHAR(20) /* longitud variable */
```

Al definir una columna de tipo carácter, se debe especificar el tamaño máximo de cualquier cadena que vaya a almacenar (20). En términos generales se debe utilizar el tipo `CHAR` cuando las cadenas a almacenar en la columna sean de la misma longitud, como por ejemplo, las abreviaturas de los nombres de las ciudades y el tipo `VARCHAR` cuando las cadenas contengan una longitud variable.

Nota: Oracle Database es una excepción en cuanto al uso del tipo `VARCHAR`. Los usuarios de Oracle deben utilizar el tipo `VARCHAR2` cuando definan columnas de cadenas de caracteres de longitud variable.

Datos de tipo texto.

Si necesita almacenar datos que excedan el límite de CHAR y VARCHAR (65.535 bytes), necesitará usar uno de los tipo de texto existentes:

```
TINYTEXT
TEXT
MEDIUMTEXT
LONGTEXT
```

Nota: Los diferentes tipos de datos de tipo texto son únicos en MySQL. Oracle utiliza un tipo de datos denominado `CLOB` (Character Large Object).

Datos de tipo numérico.

Existen varios tipos de datos numéricos que reflejan las diferentes maneras en que se utilizarán los números, tal y como se indica a continuación:

- Una columna indicando si una orden de cliente se ha enviado. Este tipo de columna se denomina Boolean y contiene un 0 (FALSE) o un 1 (TRUE).
- Una clave principal o primaria generada por el sistema para una tabla de transacciones[*]: Por lo general estos datos comienzan por 1 y van aumentando de uno en uno hasta valores muy altos.
- Información de precisión y otros datos científicos.
- ...

Para gestionar este tipo de datos y más los tipos disponibles son los siguientes:

```
TINYINT
SAMLLINT
MEDIUMINT
INT
BIGINT
FLOAT
```

DOUBLE

En las nuevas versiones de MySQL existen más tipos de datos, por ejemplo: BIT, BOLL o BOOLEAN, INTEGER, DECIMAL o DEC, DOUBLE PRECISION pero responden a la misma lógica.

Tanto los números enteros como los números decimales se pueden definir como UNSIGNED (sin signo).

[*]: Una tabla de transacciones se utiliza para almacenar información sobre transacciones que afectan a una base de datos. Una transacción representa una unidad lógica de trabajo que consiste en una o más operaciones de base de datos, como inserciones, actualizaciones o eliminaciones.

La tabla de transacciones puede contener los siguientes campos o columnas:

1. ID de transacción: Un identificador único para cada transacción. Puede ser un número autoincremental o generado de manera única.
2. Fecha y hora: La fecha y hora en que se realizó la transacción.
3. Usuario: El nombre de usuario o identificador del usuario que realizó la transacción.
4. Tipo de transacción: Puede indicar si la transacción fue una inserción, actualización, eliminación u otra operación específica.
5. Tabla afectada: El nombre de la tabla en la que se realizó la transacción.
6. Detalles: Información adicional sobre la transacción, como los valores antiguos y nuevos en el caso de una actualización, o los registros eliminados en el caso de una eliminación.

La tabla de transacciones es útil para llevar un registro de todas las modificaciones realizadas en la base de datos, lo que permite realizar auditorías, realizar seguimiento de cambios y revertir transacciones si es necesario. También puede utilizarse para generar informes y estadísticas relacionadas con las transacciones en la base de datos.

Datos de tipo indicación temporal.

Para trabajar con información sobre fecha y hora se utilizan los datos de indicación temporal (temporal), son los siguientes:

```
DATE        YYYY-MM-DD
DATETIME    YYYY-MM-DD HH-MI-SS
TIMESTAMP   YYYY-MM-DD HH-MI-SS
YEAR        YYYY
TIME        HH:MI:SS
```

Crear tablas.

Conociendo los tipos de datos disponibles (pueden variar en función de la versión MySQL u otro SGBD) es hora de ver cómo utilizar estos tipos de datos en las definiciones de las tablas. Vamos a empezar por definir una tabla que contenga información sobre una persona.

[1] Diseño.

Supongamos en una primera aproximación que para guardar datos sobre una persona (tabla `person`) necesitamos los siguientes: nombre, género, fecha de nacimiento, dirección, comidas favoritas.

Lo primero es asignar nombres de columnas y tipos de datos:

Columna	Tipo	Valores permitidos
name	VARCHAR(40)	
gender	CHAR(1)	M, F
birth_date	DATE	
address	VARCHAR(100)	
favorite_foods	VARCHAR(200)	

En el capítulo anterior se trato la normalización, que es el proceso para asegurar que no existen duplicados(aparte de las claves ajenas) o columnas compuestas, en el diseño de una base de datos.

[2] Refinamiento.

Normalicemos la tabla anterior según las siguientes conclusiones:

- `name` es en realidad un objeto compuesto, se compone de nombre y apellido.
- No existen columnas en la tabla person que garanticen la unicidad.
- La columna `favorite_foods` es una lista que contiene uno o más elementos independientes, sería mejor crear una tabla separa para estos datos. Esta tabla incluirá una clave ajena de la tabla `person` de forma que se conozca a qué persona se tiene que asignar una determinada comida.

Columna	Tipo	Valores permitidos
person_id	SMALLINT UNSIGNED	
first_name	VARCHAR(20)	
last_name	VARCHAR(20)	
gender	CHAR(1)	M, F
birth_date	DATE	
street	VARCHAR(30)	
city	VARCHAR(20)	
state	VARCHAR(20)	
country	VARCHAR(20)	
postal_code	VARCHAR(20)	

Ahora que la tabla `person` dispone de una clave primaria `person_id` para garantizar la unicidad, el siguiente paso es el de construir una tabla `favorite_food` que incluya una clave ajena a la tabla `person`.

Columna	Tipo	Valores permitidos
person_id	SMALLINT UNSIGNED	
food	VARCHAR(20)	

MUY IMPORTANTE (BASE DE LAS RELACIONES ENTRE TABLAS [1]):

Las columnas `person_id` y `food` constan de una clave primaria de la tabla `favorite_food`, y la columna `person_id` también es una clave ajena de la tabla `person`.

[3] Construir una sentencia SQL de manipulación del esquema de la base de datos.

```
CREATE TABLE person
      (person_id SMALLINT UNSIGNED,
       fname VARCHAR(20),
       lname VARCHAR(20),
       gender CHAR(1),
       birth_date DATE,
       street VARCHAR(30),
       city VARCHAR(20),
       state VARCHAR(20),
       country VARCHAR(20),
       postal_code VARCHAR(20),
       CONSTRAINT pk_person PRIMARY KEY (person_id)
      );
```

Miremos la última fila de esa sentencia. Al definir una tabla, necesita decirle al servidor de la base de datos que columnas o columnas servirán como clave primaria de la tabla. Esto se hace creando una tabla de restricciones (constraints). Existen varios tipos de restricciones que se pueden añadir a la definición de un atabla. A esta restricción se la denomina restricción de clave primaria (primary-key constraint). Se sitúa en la columna `person_id` y se le asigna el nombre `pk_person` (este nombre es solo una convención, pk_nombre_tabla para primary_key).

Existe otro tipo de restricción denominado restricción de valor (check constraint), que limita los valores permitidos de una determinada columna: MySQL permite adjuntar una restricción de comprobación en una definición de columna, tal y como puede verse a continuación:

```
      gender CHAR(1) CHECK (gender IN ('M','F')),
```

y para ello proporciona otro tipo de datos de caracteres denominado ENUM que combina la restricción de valores junto con la definición de datos:

```
      gender ENUM('M','F'),
```

Nuestra sentencia CREATE TABLE, en MySQL, quedaría entonces del modo:

```
CREATE TABLE person
      (person_id SMALLINT UNSIGNED,
       fname VARCHAR(20),
       lname VARCHAR(20),
       gender ENUM('M','F'),
       birth_date DATE,
       street VARCHAR(30),
       city VARCHAR(20),
       state VARCHAR(20),
       country VARCHAR(20),
       postal_code VARCHAR(20),
       CONSTRAINT pk_person PRIMARY KEY (person_id)
      );
```

[4] Ejecutar la sentencia SQL CREATE TABLE en la consola mysql:

```
mysql> CREATE TABLE person
    -> (person_id SMALLINT UNSIGNED,
    -> fname VARCHAR(20),
    -> lname VARCHAR(20),
    -> gender ENUM('M','F'),
    -> birth_date DATE,
```

```
    -> street VARCHAR(30),
    -> city VARCHAR(20),
    -> state VARCHAR(20),
    -> country VARCHAR(20),
    -> postal_code VARCHAR(20),
    -> CONSTRAINT pk_person PRIMARY KEY (person_id));
Query OK, 0 rows affected, 1 warning (0,19 sec)
```

Para asegurarnos de que la tabla se ha creado sin problemas podemos utilizar el comando `DESCRIBE` (o `DESC`) para consultar la definición de la tabla, como sigue:

```
mysql> DESC person;
+-------------+----------------------+------+-----+---------+-------+
| Field       | Type                 | Null | Key | Default | Extra |
+-------------+----------------------+------+-----+---------+-------+
| person_id   | smallint(5) unsigned | NO   | PRI | NULL    |       |
| fname       | varchar(20)          | YES  |     | NULL    |       |
| lname       | varchar(20)          | YES  |     | NULL    |       |
| gender      | enum('M','F')        | YES  |     | NULL    |       |
| birth_date  | date                 | YES  |     | NULL    |       |
| street      | varchar(30)          | YES  |     | NULL    |       |
| city        | varchar(20)          | YES  |     | NULL    |       |
| state       | varchar(20)          | YES  |     | NULL    |       |
| country     | varchar(20)          | YES  |     | NULL    |       |
| postal_code | varchar(20)          | YES  |     | NULL    |       |
+-------------+----------------------+------+-----+---------+-------+
10 rows in set (0,18 sec)
```

Creamos ahora la tabla favorite_food:

```
mysql> CREATE TABLE favorite_food
    -> (person_id SMALLINT UNSIGNED,
    -> food VARCHAR(20),
    -> CONSTRAINT pk_favorite_food PRIMARY KEY (person_id, food),
    -> CONSTRAINT fk_person_id FOREIGN KEY (person_id)
    -> REFERENCES person (person_id));
Query OK, 0 rows affected, 1 warning (0,19 sec)

mysql> DESCRIBE favorite_food;
+-----------+----------------------+------+-----+---------+-------+
| Field     | Type                 | Null | Key | Default | Extra |
+-----------+----------------------+------+-----+---------+-------+
| person_id | smallint(5) unsigned | NO   | PRI | NULL    |       |
| food      | varchar(20)          | NO   | PRI | NULL    |       |
+-----------+----------------------+------+-----+---------+-------+
2 rows in set (0,18 sec)
```

MUY IMPORTANTE (BASE DE LAS RELACIONES ENTRE TABLAS [2]):

Debido a que una persona puede tener más de una comida favorita, no se puede utilizar solamente la columna person_id para garantizar la unicidad en la tabla; por lo tanto esta tabla dispone de una clave primaria de dos columnas: `person_id` y `food`.

La tabla contiene otro tipo de restricción llamada restricción de clave ajena (foreign-key constrait). Esto limita los valores de la columna `person_id` de la tabla `favorite_food` para incluir únicamente los valores encontrados en la tabla `person`.

Generar datos clave numéricos.

Modifiquemos la tabla person para que su clave primaria sea autoincrementable:

MUY IMPORTANTE:

```
mysql> ALTER TABLE person
    -> MODIFY person_id SMALLINT UNSIGNED AUTO_INCREMENT;

ERROR 1833 (HY000): Cannot change column 'person_id': used in a foreign key
constraint 'fk_person_id' of table 'bank.favorite_food'
```

Recibimos un error con ese intento. Existe una referencia en person_id como clave externa de favorite_food y esto nos impide la actualización.

Eliminaremos la referencia de calve externa de person_id en favorite_food.

```
mysql> ALTER TABLE favorite_food DROP CONSTRAINT fk_person_id;
Query OK, 0 rows affected (0,22 sec)
```

Hacemos otro intento (ahora MySQL no devuelve ningún error):

```
mysql> ALTER TABLE person MODIFY person_id SMALLINT UNSIGNED AUTO_INCREMENT;
Query OK, 0 rows affected (0,22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE person;
+-------------+----------------------+------+-----+---------+----------------+
| Field       | Type                 | Null | Key | Default | Extra          |
+-------------+----------------------+------+-----+---------+----------------+
| person_id   | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| fname       | varchar(20)          | YES  |     | NULL    |                |
| lname       | varchar(20)          | YES  |     | NULL    |                |
| gender      | enum('M','F')        | YES  |     | NULL    |                |
| birth_date  | date                 | YES  |     | NULL    |                |
| street      | varchar(30)          | YES  |     | NULL    |                |
| city        | varchar(20)          | YES  |     | NULL    |                |
| state       | varchar(20)          | YES  |     | NULL    |                |
| country     | varchar(20)          | YES  |     | NULL    |                |
| postal_code | varchar(20)          | YES  |     | NULL    |                |
+-------------+----------------------+------+-----+---------+----------------+
10 rows in set (0,17 sec)

mysql> DESCRIBE favorite_food;
+-----------+----------------------+------+-----+---------+-------+
| Field     | Type                 | Null | Key | Default | Extra |
+-----------+----------------------+------+-----+---------+-------+
| person_id | smallint(5) unsigned | NO   | PRI | NULL    |       |
| food      | varchar(20)          | NO   | PRI | NULL    |       |
+-----------+----------------------+------+-----+---------+-------+
2 rows in set (0,18 sec)
```

Añadimos de nuevo la restricción de clave externa para person_id en favorite_food:

```
mysql> ALTER TABLE favorite_food ADD CONSTRAINT fk_person_id
    -> FOREIGN KEY (person_id) REFERENCES person (person_id);
Query OK, 0 rows affected (0,25 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Ahora, cuando se incluyan datos en la tabla person, hay que proporcionar un valor nulo en la columna person_id, y MySQL introducirá la información en la columna con el siguiente número disponible (por defecto, MySQL empieza con el número 1, en columnas con incremento automático).

Introducir información y modificar tablas.

Revisaremos las cuatro sentencias SQL de manipulación de datos: INSERT, UPDATE, DELETE Y SELECT.

La sentencia INSERT.

```
mysql> INSERT INTO person
    -> (person_id, fname, lname, gender, birth_date)
    -> VALUES (null, 'Alejandro', 'Donaire', 'M', '1972-05-27');
Query OK, 1 row affected (0,17 sec)

mysql> SELECT person_id, fname, lname, birth_date
    -> FROM person;
+-----------+-----------+---------+------------+
| person_id | fname     | lname   | birth_date |
+-----------+-----------+---------+------------+
|         1 | Alejandro | Donaire | 1972-05-27 |
+-----------+-----------+---------+------------+
1 row in set (0,18 sec)

mysql> SELECT person_id, fname, lname, birth_date
    -> FROM person
    -> WHERE fname = 'Alejandro';
+-----------+-----------+---------+------------+
| person_id | fname     | lname   | birth_date |
+-----------+-----------+---------+------------+
|         1 | Alejandro | Donaire | 1972-05-27 |
+-----------+-----------+---------+------------+
1 row in set (0,17 sec)

mysql> INSERT INTO favorite_food (person_id, food)
    -> VALUES (1, 'pizza');
Query OK, 1 row affected (0,20 sec)

mysql> SELECT * FROM favorite_food
    -> WHERE person_id = 1
    -> ORDER BY food;
+-----------+----------+
| person_id | food     |
+-----------+----------+
|         1 | galletas |
|         1 | nachos   |
|         1 | pizza    |
+-----------+----------+
3 rows in set (0,18 sec)
```

La cláusula ORDER BY le indica al servidor cómo clasificar u ordenar los datos devueltos por una consulta.

Introduzcamos otra fila en la tabla person para ver como actúa la columna AUTO INCREMENT:

```
mysql> INSERT INTO person (person_id, fname, lname, gender, birth_date,
    -> street, city, state, country, postal_code)
```

```
    -> VALUES (null, 'Sonia', 'Salguero', 'F', '1975-11-02',
    -> 'Pablo Neruda, 7', 'Badajoz', 'BA', 'España', '06010');
Query OK, 1 row affected (0,21 sec)

mysql> SELECT person_id, fname, lname, birth_date FROM person;
+-----------+-----------+----------+------------+
| person_id | fname     | lname    | birth_date |
+-----------+-----------+----------+------------+
|         1 | Alejandro | Donaire  | 1972-05-27 |
|         2 | Sonia     | Salguero | 1975-11-02 |
+-----------+-----------+----------+------------+
2 rows in set (0,18 sec)
```

Actualizar datos.

Incluyamos la dirección de Alejandro, no fue incluida inicialmente:

```
mysql> UPDATE person
    -> SET street = 'Doctor Lobato, 15',
    -> city = 'Cáceres',
    -> state = 'CC',
    -> country = 'España',
    -> postal_code = '08900'
    -> WHERE person_id = 1;
Query OK, 1 row affected (0,18 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Eliminar datos.

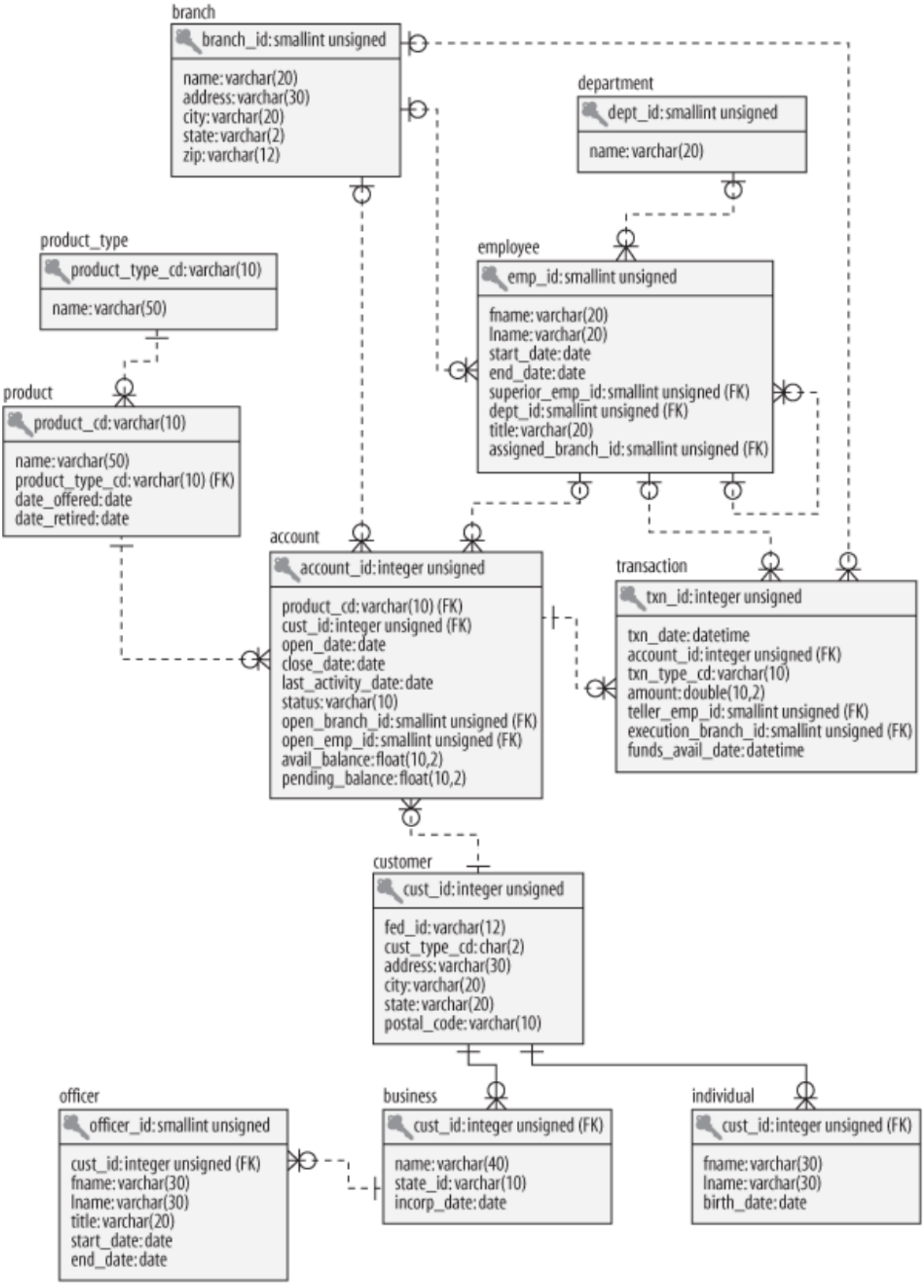
```
mysql> DELETE FROM person
    -> WHERE person_id = 2;
Query OK, 1 row affected (0,18 sec)
```

Base de datos de ejemplo.

En adelante, en estos apuntes, trabajaremos con las base de datos del archivo adjunto LearningSQLExample.sql, que hemos importado al inicio de los mismos. El esquema completo y los datos de ejemplo ya están en la base de datos si se ha importado correctamente.

Tabla	Descripción
account	Un producto específico abierto par aun cliente.
branch	Un lugar en el que se realizan transacciones.
business	Un cliente corporativo (subtipo de customer).
customer	Una persona física o jurídica conocida.
department	Un grupo de empleados del banco con una función.
employee	Una persona que trabaja en el banco.
individual	Un cliente no corporativo (subtipo de customer).
officer	Una persona autoriza a realizar operaciones.
product	Una función bancaria ofrecida a los clientes.
product_type	Un grupo de productos con una función similar.
Transaction	Una modificación realizada en el saldo de cuenta.

La siguiente imagen se muestra el esquema completo de la base de datos bank, con un diagrama de sus tablas, columnas y relaciones.


branch
branch_id: smallint unsigned
name: varchar(20)
address: varchar(30)
city: varchar(20)
state: varchar(2)
zip: varchar(12)
department
dept_id: smallint unsigned
name: varchar(20)
product_type
product_type_cd: varchar(10)
name: varchar(50)
employee
emp_id: smallint unsigned
fname: varchar(20)
lname: varchar(20)
start_date: date
end_date: date
superior_emp_id: smallint unsigned (FK)
dept_id: smallint unsigned (FK)
title: varchar(20)
assigned_branch_id: smallint unsigned (FK)
product
product_cd: varchar(10)
name: varchar(50)
product_type_cd: varchar(10) (FK)
date_offered: date
date_retired: date
account
account_id: integer unsigned
product_cd: varchar(10) (FK)
cust_id: integer unsigned (FK)
open_date: date
close_date: date
last_activity_date: date
status: varchar(10)
open_branch_id: smallint unsigned (FK)
open_emp_id: smallint unsigned (FK)
avail_balance: float(10,2)
pending_balance: float(10,2)
transaction
txn_id: integer unsigned
txn_date: datetime
account_id: integer unsigned (FK)
txn_type_cd: varchar(10)
amount: double(10,2)
teller_emp_id: smallint unsigned (FK)
execution_branch_id: smallint unsigned (FK)
funds_avail_date: datetime
customer
cust_id: integer unsigned
fed_id: varchar(12)
cust_type_cd: char(2)
address: varchar(30)
city: varchar(20)
state: varchar(20)
postal_code: varchar(10)
officer
officer_id: smallint unsigned
cust_id: integer unsigned (FK)
fname: varchar(30)
lname: varchar(30)
title: varchar(20)
start_date: date
end_date: date
business
cust_id: integer unsigned (FK)
name: varchar(40)
state_id: varchar(10)
incorp_date: date
individual
cust_id: integer unsigned (FK)
fname: varchar(30)
lname: varchar(30)
birth_date: date

Podemos borrar las tablas person y favorite_food, no las usaremos más. El comando para ello es:

```
mysql> DROP TABLE favorite_food;
Query OK, 0 rows affected (0,18 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_bank |
+----------------+
| account        |
| branch         |
| business       |
| customer       |
| department     |
| employee       |
| individual     |
| officer        |
| product        |
| product_type   |
| transaction    |
+----------------+
11 rows in set (0,18 sec)
```

Como hemos indicado anteriormente para consultar las columnas de una tabla usamos el comando DESCRIBE:

```
mysql> DESCRIBE customer;
+--------------+------------------+------+-----+---------+----------------+
| Field        | Type             | Null | Key | Default | Extra          |
+--------------+------------------+------+-----+---------+----------------+
| cust_id      | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| fed_id       | varchar(12)      | NO   |     | NULL    |                |
| cust_type_cd | enum('I','B')    | NO   |     | NULL    |                |
| address      | varchar(30)      | YES  |     | NULL    |                |
| city         | varchar(20)      | YES  |     | NULL    |                |
| state        | varchar(20)      | YES  |     | NULL    |                |
| postal_code  | varchar(10)      | YES  |     | NULL    |                |
+--------------+------------------+------+-----+---------+----------------+
7 rows in set (0,18 sec)
```

PRIMERAS CONSULTAS.

Haremos un estudio detallado de las diferentes partes de la sentencia SELECT y su interacción.

Cláusulas de consulta.

Existen diversos componente so cláusulas que forman la consulta SELECT. Mientras que solo una de ellas es obligatoria, la propia SELECT. Lo normal es utilizar dos o tres cláusulas de las seis disponibles.

Cláusula	Descripción
SELECT	Determina qué columnas se deben incluir en la consulta.
FROM	Indica las tablas de dónde extraer los datos y su unión.
WHERE	Limita el número de filas en el conjunto de resultados.
GROUP BY	Agrupa filas por valores comunes de columnas.
HAVING	Igual a WHERE pero utilizando datos agrupados.
ORDER BY	Ordena las filas del conjunto de resultados.

La cláusula SELECT.

```
mysql> SELECT * FROM department;
+---------+----------------+
| dept_id | name           |
+---------+----------------+
|       1 | Operations     |
|       2 | Loans          |
|       3 | Administration |
+---------+----------------+
3 rows in set (0,18 sec)
```

El * hace referencia a todas las columnas.

```
mysql> SELECT dept_id, name  FROM department;
+---------+----------------+
| dept_id | name           |
+---------+----------------+
|       1 | Operations     |
|       2 | Loans          |
|       3 | Administration |
+---------+----------------+
3 rows in set (0,17 sec)
```

Los resultados son idénticos, se han incluido todas las columnas de la tabla en la consulta SELECT.

Pero SELECT puede hacer mucho más. Veamos la siguiente consulta que incorpora un literal, una expresión y una función integrada en MySQL (se incluye la cláusula WHERE emp_id < 6 para retringir los resulatdos a los cinco primeros:

```
mysql> SELECT emp_id,
    -> 'ACTIVE',
    -> emp_id * 3.14159,
    -> UPPER(lname)
    -> FROM employee;
    -> WHERE emp_id < 6;
```

```
+--------+--------+------------------+--------------+
| emp_id | ACTIVE | emp_id * 3.14159 | UPPER(lname) |
+--------+--------+------------------+--------------+
|      1 | ACTIVE |          3.14159 | SMITH        |
|      2 | ACTIVE |          6.28318 | BARKER       |
|      3 | ACTIVE |          9.42477 | TYLER        |
|      4 | ACTIVE |         12.56636 | HAWTHORNE    |
|      5 | ACTIVE |         15.70795 | GOODING      |
+--------+--------+------------------+--------------+
18 rows in set (0,18 sec)
```

Si solo se necesita ejecutar una función integrada o valorar una expresión podemos ignorar por completo la cláusula FROM.

```
mysql> SELECT VERSION(), USER(), DATABASE();
+--------------------------+---------------------+------------+
| VERSION()                | USER()              | DATABASE() |
+--------------------------+---------------------+------------+
| 10.5.19-MariaDB-0+deb11u2 | user@205.188.139.18 | bank       |
+--------------------------+---------------------+------------+
1 row in set (0,18 sec)
```

Hemos realizado una consulta que llama a tres funciones integradas y no recupera datos de ninguna tabla; no es necesario el uso de FROM.

Alias de columnas.

Se puede pedir un resultado con un alias de columna (puede que las columnas tengan nombre poco explicativo). Para ello se añade el alias después de cada elemento de su cláusula SELECT.

```
mysql> SELECT emp_id,
    -> 'ACTIVE' status,
    -> emp_id * 3.14159 empid_x_pi,
    -> UPPER(lname) last_name_upper
    -> FROM employee
    -> WHERE emp_id < 6;
+--------+--------+------------+-----------------+
| emp_id | status | empid_x_pi | last_name_upper |
+--------+--------+------------+-----------------+
|      1 | ACTIVE |    3.14159 | SMITH           |
|      2 | ACTIVE |    6.28318 | BARKER          |
|      3 | ACTIVE |    9.42477 | TYLER           |
|      4 | ACTIVE |   12.56636 | HAWTHORNE       |
|      5 | ACTIVE |   15.70795 | GOODING         |
+--------+--------+------------+-----------------+
5 rows in set (0,17 sec)
```

IMPORTANTE: Esto que acabamos de ver es importante a la hora de integrar las consultas con algún lenguaje de programación. Haría el código mucho más comprensible y fácil de escribir.

Eliminar duplicados de las consultas.

En algunos casos una consulta puede devolver filas con datos duplicados, por ejemplo, la siguiente consulta a cust_id, dado que algunos clientes disponen de más de una cuenta (nuevamente usamos una cláusula WHERE para limitar los resultados de la salida). Podemos usar la palabra clave DISTINCT para mostrar el conjunto (con todos su elementos distintos) de los clientes que tienen cuentas.

```
mysql> SELECT cust_id FROM account WHERE cust_id < 5;
+---------+
| cust_id |
+---------+
|       1 |
|       1 |
|       1 |
|       2 |
|       2 |
|       3 |
|       3 |
|       4 |
|       4 |
|       4 |
+---------+
10 rows in set (0,19 sec)

mysql> SELECT DISTINCT cust_id FROM account WHERE cust_id < 5;
+---------+
| cust_id |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
+---------+
4 rows in set (0,18 sec)
```

La cláusula FROM.

La cláusula `FROM` define las tablas utilizadas por una consulta, junto con los medios para enlazarlas o vincularlas. La definición esta compuesta por dos conceptos separados pero, a la vez, relacionados, que se tratarán en las siguientes secciones.

Tablas.

Existen tres tipos de tablas:

- Tablas permanentes, creadas utilizando la sentencia CREATE TABLE.
- Tablas temporales, es decir, filas devueltas por una subconsulta.
- Tablas virtuales o Vistas, es decir creadas usando la sentencia CREATE VIEW.

Cada uno de estos tipos de tablas se pueden incluir en la cláusula FROM de una consulta. Ya hemos visto como incluir una tabla permanente, veamos los otros dos tipos.

FROM y tablas generadas en subconsultas.

Una subconsulta es una consulta contenida dentro de otra. Las subconsultas están encerradas entre parénstesis y se pueden encontrar en diversas partes de una sentencia SELECT. Sin embargo, dentro d euna cláusula FROM, una subconsulta tiene el objetivo de generar una tabla temporal que es visible desde todas las otras cláusulas de consulta y puede interactuar con otras tablas, que estén indicadas en la cláusula FROM. Veamos un ejemplo (nuevamente incluímos WHERE para limitar los resulatdos :

```
mysql> SELECT e.emp_id, e.fname, e.lname
    -> FROM (SELECT emp_id, fname, lname, start_date, title
```

```
    -> FROM employee) e
    -> WHERE e.emp_id < 6;
+--------+---------+-----------+
| emp_id | fname   | lname     |
+--------+---------+-----------+
|      1 | Michael | Smith     |
|      2 | Susan   | Barker    |
|      3 | Robert  | Tyler     |
|      4 | Susan   | Hawthorne |
|      5 | John    | Gooding   |
+--------+---------+-----------+
5 rows in set (0,23 sec)
```

La subconsulta está referenciada por la consulta que la contiene a través de su alias que, en este caso, es `e`.

FROM y Vistas.

Una vista es una consulta almacenada en el diccionario de datos. SE asemeja y actúa como una tabla, pero existen otros datos asociados con una vista(este es el motivo por el que se denomina tabla virtual). Cuando se realiza una consulta sobre una vista, su consulta se combina con la definición de la vista para crear y ejecutar una consulta final.

Creemos una vista:

```
mysql> CREATE VIEW employee_vw AS
    -> SELECT emp_id, fname, lname,
    -> YEAR(start_date) start_year
    -> FROM employee;
Query OK, 0 rows affected (0,19 sec)
```

Hagamos ahora una consulta sobre la vista:

```
mysql> SELECT emp_id, start_year FROM employee_vw WHERE emp_id < 6;
+--------+------------+
| emp_id | start_year |
+--------+------------+
|      1 |       2001 |
|      2 |       2002 |
|      3 |       2000 |
|      4 |       2002 |
|      5 |       2003 |
+--------+------------+
5 rows in set (0,18 sec)
```

Las vistas se crean por varios motivos, como por ejemplo, para ocultar columnas a los usuarios o para simplificar diseños complejos de bases de datos.

Vínculos entre tablas.

La segunda variante, en la definición sencilla que hemos hecho de la cláusula FROM, es la obligatoriedad de que si aparece más de una tabla en la cláusula FROM, las condiciones utilizadas para enlazar las tablas también se debe incluir.

La unión de múltiples tablas se tratará más adelante, pero aquí hay un ejemplo simple que introduce la cuestión:

```
mysql> SELECT employee.emp_id, employee.fname,
    -> employee.lname, department.name dept_name
    -> FROM employee INNER JOIN department
    -> ON employee.dept_id = department.dept_id
    -> WHERE employee.emp_id < 6;
+--------+---------+-----------+----------------+
| emp_id | fname   | lname     | dept_name      |
+--------+---------+-----------+----------------+
|      4 | Susan   | Hawthorne | Operations     |
|      5 | John    | Gooding   | Loans          |
|      1 | Michael | Smith     | Administration |
|      2 | Susan   | Barker    | Administration |
|      3 | Robert  | Tyler     | Administration |
+--------+---------+-----------+----------------+
5 rows in set (0,17 sec)
```

Definir los alias de tablas.

Cuando se unen múltiples tablas en una única consulta, se necesita un modo de identificar a qué tabla nos estamos refiriendo cuando referencia columnas en la sentencias. Podemos usar el nombre completo, como anteriormente, por ejemplo:

```
employee.emp_id
```

Pero también podemos asignar un alias y utilizar éste a lo largo de toda la consulta:

```
mysql> SELECT e.emp_id, e.fname,
    -> e.lname, d.name dept_name
    -> FROM employee e INNER JOIN department d
    -> ON e.dept_id = d.dept_id
    -> WHERE e.emp_id < 6;
```

Además, se puede usar la palabra `AS`:

```
mysql> SELECT e.emp_id, e.fname,
    -> e.lname, d.name dept_name
    -> FROM employee AS e INNER JOIN department AS d
    -> ON e.dept_id = d.dept_id
    -> WHERE e.emp_id < 6;
```

La cláusula WHERE.

La mayoría de las veces no se deseará recuperar todas las filas de una tabla, sino que se necesitará saber cómo filtrar esas filas que no interesan. Ésta es una función que realiza la cláusula `WHERE`. Es el mecanismo que se utiliza para filtarar las filas no deseadas en una consulta.

Ya hemos visto algunos ejempos. Veamos otros:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE title='Teller';
+--------+---------+---------+------------+-------------+
| emp_id | fname   | lname   | start_date | title       |
+--------+---------+---------+------------+-------------+
|      6 | Chris   | Tucker  | 2004-03-17 | Teller      |
+--------+---------+---------+------------+-------------+
4 rows in set (3,45 sec)
```

En WHERE se pueden incluir cuantas condiciones sena necesarias, haciendo uso de los operadores AND, OR y NOT.

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE title='Head Teller'
    -> AND start_date > '2002-01-01';
+--------+-------+---------+------------+-------------+
| emp_id | fname | lname   | start_date | title       |
+--------+-------+---------+------------+-------------+
|      6 | Helen | Fleming | 2004-03-17 | Head Teller |
|     10 | Paula | Roberts | 2002-07-27 | Head Teller |
+--------+-------+---------+------------+-------------+
2 rows in set (0,17 sec)
```

Si se combinan operador se deben usar paréntesis para separar grupos de condiciones y así evitar resultados no correctos. Como aquí:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE (title='Head Teller' AND start_date > '2002-01-01')
    -> OR (title='Teller' AND start_date > '2003-01-01');
+--------+----------+---------+------------+-------------+
| emp_id | fname    | lname   | start_date | title       |
+--------+----------+---------+------------+-------------+
|      6 | Helen    | Fleming | 2004-03-17 | Head Teller |
|      7 | Chris    | Tucker  | 2004-09-15 | Teller      |
|     10 | Paula    | Roberts | 2002-07-27 | Head Teller |
|     12 | Samantha | Jameson | 2003-01-08 | Teller      |
|     15 | Frank    | Portman | 2003-04-01 | Teller      |
+--------+----------+---------+------------+-------------+
5 rows in set (0,18 sec)
```

Las cláusulas GROUP BY y HAVING.

Estas cláusulas son más avanzadas y se verán con posterioridad. Por ejemplo, en lugar de consultar una lista de empleados y los departamentos a los que están asignados, es probable que desee ver una lista de departamentos junto con el número de empleados asignados a cada uno de ellos; esto es, agrupar datos y se utiliza la cláusula GROUP BY.

IMPORTANTE: cuando se utiliza la cláusula GROUP BY, también se puede utilizar la cláusula HAVING, que permite filtrar datos en grupo, del mismo modo que WHERE permite filtrar datos sin manipular.

La cláusula ORDER BY.

En términos generales, las filas devueltas por una consulta no salen en un orden concreto. Si necesitamos un orden debemos usar la cláusula ORDER BY.

Podemos usar varias columnas en la misma cláusula ORDER BY. Veamos un par de ejemplos. El primero es una cláusula ORDER BY haciendo uso de una columna:

```
mysql> SELECT open_emp_id, product_cd
    -> FROM account
    -> WHERE open_emp_id < 2;
    -> ORDER BY product_cd;
```

```
+-------------+------------+
| open_emp_id | product_cd |
+-------------+------------+
|           1 | CD         |
|           1 | CD         |
|           1 | CHK        |
|           1 | CHK        |
|           1 | CHK        |
|           1 | MM         |
|           1 | MM         |
|           1 | SAV        |
+-------------+------------+
8 rows in set (0,17 sec)
```

En este segundo ejemplo, hacemos uso de dos columnas. El orden en el que aparecen las columnas sí importa en los resultados buscados:

```
mysql> SELECT open_emp_id, product_cd
    -> FROM account
    -> WHERE open_emp_id > 9;
    -> ORDER BY open_emp_id, product_cd;
+-------------+------------+
| open_emp_id | product_cd |
+-------------+------------+
|          10 | BUS        |
|          10 | CD         |
|          10 | CD         |
|          10 | CHK        |
|          10 | CHK        |
|          10 | SAV        |
|          10 | SAV        |
|          13 | CHK        |
|          13 | MM         |
|          13 | SBL        |
|          16 | BUS        |
|          16 | CHK        |
|          16 | CHK        |
|          16 | CHK        |
|          16 | CHK        |
|          16 | SAV        |
+-------------+------------+
16 rows in set (0,18 sec)
```

Ordenación ascendente y descendente.

Podemos usar las palabras claves `ASC` y `DESC` junto a `ORDER BY` para ordenar.

```
mysql> SELECT account_id, product_cd, open_date, avail_balance
    -> FROM account
    -> WHERE avail_balance > 9999
    -> ORDER BY avail_balance DESC;
+------------+------------+------------+---------------+
| account_id | product_cd | open_date  | avail_balance |
+------------+------------+------------+---------------+
|         29 | SBL        | 2004-02-22 |      50000.00 |
|         28 | CHK        | 2003-07-30 |      38552.05 |
|         24 | CHK        | 2002-09-30 |      23575.12 |
|         15 | CD         | 2004-12-28 |      10000.00 |
+------------+------------+------------+---------------+
4 rows in set (0,17 sec)
```

MySQL incluye la cláusula LIMIT, pero ésta no es parte del estándar ANSI SQL. LIMIT permite ordenar los datos y descartar todas las filas menos las x primeras.

Ordenar mediante marcadores numéricos de posición.

Si se está ordenando utilizando las columnas de la cláusula SELECT, se puede optar por referenciar las columnas.

```
mysql> SELECT emp_id, title, start_date, fname, lname
    -> FROM employee
    -> ORDER BY 2, 5;
+--------+--------------------+------------+----------+-----------+
| emp_id | title              | start_date | fname    | lname     |
+--------+--------------------+------------+----------+-----------+
|     13 | Head Teller        | 2000-05-11 | John     | Blake     |
|      6 | Head Teller        | 2004-03-17 | Helen    | Fleming   |
|     16 | Head Teller        | 2001-03-15 | Theresa  | Markham   |
|     10 | Head Teller        | 2002-07-27 | Paula    | Roberts   |
|      5 | Loan Manager       | 2003-11-14 | John     | Gooding   |
|      4 | Operations Manager | 2002-04-24 | Susan    | Hawthorne |
|      1 | President          | 2001-06-22 | Michael  | Smith     |
|     17 | Teller             | 2002-06-29 | Beth     | Fowler    |
|      9 | Teller             | 2002-05-03 | Jane     | Grossman  |
|     12 | Teller             | 2003-01-08 | Samantha | Jameson   |
|     14 | Teller             | 2002-08-09 | Cindy    | Mason     |
|      8 | Teller             | 2002-12-02 | Sarah    | Parker    |
|     15 | Teller             | 2003-04-01 | Frank    | Portman   |
|      7 | Teller             | 2004-09-15 | Chris    | Tucker    |
|     18 | Teller             | 2002-12-12 | Rick     | Tulman    |
|     11 | Teller             | 2000-10-23 | Thomas   | Ziegler   |
|      3 | Treasurer          | 2000-02-09 | Robert   | Tyler     |
|      2 | Vice President     | 2002-09-12 | Susan    | Barker    |
+--------+--------------------+------------+----------+-----------+
18 rows in set (0,17 sec)
```

FILTRADO.

Hay algunos casos en los que querremos trabajar con todas las filas de una tabla, por ejemplo:

- Purgar todos los datos de una tabla utilizada para organizar las nuevas entradas de datos en un almacén.
- Modificar todas las filas de una tabla después de que se haya añadido una nueva columna.
- Capturar todas las filas de una tabla de cola de mensajes.

Construir una condición.

Una condición está hecha de una o más expresiones acompañadas de uno o más operadores. Una expresión puede ser cualquiera de las siguientes:

- Un número.
- Una columna.
- Un literal de cadena.
- Un función integrada.
- Una subconsulta.
- Una lista de expresiones.

Los operadores utilizados dentro de las condiciones incluyen:

- Operadores de comparación, como por ejemplo, además de =,!=, <, >, <>, LIKE, IN Y BETWEEN.
- Operadores aritméticos.

Veamos algunos ejemplos:

```
mysql> SELECT pt.name AS product_type, p.name AS product
    -> FROM product p INNER JOIN product_type pt
    -> ON p.product_type_cd = pt.product_type_cd
    -> WHERE pt.name = 'Customer Accounts';
+------------------+-----------------------+
| product_type     | product               |
+------------------+-----------------------+
| Customer Accounts | certificate of deposit |
| Customer Accounts | checking account      |
| Customer Accounts | money market account  |
| Customer Accounts | savings account       |
+------------------+-----------------------+
4 rows in set (0,19 sec)

mysql> SELECT pt.name AS product_type, p.name AS product
    -> FROM product p INNER JOIN product_type pt
    -> ON p.product_type_cd = pt.product_type_cd
    -> WHERE pt.name != 'Customer Accounts';
+-------------------------------+-----------------------+
| product_type                  | product               |
+-------------------------------+-----------------------+
| Individual and Business Loans | auto loan             |
| Individual and Business Loans | business line of credit |
| Individual and Business Loans | home mortgage         |
| Individual and Business Loans | small business loan   |
+-------------------------------+-----------------------+
4 rows in set (0,18 sec)
```

El operador BETWEEN.

Cuando se tiene un límite superior y un límite inferior en el rango, se puede elegir usar una condición única mediante la utilización del operador `BETWEEN`.

```
mysql> SELECT emp_id, fname, lname, start_date
    -> FROM employee
    -> WHERE start_date BETWEEN '2001-01-01' AND '2003-01-01';
+--------+---------+-----------+------------+
| emp_id | fname   | lname     | start_date |
+--------+---------+-----------+------------+
|      1 | Michael | Smith     | 2001-06-22 |
|      2 | Susan   | Barker    | 2002-09-12 |
|      4 | Susan   | Hawthorne | 2002-04-24 |
|      8 | Sarah   | Parker    | 2002-12-02 |
|      9 | Jane    | Grossman  | 2002-05-03 |
|     10 | Paula   | Roberts   | 2002-07-27 |
|     14 | Cindy   | Mason     | 2002-08-09 |
|     16 | Theresa | Markham   | 2001-03-15 |
|     17 | Beth    | Fowler    | 2002-06-29 |
|     18 | Rick    | Tulman    | 2002-12-12 |
+--------+---------+-----------+------------+
10 rows in set (0,18 sec)
```

Rangos sobre cadenas.

También es posible aplicar rangos sobre cadenas. Veamos un ejemplo:

```
mysql> SELECT cust_id, fed_id
    -> FROM customer
    -> WHERE cust_type_cd = 'I'
    -> AND fed_id BETWEEN '500-00-0000' AND '999-99-9999';
+---------+-------------+
| cust_id | fed_id      |
+---------+-------------+
|       5 | 555-55-5555 |
|       6 | 666-66-6666 |
|       7 | 777-77-7777 |
|       8 | 888-88-8888 |
|       9 | 999-99-9999 |
+---------+-------------+
5 rows in set (0,18 sec)
```

El operador IN.

Con el operador `IN`, se puede escribir una única condición sin importar el núemro de expresiones que haya en un determiando conjunto. Esto nos evita tener que usar múltiples cláusulas OR, es decir podemos sustituir esto:

```
product_cd='CHK' OR product_cd='SAV' OR product_cd='CD'
```

por esta otra cláusula:

```
mysql> SELECT account_id, product_cd, cust_id, avail_balance
    -> FROM account
    -> WHERE product_cd IN ('CHK', 'SAV', 'CD')
    -> AND account_id < 4;
```

```
+------------+------------+---------+---------------+
| account_id | product_cd | cust_id | avail_balance |
+------------+------------+---------+---------------+
|          1 | CHK        |       1 |       1057.75 |
|          2 | SAV        |       1 |        500.00 |
|          3 | CD         |       1 |       3000.00 |
+------------+------------+---------+---------------+
3 rows in set (0,18 sec)
```

El operador IN y subconsultas.

```
mysql> SELECT account_id, product_cd, cust_id, avail_balance
    -> FROM account
    -> WHERE product_cd IN (SELECT product_cd FROM product
    -> WHERE product_type_cd = 'ACCOUNT');
+------------+------------+---------+---------------+
| account_id | product_cd | cust_id | avail_balance |
+------------+------------+---------+---------------+
|          3 | CD         |       1 |       3000.00 |
|         15 | CD         |       6 |      10000.00 |
|         17 | CD         |       7 |       5000.00 |
|         23 | CD         |       9 |       1500.00 |
|          1 | CHK        |       1 |       1057.75 |
|          4 | CHK        |       2 |       2258.02 |
|          7 | CHK        |       3 |       1057.75 |
|         10 | CHK        |       4 |        534.12 |
|         13 | CHK        |       5 |       2237.97 |
|         14 | CHK        |       6 |        122.37 |
|         18 | CHK        |       8 |       3487.19 |
|         21 | CHK        |       9 |        125.67 |
|         24 | CHK        |      10 |      23575.12 |
|         28 | CHK        |      12 |      38552.05 |
|          8 | MM         |       3 |       2212.50 |
|         12 | MM         |       4 |       5487.09 |
|         22 | MM         |       9 |       9345.55 |
|          2 | SAV        |       1 |        500.00 |
|          5 | SAV        |       2 |        200.00 |
+------------+------------+---------+---------------+
21 rows in set (0,18 sec)
```

El operador `IN` tiene su contraparte `NOT IN`, se usa del mismo modo.

Condiciones de correspondencia.

Puede que queramos encontrar todos los empleados cuyos apellidos empiecen con 'T'. En este caso, se debe utiliar una función integrada, `LEFT()`, para separar la primera letra de la columna `lname`, como por ejemplo:

```
mysql> SELECT emp_id, fname, lname
    -> FROM employee
    -> WHERE LEFT(lname, 1) = 'T';
+--------+--------+--------+
| emp_id | fname  | lname  |
+--------+--------+--------+
|      3 | Robert | Tyler  |
|      7 | Chris  | Tucker |
|     18 | Rick   | Tulman |
+--------+--------+--------+
3 rows in set (0,19 sec)
```

El operador LIKE y la utilización de carcateres comodín.

El guión bajo ocupa el lugar de un único carácter, mientras que el signo de porcentaje puede ser sustituido por un número variable de caracteres.

Carácter comodín	Coincidencias
_ (guión bajo)	Exactamente un carácter.
%	Cualquier número de caracteres (incluido cero).

```
mysql> SELECT lname
    -> FROM employee
    -> WHERE lname LIKE '_a%e%';
+-----------+
| lname     |
+-----------+
| Barker    |
| Hawthorne |
| Parker    |
| Jameson   |
+-----------+
4 rows in set (0,18 sec)
```

Utilización de expresiones regulares.

```
mysql> SELECT emp_id, fname, lname
    -> FROM employee
    -> WHERE lname REGEXP '^[FG]';
+--------+-------+----------+
| emp_id | fname | lname    |
+--------+-------+----------+
|      5 | John  | Gooding  |
|      6 | Helen | Fleming  |
|      9 | Jane  | Grossman |
|     17 | Beth  | Fowler   |
+--------+-------+----------+
4 rows in set (0,18 sec)
```

Oracle Database también soporta el uso de estas expresiones. Se utiliza la función `REGEXP_LIKE` en lugar del operador `REGEXP` mostrado en el ejemplo anterior.

NULL.

Cuando se trabaja con NULL, se debe recordar lo siguiente:

- Una expresión pueser ser nula, pero nunca puede igualarse a un valor nulo.
- Dos valores nulos nunca son iguales entre ellos.

Para comprobar si una expresión es nula se utiliza el oeprador `IS NULL`. Este oeprador tiene su comtraparte `IS NOT NULL` que operada del mismo modo.

```
mysql> SELECT emp_id, fname, lname, superior_emp_id
    -> FROM employee
    -> WHERE superior_emp_id IS NULL;
```

```
+--------+---------+-------+-----------------+
| emp_id | fname   | lname | superior_emp_id |
+--------+---------+-------+-----------------+
|      1 | Michael | Smith |            NULL |
+--------+---------+-------+-----------------+
1 row in set (0,18 sec)
```

La consulta devuelve todos los empleados que no tienen jefe. A continuación, se muestra la misma consulta utilizando '= NULL' en lugar de IS NULL:

```
mysql> SELECT emp_id, fname, lname, superior_emp_id
    -> FROM employee
    -> WHERE superior_emp_id = NULL;
Empty set (0,18 sec)
```

La consulta no devuelve ninguna fila. Este suele ser un error muy común realizado por programadores SQL sin experiencia. Además, el servidor de la base de datos tampoco le avisará sobre el error que se ha cometido, por lo tanto, se debe tener cuidado cuando se construyan las condiciones que comprueban los valores nulos.

IMPORTANTE: otro ejemplo del uso de NULL:

```
mysql> SELECT emp_id, fname, lname, superior_emp_id
    -> FROM employee
    -> WHERE superior_emp_id != 6 OR superior_emp_id IS NULL;
+--------+----------+-----------+-----------------+
| emp_id | fname    | lname     | superior_emp_id |
+--------+----------+-----------+-----------------+
|      1 | Michael  | Smith     |            NULL |
|      2 | Susan    | Barker    |               1 |
|      3 | Robert   | Tyler     |               1 |
|      4 | Susan    | Hawthorne |               3 |
|      5 | John     | Gooding   |               4 |
|      6 | Helen    | Fleming   |               4 |
|     10 | Paula    | Roberts   |               4 |
|     11 | Thomas   | Ziegler   |              10 |
|     12 | Samantha | Jameson   |              10 |
|     13 | John     | Blake     |               4 |
|     14 | Cindy    | Mason     |              13 |
|     15 | Frank    | Portman   |              13 |
|     16 | Theresa  | Markham   |               4 |
|     17 | Beth     | Fowler    |              16 |
|     18 | Rick     | Tulman    |              16 |
+--------+----------+-----------+-----------------+
15 rows in set (0,17 sec)
```

CONSULTAR MÚLTIPLES TABLAS. JOIN.

Debido a que el diseño de base de datos relacional establece u obliga a que las entidades independientes estén situadas en tablas separadas, necesitamos un mecanismo para agrupar múltiples tablas en una misma consulta. A este mecanismo se le conoce como unión (`JOIN`). Estudiaremos primero la unión más sencilla, la unión interna (`INNER JOIN`). Más adelante, se mostrarán los diferentes tipos de uniones.

Left outer join

Inner join

Right outer join

Full outer join

¿Qué es una unión?

Supongamos que queremos realizar una consulta que engloba datos en están en tablas separadas. Por ejemplo las tablas `employee` y `department` contienen los datos de los empleados y el departamento en el que trabajan respectivamente. Aquí la descripción de la tabla department:

```
mysql> DESCRIBE department;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| dept_id | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| name    | varchar(20)          | NO   |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
2 rows in set (0,18 sec)
```

Digamos que queremos recuperar el nombre y los apellidos de cada empleado junto con el nombre del departamento al que esta asignado cada empleado (Ver el diagrama entidad-relacción ER de la base de datos). Esta consulta necesitará recuperar las siguientes columnas:

```
employee.lname
employee.fname
department.name
```

IMPORTANTE: ¿Cómo podemos recuperar datos de ambas tablas en la misma consulta? La respuesta reside en la columna `employee.dept_id`, que es la clave ajena de la tabla `department`. La siguiente consulta, indica al servidor que utilice la columna `employee.dept_id` como puente entre ambas tablas. A este tipo de operación se le llama unión.

```
mysql> SELECT e.fname, e.lname, d.name
    -> FROM employee AS e JOIN department AS d;
+----------+-----------+----------------+
| fname    | lname     | name           |
+----------+-----------+----------------+
| Michael  | Smith     | Operations     |
| Michael  | Smith     | Loans          |
| Michael  | Smith     | Administration |
| Susan    | Barker    | Operations     |
| Susan    | Barker    | Loans          |
| Susan    | Barker    | Administration |
| Robert   | Tyler     | Operations     |
| Robert   | Tyler     | Loans          |
| Robert   | Tyler     | Administration |
| Susan    | Hawthorne | Operations     |
| Susan    | Hawthorne | Loans          |
| Susan    | Hawthorne | Administration |
| John     | Gooding   | Operations     |
| John     | Gooding   | Loans          |
| John     | Gooding   | Administration |
| Helen    | Fleming   | Operations     |
| Helen    | Fleming   | Loans          |
| Helen    | Fleming   | Administration |
| Chris    | Tucker    | Operations     |
| Chris    | Tucker    | Loans          |
| Chris    | Tucker    | Administration |
| Sarah    | Parker    | Operations     |
| Sarah    | Parker    | Loans          |
| Sarah    | Parker    | Administration |
| Jane     | Grossman  | Operations     |
| Jane     | Grossman  | Loans          |
| Jane     | Grossman  | Administration |
| Paula    | Roberts   | Operations     |
| Paula    | Roberts   | Loans          |
| Paula    | Roberts   | Administration |
| Thomas   | Ziegler   | Operations     |
| Thomas   | Ziegler   | Loans          |
| Thomas   | Ziegler   | Administration |
| Samantha | Jameson   | Operations     |
| Samantha | Jameson   | Loans          |
| Samantha | Jameson   | Administration |
| John     | Blake     | Operations     |
| John     | Blake     | Loans          |
| John     | Blake     | Administration |
| Cindy    | Mason     | Operations     |
| Cindy    | Mason     | Loans          |
| Cindy    | Mason     | Administration |
| Frank    | Portman   | Operations     |
| Frank    | Portman   | Loans          |
| Frank    | Portman   | Administration |
| Theresa  | Markham   | Operations     |
| Theresa  | Markham   | Loans          |
| Theresa  | Markham   | Administration |
| Beth     | Fowler    | Operations     |
| Beth     | Fowler    | Loans          |
| Beth     | Fowler    | Administration |
| Rick     | Tulman    | Operations     |
| Rick     | Tulman    | Loans          |
| Rick     | Tulman    | Administration |
+----------+-----------+----------------+
54 rows in set (0,17 sec)
```

Pero solo tenemos 18 empleados y tes departamentos, y MySQL nos ha devuelto la consulta con 54 resultados. Lo que ha ocurrido es que el conjunto de 18 empleados se ha repetidos 3 veces, con datos idénticos excepto el nombre del departamento. Se generó el producto cartesiano, que es cualquier permutación de las tablas. A este tipo de unión se le denomina unión cruzada `CROSS JOIN`, pero normalmente no se usa. Debemos añadir más información, para ello usamos la subcláusula `ON` en la cláusula `FROM` y solicitamos una unión interna `INNER JOIN` (Si no se especifica el tipo de unión -INNER-, el servidor realizará por defecto una **unión interna**, INNER JOIN. Sin embargo, existen varios tipos de uniones, y por lo tanto, debemos acostumbrarnos a especificar el tipo de unión exacto).

```
mysql> SELECT e.fname, e.lname, d.name
    -> FROM employee AS e INNER JOIN department AS d
    -> ON e.dept_id = d.dept_id;
+----------+-----------+----------------+
| fname    | lname     | name           |
+----------+-----------+----------------+
| Susan    | Hawthorne | Operations     |
| Helen    | Fleming   | Operations     |
| Chris    | Tucker    | Operations     |
| Sarah    | Parker    | Operations     |
| Jane     | Grossman  | Operations     |
| Paula    | Roberts   | Operations     |
| Thomas   | Ziegler   | Operations     |
| Samantha | Jameson   | Operations     |
| John     | Blake     | Operations     |
| Cindy    | Mason     | Operations     |
| Frank    | Portman   | Operations     |
| Theresa  | Markham   | Operations     |
| Beth     | Fowler    | Operations     |
| Rick     | Tulman    | Operations     |
| John     | Gooding   | Loans          |
| Michael  | Smith     | Administration |
| Susan    | Barker    | Administration |
| Robert   | Tyler     | Administration |
+----------+-----------+----------------+
18 rows in set (0,18 sec)
```

Si los nombres de las columans utilizados par aunir las dos tablas son idénticos, lo que es cierto en la consulta anterior, se pued eusar la subcláusula `USING`, como sigue:

```
mysql> SELECT e.fname, e.lname, d.name
    -> FROM employee AS e INNER JOIN department AS d
    -> USING (dept_id);
```

Esta consulta devolvería el mismo resulatdo que la anterior.

JOIN avanzado. Unión de tres tablas - utilización de subconsultas como tablas.

Ejemplo de una consulta en la que se usan varias tablas generadas por subconsultas utilizando las cláusulas `INNER JOIN` para obtener información específica de una base de datos. Aquí está la descripción de la consulta paso a paso:

1. La consulta principal selecciona las siguientes columnas de la tabla "account": `account_id`, `cust_id`, `open_date` y `product_cd`.

2. Se utiliza la cláusula INNER JOIN para unir la tabla `account` con una subconsulta que selecciona `emp_id` y `assigned_branch_id` de la tabla

`employee`. Esta subconsulta filtra los empleados que tienen una fecha de inicio anterior o igual al 1 de enero de 2003 y que tienen el título de "Teller" o "Head Teller".

3. A su vez, se utiliza otra cláusula INNER JOIN para unir la tabla resultante del paso anterior con una subconsulta que selecciona `branch_id` de la tabla `branch`. Esta subconsulta filtra las sucursales que tienen el nombre "Woburn Branch".

4. La condición de unión se establece entre el `open_emp_id` de la tabla `account` y el `emp_id` de la subconsulta de empleados, así como entre el `assigned_branch_id` de la subconsulta de empleados y el `branch_id` de la subconsulta de sucursales.

```
SELECT a.account_id, a.cust_id, a.open_date, a.product_cd
FROM account AS a INNER JOIN
      (SELECT emp_id, assigned_branch_id
            FROM employee
            WHERE start_date <= '2003-01-01'
                  AND (title = 'Teller' OR title = 'Head Teller')) AS e
      ON a.open_emp_id = e.emp_id
      INNER JOIN
            (SELECT branch_id
            FROM branch
            WHERE name = 'Woburn Branch') AS b
      ON e.assigned_branch_id = b.branch_id;
```

En resumen, esta consulta devuelve las cuentas que fueron abiertas por los empleados con título de "Teller" o "Head Teller" antes o en el 1 de enero de 2003 y están asignados a la sucursal llamada "Woburn Branch". Proporciona información específica de esas cuentas, como el ID de cuenta, el ID de cliente, la fecha de apertura y el código de producto.

```
+------------+---------+------------+------------+
| account_id | cust_id | open_date  | product_cd |
+------------+---------+------------+------------+
|          1 |       1 | 2000-01-15 | CHK        |
|          2 |       1 | 2000-01-15 | SAV        |
|          3 |       1 | 2004-06-30 | CD         |
|          4 |       2 | 2001-03-12 | CHK        |
|          5 |       2 | 2001-03-12 | SAV        |
|         17 |       7 | 2004-01-12 | CD         |
|         27 |      11 | 2004-03-22 | BUS        |
+------------+---------+------------+------------+
7 rows in set (0,20 sec)
```

TRABAJAR CON CONJUNTOS.

Aunque se puede interactuar fila por fila con la información de una base de datos, las bases de datos relacionales tratan realmente de conjuntos. Analizaremos ahora cómo se pueden combinar múltiples tablas utilizando varios operadores de conjuntos.

Primera teoría de conjuntos.

Las operaciones de conjuntos disponibles son las siguientes:

Operación	Operador
Unión	A UNION B
Intersección	A INTERSECT B
Diferencia	A EXCEPT B
Unión – Intersección	(A UNION B) EXCEPT (A INTERSECT B)

Se deben seguir las siguientes pautas al ejecutar operaciones de conjunto sobre tablas reales:

- Ambas tablas deben tener el mismo números de columnas.
- Los tipos de datos de cada columna de ambas tablas deben ser los mismos (o el servidor debe ser capaz de convertirlos).

Las operaciones sobre conjuntos se realizan situando un operador de conjuntos entre dos sentencias `SELECT`, por ejemplo:

```
mysql> SELECT 1 num, 'abc' str
    -> UNION
    -> SELECT 9 num, 'xyz' str;
+------+------+
| num  | str  |
+------+------+
|  1   | abc  |
|  9   | xyz  |
+------+------+
2 rows in set (0,02 sec)
```

El operador UNION.

Los operadores `UNION` y `UNION ALL` permiten combinar múltiples tablas. La diferencia está en que, cuando se quiere combinar dos tablas que incluyan todas las filas de ambas en el resultado final, incluso cuando se tenga que pagar por tener duplicados, se necesita utilizar el operador `UNION ALL`. Con `UNION ALL` el número de filas de la tabla final siempre será igual a la suma del número de filas en las tablas originales.

Veamos las filas que contienen cada una de las dos tablas sobre las que luego haremos las operaciones de conjunto. Usaremos las tablas `individual` y `business` de la base de datos de nuestro ejemplo `bank`:

```
mysql> SELECT cust_id, lname AS name
    -> FROM individual;
```

```
+---------+---------+
| cust_id | name    |
+---------+---------+
|       1 | Hadley  |
|       2 | Tingley |
|       3 | Tucker  |
|       4 | Hayward |
|       5 | Frasier |
|       6 | Spencer |
|       7 | Young   |
|       8 | Blake   |
|       9 | Farley  |
+---------+---------+
9 rows in set (0,17 sec)

mysql> SELECT cust_id, name
    -> FROM business;
+---------+------------------------+
| cust_id | name                   |
+---------+------------------------+
|      10 | Chilton Engineering    |
|      11 | Northeast Cooling Inc. |
|      12 | Superior Auto Body     |
|      13 | AAA Insurance Inc.     |
+---------+------------------------+
4 rows in set (0,17 sec)

mysql> SELECT cust_id, lname AS name
    -> FROM individual
    -> UNION ALL
    -> SELECT cust_id, name
    -> FROM business;
+---------+------------------------+
| cust_id | name                   |
+---------+------------------------+
|       1 | Hadley                 |
|       2 | Tingley                |
|       3 | Tucker                 |
|       4 | Hayward                |
|       5 | Frasier                |
|       6 | Spencer                |
|       7 | Young                  |
|       8 | Blake                  |
|       9 | Farley                 |
|      10 | Chilton Engineering    |
|      11 | Northeast Cooling Inc. |
|      12 | Superior Auto Body     |
|      13 | AAA Insurance Inc.     |
+---------+------------------------+
13 rows in set (0,17 sec)
```

La consulta devuelve 13 clientes, 9 filas provienen de la tabla `individual` y otras 4 provienen de la tabla `business`. Si se quiere que la tabla combinada excluya las filas duplicadas, se tiene que utilizar el operador `UNION`, en lugar del operador `UNION ALL`.

En resumen, tanto `UNION` como `UNION ALL` se utilizan para combinar los resultados de dos o más consultas en una sola tabla. Sin embargo, hay una diferencia importante entre ambas:

UNION: La cláusula UNION se utiliza para combinar el resultado de dos o más consultas y devolver un conjunto de resultados único sin duplicados. Cuando se utiliza UNION, el motor de la base de datos elimina automáticamente las filas duplicadas del conjunto de resultados combinado.

UNION ALL: La cláusula UNION ALL también combina el resultado de dos o más consultas, pero a diferencia de UNION, no elimina las filas duplicadas. El conjunto de resultados combinado con UNION ALL incluirá todas las filas de cada consulta, incluso si hay duplicados.

Si se desea eliminar las filas duplicadas del resultado combinado, se debe usar UNION. Si no se necesita eliminar duplicados y se desea incluir todas las filas de cada consulta, se puede utilizar UNION ALL. Es importante tener en cuenta que UNION ALL es generalmente más eficiente en términos de rendimiento, ya que no requiere la operación adicional de eliminación de duplicados.

El operador INTERSECT.

Las siguientes dos consultas operadas por INTERSECT no tienen elementos en común por lo tanto se devuelve el conjunto vacío (Empty set):

```
mysql> SELECT emp_id, fname, lname
    -> FROM employee
    -> INTERSECT
    -> SELECT cust_id, fname, lname
    -> FROM individual;
Empty set (0,17 sec)
```

El siguiente ejemplo muestra una consulta que sí devuelve un resultado no vacío:

```
mysql> SELECT emp_id
    -> FROM employee
    -> WHERE assigned_branch_id = 2
    -> AND (title = 'Teller' OR title = 'Head Teller')
    -> INTERSECT
    -> SELECT DISTINCT open_emp_id
    -> FROM account
    -> WHERE open_branch_id = 2;
+--------+
| emp_id |
+--------+
|     10 |
+--------+
1 row in set (0,18 sec)
```

Para comprender mejor este ejemplo conviene inspeccionar los contenidos de las tablas utilizadas, no lo haremos aquí.

El operador EXCEPT.

El operador EXCEPT devuelve la primera tabla menos cualquier coincidencia encontrada en la segunda tabla.

```
mysql> SELECT emp_id
    -> FROM employee
    -> WHERE assigned_branch_id = 2
    -> AND (title = 'Teller' OR title = 'Head Teller')
    -> EXCEPT
```

```
    -> SELECT DISTINCT open_emp_id
    -> FROM account
    -> WHERE open_branch_id = 2;
+--------+
| emp_id |
+--------+
|     11 |
|     12 |
+--------+
2 rows in set (0,18 sec)
```

Igualmente, para comprender mejor este ejemplo conviene inspeccionar los contenidos de las tablas utilizadas, pero no lo haremos aquí.

AGRUPAR Y AGREGAR.

Nos centraremos ahora en cḿo se pueden agrupar o gregar los datos para permitr a los usuarios que interactuen con ellos a nivel de granularidad más alto que el que está almacenado en la propia base de datos.

Conceptos de agrupación.

Supongamos que queremos conocer cuántas cuentas a abierto cada cajero. Podemos escribir una consulta con datos son agrupar:

```
mysql> SELECT open_emp_id FROM account;
+-------------+
| open_emp_id |
+-------------+
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          13 |
|          13 |
|          13 |
|          16 |
|          16 |
|          16 |
|          16 |
|          16 |
|          16 |
+-------------+
24 rows in set (3,42 sec)
```

Los empleados 1, 10, 13 y 16 han abierto cuentas. Ahora hacemos la misma consulta agrupando los datos:

```
mysql> SELECT open_emp_id
    -> FROM account
    -> GROUP BY open_emp_id;
+-------------+
| open_emp_id |
+-------------+
|           1 |
|          10 |
|          13 |
|          16 |
+-------------+
4 rows in set (0,18 sec)
```

El conjunto de resultados contiene una fila por cada valor distinto en la columna `open_emp_id`, dando como resultado cuatro filas en lugar de 24. El motivo

de obtener un conjuto de resultados más pequeño es que cada uno de los cuatro empleados abrió másde una cuenta.

Para comprobar cuántas cuentas ha aboerto cada cajero, podemos utilizar una funciónde agregación en la cláusula `SELECT` para contar el núemro de filas ecistente en cada grupo:

```
mysql> SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          13 |        3 |
|          16 |        6 |
+-------------+----------+
4 rows in set (0,18 sec)
```

Esta consulta contesta a nuestra pregunta inicial, conocer cuántas cuentas ha abierto cada cajero.

Al agrupar los datos, puede que necesitemos filtrar los que no queramos que aparezcan en el conjunto de resultados que esté basado en lso grupos. Podríamos pensar hacerlo conuna cláusula WHERE, como aquí (pero la siguinet consulta devolverá un error):

```
mysql> SELECT open_emp_id, CONY(*) how_many
    -> FROM account
    -> WHERE COUNT(*) > 4
    -> GROUP BY open_emp_id, product_cd;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near '*)
how_many. FROM account WHERE count(*) > 4 GROUP BY open_emp_id, product_cd' at
line 1.
```

No podemos referirnos a la función de agregación `COUNT(*)` en la cláusula `WHERE`, porque los grupos no se han generado al mismo tiempo que se evaluó esta cláusula. Pero podemos indicar las condiciones de filtrado del grupo en una cláusula `HAVING`.

Cláusula HAVING.

```
mysql> SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id
    -> HAVING COUNT(*) > 4;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          16 |        6 |
+-------------+----------+
3 rows in set (0,17 sec)
```

El conjunto de resultados comprende a aquellos empleados que han abierto 5 cuentas o más.

Funciones de agregación.

Las funciones de agregación realizan una función específica sobre todas las filas de un grupo. Aunque todos los servidores de bases de datos disponen de su propio conjunto de funciones de agregación, las más comunes implementadas por los principales servidores son:

Función	Descripción
MAX()	Devuelve el valor máximo dentro de un conjunto.
MIN()	Devuelve el valor mínimo dentro de un conjunto.
AVG()	Devuelve el valor medio dentro de un conjunto.
SUM()	Devuelve la suma de los valores del conjunto.
COUNT()	Devuelve el número de valores de un conjunto.

Con estas funciones podemos analizar los saldos disponibles en las cuentas:

```
mysql> SELECT MAX(avail_balance) max_balance,
    -> MIN(avail_balance) min_valance,
    -> AVG(avail_balance) avg_balance,
    -> SUM(avail_balance) tot_balance,
    -> COUNT(*) num_accunts
    -> FROM account
    -> WHERE product_cd = 'CHK';
+-------------+-------------+-------------+-------------+-------------+
| max_balance | min_balance | avg_balance | tot_balance | num_accunts |
+-------------+-------------+-------------+-------------+-------------+
|    38552.05 |      122.37 | 7300.800985 |    73008.01 |          10 |
+-------------+-------------+-------------+-------------+-------------+
1 row in set (0,18 sec)
```

SUBCONSULTAS.

Las subconsultas son una herramienta muy potente que se pueden utilizar en las cuatro sentencias de manipulación de datos SQL.

¿Qué es una subconsulta?

Una subconsulta es uan consulta dentro de otra sentencia SQL (a la que nos referiremos como sentencia contenedora). Una subconsulta siempre está entre paréntesis y normalmente, se ejecuta con anterioridad a la sentecnia que la contiene. Al igual que cualquier consulta, uan subconsulta devuelve una tabla que puede consistir en: una única fila conuna única columna; múltiples filas con una única columna; múltiples filas y columnas.

Ya hemos posido ver subconsultas en eejemplos anteriores, pero mostremos nuevamente un ejemplo sencillo:

```
mysql> SELECT account_id, product_cd, cust_id, avail_balance
    -> FROM account
    -> WHERE account_id = (SELECT MAX(account_id) FROM account);
+------------+------------+---------+---------------+
| account_id | product_cd | cust_id | avail_balance |
+------------+------------+---------+---------------+
|         29 | SBL        |      13 |      50000.00 |
+------------+------------+---------+---------------+
1 row in set (0,18 sec)
```

Tipos de subconsultas.

Algunas consultas son totalmente autocontenidas, denominas subconsultas no correlacionadas, mientras que otras referencian columnas de la sentencia contenedora, denominadas subconsultas correlacionadas.

Subconsultas no correlacionadas.

El ejemplo anterior era de una subconsulta no correlacionada, puede ejecutarse sola y no hace referencia a nada de la sentencia contenedora: La mayoría de las subconsultas que encontremos serán de este tipo, a menos que estemos escribiendo sentencias `UPDATE` o `DELETE`, que frecuentemente utilizan subconsultas correlacionadas. Además, la consulta anterior devuelve una tabla formada por una sola fila y una única columna:

```
mysql> SELECT MAX(account_id) FROM account;
+-----------------+
| MAX(account_id) |
+-----------------+
|              29 |
+-----------------+
1 row in set (0,18 sec)
```

A este tipo de subconsulta se la denomina subconsulta escalar y puede aparecer en cualquier parte de una condición, utilizando los operadores habituales (=, <>, <, >, >=, <=). Veamos otro ejemplo de subconsulta escalar, en este caso con una condición de desigualdad:

```
mysql> SELECT account_id, product_cd, cust_id, avail_balance
    -> FROM account
    -> WHERE open_emp_id <> (SELECT e.emp_id
    -> FROM employee AS e INNER JOIN branch AS b
    -> ON e.assigned_branch_id = b.branch_id
    -> WHERE e.title = 'Head Teller' AND b.city = 'Woburn');
```

```
+------------+------------+---------+---------------+
| account_id | product_cd | cust_id | avail_balance |
+------------+------------+---------+---------------+
|          7 | CHK        |       3 |       1057.75 |
|          8 | MM         |       3 |       2212.50 |
|         10 | CHK        |       4 |        534.12 |
|         11 | SAV        |       4 |        767.77 |
|         12 | MM         |       4 |       5487.09 |
|         13 | CHK        |       5 |       2237.97 |
|         14 | CHK        |       6 |        122.37 |
|         15 | CD         |       6 |      10000.00 |
|         18 | CHK        |       8 |       3487.19 |
|         19 | SAV        |       8 |        387.99 |
|         21 | CHK        |       9 |        125.67 |
|         22 | MM         |       9 |       9345.55 |
|         23 | CD         |       9 |       1500.00 |
|         24 | CHK        |      10 |      23575.12 |
|         25 | BUS        |      10 |          0.00 |
|         28 | CHK        |      12 |      38552.05 |
|         29 | SBL        |      13 |      50000.00 |
+------------+------------+---------+---------------+
17 rows in set (0,18 sec)
```

Subconsultas de columnas únicas y múltiples filas.

Si una subconsulta devuelve más de una fila, entonces no se podrá utilizar en uno de los lados de una condición de igualdad. Sin embargo, existen cuatro operadores adicionales que se pueden utilizar para construir condiciones con este tipo de subconsultas.

El Operador IN.

Aunque no se puede igualar un valor único a un conjunto de valores, si se puede comprobar si un único valor pertenece a un conjunto de valores. El siguiente ejemplo , aunque no utiliza una subconsulta, muestra como construir una condición que utilice el operador `IN`, para localizar un valor dentro de un conjunto de valores.

```
mysql> SELECT branch_id, name, city
    -> FROM branch
    -> WHERE name IN ('Headquarters', 'Quincy Branch');
+-----------+---------------+---------+
| branch_id | name          | city    |
+-----------+---------------+---------+
|         1 | Headquarters  | Waltham |
|         3 | Quincy Branch | Quincy  |
+-----------+---------------+---------+
2 rows in set (0,17 sec)
```

El operador `IN` comprueba si se pueden encontrar las cadenas en la columna `name`; si es así, la condición se cumple y la fila se añade al conjunto de resultados. La siguiente consulta utiliza el operador IN con una subconsulta en la parte derecha de la condición de filtrado, para ver qué empleados supervisan a otros.

```
mysql> SELECT emp_id, fname, lname, title
    -> FROM employee
    -> WHERE emp_id IN
    -> (SELECT superior_emp_id FROM employee);
```

```
+--------+---------+-----------+--------------------+
| emp_id | fname   | lname     | title              |
+--------+---------+-----------+--------------------+
|      1 | Michael | Smith     | President          |
|      3 | Robert  | Tyler     | Treasurer          |
|      4 | Susan   | Hawthorne | Operations Manager |
|      6 | Helen   | Fleming   | Head Teller        |
|     10 | Paula   | Roberts   | Head Teller        |
|     13 | John    | Blake     | Head Teller        |
|     16 | Theresa | Markham   | Head Teller        |
+--------+---------+-----------+--------------------+
7 rows in set (0,18 sec)
```

El operado `IN` tiene su contraparte `NOT IN`, que se utiliza del mismo modo.

El operador ALL.

Mientras que el operador `IN` se utiliza cuando una expresión se puede encontrar dentro de un conjunto de expresiones, el operador `ALL` le permite realizar comparaciones entre un único valor y todos los valores del conjunto. Es decir, el operador `ALL` se utiliza para comparar un valor con todos los valores de un conjunto resultante de una subconsulta. La función del operador `ALL` es verificar si una condición se cumple para todos los valores del conjunto resultante.

La sintaxis básica del operador ALL es la siguiente:

```
value [operador de comparación] ALL (subconsulta)
```

El operador ALL funciona de la siguiente manera:

- La subconsulta se ejecuta y devuelve un conjunto de valores.
- El valor se compara con cada uno de los valores del conjunto resultante utilizando el operador de comparación especificado.
- La condición se considera verdadera si la comparación es verdadera para todos los valores del conjunto.

```
mysql> SELECT emp_id, fname, lname, title
    -> FROM employee
    -> WHERE emp_id <> ALL (SELECT superior_emp_id
    -> FROM employee
    -> WHERE superior_emp_id IS NOT NULL);
+--------+----------+----------+----------------+
| emp_id | fname    | lname    | title          |
+--------+----------+----------+----------------+
|      2 | Susan    | Barker   | Vice President |
|      5 | John     | Gooding  | Loan Manager   |
|      7 | Chris    | Tucker   | Teller         |
|      8 | Sarah    | Parker   | Teller         |
|      9 | Jane     | Grossman | Teller         |
|     11 | Thomas   | Ziegler  | Teller         |
|     12 | Samantha | Jameson  | Teller         |
|     14 | Cindy    | Mason    | Teller         |
|     15 | Frank    | Portman  | Teller         |
|     17 | Beth     | Fowler   | Teller         |
|     18 | Rick     | Tulman   | Teller         |
+--------+----------+----------+----------------+
11 rows in set (0,18 sec)
```

En resumen, el operador ALL se utiliza para realizar una comparación con todos los valores de un conjunto resultante de una subconsulta, y la condición se considera verdadera si la comparación es verdadera para todos los valores del conjunto.

El operador ANY.

El operador `ANY` se utiliza para comparar un valor con al menos uno de los valores de un conjunto resultante de una subconsulta. La función del operador `ANY` es verificar si una condición se cumple para al menos uno de los valores del conjunto resultante.

La sintaxis básica del operador `ANY` es la siguiente:

```
value [operador de comparación] ANY (subconsulta)
```

El operador `ANY` funciona de la siguiente manera:

- La subconsulta se ejecuta y devuelve un conjunto de valores.
- El valor se compara con cada uno de los valores del conjunto resultante utilizando el operador de comparación especificado.
- La condición se considera verdadera si la comparación es verdadera para al menos uno de los valores del conjunto.

En la siguiente consulta el operador de condición filtra las filas de la tabla `account` donde el valor de `avail_balance` es mayor que al menos uno de los valores de la subconsulta.

```
mysql> SELECT account_id, cust_id, product_cd, avail_balance
    -> FROM account
    -> WHERE avail_balance > ANY (SELECT a.avail_balance
    -> FROM account AS a
    -> INNER JOIN individual AS i ON a.cust_id = i.cust_id
    -> WHERE i.fname = 'Frank' AND i.lname = 'Tucker');
+------------+---------+------------+---------------+
| account_id | cust_id | product_cd | avail_balance |
+------------+---------+------------+---------------+
|          3 |       1 | CD         |       3000.00 |
|          4 |       2 | CHK        |       2258.02 |
|          8 |       3 | MM         |       2212.50 |
|         12 |       4 | MM         |       5487.09 |
|         13 |       5 | CHK        |       2237.97 |
|         15 |       6 | CD         |      10000.00 |
|         17 |       7 | CD         |       5000.00 |
|         18 |       8 | CHK        |       3487.19 |
|         22 |       9 | MM         |       9345.55 |
|         23 |       9 | CD         |       1500.00 |
|         24 |      10 | CHK        |      23575.12 |
|         27 |      11 | BUS        |       9345.55 |
|         28 |      12 | CHK        |      38552.05 |
|         29 |      13 | SBL        |      50000.00 |
+------------+---------+------------+---------------+
14 rows in set (0,20 sec)
```

En resumen, el operador ANY se utiliza para realizar una comparación con al menos uno de los valores de un conjunto resultante de una subconsulta, y la

condición se considera verdadera si la comparación es verdadera para al menos uno de los valores del conjunto.

Subconsultas de múltiples columnas.

En ocasiones se pueden utilizar subconsultas que devuelven dos o más columnas.

La siguiente consulta selecciona las columnas `account_id`, `product_cd` y `cust_id` de la tabla `account`. La condición utilizada en la cláusula WHERE es

```
(open_branch_id, open_emp_id) IN (SELECT b.branch_id, e.emp_id ...)
```

Esta condición filtra las filas de la tabla account donde la combinación de `open_branch_id` y `open_emp_id` se encuentra en el conjunto de resultados de la subconsulta.

La subconsulta

```
SELECT b.branch_id, e.emp_id FROM branch AS b INNER JOIN employee AS e ON
b.branch_id = e.assigned_branch_id WHERE b.name = 'Woburn Branch' AND
(e.title = 'Teller' OR e.title = 'Head Teller')
```

se utiliza para obtener pares de valores de `branch_id` y `emp_id` de las tablas `branch` y `employee`. La condición

```
b.name = 'Woburn Branch'
```

filtra las filas de la tabla `branch` para obtener solo la sucursal con el nombre 'Woburn Branch'. Además, la condición

```
e.title = 'Teller' OR e.title = 'Head Teller'
```

filtra las filas de la tabla `employee` para obtener empleados con el título 'Teller' o 'Head Teller'.

```
mysql> SELECT account_id, product_cd, cust_id
    -> FROM account
    -> WHERE (open_branch_id, open_emp_id) IN
    -> (SELECT b.branch_id, e.emp_id
    -> FROM branch AS b INNER JOIN employee AS e
    -> ON b.branch_id = e.assigned_branch_id
    -> WHERE b.name = 'Woburn Branch'
    -> AND (e.title = 'Teller' OR e.title = 'Head Teller'));
+------------+------------+---------+
| account_id | product_cd | cust_id |
+------------+------------+---------+
|          1 | CHK        |       1 |
|          2 | SAV        |       1 |
|          3 | CD         |       1 |
|          4 | CHK        |       2 |
|          5 | SAV        |       2 |
|         17 | CD         |       7 |
|         27 | BUS        |      11 |
+------------+------------+---------+
7 rows in set (0,18 sec)
```

En resumen, la consulta selecciona las filas de la tabla `account` donde la combinación de `open_branch_id` y `open_emp_id` se encuentra en el conjunto de resultados de la subconsulta, que obtiene los valores de `branch_id` y `emp_id` para

la sucursal 'Woburn Branch' y los empleados con los títulos 'Teller' o 'Head Teller'.

Subconsultas correlacionadas.

Una subconsulta correlacionada es dependiente de su sentencia contenedora, a partir de la cual se referencian una o más columnas. Veamos un ejemplo.

La siguiente consulta selecciona las columnas `cust_id`, `cust_type_cd` y `city` de la tabla `customer`. La condición utilizada en la cláusula WHERE es

```
2 = (SELECT COUNT(*) FROM account AS a WHERE a.cust_id = c.cust_id)
```

Esta condición filtra las filas de la tabla customer donde el número de cuentas asociadas al cliente es igual a 2.

La subconsulta

```
SELECT COUNT(*) FROM account AS a WHERE a.cust_id = c.cust_id
```

se utiliza para contar el número de filas en la tabla account donde el `cust_id` coincide con el `cust_id` de la tabla `customer`. Luego, se compara ese número de cuentas con el valor 2.

```
mysql> SELECT c.cust_id, c.cust_type_cd, c.city
    -> FROM customer AS c
    -> WHERE 2 = (SELECT COUNT(*)
    -> FROM account AS a
    -> WHERE a.cust_id = c.cust_id);
+---------+--------------+---------+
| cust_id | cust_type_cd | city    |
+---------+--------------+---------+
|       2 | I            | Woburn  |
|       3 | I            | Quincy  |
|       6 | I            | Waltham |
|       8 | I            | Salem   |
|      10 | B            | Salem   |
+---------+--------------+---------+
5 rows in set (0,17 sec)
```

En resumen, la consulta selecciona las filas de la tabla customer donde el cliente tiene exactamente 2 cuentas asociadas en la tabla account.

El operador EXISTS.

El operador que se utiliza con más frecuencia para construir condiciones que utilizan subconsultas correlacionadas es el operador `EXISTS`. Este operador se usa cuando se quiere identificar que existe una relación sin descartar la cantidad.

El operador `EXISTS` se utiliza para verificar si una subconsulta devuelve al menos un resultado. La función del operador `EXISTS` es evaluar una condición y devolver un valor booleano (verdadero o falso) basado en si la subconsulta tiene algún resultado.

La sintaxis básica del operador `EXISTS` es la siguiente:

```
EXISTS (subconsulta)
```

El operador EXISTS funciona de la siguiente manera:

- La subconsulta se ejecuta y devuelve un conjunto de resultados (puede ser una sola columna o varias columnas).
- El operador EXISTS verifica si hay al menos una fila en el conjunto de resultados de la subconsulta.
- Si la subconsulta tiene al menos un resultado, el operador EXISTS devuelve verdadero (TRUE). De lo contrario, devuelve falso (FALSE).

El operador EXISTS se utiliza comúnmente en combinación con la cláusula WHERE para filtrar filas en una consulta principal basada en la existencia de registros en una subconsulta.

```
mysql> SELECT a.account_id, a.product_cd, a.cust_id, a.avail_balance
    -> FROM account AS a
    -> WHERE EXISTS (SELECT 1
    -> FROM transaction AS t
    -> WHERE t.account_id = a.account_id
    -> AND t.txn_date = '2005-01-22');
Empty set (0,18 sec)
```

La anterior consulta selecciona las columnas account_id, product_cd, cust_id y avail_balance de la tabla account. La condición utilizada en la cláusula WHERE es

> EXISTS (SELECT 1 FROM transaction AS t WHERE t.account_id = a.account_id AND t.txn_date = '2005-01-22').

IMPORTANTE: Esta condición filtra las filas de la tabla account donde exista al menos un registro en la tabla transaction que cumpla las condiciones especificadas.

La subconsulta

> SELECT 1 FROM transaction AS t WHERE t.account_id = a.account_id AND t.txn_date = '2005-01-22'

se utiliza para verificar si hay al menos un registro en la tabla transaction que cumple las condiciones de igualdad entre account_id y la fecha de transacción '2005-01-22'.

En decir, la consulta selecciona las filas de la tabla account donde existe al menos un registro en la tabla transaction que tiene el mismo account_id y una fecha de transacción igual a '2005-01-22'.

IMPORTANTE: Si se observa la cláusula SELECT de la consulta, se verá que contiene un único literal (1), ya que la condición de la consulta contenedora sólo precisa conocer cuántas filas se han devuelto. Los datos reales devueltos por la consulta son irrelevantes. Sin embargo, la convención al usar EXISTS es especificar SELECT 1 o SELECT *.

En resumen, el operador EXISTS se utiliza para verificar la existencia de al menos un resultado en una subconsulta y devuelve verdadero o falso en función de esa existencia.

Existe la contraparte de EXISTS, que es NOT EXISTS, su uso es similar.

UNIONES DE TABLAS (JOINS).

Ya hemos visto la unión `INNER JOIN`. Ahora veremos otros métodos en los que se pueden unir las tablas, incluidas la unión externa y la unión cruza.

Uniones externas.

Recordemos por un momento la unión `INNER JOIN`. Analicemos el contenido de las tablas `account` y `customer`.

```
mysql> SELECT account_id, cust_id
    -> FROM account;
+------------+---------+
| account_id | cust_id |
+------------+---------+
|          1 |       1 |
|          2 |       1 |
|          3 |       1 |
|          4 |       2 |
|          5 |       2 |
|          7 |       3 |
|          8 |       3 |
|         10 |       4 |
|         11 |       4 |
|         12 |       4 |
|         13 |       5 |
|         14 |       6 |
|         15 |       6 |
|         17 |       7 |
|         18 |       8 |
|         19 |       8 |
|         21 |       9 |
|         22 |       9 |
|         23 |       9 |
|         24 |      10 |
|         25 |      10 |
|         27 |      11 |
|         28 |      12 |
|         29 |      13 |
+------------+---------+
24 rows in set (0,18 sec)
```

```
mysql> SELECT cust_id
    -> FROM customer;
+---------+
| cust_id |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
|       5 |
|       6 |
|       7 |
|       8 |
|     ...|
|      13 |
+---------+
13 rows in set (0,18 sec)
```

En la tabla account hay 24 cuentas para 13 clientes distintos, con ID de cliente que va desde el 1 al 13. En la tabla customer hay 13 filas con ID del 1 al 13, por lo tanto cada ID de cliente está incluido por lo menos una vez, en la tabla account.

IMPORTANTE: Cuando las dos tablas se unen en la columna cust_id, se espera que las 24 filas se incluyan en el conjunto de resultados:

```
mysql> SELECT a.account_id, c.cust_id
    -> FROM account AS a INNER JOIN customer AS c
    -> ON a.cust_id = c.cust_id;
+------------+---------+
| account_id | cust_id |
+------------+---------+
|          1 |       1 |
|          2 |       1 |
|          3 |       1 |
|          4 |       2 |
|          5 |       2 |
|          7 |       3 |
|          8 |       3 |
|         10 |       4 |
|         11 |       4 |
|         12 |       4 |
|         13 |       5 |
|         14 |       6 |
|         15 |       6 |
|         17 |       7 |
|         18 |       8 |
|         19 |       8 |
|         21 |       9 |
|         22 |       9 |
|         23 |       9 |
|         24 |      10 |
|         25 |      10 |
|         27 |      11 |
|         28 |      12 |
|         29 |      13 |
+------------+---------+
24 rows in set (0,18 sec)
```

¿Pero qué ocurre si se une la tabla account a otra tabla, por ejemplo business?

```
mysql> SELECT a.account_id, b.cust_id, b.name
    -> FROM account AS a INNER JOIN business AS b
    -> ON a.cust_id = b.cust_id;
+------------+---------+------------------------+
| account_id | cust_id | name                   |
+------------+---------+------------------------+
|         24 |      10 | Chilton Engineering    |
|         25 |      10 | Chilton Engineering    |
|         27 |      11 | Northeast Cooling Inc. |
|         28 |      12 | Superior Auto Body     |
|         29 |      13 | AAA Insurance Inc.     |
+------------+---------+------------------------+
5 rows in set (0,17 sec)
```

En lugar de 24 filas en el conjunto de resultados, ahora solo hay 5. Veamos la tabla business, para ver por qué esto es así:

```
mysql> SELECT cust_id, name
    -> FROM business;
+---------+------------------------+
| cust_id | name                   |
+---------+------------------------+
|      10 | Chilton Engineering    |
|      11 | Northeast Cooling Inc. |
|      12 | Superior Auto Body     |
|      13 | AAA Insurance Inc.     |
+---------+------------------------+
4 rows in set (0,17 sec)
```

De las 13 filas de la tabla `customer`, solo cuatro son clientes institucionales y como uno de éstos tiene dos cuentas, hay un total de cinco filas en la tabla `account` que están vinculadas a este tipo de clientes.

Pero, ¿qué ocurre si lo que se quiere es que la consulta devuelva todas las cuentas, pero que solo se incluya el nombre de la compañía si la cuenta está vinculada a un cliente institucional? Éste es un ejemplo de uso de una unión externa entre las tablas `account` y `business`, como por ejemplo:

```
mysql> SELECT a.account_id, a.cust_id, b.name
    -> FROM account AS a LEFT OUTER JOIN business AS b
    -> ON a.cust_id = b.cust_id;
+------------+---------+------------------------+
| account_id | cust_id | name                   |
+------------+---------+------------------------+
|          1 |       1 | NULL                   |
|          2 |       1 | NULL                   |
|          3 |       1 | NULL                   |
|          4 |       2 | NULL                   |
|          5 |       2 | NULL                   |
|          7 |       3 | NULL                   |
|          8 |       3 | NULL                   |
|         10 |       4 | NULL                   |
|         11 |       4 | NULL                   |
|         12 |       4 | NULL                   |
|         13 |       5 | NULL                   |
|         14 |       6 | NULL                   |
|         15 |       6 | NULL                   |
|         17 |       7 | NULL                   |
|         18 |       8 | NULL                   |
|         19 |       8 | NULL                   |
|         21 |       9 | NULL                   |
|         22 |       9 | NULL                   |
|         23 |       9 | NULL                   |
|         24 |      10 | Chilton Engineering    |
|         25 |      10 | Chilton Engineering    |
|         27 |      11 | Northeast Cooling Inc. |
|         28 |      12 | Superior Auto Body     |
|         29 |      13 | AAA Insurance Inc.     |
+------------+---------+------------------------+
24 rows in set (0,17 sec)
```

Una unión externa incluye todas las filas de una tabla así como los datos de la segunda tabla, solo si encuentra filas coincidentes. En este caso, todas las filas de la tabla `account` están incluidas, porque se especificó una unión externa por la izquierda (`LEFT OUTER JOIN`) y la tabla account está en el lado izquierdo de la definición de unión. IMPORTANTE: La columna `name` es `NULL` para todas las filas, excepto para los cuatro clientes institucionales.

En definitiva, LEFT OUTER JOIN se utiliza para combinar dos tablas basándose en una condición de igualdad, pero también incluye todas las filas de la tabla izquierda (tabla de referencia) en el resultado, incluso si no hay coincidencias en la tabla derecha (tabla combinada).

LEFT OUTER JOIN realiza lo siguiente:

1. Combina las filas de la tabla de referencia (tabla izquierda) con las filas correspondientes de la tabla combinada (tabla derecha) basándose en la condición de igualdad especificada.

2. Si no hay una coincidencia en la tabla derecha para una fila en la tabla izquierda, se incluye una fila con valores NULL para las columnas de la tabla derecha en el resultado.

3. Solo se incluyen las filas de la tabla izquierda que cumplen la condición de igualdad.

En resumen, el LEFT OUTER JOIN combina las filas de dos tablas basándose en una condición de igualdad y también incluye todas las filas de la tabla izquierda, incluso si no hay coincidencias en la tabla derecha.

Existe la contraparte unión externa por la derecha (RIGHT OUTER JOIN) que opera de un modo similar. Veamos un ejemplo de ambas y sus diferencias:

```
mysql> SELECT c.cust_id, b.name
    -> FROM customer AS c LEFT OUTER JOIN business AS b
    -> ON c.cust_id = b.cust_Id;
+---------+------------------------+
| cust_id | name                   |
+---------+------------------------+
|       1 | NULL                   |
|       2 | NULL                   |
|       3 | NULL                   |
|       4 | NULL                   |
|       5 | NULL                   |
|       6 | NULL                   |
|       7 | NULL                   |
|       8 | NULL                   |
|       9 | NULL                   |
|      10 | Chilton Engineering    |
|      11 | Northeast Cooling Inc. |
|      12 | Superior Auto Body     |
|      13 | AAA Insurance Inc.     |
+---------+------------------------+
13 rows in set (0,17 sec)

mysql> SELECT c.cust_id, b.name
    -> FROM customer AS c RIGHT OUTER JOIN business AS b
    -> ON c.cust_id = b.cust_Id;
+---------+------------------------+
| cust_id | name                   |
+---------+------------------------+
|      10 | Chilton Engineering    |
|      11 | Northeast Cooling Inc. |
|      12 | Superior Auto Body     |
|      13 | AAA Insurance Inc.     |
+---------+------------------------+
4 rows in set (0,18 sec)
```

En RIGHT OUTER JOIN el número de filas del conjunto de resulatdos lo determina el núemro de filas en la tabal business, motivo por el cual hay solo cuatro filas en el conjunto de resulatdos.

Autouniones externas.

IMPORTANTE: En una autounión una tabal se une consigo misma, utilizando alias o nombres de tabla diferentes para referirse a las dos instancias de la tabla. Se utiliza para combinar información de una tabla basándose en relaciones entre filas dentro de la misma tabla.

Veamos el contenido de la tabla employee.

```
mysql> SELECT emp_id, fname, lname, superrior_emp_id, title
    -> FROM employee;
+--------+----------+-----------+-----------------+--------------------+
| emp_id | fname    | lname     | superior_emp_id | title              |
+--------+----------+-----------+-----------------+--------------------+
|      1 | Michael  | Smith     |            NULL | President          |
|      2 | Susan    | Barker    |               1 | Vice President     |
|      3 | Robert   | Tyler     |               1 | Treasurer          |
|      4 | Susan    | Hawthorne |               3 | Operations Manager |
|      5 | John     | Gooding   |               4 | Loan Manager       |
|      6 | Helen    | Fleming   |               4 | Head Teller        |
|      7 | Chris    | Tucker    |               6 | Teller             |
|      8 | Sarah    | Parker    |               6 | Teller             |
|      9 | Jane     | Grossman  |               6 | Teller             |
|     10 | Paula    | Roberts   |               4 | Head Teller        |
|     11 | Thomas   | Ziegler   |              10 | Teller             |
|     12 | Samantha | Jameson   |              10 | Teller             |
|     13 | John     | Blake     |               4 | Head Teller        |
|     14 | Cindy    | Mason     |              13 | Teller             |
|     15 | Frank    | Portman   |              13 | Teller             |
|     16 | Theresa  | Markham   |               4 | Head Teller        |
|     17 | Beth     | Fowler    |              16 | Teller             |
|     18 | Rick     | Tulman    |              16 | Teller             |
+--------+----------+-----------+-----------------+--------------------+
18 rows in set (0,17 sec)
```

Hagamos un SELF-JOIN (autounión) tanto interno INNER JOIN como externo LEFT OUTER JOIN en la tabla. En el ejemplo siguiente se une employee con ella misma, para generar una lista de empleados y sus supervisores:

```
mysql> SELECT e.fname, e.lname,
    ->  e_mgr.fname AS mgr_fname, e_mgr.lname AS mgr_lname
    -> FROM employee AS e INNER JOIN employee AS e_mgr
    -> ON e.superior_emp_id = e_mgr.emp_id;
+----------+-----------+-----------+-----------+
| fname    | lname     | mgr_fname | mgr_lname |
+----------+-----------+-----------+-----------+
| Susan    | Barker    | Michael   | Smith     |
| Robert   | Tyler     | Michael   | Smith     |
| Susan    | Hawthorne | Robert    | Tyler     |
| John     | Gooding   | Susan     | Hawthorne |
| Helen    | Fleming   | Susan     | Hawthorne |
| Chris    | Tucker    | Helen     | Fleming   |
| ...      | ...       | ...       | ...       |
+----------+-----------+-----------+-----------+
17 rows in set (0,18 sec)
```

Esta consulta funciona bien excepto por un pequeño detalle, lso empleados que no tienen un supervisor se han quedado fuera del conjunto de resultados. Sin embargo, si se cambia la unión por una externa, el conjunto de resultados incluirá todos los empleados, incluidos aquellos sin supervisores:

```
mysql> SELECT e.fname, e.lname,
    ->  e_mgr.fname AS mgr_fname, e_mgr.lname AS mgr_lname
    -> FROM employee AS e LEFT OUTER JOIN employee AS e_mgr
    -> ON e.superior_emp_id = e_mgr.emp_id;
+----------+-----------+-----------+-----------+
| fname    | lname     | mgr_fname | mgr_lname |
+----------+-----------+-----------+-----------+
| Michael  | Smith     | NULL      | NULL      |
| Susan    | Barker    | Michael   | Smith     |
| Robert   | Tyler     | Michael   | Smith     |
| Susan    | Hawthorne | Robert    | Tyler     |
| John     | Gooding   | Susan     | Hawthorne |
| Helen    | Fleming   | Susan     | Hawthorne |
| Chris    | Tucker    | Helen     | Fleming   |
| Sarah    | Parker    | Helen     | Fleming   |
| Jane     | Grossman  | Helen     | Fleming   |
| Paula    | Roberts   | Susan     | Hawthorne |
| Thomas   | Ziegler   | Paula     | Roberts   |
| Samantha | Jameson   | Paula     | Roberts   |
| John     | Blake     | Susan     | Hawthorne |
| Cindy    | Mason     | John      | Blake     |
| Frank    | Portman   | John      | Blake     |
| Theresa  | Markham   | Susan     | Hawthorne |
| Beth     | Fowler    | Theresa   | Markham   |
| Rick     | Tulman    | Theresa   | Markham   |
+----------+-----------+-----------+-----------+
18 rows in set (0,17 sec)
```

Este conjunto de resulatdos incluye a Michael Smith, que es el presidente del banco y por lo tanto no tiene supervisor. La consulta utiliza una unión externa por la izquierda.

Uniones cruzadas (CROSS JOIN).

El producto cartesiano es básicamente el resultado de unir múltiples tablas sin especificar ninguna condición de unión.

`CROSS JOIN` combina todas las filas de una tabla con todas las filas de otra tabla, generando todas las combinaciones posibles. No se establece una condición de unión entre las tablas, lo que resulta en un producto cartesiano de las filas.

Veamos el contenido de las tablas `product` y `product_type`:

```
mysql> SELECT * FROM product_type;
+-----------------+-------------------------------+
| product_type_cd | name                          |
+-----------------+-------------------------------+
| ACCOUNT         | Customer Accounts             |
| INSURANCE       | Insurance Offerings           |
| LOAN            | Individual and Business Loans |
+-----------------+-------------------------------+
3 rows in set (0,18 sec)
```

```
mysql> SELECT product_cd, name, product_type_cd FROM product;
+------------+-------------------------+-----------------+
| product_cd | name                    | product_type_cd |
+------------+-------------------------+-----------------+
| AUT        | auto loan               | LOAN            |
| BUS        | business line of credit | LOAN            |
| CD         | certificate of deposit  | ACCOUNT         |
| CHK        | checking account        | ACCOUNT         |
| MM         | money market account    | ACCOUNT         |
| MRT        | home mortgage           | LOAN            |
| SAV        | savings account         | ACCOUNT         |
| SBL        | small business loan     | LOAN            |
+------------+-------------------------+-----------------+
8 rows in set (0,18 sec)
```

Hagamos su producto cartesiano con `CROSS JOIN`:

```
mysql> SELECT pt.name, p.product_cd, p.name
    -> FROM product AS p CROSS JOIN product_type AS pt;
+------------------------------+------------+-------------------------+
| name                         | product_cd | name                    |
+------------------------------+------------+-------------------------+
| Customer Accounts            | AUT        | auto loan               |
| Insurance Offerings          | AUT        | auto loan               |
| Individual and Business Loans | AUT        | auto loan               |
| Customer Accounts            | BUS        | business line of credit |
| Insurance Offerings          | BUS        | business line of credit |
| Individual and Business Loans | BUS        | business line of credit |
| Customer Accounts            | CD         | certificate of deposit  |
| Insurance Offerings          | CD         | certificate of deposit  |
| Individual and Business Loans | CD         | certificate of deposit  |
| Customer Accounts            | CHK        | checking account        |
| Insurance Offerings          | CHK        | checking account        |
| Individual and Business Loans | CHK        | checking account        |
| Customer Accounts            | MM         | money market account    |
| Insurance Offerings          | MM         | money market account    |
| Individual and Business Loans | MM         | money market account    |
| Customer Accounts            | MRT        | home mortgage           |
| Insurance Offerings          | MRT        | home mortgage           |
| Individual and Business Loans | MRT        | home mortgage           |
| Customer Accounts            | SAV        | savings account         |
| Insurance Offerings          | SAV        | savings account         |
| Individual and Business Loans | SAV        | savings account         |
| Customer Accounts            | SBL        | small business loan     |
| Insurance Offerings          | SBL        | small business loan     |
| Individual and Business Loans | SBL        | small business loan     |
+------------------------------+------------+-------------------------+
24 rows in set (0,18 sec)
```

Esta consulta genera el producto cartesiano de las tablas `product` y `porduct_type`, dando como resultado 24 filas (8*3).

En general, el `CROSS JOIN` se utiliza en situaciones específicas donde se requiere generar todas las combinaciones posibles o realizar análisis comparativos entre diferentes conjuntos de datos sin restricciones de unión específicas. Sin embargo, su uso es menos común en comparación con otros tipos de joins como el `INNER JOIN` o el `LEFT OUTER JOIN`, que se utilizan con más frecuencia para combinar tablas basándose en condiciones de unión específicas.

Unión natural.

NATURAL JOIN es un tipo de join que combina dos o más tablas basándose en las columnas que tienen los mismos nombres y tipos de datos. En otras palabras, el NATURAL JOIN realiza automáticamente una unión basada en las columnas coincidentes entre las tablas, sin necesidad de especificar explícitamente las condiciones de igualdad.

NATURAL JOIN realiza lo siguiente:

1. Examina las columnas de ambas tablas y encuentra las columnas que tienen los mismos nombres y tipos de datos.
2. Realiza una unión basándose en las columnas coincidentes de las tablas.
3. Devuelve las filas que tienen valores coincidentes en las columnas correspondientes.

Es importante tener en cuenta que el NATURAL JOIN no utiliza la cláusula ON para especificar una condición de igualdad entre las columnas, ya que asume que las columnas coincidentes son las que se deben utilizar para la unión. Esto significa que si hay columnas adicionales con los mismos nombres en las tablas que no deberían considerarse para la unión, se incluirán en el resultado.

Es importante tener precaución al utilizar el NATURAL JOIN, ya que puede generar resultados inesperados si las columnas coincidentes no se utilizan de manera correcta o si hay columnas adicionales con los mismos nombres que no deberían considerarse en la unión.

En resumen, el NATURAL JOIN es un tipo de join que combina tablas basándose en las columnas que tienen los mismos nombres y tipos de datos. Realiza automáticamente la unión basándose en las columnas coincidentes y no requiere especificar explícitamente las condiciones de igualdad.

```
mysql> SELECT a.account_id, a.cust_id, c.cust_type_cd, c.fed_id
    -> FROM account AS a NATURAL JOIN customer AS c;
+------------+---------+--------------+-------------+
| account_id | cust_id | cust_type_cd | fed_id      |
+------------+---------+--------------+-------------+
|          1 |       1 | I            | 111-11-1111 |
|          2 |       1 | I            | 111-11-1111 |
|          3 |       1 | I            | 111-11-1111 |
|          4 |       2 | I            | 222-22-2222 |
|          5 |       2 | I            | 222-22-2222 |
|          7 |       3 | I            | 333-33-3333 |
|          8 |       3 | I            | 333-33-3333 |
|         10 |       4 | I            | 444-44-4444 |
|         11 |       4 | I            | 444-44-4444 |
|         12 |       4 | I            | 444-44-4444 |
|        ... |     ... | ...          |         ... |
|         22 |       9 | I            | 999-99-9999 |
|         23 |       9 | I            | 999-99-9999 |
|         24 |      10 | B            | 04-1111111  |
|         25 |      10 | B            | 04-1111111  |
|         27 |      11 | B            | 04-2222222  |
|         28 |      12 | B            | 04-3333333  |
|         29 |      13 | B            | 04-4444444  |
+------------+---------+--------------+-------------+
24 rows in set (0,17 sec)
```

LÓGICA CONDICIONAL.

En algunas ocasiones puede que queramos que nuestras sentencias SQL realicen una acción u otra, dependiendo de los valores de algunas columnas o expresiones. Abordaremos ahora la cuestión de cómo escribir sentencias que se puedan comportar de forma diferente, dependiendo de los datos que se encuentren durante la ejecución de la sentencia.

Expresión CASE.

Por ejemplo, cuando se consulta la información sobre clientes, puede que queramos recuperar las columnas fname/lname de la tabla individual, o la columna name de la tabla business, dependiendo en qué tipo de cliente nos encontremos. Si se utilizan uniones externas se pueden devolver ambas cadenas y se puede dejar que el usuario averigüe cuál debe usar, según se indica en el siguiente ejemplo:

```
mysql> SELECT c.cust_id, c.fed_id, c.cust_type_cd,
    -> CONCAT(i.fname, ' ', i.lname) AS indivname,
    -> b.name AS business_name
    -> FROM customer AS c LEFT OUTER JOIN individual AS i
    -> ON c.cust_Id = i.cust_id
    -> LEFT outer JOIN business AS b
    -> ON c.cust_id = b.cust_id;
+---------+-------------+--------+-----------------+------------------------+
| cust_id | fed_id      | c_t_cd | indivname       | business_name          |
+---------+-------------+--------+-----------------+------------------------+
|       1 | 111-11-1111 | I      | James Hadley    | NULL                   |
|       2 | 222-22-2222 | I      | Susan Tingley   | NULL                   |
|       3 | 333-33-3333 | I      | Frank Tucker    | NULL                   |
|       4 | 444-44-4444 | I      | John Hayward    | NULL                   |
|       5 | 555-55-5555 | I      | Charles Frasier | NULL                   |
|       6 | 666-66-6666 | I      | John Spencer    | NULL                   |
|       7 | 777-77-7777 | I      | Margaret Young  | NULL                   |
|       8 | 888-88-8888 | I      | Louis Blake     | NULL                   |
|       9 | 999-99-9999 | I      | Richard Farley  | NULL                   |
|      10 | 04-1111111  | B      | NULL            | Chilton Engineering    |
|      11 | 04-2222222  | B      | NULL            | Northeast Cooling Inc. |
|      12 | 04-3333333  | B      | NULL            | Superior Auto Body     |
|      13 | 04-4444444  | B      | NULL            | AAA Insurance Inc.     |
+---------+-------------+--------+-----------------+------------------------+
13 rows in set (0,18 sec)
```

En la tabla anterior, un usuario puede consultar el valor de la columna `c_t_cd` y decidir si usa la columna `indiv_name` o la columna `business_name`. Sin embargo, en su lugar puede usar la lógica condicional mediante una expresión `CASE`, para determinar el tipo de cliente y devolver la cadena adecuada, según se muestra a continuación:

```
mysql> SELECT c.cust_id, c.fed_id,
    -> CASE
    ->   WHEN c.cust_type_cd = 'I' THEN
    ->     CONCAT(i.fname, ' ', i.lname)
    ->   WHEN c.cust_type_cd = 'B' THEN
    ->     b.name
    ->   ELSE 'Desconocido'
    -> END AS name
    -> FROM customer AS c LEFT OUTER JOIN individual AS i
    ->   ON c.cust_id = i.cust_id
    ->   LEFT OUTER JOIN business AS b
    ->   ON c.cust_id = b.cust_id;
```

```
+---------+-------------+------------------------+
| cust_id | fed_id      | name                   |
+---------+-------------+------------------------+
|       1 | 111-11-1111 | James Hadley           |
|       2 | 222-22-2222 | Susan Tingley          |
|       3 | 333-33-3333 | Frank Tucker           |
|       4 | 444-44-4444 | John Hayward           |
|       5 | 555-55-5555 | Charles Frasier        |
|       6 | 666-66-6666 | John Spencer           |
|       7 | 777-77-7777 | Margaret Young         |
|       8 | 888-88-8888 | Louis Blake            |
|       9 | 999-99-9999 | Richard Farley         |
|      10 | 04-1111111  | Chilton Engineering    |
|      11 | 04-2222222  | Northeast Cooling Inc. |
|      12 | 04-3333333  | Superior Auto Body     |
|      13 | 04-4444444  | AAA Insurance Inc.     |
+---------+-------------+------------------------+
13 rows in set (0,18 sec)
```

Esta versión de la consulta devuelve una única columna name, que se genera mediante la expresión condicional que empieza en la segunda línea de la consulta que, en este caso, comprueba el valor de la columna `cust_type_cd` y devuelve el nombre y los apellidos del individuo o la denominación o rezón aocial de la compañía.

Nota: La función `CONCAT` en MySQL se utiliza para concatenar o combinar múltiples cadenas en una sola cadena. Puedes proporcionar uno o más argumentos a la función `CONCAT`, que serán concatenados en el orden en que se especifican.

La mayoría de los principales servidores de bases de datos incluyen funciones integradas designadas para simular el funcionamiento de la sentencia `if-then-else`, que se encuentra en casi todos los lenguajes de programación. Por ejemplo la función `decode()` de Oracle o la función `if()` de MySQL. Las expresión condicionales `CASE` también están diseñadas para facilitar la lógica `if-then-else`, pero gozan de las siguientes dos ventajas frente a las funciones integradas:

1. La expresión condicional forma parte del estándar SQL y se ha implementado por Oracle Database, SQL Server y MySQL.

2. Las expresiones condicionales se construyen dentro de la gramática SQL y se pueden incluir en las sentencias `SELECT`, `INSERT`, `UPDATE` Y `DELETE`.

APÉNDICE: EXTENSIONES DE SELECT EN MYSQL AL LENGUAJE SQL.

MySQL incorpora una serie de extensiones al lenguaje SQL. Ahora trataremos las extensiones de SELECT, las más importantes. La implementación de MySQL de la sentencia SELECT incluye dos cláusulas adicionales: LIMIT y INTO OUTFILE.

La cláusula LIMIT.

Pude que no estemos interesados en todas las filas que devuelve una consulta. Para ayudarnos con esto la sentencia SELECT MySQL incluye la cláusula LIMIT, que permite restringir el número de filas devueltas por la consulta.

```
MariaDB > SELECT open_emp_id. COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          13 |        3 |
|          16 |        6 |
+-------------+----------+
4 rows in set (0.180 sec)

MariaDB > SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id
    -> LIMIT 3;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          13 |        3 |
+-------------+----------+
3 rows in set (0.179 sec)

MariaDB > SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id
    -> ORDER BY  how_many DESC
    -> LIMIT 3;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          16 |        6 |
+-------------+----------+
3 rows in set (0.181 sec)
```

Con esta última consulta averiguamos los tres cajeros que abrieron más cuentas, haciendo uso de ORDER BY y LIMIT.

Segundo parámetro opcional de la cláusula LIMIT.

Si pasamos dos parámetros a LIMIT, el primero designa en qué registro se debe empezar a añadir registros al conjunto de resultados final y el segundo designa

cuántos registros debe incluir. MySQL designa el primer registro como registro 0; por lo tanto para encontrar el mejor tercer cajero la consulta sería la siguiente:

```
MariaDB > SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id
    -> ORDER BY how_many DESC
    -> LIMIT 2, 1;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|          16 |        6 |
+-------------+----------+
1 row in set (0.177 sec)
```

A las consultas que incluyen la cláusula `LIMIT` junto con una cláusula `ORDER BY` se las denomina consultas de clasificación, porque permiten clasificar los datos.

La cláusula INTO OUTFILE.

MySQL incluye la cláusula INTO OUTFILE que permite indicar el nombre de un archivo sobre el que se escribirán los datos de salida de una consulta, es decir, MySQL permite que el resultado de una consulta se escriba en un archivo.

```
MariaDB [bank]> SELECT emp_id, fname, lname, start_date
    -> INTO OUTFILE '/tmp/emp_list.txt'
    -> FROM employee;
Query OK, 18 rows affected (0.177 sec)
```

La consulta escribirá en el archivo `/tmp/emp_list.txt` el siguiente contenido:

```
1       Michael Smith   2001-06-22
2       Susan   Barker  2002-09-12
3       Robert  Tyler   2000-02-09
...
14      Cindy   Mason   2002-08-09
15      Frank   Portman 2003-04-01
16      Theresa Markham 2001-03-15
17      Beth    Fowler  2002-06-29
18      Rick    Tulman  2002-12-12
```

Para poder ejecutar la consulta anterior con éxito debemos otorgarle permisos al usuario para escribir en disco. Lo hacemos del siguiente modo:

```
mysql -u root -p
```

```
MariaDB [(none)]> GRANT FILE ON *.* to 'user'@'%';
Query OK, 0 rows affected (0,010 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,005 sec)
```

El formato de salida por defecto utiliza tabuladores y retornos de carro después de cada registro. Si queremos tener un mayor control sobre el formato de salida, existen varias subcláusulas adicionales: `FIELDS TERMINATED` y `LINES TERMINATED`. Veamos un ejemplo del uso de una de ellas:

```
MariaDB > SELECT emp_id, fname, lname, start_date
    -> INTO OUTFILE '/tmp/emp_list_delim.txt'
    -> FIELDS TERMINATED BY ','
    -> FROM employee;
Query OK, 18 rows affected (0.177 sec)
```

Con la salida:

```
1,Michael,Smith,2001-06-22
2,Susan,Barker,2002-09-12
3,Robert,Tyler,2000-02-09
...
14,Cindy,Mason,2002-08-09
15,Frank,Portman,2003-04-01
16,Theresa,Markham,2001-03-15
17,Beth,Fowler,2002-06-29
18,Rick,Tulman,2002-12-12
```