SQL Cheat Sheet

Crear una tabla

```
CREATE TABLE IF NOT EXISTS `tabla_B` (
  `id_tabla_B` INT NOT NULL AUTO_INCREMENT,
  `id_tabla_A` INT NOT NULL,
  PRIMARY KEY (`id_tabla_B`),
  CONSTRAINT `fk_tablaB_tablaA`
    FOREIGN KEY (`id_tabla_A`)
    REFERENCES `tabla_A`  (id_tabla_A)
    ON DELETE opcion_referencia
    ON UPDATE opcion_referencia)
```

Tipos de Data

Numerico

SMALLINT: un número entero que ocupa 16 bits de almacenamiento (2 Bytes).

MEDIUMINT un número entero que ocupa 24 bits de almacenamiento (3 Bytes).

INT/INTEGER un número entero de hasta 10 dígitos. Ocupa 32 bits de almacenamiento (4 Bytes).

BIGINT Entero de hasta 19 dígitos. Ocupa 64 bits de almacenamiento (8 Bytes).

FLOAT número decimal con 7 dígitos de precisión decimal. Ocupa 32 bits de almacenamiento (4 Bytes).

DOUBLE número decimal con 15 dígitos de precision decimal. Ocupa 64 bits de almacenamiento (8 Bytes).

BOOL/BOOLEAN utilizado para comprobaciones True-False. Un 0 es considerado False y cualquier otro valor es True.

UNSIGNED to disallow negative values

ZEROFILL to fill zeros in all available spaces to the left; also automatically assigns UNSIGNED to the column

Texto

CHAR(tamaño) Cadena de caracteres de una longitud fija especificada, hasta 255 car.

VARCHAR(tamaño) Cadena de caracteres de tamaño variable. Se especifica el máximo tamaño que podrá tener hasta 65,535 car.

BINARY(tamaño) igual que CHAR pero almacena cadenas de caracteres binarios.

VARBINARY(tamaño) igual que VARCHAR pero almacena cadenas de caracteres binarios.

TINYTEXT almacena una cadena de caracteres con una longitud máxima de 255 caracteres.

TEXT almacena una cadena de caracteres con una longitud máxima de 65,535 caracteres.

MEDIUMTEXT almacena una cadena de caracteres con una longitud máxima de 16,777,215 car.

LONGTEXT almacena una cadena de caracteres con una longitud máxima de 4,294,967,295 car.

ENUM(val1, val2, val3, ...) una cadena de caracteres que puede tomar un solo valor de los indicados en la lista.

Fechas

DATE Fecha con formato AAAA-MM-DD.

TIME Hora con formato HH:MM:SS.

DATETIME Fecha y hora con formato AAAA-MM-DD HH:MM:SS.

TIMESTAMP Un timestamp es una representación de la fecha y hora actual. El formato es: AAAA-MM-DD hh:mm:ss. El intervalo permitido va desde '1970-01-01 00:00:01' UTC hasta el '2038-01-09 03:14:07' UTC. El gestor realiza la conversión de la fecha de tu zona horaria a UTC y viceversa automáticamente.

YEAR Un año en formato de cuatro dígitos. Los valores permitidos van desde 1901 a 2155 (aunque el 0000 también es un valor admitido).

Restricciones

NOT NULL sirve para indicar que la columna en cuestión no puede dejarse vacía

PRIMARY KEY se usa para indicar que la columna servirá como clave principal de la tabla.

UNIQUE define índice único; no permite que haya valores duplicados en la tabla

CONSTRAINT Esta clausula es opcional (excepto por los FOREIGN KEY). Sirve para poner nombre a las restricciones, que deben de cumplir los datos.

REFERENCES tabla [(columna)]: se usa para definir columnas como claves foráneas de la tabla. Con REFERENCES se pueden indicar con qué columnas de qué tablas se corresponde esta clave foránea. Si no se especifica la columna de la tabla externa, se asume que se corresponde con su clave primaria.

CHECK (expresion condicional): sirve para asegurarse de que los valores en una columna cumplen una determinada condición.

DEFAULT sirve para establecer un valor por defecto para la columna si no tiene valor.

Foreign Key Sintaxis:

```
CONSTRAINT `fk_tablahija_tablamadre`
FOREIGN KEY (columna_tablahija)
REFERENCES table_madre (columna_madre)
        [ON DELETE reference_option]
        [ON UPDATE reference_option]
```

ON DELETE donde indicamos que hacer cuando se elimina un registro en la tabla madre

ON UPDATE donde indicamos que hacer cuando se actualiza un

Para ON DELETE y ON UPDATE podemos usar:

RESTRICT rechaza el borrado o la actualización de la columna clave en la tabla "madre"

CASCADE borrar o actualizar una fila/registro en la tabla "madre" hace que las filas correspondiente de la tabla "hija" se borren o se actualicen en consecuencia

SET NULL borrar o actualizar una fila/registro en la tabla "madre" hace que la columna correspondiente de la tabla "hija" se actualice al valor NULL para los registros afectados

Alteración de Tablas

ADD COLUMN: para añadir una columna

```
ALTER TABLE table_name
ADD COLUMN column_name data_type restrictions
```

ADD CONSTRAINT: añadir una restricción

- se puede usar con CHECK, FOREIGN KEY, PRIMARY KEY y UNIQUE

Para anadir una restriccion a una columna:

```
ALTER TABLE table_name
ADD CONSTRAINT restriction_name
    FOREIGN KEY (column_name)
    REFERENCES mother_table (referenced_column)
    ON UPDATE CASCADE;
```

Para anadir una restriccion a una tabla:

```
ALTER TABLE table_name
ADD CONSTRAINT
    PRIMARY KEY (column1, column 2, ...)
```

RENAME COLUMN TO: cambiar el nombre de una columna

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```

RENAME TO: cambiar el nombre de una table

```
ALTER TABLE table name
RENAME TO new_table_name;
```

MODIFY: Cambiar el tipo de data de una columna

```
ALTER TABLE table_name
MODIFY COLUMN column_name data_type;
```

DROP COLUMN: quitar una columna o una table

```
ALTER TABLE table_name
DROP COLUMN column_name1, column_name2, ..;
```

DROP CONSTRAINT: quitar una restriccion de una table o columna

```
ALTER TABLE table_name
DROP CONSTRAINT restriction_name;
```

*** para cambiar las restricciones de una columna o table hay que quitarlas y luego añadirlas con ADD ***

Insercion de datos

INSERT INTO: para insertar datos en una tabla existente:

```
INSERT INTO nombre_tabla (columnas)
VALUES (valores_columnas), (valores_columnas),
(valores_columnas)...;
```

UPDATE: Para actualizar valores ya existentes en la tabla:

```
UPDATE nombre_tabla SET columna1= valor1,
columna2=valor2 WHERE condicion= valor_condicion;
```

DELETE FROM: para borrar registros de una tabla:

```
DELETE FROM nombre_tabla
WHERE condicion= valor_condicion;
```

SIEMPRE USAR WHERE PARA UPDATE Y DELETE FROM

Consultas base de datos

SELECT FROM para hacer consultas; usar * para seleccionar todo

```
SELECT nombre_col1, nombre_col2,....
FROM tabla;
```

WHERE para filtrar una consulta (o establecer condiciones de la consulta) por los valores de sus atributos (columnas):

```
SELECT *
FROM tabla
WHERE columna = condicion;
```

WHERE NOT: para filtrar excluyendo

```
SELECT * FROM tabla
WHERE NOT columna = condicion;
```

NOT NULL/IS NULL: Selecciona aquellos registros cuyo valor en la columna no sea o sea nulo

```
SELECT * FROM tabla
WHERE  columna NOT NULL; o WHERE columna IS NULL
```

ORDER BY: Permite ordenar las consultas, por defecto las ordena de forma ascendente o usar **DESC** para descendente

```
SELECT nombre_col1, nombre_col2,....
FROM tabla
ORDER BY nombre_col DESC;
```

DISTINCT: Permite filtrar la columna por aquellos valores que sean únicos, es decir, nos quita los duplicados. Se puede usar para mas de una columna

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla;
```

LIMIT numero_valores : Limita el número de valores seleccionados de la consulta. Por defecto MySQL selecciona un máximo de 1000 registros.

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla
LIMIT numero_valores;
```

OFFSET numero_valores: Se utiliza para descartar los primeros (numero_valores) indicados en la consulta.

```
SELECT nombre_col1, nombre_col2,....
FROM tabla
LIMIT numero_valores1
OFFSET numero_valores2;
```

BETWEEN: Es una condición del WHERE. Permite filtrar por rangos de datos, considerando la columna por la que estamos filtrando.

- Solo acepta valores numéricos.
- Es equivalente a filtar la columna numérica utilizando < = y >=

```
SELECT nombre_col1, nombre_col2,....
FROM tabla
WHERE columna BETWEEN valor1 AND valor2;
```

AS: Se utiliza para darle un alias (nombre) a una columna de forma temporal, no modifica el nombre real de la columna.

```
SELECT nombre_col1 AS nombre_columna1_alias,  nombre_col2 AS
nombre_columna2_alias ,....
FROM tabla
WHERE columna;
```

IN: Nos permite filtrar utilizando uno o varios elementos de la columna por la que estamos filtrando

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla
WHERE columna IN( valor1, valor2);
```

Operadores

```
AND
OR
<> para excluir aquellos valores que no cumplan la condición
= para seleccionar aquellos que cumplan la condición
> mayor que
>= mayor o igual que
< menor que
<= menor o igual que
```

Funciones agregadas

MIN() y **MAX()** devuelven unicamente el valor mas pequeno o el valor mayor de la columna

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

SUM() realiza la suma de todas las entradas en la columna indicada

```
SELECT SUM(columna)
FROM tabla;
```

AVG() devuelve el valor medio (average) del atributo especificado

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

COUNT() es una función que devuelve el número de registros (filas) tiene la tabla-resultado original

```
SELECT COUNT(columna)
FROM tabla;
-   se puede usar con distinct:
SELECT COUNT(DISTINCT columna)
FROM tabla;
```

GROUP BY() agrupa las filas resultado según los valores de uno de sus atributos

```
SELECT COUNT(columna)
FROM tabla
GROUP BY columna;
```

HAVING se usa para imponer condiciones a los grupos creados con GROUP BY una vez creados

```
SELECT COUNT(columna)
FROM tabla
GROUP BY columna
HAVING condicion;
```

CASE se puede usar con SELECT para crear nuevas categorías basado en condiciones, o con WHERE para crear una condición para filtrar

```
SELECT
CASE
    WHEN condicion THEN “nombre_categoria"
    WHEN condicion2 THEN “nombre_categoria2"
    ELSE “nombre_categoria3"
    END AS AliasColumna, columna1, columna2
FROM tabla;

SELECT columna1, columna2
FROM tabla
WHERE columna >
    (SELECT CASE
    WHEN condicion1 THEN segunda_parte condicion_WHERE
    WHEN condicion1 THEN segunda_parte condicion_WHERE
    ELSE segunda_parte condicion_WHERE
    END);
```

SQL Cheat Sheet

Consultas en multiples tablas

CROSS JOIN realiza un producto cartesiano de dos tablas: se combinan todos los registros de la primera table con todos los registros de la segunda table

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1
FROM tabla1
CROSS JOIN tabla2
WHERE tabla1.columna1 = tabla2.columna1;
```

NATURAL JOIN junta las tablas juntando automáticamente la columnas que tienen en común, sin usar WHERE

```
SELECT columna1, columna2, columna3
FROM tabla1 NATURAL JOIN tabla2;
```

INNER JOIN junta dos tablas usando columnas que no se llamen igual en ambas
ON se usa cuando las columnas a asociar no se llaman igual en ambas tablas

```
SELECT columna1, columna2, tabla2.columna1
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

LEFT JOIN selecciona todos los registros de la primera tabla y solo los registros coincidentes de la segunda

```
SELECT tabla1.columna1, tabla1.columna2,
tabla2.columna1...
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

RIGHT JOIN selecciona todos los registros de la segunda tabla y solo los registros coincidentes de la primera

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1,
....
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

FULL OUTER JOIN devuelve todos los registros de ambas tablas, tengan entradas asociadas en la otra tabla o no

```
SELECT tabla1.columna1, tabla1.columna2,
tabla2.columna1...
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1
UNION
SELECT tabla1.columna1, tabla1.columna2,
tabla2.columna1...
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

SELF JOIN es un caso concreto de NATURAL JOIN en el que una tabla se combina consigo misma en vez de con una segunda tabla; hay que usar aliases

```
SELECT A.columna1 AS Nombre1, A.columna2 AS Nombre2,
B.columna1 AS Nombre3, B.columna2 AS Nombre4,
FROM nombre_tabla AS A, nombre_tabla AS B
WHERE A.columna1 <> B.columna1;
```

UNION combina los resultados de dos o más instrucciones del tipo SELECT, y los devuelven sin filas duplicadas para las tablas utilizadas

- cada instrucción de tipo SELECT en la UNION debe tener el mismo número de columnas
- las columnas deben tener tipos de datos iguales (int, str, etc)

```
SELECT columna1
FROM tabla1
UNION
SELECT columna1
FROM tabla2;
```

UNION ALL combina los resultados de dos o más instrucciones del tipo SELECT, permitiendo duplicados

```
SELECT columna1
FROM tabla1
UNION ALL
SELECT columna1
FROM tabla2;
```

UNION y UNION ALL se pueden usar con ORDER BY y LIMIT

IN (INTERSECT) nos devuelve las filas de la primera tabla que son idénticas a las de la segunda tabla; no devuelve duplicados y devuelve valores coincidentes que son nulos

```
SELECT columna1
FROM tabla1
WHERE columna1 IN (
    SELECT columna
    FROM tabla2);
```

NOT IN (EXCEPT) nos devuelve la filas de la primera tabla, que no se pueden encontrar en la segunda tabla

```
SELECT columna1
FROM tabla1
WHERE columna1 NOT IN (
    SELECT columna
    FROM tabla2);
```

Orden de escritura	Orden de ejecución
SELECT	FROM
DISTINCT	JOIN
FROM	WHERE
JOINS	GROUP BY
WHERE	HAVING
GROUP BY	SELECT
HAVING	DISTINCT
ORDER BY	ORDER BY
LIMIT - OFFSET	LIMIT - OFFSET

Subconsultas

- se pueden usar como condicion con HAVING o WHERE

```
SELECT id_empleada, nombre
FROM empleadas
WHERE id_empleada IN (
    SELECT id_empleada
    FROM empleadas_en_proyectos
    WHERE id_proyecto=2);
```

- se pueden usar con los operadores IN, NOT IN, ANY y ALL

ANY: la condicion tiene que cumplir para al menos un valor en la subconsulta

ALL: la condicion tiene que cumplir para todos los valores en la subconsulta (usar con >= o <= o no devuelve nada)

Subqueries correlacionadas

- una manera de comparar cada fila de una tabla contra un valor relacionada
- se usa cuando la subquery devuelve un resultado diferente de la query principal
- se ejecuta por cada fila de la consulta principal

```
SELECT * (o columnas)
FROM tabla1 AS t1
WHERE columna o condición (
    SELECT columna o operacion (columna)
    FROM tabla2 AS t2
    WHERE t1.id = t2.id)
```

Wildcards y Regex

LIKE y NOT LIKE para usar patrones para filtrar por strings; LIKE para incluir y NOT LIKE para excluir

```
SELECT columna1, columna2, ...
FROM tabla
WHERE columna LIKE 'patron';
```

% para que las coincidencias sean de un string de cero o más caracteres

_ realiza las coincidencias para cualquier carácter individual

- caracteres reservadas de SQL se tienen que usar con \ delante:

```
\0 caracter nulo de ASCII
\' comillas simple
\« comillas dobles
  nueva linea
\\ barra invertida
\% porcentaje
\_ barra baja
```

REGEX

```
SELECT columna1, columna2, ...
FROM tabla
WHERE columna REGEXP 'patron_regex';
```

CTE (Common Table Expression)

```
WITH nombre_CTE [(nombres_columnas)]
  AS (consulta_CTE);
```

Antes de SELECT, UPDATE o DELETE:

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

```
SELECT columna1, columna2
FROM nombre_CTE;
```

Al comienzo de subconsultas dentro de otra consulta SELECT:

```
SELECT ...
WHERE id IN (
    WITH ...
    SELECT ...
    ) ...

SELECT ...
FROM (
    WITH ...
    SELECT ...
    ) AS dt ...
```

Justo antes del SELECT dentro de otras sentencias que contienen un SELECT:

```
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
CREATE TABLE ... WITH ... SELECT ...

WITH cte1 AS (SELECT ...),
      cte2 AS (SELECT ...)
SELECT ...

WITH cte1 AS (SELECT ...),
     cte2 AS (SELECT ...
              FROM cte1)
SELECT ...
```

MySQL Connector/Python

Conectar a una base de datos

import mysql.connector para importar MySQL Connector

```
pip install mysql-connector
pip install mysql-connector-Python
```

connect() para conectar a una base de datos:

```
variable_cnx = mysql.connector.connect (user='root',
                    password='AlumnaAdalab',
                    host='127.0.0.1',
                    database='nombre_BBDD')
```

cnx.close() desconectar de la base de datos

Añadir errores

from mysql.connector import errorcode importar errores

mysql.connector.Error se puede usar en un try/except

```
try:
    accion
except mysql.connector.Error as err:
    print(err)
    print("Error Code:", err.errno)
    print("SQLSTATE", err.sqlstate)
    print("Message", err.msg)
```

MySQL Connector/Python

Realizar queries

variable_cursor = cnx.cursor() crear el objeto cursor que nos permite comunicar con la base de datos

variable_cursor.close() desconectar el cursor

variable_query = ("SQL Query") guardar un query en un variable

variable_cursor.execute(variable_query) ejecutar el query; devuelve una lista de tuplas

import datetime sacar fechas en el formato AAAA-MM-DD

datetime.date(AAAA, M, D) devuelve el formato de fecha

variable_query = "SQL Query... %s AND %s") query dinamica

variable_cursor.execute(query, (variable1, variable2)) valores que van en lugar de los %s

variable_cursor.execute("SHOW DATABASES") mostrar las BBDD

variable_cursor.execute("SHOW TABLES") mostrar las tablas de la BBDD indicado en la conexión

variable_cursor.execute("SHOW TABLES")

variable_cursor.execute("SHOW COLUMNS FROM bbdd.table") mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information_schema

Obtener resultados de una query

variable_cursor.fetchone() devuelve el primer resultado

variable_cursor.fetchall() devuelve todos los resultados como iterable – cada fila es una tupla

Crear y alterar una base de datos

variable_cursor.execute("CREATE DATABASE nombre_BBDD")

variable_cursor.execute("CREATE TABLE nombre_tabla (nombre_columna TIPO, nombre_columna2 TIPO2)")

variable_cursor.execute("ALTER TABLE nombre_tabla ALTERACIONES")

Insertar datos

variable_query = "INSERT INTO nombre_tabla (columna1, columna2) VALUES (%s, %s)"

variable_valores = (valor1, valor2)

variable_cursor.execute(variable_query, variable_valores)

otro método:

variable_query = "UPDATE nombre_tabla SET nombre_columna = "nuevo_valor" WHERE nombre_columna = "valor"

Insertar múltiples filas a una tabla

variable_valores_en_tuplas = ((valor1columna1, valor1columna2), (valor2columna1, valor2columna2))
variable_cursor.executemany(variable_query, variable_valores_en_tuplas)

variable_conexion.commit() después de ejecutar la inserción, para que los cambios efectúen en la BBDD

variable_conexion.rollback() se puede usar después de execute y antes de commit para deshacer los cambios

print(variable_cursor.rowcount, "mensaje") imprimir el numero de filas en las cuales se han tomado la accion

Eliminar registros

variable_query = "DROP TABLE nombre_tabla"