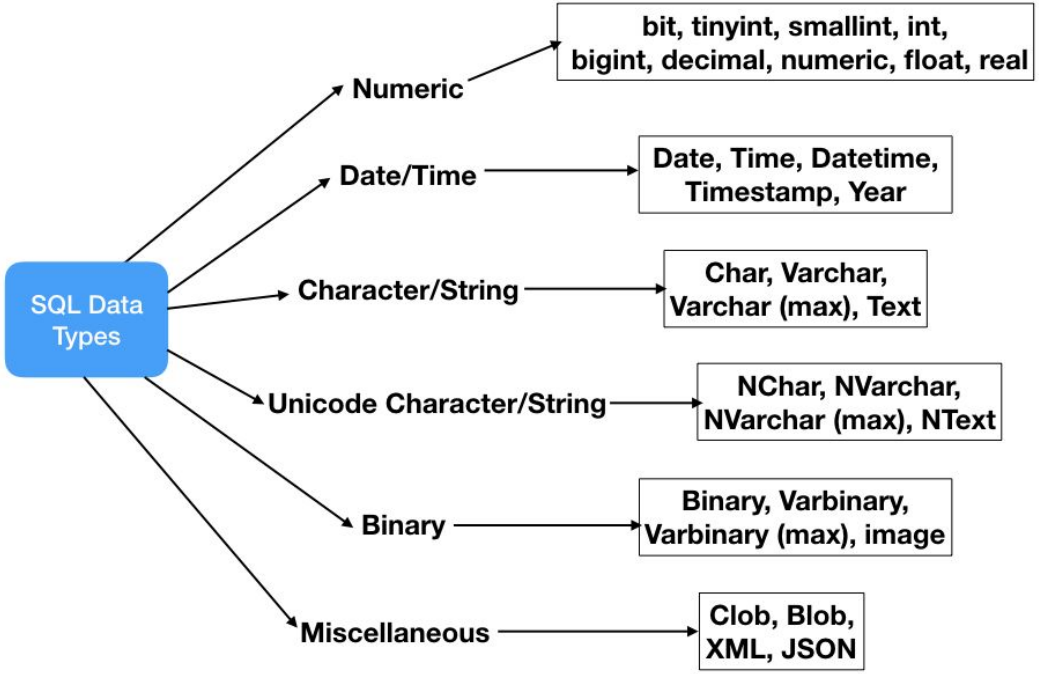
Bases de Datos

SQL

Tipos de Dato

Tipos de dato



Tipos de dato

- Los tipos de dato (data types) tienen importantes características sobre los tipos de valores que puede representar.
- Cuanto puede llegar a ocupar cada valor
  - **Longitud fija**: todos los valores de un tipo ocuparan la misma cantidad de espacio
  - **Longitud variable**: la cantidad de espacio depende del valor específico que está siendo almacenado
- Van a determinar cómo MySQL realiza comparaciones y ordenamientos para ese tipo de dato y también si puede ser indexado.

Tipos de dato

- La definición apropiada de los tipos de datos en una tabla es importante para la completa optimización de nuestras bases de datos.
- Siempre debemos usar únicamente el tipo y tamaño de datos que realmente necesitamos usar.
    - Por ejemplo, no definir un campo con una longitud de 10 caracteres si sabemos de antemano que solo usaremos 2 caracteres.

Tipos Numéricos

INT −Un entero de tamaño fijo que puede ser signed o unsigned En caso de ser signed, el rango permitido es desde -2147483648 a 2147483647. Si es unsigned, el rango permitido es de 0 a 4294967295.

TINYINT − Un entero corto que puede ser signed o unsigned. Si es signed, el rango permitido es de -128 a 127. Si es unsigned, el rango permitido es de 0 a 255.

SMALLINT − Un entero corto que puede ser signed o unsigned. Si es signed, el rango permitido es de -32768 a 32767. Si es unsigned, el rango permitido es de 0 a 65535.

MEDIUMINT − Es un entero de tamaño medio que puede ser signed o unsigned. Si es signed, el rango permitido es de -8388608 a 8388607. Si es unsigned, el rango permitido es de 0 a 16777215.

Tipos Numéricos

BIGINT − Un entero largo que puede ser signed o unsigned. Si es signed, el rango permitido es de -9223372036854775808 a 9223372036854775807. Si es unsigned, el rango permitido es de 0 a 18446744073709551615.

FLOAT(M,D) − Un número del tipo punto flotante que no puede ser unsigned. Podemos definir la longitud a desplegar (M) y el número de decimales (D) . Esto no es requerido y será por defecto 10,2, donde 2 es el número de decimales y 10 es el número total de dígitos (incluyendo decimales). La precisión decimal puede llegar hasta 24 posiciones para un FLOAT.

DOUBLE(M,D) Un número de punto flotante de doble precisión que no puede ser unsigned. Podemos definir la longitud a desplegar (M) y el número de decimales (D). Esto no es requerido y el valor por defecto será de 16,4, donde 4 es el número de decimales. La precisión decimal puede tomar hasta 53 posiciones para un DOUBLE. REAL es un sinónimo para DOUBLE.

NUMERIC(M,D) | **DECIMAL(M,D) −** Un número de tipo flotante desempaquetado que no puede ser unsigned. La definición de la longitud a desplegar (M) y el número de decimales (D) es requerido. NUMERIC es un sinónimo para DECIMAL. **Mejor que FLOAT o DOUBLE (estos últimos sólo reservados para programas matemáticos serios).**

Tipos Texto

CHAR(N) − Una cadena de longitud fija entre 1 y 255 caracteres. Definir una longitud no es obligatorio, por defecto es 1.

VARCHAR(N) − Una cadena de longitud variable entre 1 y 255 caracteres. Por ejemplo, VARCHAR(25). Es obligatorio definir una longitud para este tipo de datos.

BLOB o TEXT − Es un campo con una longitud máxima de 65535 caracteres. BLOBs significa "Binary Large Objects" y son usados para almacenar grandes cantidades de datos binarios, tales como imágenes u otros tipos de archivos. Campos definidos como TEXT permiten manejar grandes cantidades de datos. La diferencia entre las dos es que el ordenamiento y comparación sobre los datos ordenados son **case sensitive** en BLOBs y no son **case sensitive** en los campos TEXT. En ambos casos no es requerido especificar la longitud.

TINYBLOB o TINYTEXT − Un tipo de datos BLOB o TEXT con una longitud máxima de 255 caracteres. No es requerido especificar longitud.

MEDIUMBLOB o MEDIUMTEXT − Una columna BLOB o TEXT con una longitud máxima de 16777215 caracteres.

Tipos Texto

LONGBLOB o LONGTEXT – Una columna BLOB o TEXT con una longitud máxima de 4294967295 caracteres. No es requerido especificar la longitud.

ENUM – Es una enumeración, lo cual permite almacenar una lista. Cuando definimos un ENUM, estamos creando una lista de elementos, por ejemplo, si queremos que un campo contenga un campo con los valores de "A" o "B" o "C", deberíamos definir un tipo ENUM ('A', 'B', 'C') y solo esos valores o NULL podrían llenar ese campo.

Tipos Fecha

DATE − Una fecha en formato YYYY-MM-DD , con valores entre 1000-01-01 y 9999-12-31. Por ejemplo, Diciembre 30th, 1973 sería almacenado como 1973-12-30.

DATETIME −Una combinación de fecha y tiempo en formato YYYY-MM-DD HH:MM:SS , con valores entre 1000-01-01 00:00:00 y 9999-12-31 23:59:59. Por ejemplo, 3:30 en la tarde del 30 de Diciembre, 1973 debería ser almacenado como 1973-12-30 15:30:00.

TIMESTAMP − Un timestamp en donde se almacenaría por ejemplo una fecha/hora como 3:30 en la tarde del 30th de Diciembre de 1973 en la siguiente forma 19731230153000 (YYYYMMDDHHMMSS).

TIME − Almacena la hora en un formato HH:MM:SS format.

YEAR(M) − Almacena un año en un formato de 2-digitos o 4-digitos. Si la longitud es especificada como 2 (por ejemplo YEAR(2)), YEAR podría estar entre 1970 a 2069 (70 to 69). Si la longitud es especificada como 4, entonces YEAR puede estar entre 1901 a 2155. La longitud por defecto es 4.

Otros Tipos

BOOL | BOOLEAN - Verdadero o falso. Cero es considerado falso y cualquier otro valor diferente de cero (incluido NULL) es considerado verdadero. Se puede usar las constantes TRUE y FALSE para evaluar 1 y 0 respectivamente.

JSON | XML

NULL: A nivel de BBDD y en MySQL en particular se puede definir NULL como un tipo de datos no definido, por lo general esto significa que **no tiene valor o dicho valor es desconocido** o no es aplicable. Podemos insertar valores NULL dentro de tablas y recuperarlos, así como probar si un valor es NULL.

Normas de Estilo

Normas de Estilo SQL

- Nombres de tablas y campos:
  - Siempre singular
  - En minúsculas y snake case: nombre, primer_apellido, id, dni, telefono, email
  - Siempre únicos
  - Máximo 30 caracteres
  - Sin símbolos excepto _
  - Mejor en inglés
- Palabras reservadas en mayúsculas: SELECT, CREATE, UPDATE, DELETE, SUM, AVG, CONSTRAINT...
- Comentarios: -- y /* */

Palabras Reservadas

- https://www.w3schools.com/sql/sql_ref_keywords.asp

Crear las Tablas y sus Relaciones

Sentencias del LDD (Lenguaje de Definición de Datos)

Operaciones sobre la estructura:

- **CREATE TABLE**: Crear tablas, campos de índices
- **ALTER TABLE**: Modificar tablas al agregar o cambiar la definición de los campos campos.
- **DROP TABLE**: Eliminar tablas e índices

CREATE TABLE

- Es necesario empezar por las tablas independientes
- Las restricciones de relación se deben cumplir en el momento que se crea la tabla

CREATE TABLE nombre_tabla

Restricciones (Constraints)

- **NOT NULL**: Elemento del que siempre tiene que existir un valor en cada registro. Da un error si se trata de hacer una inserción sin ese campo.
- **UNIQUE**: aquellos atributos no pertenecientes a la clave que no deben tener nunca valores repetidos. Da un error si se trata de insertar un valor repetido.
- **AUTO_INCREMENT**: Genera de forma automática un valor secuencial que nunca se repetirá. Lo genera el motor SQL. Se puede establecer el valor inicial. Nosotros no lo vamos a poder modificar. Muy importante.
- **DEFAULT**: Valor por defecto
- **UNSIGNED**: Sólo valores positivos (sin signo, valores absolutos)

ALTER TABLE

Existen situaciones en las que requerimos alterar la estructura de una tabla previamente creada, este puede incluir algo tan sencillo como añadir una columna o eliminar una existente o cambiar el tipo de dato de una columna.

Si requerimos cambiar el tipo de datos de una columna lo podemos hacer, sin embargo es muy importante saber que la conversión puede tener un impacto sobre los datos que ya tenemos almacenados en ese campo en esa tabla.

El cambio tiene que ser compatible con la información que ya está almacenada.

ALTER TABLE

- Añadir una nueva columna:

  ALTER TABLE nombre_tabla
  ADD nombre_columna DATATYPE ;

- Quitar una columna:

  ALTER TABLE nombre_tabla
  DROP nombre_columna;

- Modificar una columna existente

  ALTER TABLE nombre_tabla
  MODIFY COLUMN nombre_columna DATATYPE ;

ALTER TABLE

- Recomendación: Si es imprescindible hacer un cambio en el tipo de datos de una tabla existente en producción, será necesario avisar de que el servidor no va a estar disponible (planificar downtime) para evitar problemas vinculados a la NO DISPONIBILIDAD
- Cuando se cambia un dato se crea un bloqueo de esquema (scheme lock)

DROP TABLE

- Elimina la tabla de la base de datos

DROP TABLE nombre_tabla

Manipular los datos

Sentencias del LMD (Lenguaje de Manipulación de Datos)

Operaciones sobre la información:

- **SELECT**: buscar entre los datos
- **INSERT**: añadir nuevos datos
- **UPDATE**: actualizar los datos existentes
- **DELETE**: eliminar todos o parte de los datos

No te olvides de poner el where en el delete from

https://www.youtube.com/watch?v=i_cVJgIz_Cs

INSERT

En una sentencia INSERT de SQL debes especificar la tabla dentro de la cual deseamos insertar una fila de datos y los valores a insertar. La sentencia INSERT tiene diferentes formas:

- INSERT INTO tbl_name VALUES (value1, value2, ...);
- INSERT INTO tbl_name (col_name1, col_name2, ...) VALUES (value1, value2, ...);
- INSERT INTO tbl_name SET col_name1 = value1, col_name2 = value2, ... ;

INSERT

- Para insertar datos habrá que tener en cuenta la estructura de la tabla.
- Habrá que respetar los tipos de dato que hayamos declarado
- No se aceptará que en un campo no se inserte información si este está creado como NOT NULL, en caso de que no se especifique al crearlo sí que permitirá que no se inserten datos en ese campo.
- Si usamos valores que están fuera de rango o no se corresponden con el tipo de datos obtendremos errores como los siguientes:

```
mysql> INSERT INTO Ventas(IdProducto,FechaVenta,IdVendedor,Monto) VALUES(123456789012345,'2016-01-01 05:01:20',2,45674.89);
ERROR 1264 (22003): Out of range value for column 'IdProducto' at row 1
mysql>
```

UPDATE

- Una vez que tenemos datos almacenados en nuestras tablas lo normal es tener que actualizarlos a menudo

UPDATE nombre_tabla
SET columna_a_modificar
WHERE filtro o predicado de las columnas que queremos cambiar;

DELETE

- Sirve para eliminar determinados registros.
- DANGER DANGER: hay que establecer un filtro para evitar eliminar **TODAS** las filas almacenadas en esas tabla

UPDATE nombre_tabla
SET columna_a_modificar
WHERE filtro o predicado de las columnas que queremos cambiar;

SELECT

- Sirve para obtener información.
- Tiene importantes modificadores que se pueden combinar para permitirnos obtener la información en la cual estamos interesados.

SELECT campo FROM tabla WHERE condicion ORDER BY secuencia de clasificación;

SELECT *

- El carácter * permite devolver TODOS los registros de una tabla
- Si son muchos registros, tardará mucho

SELECT campo1, campo2

- Se pueden poner los nombres de los campos que se quieren seleccionar
- Estos campos pueden pertenecer a varias tablas
- Se pueden nombrar anteponiendo el nombre de la tabla a la que pertenecen: nombre_tabla.nombre_campo
- Se le puede dar un nombre a la presentación de datos de la consulta: nombre_campo “Nombre del Campo”
- Se pueden poner alias a los nombres de los campos: nombre_campo as n_campo

SELECT * FROM nombre_tabla1, nombre_tabla2...

- FROM nos permite elegir qué tabla o tablas vamos a hacer que participen en la consulta.
- Si es sólo una tabla se pone directamente el nombre de la tabla
- Si son varias tablas en las que vamos a buscar o que vamos a relacionar, se podrán todos los nombres de las tablas que participan separados por comas

SELECT * FROM nombre_tabla WHERE

- WHERE permite añadir modificadores para filtrar las filas retornadas por FROM.
- Solo las filas para las cuales la expresión lógica evalúa a TRUE o verdadero son retornados por WHERE.
- Se pueden encadenar o combinar modificadores usando AND, OR o NOT
- Muy importante en términos de rendimiento de la consulta
- Los filtros ayudarán a que la consulta se realice mejor, de forma más quirúrgica con respecto a la información que queremos obtener
- También reducirán el tráfico de red (**network traffic**) creados por todas las posibles filas devueltas en la consulta.

Técnicas de filtrado de datos

Modificadores para consultas SELECT

MODIFICADORES

SELECT [ALL | DISTINCT]

<*nombre_campo*> [{,<*nombre_campo*>}]

FROM <*nombre_tabla*> [{,<*nombre_tabla*>}]

[**WHERE** <*condicion*> [{ **AND**|**OR** <condicion>}]]

[**GROUP BY** <*nombre_campo*> [{,<*nombre_campo* >}]]

[**HAVING** <*condicion*>[{ **AND**|**OR** <condicion>}]]

[**ORDER BY** <*nombre_campo*>|<indice_campo> [**ASC** | **DESC**]

[{,<*nombre_campo*>|<indice_campo> [**ASC** | **DESC**]}]]

COUNT

- Cuenta el número de tuplas devueltas por una consulta
- Usar count(*) es mucho más lento que usar count(id)

```
SELECT COUNT(*) FROM earthquake;
```

AND y OR

```
select nombre, apellido1, apellido2 from personas
  where (edad>25 AND edad<50);
select nombre, apellido1, apellido2 from personas
  where (nombre="Luis" OR nombre="Pedro");
```

ORDER BY

- Ordena los registros devueltos como resultado de la consulta
- Sirve para presentar la información de forma ordenada
- Es la última en todas las consultas
- Es la última en ser procesada por el motor
- Se puede ordenar por más de una columna
- Existen los modificadores:
  - **ASC**: ordena la información de menor a mayor
  - **DESC**: ordena la información de mayor a menor

ORDER BY: Ejemplos

- Ordenar por edad descendente (los mayores arriba)

```
select * from personas order by edad desc;
```

- Ordenar por varios campos

```
select * from personas order by apellido1, apellido2,
nombre;
```

LIMIT

- Se utiliza para delimitar el número de resultados devueltos

```
1  SELECT *
2  FROM earthquake
3  WHERE occurred_on >= '2010-01-01' AND occurred_on <= '2010-12-31'
4  ORDER BY magnitude DESC
5  LIMIT 1;
```

LIKE

- Filtra los resultados devueltos para obtener sólo los valores que tengan parte de la cadena dada

```
select * from personas where apellido1 like "g%";
```

expresión *like*	significado
"g%"	Que empiece por **g**
"%g"	Que termine por **g**
"%g%"	Que tenga una **g**
"_____"	Que tenga cinco caracteres

MIN y MAX

- Se utilizan para hallar el menor/mayor valor de una serie de valores dada

```
1  SELECT MIN(occurred_on), MAX(occurred_on)
2  FROM earthquake;
```

GROUP BY

- Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.

```
select provincia from localidades group by provincia;
```

```
select provincia, count(*) from localidades
  group by provincia;
```

ALL y DISTINCT

- ALL después de SELECT indicará que se quieren seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.
- DISTINCT después de SELECT suprimirá aquellas filas del resultado que tengan igual valor que otras.

```
SELECT nombre FROM personas
```

nombre
ANTONIO
LUIS
ANTONIO

```
SELECT DISTINCT nombre FROM personas
```

nombre
ANTONIO
LUIS

HAVING

- La sentencia HAVING permite especificar un predicado o filtro a nivel de grupos (en lugar de filtros individuales). Solo los grupos para los cuales la expresión lógica en la cláusula HAVING evalúa un valor TRUE o Verdadera por la fase HAVING del procesamiento lógico.

Subconsulta

Una subconsulta es una consulta SQL insertada dentro de una consulta mas grande o externa.

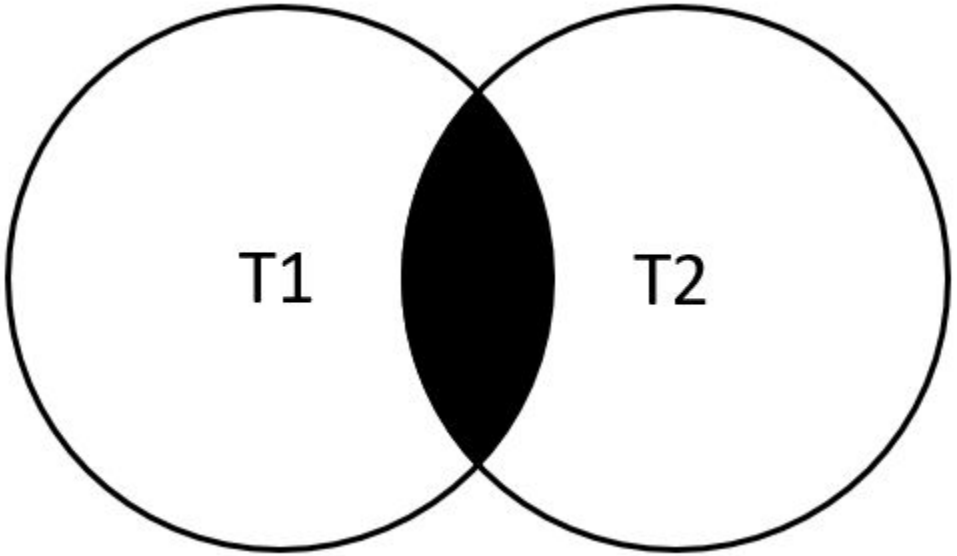
Puede estar enlazada dentro de una cláusula SELECT, INSERT, UPDATE, DELETE, SET, o incluso dentro de otra subconsulta.

Una subconsulta se suele añadir dentro de la cláusula WHERE de otra sentencia SELECT.

JOINS

JOIN

- Sirve para obtener datos de dos o más tablas en una única consulta
- Las más comunes son INNER JOIN (también llamado JOIN)

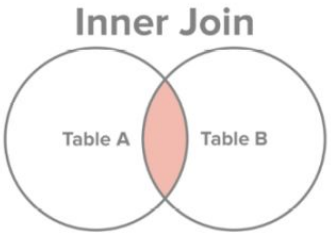


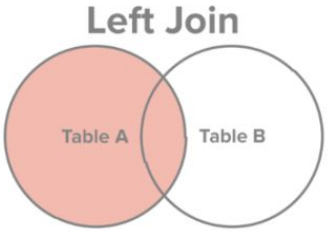
JOIN

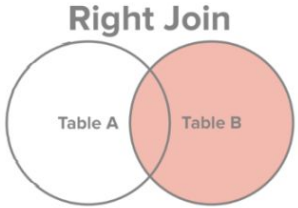
- **(INNER) JOIN**: se devuelven las filas de ambas tablas que tienen valores relacionados que cumplen las condiciones

- **LEFT (OUTER) JOIN**: devuelve todas las filas de la tabla de la izquierda y sólo las filas que cumplen las condiciones de la tabla de la derecha

- **RIGHT (OUTER) JOIN**: devuelve las filas de la tabla de la derecha y sólo las filas que cumplen las condiciones de la tabla de la izquierda







JOIN

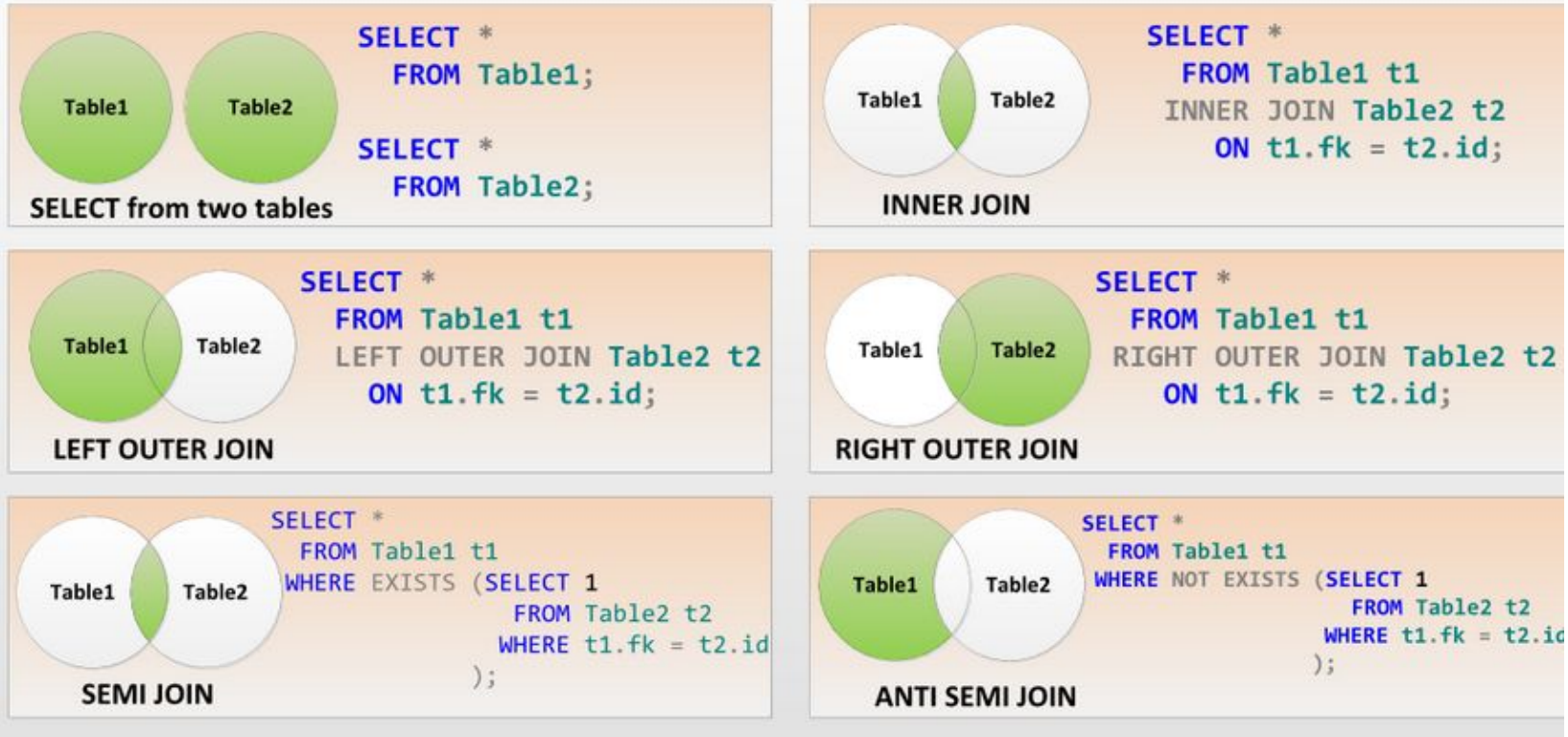


Table1
Table2
SELECT *
FROM Table1;
SELECT *
FROM Table2;
SELECT from two tables
Table1
Table2
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id;
INNER JOIN
Table1
Table2
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
LEFT OUTER JOIN
Table1
Table2
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id;
RIGHT OUTER JOIN
Table1
Table2
SELECT *
FROM Table1 t1
WHERE EXISTS (SELECT 1
FROM Table2 t2
WHERE t1.fk = t2.id
);
SEMI JOIN
Table1
Table2
SELECT *
FROM Table1 t1
WHERE NOT EXISTS (SELECT 1
FROM Table2 t2
WHERE t1.fk = t2.id
);
ANTI SEMI JOIN

JOIN

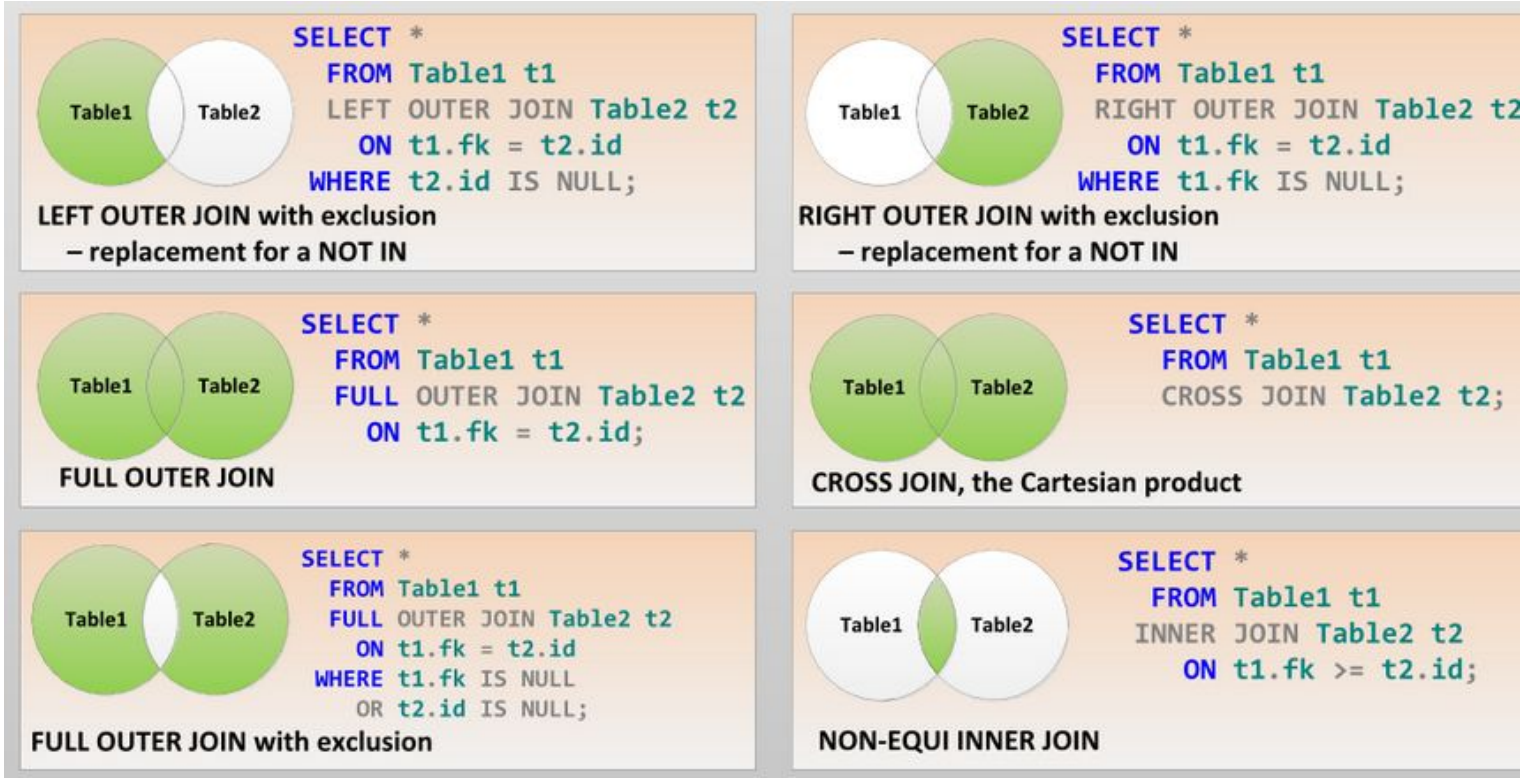
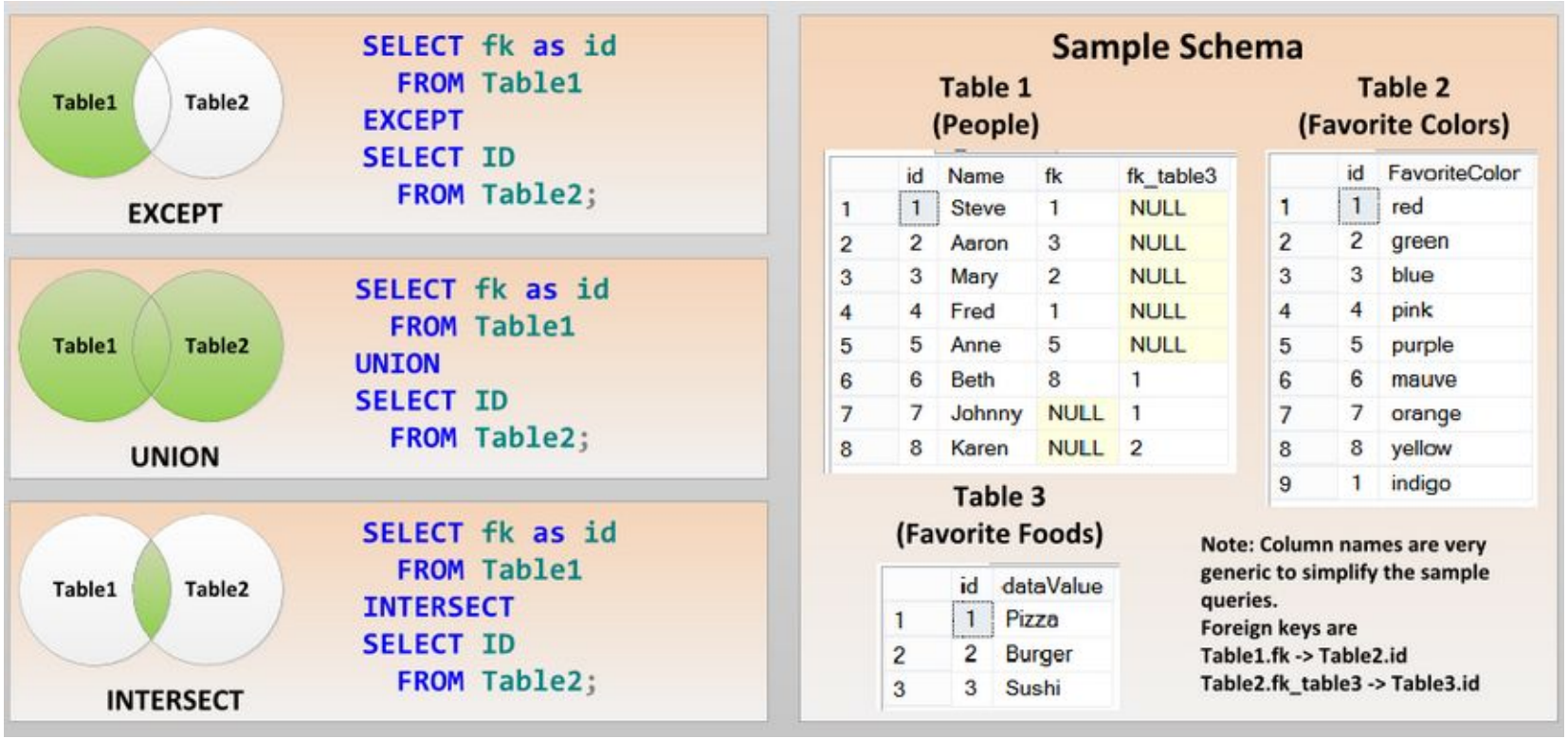


Table1
Table2
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t2.id IS NULL;
LEFT OUTER JOIN with exclusion
– replacement for a NOT IN
Table1
Table2
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t1.fk IS NULL;
RIGHT OUTER JOIN with exclusion
– replacement for a NOT IN
Table1
Table2
SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
ON t1.fk = t2.id;
FULL OUTER JOIN
Table1
Table2
SELECT *
FROM Table1 t1
CROSS JOIN Table2 t2;
CROSS JOIN, the Cartesian product
Table1
Table2
SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t1.fk IS NULL
OR t2.id IS NULL;
FULL OUTER JOIN with exclusion
Table1
Table2
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk >= t2.id;
NON-EQUI INNER JOIN

JOIN



	id	Name	fk	fk_table3
1	1	Steve	1	NULL
2	2	Aaron	3	NULL
3	3	Mary	2	NULL
4	4	Fred	1	NULL
5	5	Anne	5	NULL
6	6	Beth	8	1
7	7	Johnny	NULL	1
8	8	Karen	NULL	2

	id	FavoriteColor
1	1	red
2	2	green
3	3	blue
4	4	pink
5	5	purple
6	6	mauve
7	7	orange
8	8	yellow
9	1	indigo

	id	dataValue
1	1	Pizza
2	2	Burger
3	3	Sushi

JOIN

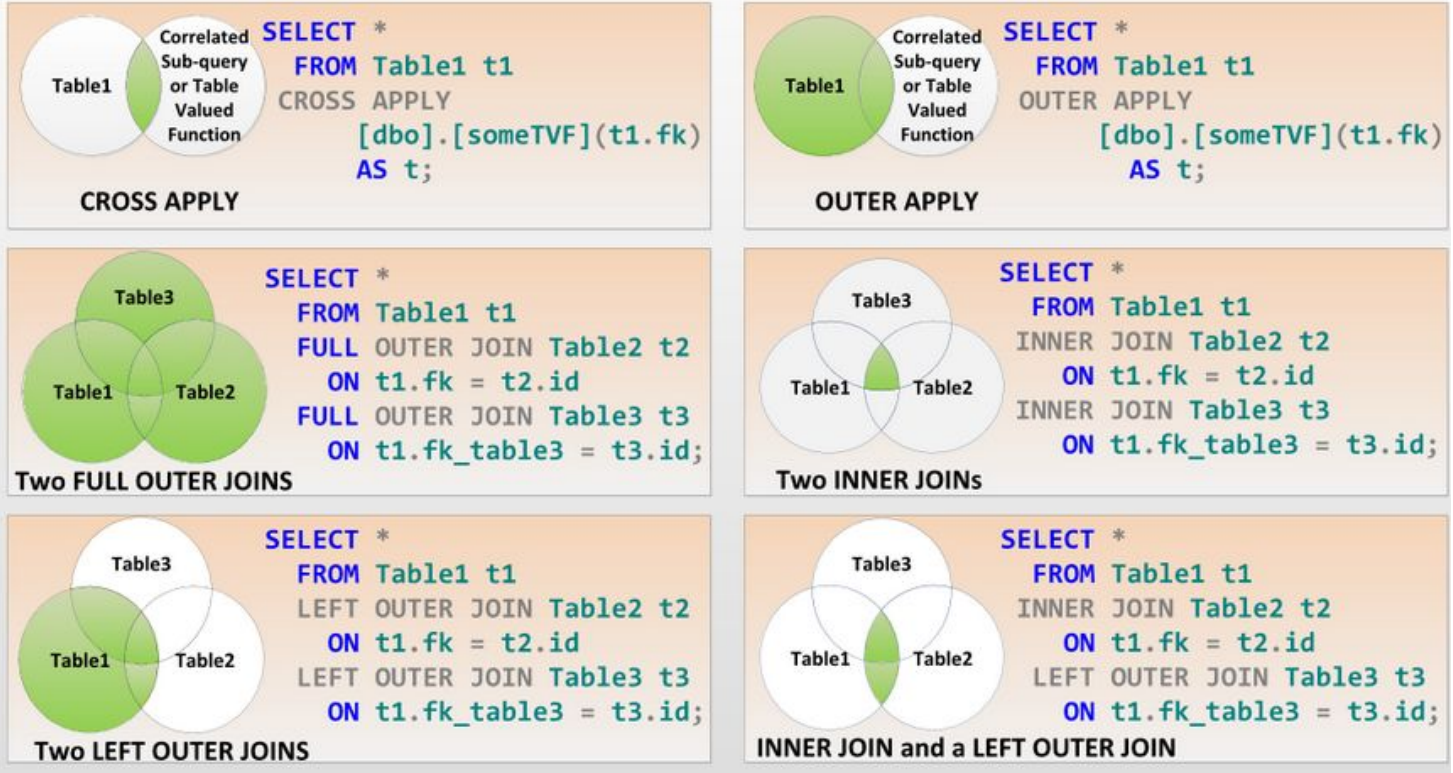


Table1
Correlated Sub-query or Table Valued Function
SELECT *
FROM Table1 t1
CROSS APPLY
[dbo].[someTVF](t1.fk)
AS t;
CROSS APPLY
Table1
Correlated Sub-query or Table Valued Function
SELECT *
FROM Table1 t1
OUTER APPLY
[dbo].[someTVF](t1.fk)
AS t;
OUTER APPLY
Table3
Table1
Table2
SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
ON t1.fk = t2.id
FULL OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
Two FULL OUTER JOINS
Table3
Table1
Table2
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
INNER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
Two INNER JOINs
Table3
Table1
Table2
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
Two LEFT OUTER JOINS
Table3
Table1
Table2
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
INNER JOIN and a LEFT OUTER JOIN

Funcionalidades de la BD

INDEX

- Se pueden crear índices sobre uno o varios campos de una tabla
- Los índices sirven para acelerar consultas sobre una tabla
- Se mejora mucho el rendimiento en las consultas
- Se usa cuando las consultas se suelen hacer sobre unas columnas específicas o los valores suelen obtenerse en un orden concreto
- Se crea como un índice de contenidos de la tabla
- Cualquier búsqueda que contenga ese campo indexado será más rápida

```
CREATE INDEX person_first_name_idx
ON person (first_name);
```

Backup: MYSQLDUMP

- Sirve para realizar una copia de seguridad de la BD en un archivo *.sql*
- Permite “llevarse la BD a otro ordenador”
- Vuelca todo el contenido a un archivo (Estructura + Datos)
- Pasos:
  - inicio > “cmd”
  - *mysqldump -u nombre_usuario -p nombre_bd > nombre_bd.sql*
  - insertar password

```
C:\Program Files\MySQL\MySQL Shell 8.0>mysqldump -u root -p camping > camping.sql
Enter password: ****
```