**RDBMS = Relational Data Base Management System**

**PARTE 1:** Movimientos básicos en una sola tabla

SQL = Base de Datos Relacionales, suelen almacenar la información en forma de tabla, las cuales tienen filas y columnas.

NoSQL = Base de Datos NO Relacionales, tienen distintas maneras de almacenar la información en la base de datos, entre ellas se encuentra: JSON, BSON, BLOB, Key-Value.

Columnas= Hacen referencia al dato mismo que se está guardando, ej: si tu nombre es Felipe, la columna se llamaría nombre, que es el dato que esta guardando.

Filas= la parte horizontal de nuestras tablas(las filas), se conocen como registros, cada espacio se lo conoce como dato, ej: la tabla tiene 2 columnas nombe, edad, Felipe es un datos 20 es otro todos los datos de esa misma fila junta forman el registro. María, 22 serían otro registro.

Consultas:

create database NombreDeLaBaseDeDatos; (crea base de datos con el nombre que le ponga)

show databases; (muestra todas las bases de datos)

**CREAR TABLA:**

use NombreDeLaDB (importante para que sepa en que db se crea la tabla)

CREATE TABLE animales (

id int,

tipo varchar(255),

estado varchar(255),

PRIMARY KEY (id)

); (crea la tabla animales)

**MODIFICAR UNA TABLA:**

ALTER TABLE animales MODIFY COLUMN id int auto\_increment;(modifica la tabla para que id sea autoincremental, **siempre hay que poner que tipo de dato es**, en este caso int, por más que ya haya sido declarado)

**AGREGAR DATOS A LA TABLA:**

INSERT INTO (tipo, estado) VALUES (‘chanchito’, ‘feliz’)

SHOW CREATE TABLE animales (nos muestra el comando con el que fue creada la tabla, por ejemplo en este caso que la cree sin autoincrement pero después lo agregue si quisiera saber como se usa el comando para el auto incremenet con el SHOWCREATETABLE me mostraría como se hubiese creado de inicio)

**MODIFICAR DATOS DE UNA TABLA**:

**UPDATE** animales **SET** estado = ‘triste’ WHERE id = 1;

**Comentar una línea en las consultas** se utiliza el doble guion (--) al principio de la línea.

**TIPOS DE CONSULTAS:**

SELECT \* FROM animales; (nos muestra todos los registros de la tabla animales)

SELECT \* FROM animales **LIMIT** 1;(limit indica cuantos resultados quiero como máximo)

SELECT \* FROM animales WHERE id **>** 1; (no tiene mucho sentido aca se usaria mas para edades )

SELECT \* FROM animales WHERE id **=** 1;(nos muestra los elementos con id 1)

SELECT \* FROM animales WHERE tipo = ‘chanchito’;(nos muestra los elementos que de tipo tengan chanchito)

SELECT \* FROM animales WHERE name **LIKE** ‘%chito%’; busca dentro de la tabla en la columna name todo lo que tenga la cadena ‘chito’ tiene 3 tipos de uso:

**%gmail%** puede tener antes o después algo pero tiene que tener gmail en algún lugar

**%gmail** puede tener algo o no antes, pero tiene que terminar con gmail.

**gmail%** tiene que emepezar con gmail.

**SELECT u.name, u. money+(u. money \*0.5) FROM users u;** Nos devuele la plata del usuario mas un 50%;

SELECT \* FROM user **ORDER BY** edad asc; busca todos los usuarios y los ordena pore dad ascendente IMPORTANTE también se puede usar asc y desc para ordenar alfabéticamente.

SELECT \* FROM user **ORDER BY** name desc; devuele los usuarios por nombre alfabetico descendiente. IMPORTANTE el ORDER BY puede ir sin el WHERE, pero, si hay un WHERE, el ORDER BY tiene que ir después de este.

SELECT **max(**edad**) AS** mayor FROM user; nos devuelve el de mayor edad y “crea” una columna mayor para mostrar el resultado.

SELECT **min(**edad**) AS** ‘el menor’ FROM user; nos devuelve el menor y a diferencia del max este tiene ‘’ en el AS porque hay un espacio entre las palabras.

SELECT **AVG(**u.edad**)** FROM user u; nos devuelve el promedio de la columna edad.

SELECT **SUM(**u.edad**)** FROM user u; nos devuelve la suma del total de las edades, a diferncia de count() que es un contador, osea que cuenta cuantas veces esta el elemento, NO su valor.

SELECT id, name FROM user; esto solo nos devuelve los campos requeridos (id, name) de la tabla user.

SELECT id, name **AS** nombre FROM user; esto nos devuelve la columna id y name, pero name la devuelve como nombre(puede servir para alguna referencia visual).

SELECT id, name, email AS correo FROM user WHERE email LIKE %gmail%; devuelve los campos id, name, email lo devuelve como correo de la table Usuario cuando tengan en el mail una parte ‘gmail’

**ELIMINAR DATOS DE UNA TABLA:**

DELETE FROM animales WHERE estado = ‘feliz’;

**CAMBIAR NOMBRE DE UNA TABLA:**

**RENAME TABLE** products **TO** product; cambia el nombre de la table products a product .

**OPERADORES AND OR NOT y mas**

SELECT \* FROM animales WHERE estado = ‘feliz’ **AND** tipo = ‘chanchito’

SELECT \* FROM animals WHERE edad **BETWEEN** 10 **AND** 30;

SELECT \* FROM animals WHERE edad **BETWEEN** 10 **AND** 30 **OR** name = ‘claudio’;

**PARTE 2:** Movimientos básicos entre distintas tablas.

**CREAR UNA TABLA CON FOREIGN KEY:**

CREATE TABLE products (

id int NOT NULL AUTO\_INCREMET,

name varchar(50) NOT NULL,

created\_by int NOT NULL,

marca varchar(50) NOT NULL,

PRIMARY KEY(id),

UNIQUE ‘name\_UNIQUE’ (‘name’ asc) VISIBLE;

**FOREIGN KEY** (created\_by) **REFERENCES** user(id)

);

**CREAR UNA TABLA CON UNIQUE:**

CREATE TABLE alumnos(

id int NOT NULL AUTO\_INCREMENT,

name varchar(50) NOT NULL,

email varchar(50) NOT NULL,

**UNIQUE INDEX** **email\_UNIQUE** (**email ASC**) **VISIBLE**,

PRIMARY KEY (id)

);

**AGREGAR MAS DE 1 FILA:**

INSERT INTO product (name, created\_by, marca)

VALUES

(‘ipad’, 1, ‘apple’),

(‘iphone’, 2, ‘apple’),

(‘motog32’, 3, ‘motorola’);

**LEFT JOIN:**

(Uno a muchos) Nos devuelve todos los elementos de la primera tabla y sólo los elementos de la 2 que tengan relación con la primera.

**SELECT** **u**.id, **u**.name, **p**.id, **p**.name FROM user **u** **LEFT JOIN** product **p** on u.id = p.created\_by;

Nos devuelve **todos** los usuarios y **sólo** los productos que hayan sido creados(created\_by), por usuarios de la table user. “u” y “p”, los usamos para hacer más fácil el llamado en la función, va a devolver todos los usuarios y si alguno no creo producto ese campo esta quedará vacío y si un producto no hubiese sido creado por un usario no aparecería en la consulta por eso es LEFT JOIN porque prevalece una tabla sobre otra.

**RIGHT JOIN:**

**SELECT** **u**.id, **u**.name, **p**.id, **p**.name FROM user **u** **RIGHT JOIN** product **p** on u.id = p.created\_by;

Siguiendo el mismo caso anterior, hace lo opuesto trae todos los productos si o si y solo completa el campo u.name si es que tienen relación con la segunda tabla(products)

**INNER JOIN:**

**SELECT** **u**.id, **u**.name, **p**.id, **p**.name FROM user **u** **INNER JOIN** product **p** on u.id = p.created\_by;

Nos devuelve solo los user y product que tengan relación entre las 2 tablas.



**LEFT JOIN RIGHT JOIN INNER JOIN**

**CROSS JOIN:**

**SELECT** **u**.id, **u**.name, **p**.id, **p**.name FROM user **u** **CROSS JOIN** product **p;**

Devuelve por cada elemento un cruce con todos los elementos de la otra tabla, nose para que serviría CUIDADO! Devuelve resultados grandes.

**GRUOP BY:**

SELECT **count(**id**)**, marca FROM product **GROUP BY** marca;

Nos devuelve la cuenta de cuantos productos hay por marca.

SELECT **count(p.id),** u.name **FROM** product p

**LEFT JOIN** user u ON u.id= p.created\_by

**GROUP BY** p.created\_by

**HAVING count**(p.id**) >=** 2**;**

nos devuelve un Contador con los usuarios que hayan creado mas de 2 productos.

**ELIMINAR UNA TABLA**

**DROP TABLE** product;

Elimina la tabla product.

**ELIMINAR UNA COLUMNA DE UNA TABLA**

**ALTER TABLE** product **DROP COLUMN** name;

**AGREGAR UNA COLUMNA A UNA TABLA EXISTENTE**

**ALTER TABLE** product **ADD COLUMN** name varchar(50) NOT NULL;