Data Analytics

-

SQL

-

Sentencias I

gamma tech school

SQL - SENTENCIAS I

Índice

1.Sentencia - cláusula

Sentencia

Instrucción con la que le decimos a la base de datos lo que queremos hacer.

- Recupera datos
- Inserta datos
- Actualiza lo que tienes
- Elimina datos
- Crea tablas
- y mucho más

Características

- El lenguaje es estructurado para que la base de datos lo pueda entender.
- Son precisas y específicas.
- Terminan con un punto y coma **;**

1.Sentencia - cláusula

Cláusula

Condición o instrucción adicional que especifica detalles sobre cómo debe ejecutarse una operación en la base de datos.

Estas instrucciones se incluyen en la sentencia para refinar el resultado.

Cláusulas principales

- **WHERE**: condición para filtrar
- **GROUP BY**: agrupa los registros
- **HAVING**: filtra las agrupaciones
- **ORDER BY**: ordena los registros
- **LIMIT**: restringe las filas devueltas de la consulta

Características

- Se usan para personalizar el comportamiento de una sentencia
- pueden combinarse y anidarse
- mejoran la eficiencia y relevancia de una query

2. Definición de datos - DDL

Definición de datos

- Definir y manejar todas las estructuras de la base de datos
- Crear tablas
- Definir columnas
- Definir tipos de datos de cada columna
- Relaciones entre tablas

Es como si fuera el arquitecto, quien dibuja los planos.

Sentencias principales:

- create: atención a los tipos de datos, PK FK
- alter: modificar una tabla existente
- drop: eliminar
- truncate: borrar contenido de una tabla
- rename

2. Definición de datos - DDL

Crear base de datos

```
CREATE DATABASE nombre_db;
```

Eliminar base de datos

```
DROP DATABASE IF EXISTS nombre_db;
```

Crear tabla si no existe

```
CREATE TABLE IF NOT EXISTS nombre_tabla (
  id DATA_TYPE PRIMARY KEY,
  atributo1 DATA_TYPE RESTRICCIONES,
  atributo2 VARCHAR(3)NOT NULL,
  atributo3 INT NULL,
  atributo4 DATE,
  es_cliente_activo BOOLEAN
);
```

2. Definición de datos - DDL

Crear alteraciones

```
ALTER TABLE nombre_tabla_A (
  ADD FOREIGN KEY(tabla_b),
  REFERENCES(columna_tabla_b),
  ON DELETE SET NULL;
```

Renombrar una tabla

```
RENAME TABLE nombre_tabla TO nuevo_nombre;
```

- Puede hacerse a través de la sentencia ALTER TABLE.
- RENAME puede renombrar varias tablas en la misma sentencia.
- ALTER TABLE ... RENAME sólo puede renombrar una tabla por sentencia.

```
ALTER TABLE nombre_tabla RENAME nuevo_nombre;
```

Borrar la base de datos

```
DROP DATABASE nombre_base_datos;
```

Borrar una tabla

```
DROP TABLE IF EXISTS nombre_tabla;
```

- CUIDADO. 'DROP' va a eliminar todo.
- Para poder usarlos se deben tener permisos.
- DROP TABLE: va a eliminar la tabla definida y toda su información.

Vaciar una tabla

```
TRUNCATE TABLE nombre_tabla;
```

- Elimina y vuelve a crear la tabla vacía.
- Requiere los permisos de DROP.
- Similar a la sentencia DELETE.

3. Manipulación de datos - DML

Data Manipulation Language

- Permite realizar diferentes acciones sobre los datos que se encuentran en la base de datos.
  - Recuperar
  - Almacenar
  - Modificar
  - Eliminar
  - Insertar
  - Actualizar

Sentencias principales

- SELECT: consultar registros
- INSERT: cargar datos a la base de datos
- UPDATE: modifica los valores de los campos seleccionados
- DELETE: elimina datos de una tabla

Operaciones CRUD

- **C**reate: crea nuevos datos,
- **R**ead: lee los datos.
- **U**pdate: actualiza los datos.
- **D**elete: borra los datos.

3. Manipulación de datos - DML

Añadir datos a una tabla

```
UPDATE nombre_tabla
SET columna = registro, producto = ‘bicicleta’
WHERE condición;
```

Modificar datos de una tabla

```
INSERT INTO nombre_tabla (columna1, columna 2)
VALUES(valor1, valor2);
```

Borrar la base de datos

```
DELETE FROM nombre_tabla WHERE condición;
```

ATENCIÓN
Si no filtras los datos con una condición borrarás la tabla entera.

4. SELECT - query

SELECT

Sentencia con la que se abre una **consulta** a la base de datos. Recibe un nombre especial: **query**.

Se puede sacar información de una o varias tablas, hacer operaciones de UNION, subqueries...

- SELECT: para indicar las columnas, separadas por comas ,
- FROM: para indicar las tablas a las que hace referencia.
  - JOIN: para unir la tabla seleccionada con otras.
- WHERE: filtrar por condiciones.
- GROUP BY: agrupar datos cuando usamos operaciones de agregación.
  - HAVING: parecido al WHERE, para filtrar en las agrupaciones.
- ORDER BY: indicaciones para ordenar los datos.
- LIMIT: limitar la cantidad de filas del resultado.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

4. SELECT - query

ORDEN DE EJECUCIÓN

La sentencia SELECT se compone de varias cláusulas siguiendo la siguiente secuencia:

1. FROM: obtiene la fuente de los datos. Si hay subquery es primero se lee la subquery.
2. JOIN: hace las combinaciones con las otras tablas.
3. WHERE: filtra según las condiciones que se indiquen.
4. GROUP BY: agrupa los datos.
5. HAVING: filtra los datos agrupados.
6. SELECT: selecciona los datos indicados.
7. DISTINCT: elimina los datos duplicados.
8. UNION, EXCEPT, INTERCEPT: aplica la operación indicada.
9. ORDER BY: ordena los datos del resultado.
10. LIMIT: limita la cantidad de filas del resultado.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

4. SELECT - query

Seleccionar todos los datos de una tabla

```
SELECT *
FROM  tabla;
```

Selecciona columnas específicas de una tabla

```
SELECT columna1, columna2, columna3
FROM tabla;
```

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

4.1. WHERE - query

Selección con condicional

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna = ‘valor’;
```

Selección con condicional y AND

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna = ‘valor’ AND columna = ‘valor’;
```

Selección con condicional y OR

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna = ‘valor’ OR columna = ‘valor’;
```

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

4.1. WHERE - query

Selección con condicional, IN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna IN (‘valor1’, ‘valor2’);
```

Selección con condicional, NOT IN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna NOT IN (‘valor1’, ‘valor2’);
```

Selección con condicional, NULL

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna IS/IS NOT NULL;
```

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

4.1. WHERE - query

Selección con condicional, IN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna IN (‘valor1’, ‘valor2’);
```

Selección con condicional, NOT IN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna NOT IN (‘valor1’, ‘valor2’);
```

Selección con condicional, NULL

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna IS/IS NOT NULL;
```

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

4.1. WHERE - query

Selección con texto, empieza por

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna LIKE ‘A%’;
```

Selección con texto, acaba por

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna LIKE ‘%A’;
```

Selección con texto, contiene…

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna LIKE ‘%A%’;
```

Caracteres especiales

- **%:** cantidad variable de caracteres.
- **_:** un único caracter.

4.1. WHERE - query

Selección con texto, _

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna LIKE 'CARL_';
```

Selección con condicional, BETWEEN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna >= 'valor1' AND columna <= 'valor2';
```

Selección con condicional, BETWEEN

```
SELECT columna1, columna2, columna3
FROM  tabla
WHERE columna BETWEEN 'valor1' AND 'valor2';
```

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

4.2. DISTINCT, ORDER BY - query

Selección de valores únicos

```
SELECT DISTINCT columna
FROM  tabla;
```

Selección con ORDER BY

```
SELECT columna1, columna2, columna3
FROM  tabla
ORDER BY columna1 [ASC]/DESC;
```

Por defecto la cláusula ORDER BY tiene el argumento ASC.

- **ASC**: ordena de menor a mayor.
- **DESC**: ordena de mayor a menor.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

4.3. AGREGACIONES - query

Selección del valor máximo

```
SELECT MAX(columna1)
FROM  tabla;
```

Selección de la media

```
SELECT AVG(columna1)
FROM  tabla;
```

Selección del valor mínimo

```
SELECT MIN(columna1)
FROM  tabla;
```

Selección del conteo de filas

```
SELECT COUNT(columna1)
FROM  tabla;
```

GROUP BY

Si hay varias columnas seleccionadas y se hace una agregación solo en una de ellas recuerda siempre agrupar el resto de columnas.

COUNT()

- **COUNT(*)**: se fija en toda la tabla.
- **COUNT(DISTINCT 'columna')**: se fija en los valores únicos de esa columna.
- No va a contar los valores **NULL**.

4.3. AGREGACIONES - query

```sql
SELECT
 columna_a
 , columna_b
 , SUM(columna_c)
FROM tabla
WHERE (condición_si_es_necesario_filtrar)
GROUP BY columna_a, columna_b
HAVING (condición_que_filtra_la_agrupación)
ORDER BY (columna_por_la_que_ordenar_la_información);
```

```sql
SELECT
 columna_a
 , columna_b
 , SUM(columna_c)
FROM tabla
WHERE (condición_si_es_necesario_filtrar)
GROUP BY 1, 2
HAVING (condición_que_filtra_la_agrupación)
ORDER BY (columna_por_la_que_ordenar_la_información);
```

Esta query hará en este caso una suma, **SUM()**. Y va a agrupar todos aquellos registros que sean iguales en las columna_a y columna_b para poder hacer correctamente la operación.

HAVING es muy parecido a WHERE. Pero en él se pone una condición que puede tenerse en cuenta una vez se ha hecho el **GROUP BY**.

- Las columnas destacadas en **negrita** son las que tengo que indicar al GROUP BY qué hacer con ellas, y cómo a través del HAVING.
- En la cláusula GROUP BY se puede referenciar a las columnas por su nombre, o por su posición (con números).

4.4. Relación entre tablas

Introducción

Para preservar la eficiencia en el almacenamiento de la información de la base de datos se reparte agrupándola en diferentes tablas.

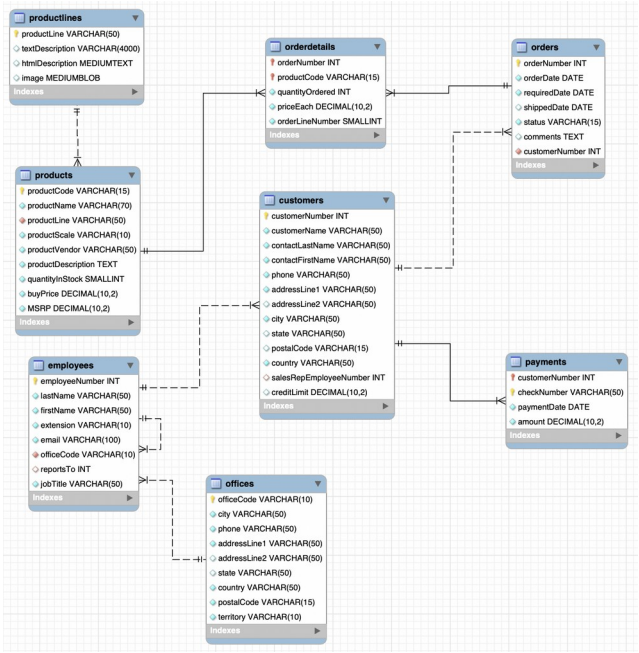
Imagina una plataforma de suscripción para ver contenido audiovisual en streaming. En la base de datos está la información de todos los usuarios registrados, el tipo de suscripción que tienen y el contenido que ven.

Si todo está contenido en la misma tabla todos los usuarios que tengan el mismo plan de suscripción harán que esa información se repita a lo largo de la tabla, además cada usuario va a ver diverso contenido y eso hará que a su vez se multipliquen los registros del mismo usuario.

Para evitar eso se puede distribuir teniendo la tabla de los usuarios, la tabla de los planes de suscripción y la tabla de los títulos que hay en el catálogo.

Estas tablas pueden relacionarse entre ellas si queremos ver por ejemplo el tipo de suscripción que tiene cada usuario juntando la información de las tablas donde está la información de los usuarios y la información de las suscripciones.

En la siguiente imagen puedes ver el diagrama de las relaciones de una base de datos de un concesionario. Hay tablas para las diferentes líneas de producto, todos los productos que se venden, todos los pedidos, los detalles de los pedidos, los clientes, los pagos de los clientes, los empleados y las diferentes oficinas que hay. Las flechas de unión indican las tablas que están relacionadas entre sí.



4.4. Relación entre tablas (PK vs KF)

PRIMARY KEY (PK)

Toda tabla de una base de datos va a tener una columna que será el identificador de esa fila. Debe cumplir una serie de características:

- Ningún valor puede ser NULL.
- Cada registro debe ser único (en una tabla de alumnos no puede haber dos alumnos con el mismo identificador).
- Una tabla no puede contener más de una columna como PRIMARY KEY.

FOREIGN KEY (FK)

Cuando una clave primaria de una tabla aparece en otra tabla recibe el nombre de FOREIGN KEY, por ser el identificador único de las filas de otra tabla.

Así se puede relacionar una tabla con otra.

Esto es muy importante para poder unir unas tablas con otras. En la mayoría de los casos el nexo de unión entre dos tablas es la clave primaria de una tabla con la clave foránea de la otra tabla, siendo clave primaria de la primera tabla.

4.4. Relación entre tablas (JOIN)

JOIN

JOIN es la cláusula que nos permite unir tablas. A lo largo de la query después de indicar qué columnas necesitas indicas de qué tabla debe seleccionar esa información en el FROM. Ahora hay que dar un paso más y unirlo con otra tabla. En la siguiente línea se indica el tipo de JOIN y la tabla con la que hay que hacer la unión.

Al unir tablas hay que indicarle cuál es el punto de union. **ON** la columna de una tabla sea igual = a la columna de la otra tabla.

Cuando se trabaja con diferentes tablas hay que tener en cuenta que la sintaxis varía. Siempre que nos refiramos a cualquier columna hay que decirle de qué tabla es esa información.

tabla.columna

En el ejemplo de la derecha podrás ver una query de ejemplo estándar para comprender la estructura.

```
SELECT
 tabla1.columna1
 , tabla2.columna_1
 , tabla2.columna_2
FROM tabla1
JOIN tabla2
 ON tabla1.columna_común = tabla2.columna_común
WHERE (condición_si_es_necesario_filtrar)
ORDER BY (columna_por_la_que_ordenar_la_información);
```

4.4. Relación entre tablas (JOIN)

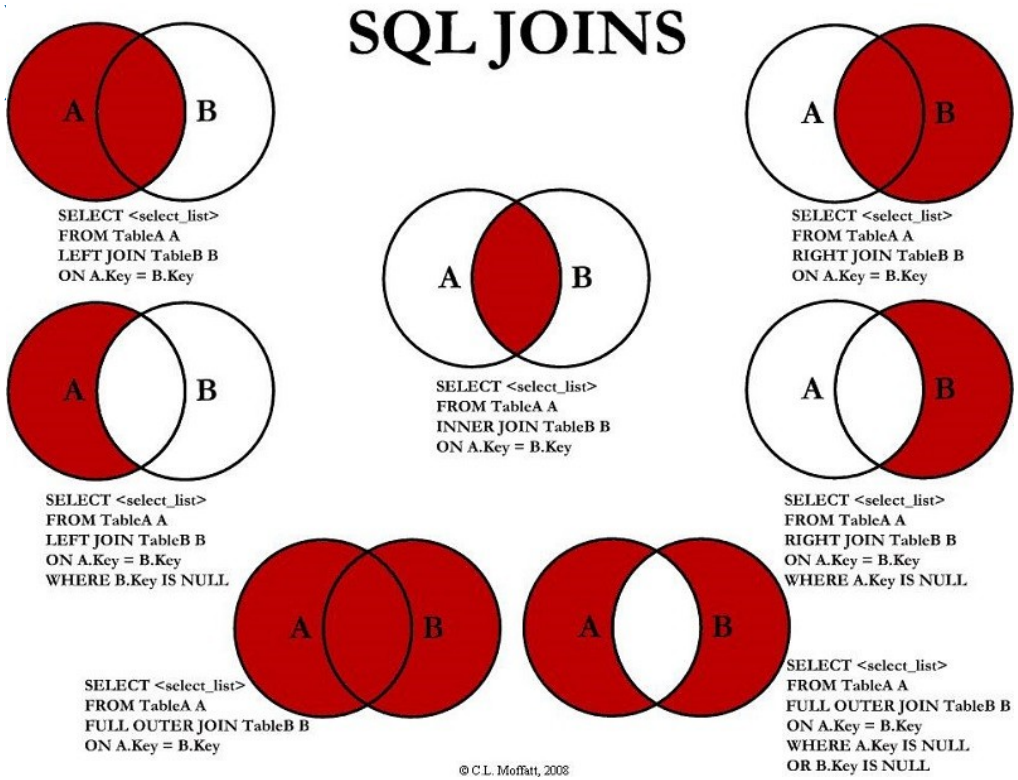
Tipos de JOIN

Hay diferentes maneras de hacer la unión entre las tablas. Puede hacerse:

- uniendo **toda** la información de una y otra tabla.
- uniendo únicamente la información que es **coincidente** en ambas tablas.
- uniendo **toda** la información de una tabla con aquella información **coincidente** de otra tabla.

INNER JOIN	muestra únicamente las coincidencias en ambas tablas
LEFT JOIN	muestra todos los datos de la tabla de la izquierda y únicamente los coincidentes en la tabla de la derecha. Aquellas filas en las que no haya coincidencia se rellenan con NULL
RIGHT JOIN	muestra todos los datos de la tabla de la derecha y únicamente los coincidentes en la tabla de la izquierda. Aquellas filas en las que no haya coincidencia se rellenan con NULL
FULL (OUTER) JOIN	muestra todos los datos de ambas tablas, uniendo las filas que coinciden, y mostrando el resto de la información de ambas tablas aunque no haya coincidencias. Todos aquellos datos sin coincidencias serán rellenados con valores NULL

4.4. Relación entre tablas (JOIN



4.4. Relación entre tablas

```sql
SELECT
 t1.columna1
 , t2.columna_1
 , t2.columna_2
FROM tabla1 [AS] t1
JOIN tabla2 [AS] t2
 ON t1.columna_común = t2.columna_común
WHERE (condición_si_es_necesario_filtrar)
ORDER BY (columna_por_la_que_ordenar_la_información);
```

```sql
SELECT
 tabla1.columna1
 , tabla2.columna_1
 , tabla2.columna_2
FROM tabla1
JOIN tabla2
 ON tabla1.columna_común = tabla2.columna_común
WHERE (condición_si_es_necesario_filtrar)
ORDER BY (columna_por_la_que_ordenar_la_información);
```

Alias —> AS

Dar un alias a las tablas puede hacerse con o sin **AS**.

Usar los alias va a ponerte la vida más fácil, escribiendo menos.