SQL

1. bases de datos relacionales SQL
2. sentencias basicas
3. tablas, tipos y restricciones
4. diseño de base de datos
5. SQL insert update SELECT y delete
6. funciones en SQL y sus clausulas
7. agrupamiento
8. subconsultas basicas
9. consultas multitabla y JOIN
10. bloque de ejercicios con SQL y PHP

introduccion a SQL

que es una base de datos

sistemas gestores de bases de datos SGBD ***Database Management System***

elementos de una base de datos

1. tablas
2. columnas ***campos*** y filas ***registros***

Formato de una tabla que representa una entidad

Usuario

```
columna ->  id | name | email | fecha |
fila    ->  1 jared jared@gmail.com 28-05-2021
fila    ->  2 kenneth kenneth@gmail.com 28-05-2021
fila    ->  3 CRKJ crkj@gmail.com 28-05-2021
```

comandos de la consola de mysql

1. help ***ayuda***

2. status ***informacion de mysql***
3. ; para cerrar sentencias

sentencias básicas en estructura

1. create database usuarios; ***crea base de dadtos***
2. show databases; ***muestra las bases de datos creadas***
3. show tables; ***muestra las tablas creadas***
4. use usuarios; ***seleccionar database***
5. drop database usuarios; ***borrar base de datos***
6. desc usuarios; ***abreviado, estructura de la base de datos***
7. describe usuarios_table; ***estructura de la tabla***

sentencias básicas en consultas

1. SELECT * FROM usuarios; ***muestra los datos insertados en la tabla***

crear tablas

```
create table usuarios(
    id int,
    nombre varchar(100),
    apellidos varchar(255),
    email varchar(100),
    password varchar(255)
);
```

eliminar una tabla

1. drop table usuarios;

administrador visual phpmyadmin

agregar usuario a la base de datos

netbeans agrega un gestor de la base de datos

restricciones de integridad básicas

1. not null ***nunca puede ser nulo, siempre debe llevar una dato***
2. null ***que puede ser nulo el valor***
3. default ***default 'hola'***, valor por defecto
4. auto_increment ***incrementara automaticamente***
5. primary key ***llave primaria***
6. CONSTRAINT pk_usuarios PRIMARY KEY (id); ***dice que campo será la llave primaria***

modificar tablas

renombrar tablas

```
ALTER TABLE usuarios RENAME TO usuarios_renombre;
```

cambiar nombre de una columna

```
ALTER TABLE usuarios CHANGE apellidos apellido var(100) null;
```

modificar columna sin cambiar nombre

```
ALTER TABLE usuarios MODIFY apellido varchar(50) not null;
```

añadir columna a un tabla

```
ALTER TABLE usuarios ADD website varchar(100) not null;
```

añadir restriccion a columna campo unique

campo unico en la tabla, no pueden haber dos email iguales

```
ALTER TABLE usuario ADD CONSTRINT uq_email UNIQUE(email);
```

borrar una columna, aunque tenga contenido

```
ALTER TABLE usuarios DROP website;
```

Diseño de bases de datos

nos permite saber que estructura bamos a tener en nuestra base de datos

Programa para diseñar con diagramas open source ***Dia Diagram Editor***

UML todo ese tipo de diseño de bases de datos complejo

tablas y relaciones

crear un blog

tablas usuarios id (clave primaria) ***primary key*** nombre apellido email password fecha_registro

posts id (clave primaria) ***primary key*** usuario_id (clave ajena) ***foranea*** categoria_id (clave ajena) ***foranea*** titulo descripcion fecha

categorias id (clave primaria) ***primary key*** nombre

pasar a SQL el diseño

blog_master

1. tabla usuarios

```
CREATE TABLE usuarios (
    id int(255) auto_increment not null,
    nombre varchar(100) not null,
    apellido varchar(100) not null,
    email varchar(255) not null,
    password varchar(255) not null,
    fecha date not null,
    CONSTRAINT pk_usuarios PRIMARY KEY(id),
    CONSTRAINT uq_email UNIQUE(email)
)ENGINE=InnoDb;
```

2. tabla categorias

```
CREATE TABLE categorias (
    id int(255) auto_increment not null,
    nombre varchar(100) not null,
    CONSTRAINT pk_categorias PRIMARY KEY(id)
)ENGINE=InnoDb;
```

3. tabla posts

```
CREATE TABLE posts (
    id int(255) auto_increment not null,
    usuario_id int(255),
    categoria_id int(255),
    titulo varchar(255) not null,
    descripcion MEDIUMTEXT,
    fecha date not null,

    CONSTRAINT pk_posts PRIMARY KEY(id),

    # relacion posts + usuario(id)
    CONSTRAINT fk_posts_usuario FOREIGN KEY (usuario_id) REFERENCES
usuarios(id),

    # relacion posts + categorias(id)
    CONSTRAINT fk_posts_categoria FOREIGN KEY (categoria_id) REFERENCES
categorias(id)
)ENGINE=InnoDb;
```

que significan estas sentencias

1. CONSTRAINT ***restriccion***
2. fk_ ***foreign key*** llave foranea (ajena)
3. pk_ ***primary key*** llave primaria (identificadora de la tabla)
4. references ***referencia*** asociando la key de otra tabla
5. mediumtext ***contenedor mas grande***
6. ENGINE=InnoDb ***el motor de la base de datos***
    - asegura que las llaves ajenas funcionen correctamente
    - asegura rendimiento en consultas insert, update etc.
7. ENGINE=MyISAM ***el motor de la base de datos*** - asegura rendimiento a gran escala las consultas de tipo SELECT - no mantiene la integridad en relaciones
8. uq_ UNIQUE(email) ***restringimos UNIQUE*** no dejara meter el mismo dato en este campo

los que hacemos es hacer relacion del campo de esta tabla, con el id de la tabla que queremos asociar

codigo final aplicado en workbench

```
# drop database blog_master; #eliminar database
create database blog_master; # crear database
use blog_master; #seleccionar database

CREATE TABLE usuarios (
    id int auto_increment not null,
    nombre varchar(100) not null,
    apellido varchar(100) not null,
    email varchar(255) not null,
    password varchar(255) not null,
    fecha date not null,
    CONSTRAINT pk_usuarios PRIMARY KEY(id),
    CONSTRAINT uq_email UNIQUE(email)
)ENGINE=InnoDb;

CREATE TABLE categorias (
    id int auto_increment not null,
    nombre varchar(100) not null,
    CONSTRAINT pk_categorias PRIMARY KEY(id)
)ENGINE=InnoDb;

CREATE TABLE posts (
    id int auto_increment not null,
    usuario_id int,
    categoria_id int,
    titulo varchar(255) not null,
    descripcion MEDIUMTEXT,
    fecha date not null,

    CONSTRAINT pk_posts PRIMARY KEY(id),

    # relacion posts + usuario(id)
    CONSTRAINT fk_posts_usuario FOREIGN KEY (usuario_id) REFERENCES usuarios(id),

    # relacion posts + categorias(id)
    CONSTRAINT fk_posts_categoria FOREIGN KEY (categoria_id) REFERENCES
categorias(id)
)ENGINE=InnoDb;
```

Cascade

permite que acciones afecten a las edmas llaves foraneas

ON DELETE CASCADE SET

1. null ***que el campo se vuelva null***
2. default ***que deje por default el valor***
3. no action ***que no haga nada***

ON UPDATE CASCADE

1. actualizacion en cascada ***actualiza***

INSERT y SELECT

DML (Lenguaje de manipulacion de datos) 1. SELECT 2. insert 3. update 4. delete

INSERT

1. el id como es autoincrement, se pasa null ya que es automatico su valor
2. la fecha debe estar en formato americano, ***año mes día***

```
INSERT INTO usuarios
VALUES (null, 'Jared', 'Latorre', 'jared@gmail.com','321654','2019-05-01');
```

INSERT ciertos campos

1. para poder insertar ciertos datos a la tabla, el campo debe permitir valores null
2. estos son registros con valores null ya

```
INSERT INTO usuarios(email, password)
VALUES ('admin@admin.com','321654987');
```

SELECT

1. hace la consulta de todas las columnas de la tabla

```
SELECT * FROM usuarios;
```

SELECT por campos especificos

1. consulta por campos especificos

```
SELECT email, nombre, apellido FROM usuarios;
```

operadores aritmeticose en SQL

permite hacer operaciones y mostrar el resultado en un nuevo campo en todas los registros

```
SELECT email, (4+7) FROM usuarios; #11
SELECT email, (4-7) FROM usuarios; #-3
SELECT email, (4*7) FROM usuarios; #28
SELECT email, (4/7) FROM usuarios; #0.5714
SELECT email, (7%2) FROM usuarios; #1
```

Resultaría asi:

```
email            | (4+7)
jared@gmail.com  |  11
jared2@gmail.com |  11
```

operaciones de comparacion

1. = ***igual***
2. != ***diferente***
3. <> ***menor y mayor que***
4. <= >= ***menor o igual y mayor o igual***
5. between ***entre A and B***
6. in ***en***
7. is null ***es nulo***
8. is not null ***no es nulo***
9. like ***comparar cadenas de caracteres “es como”, ayuda a buscar si contiene***
10. LIKE “%a%”; ***busca por delante y por detras***
11. not like ***si no contiene***

operadores logicos

2. or
3. and
4. not

operaciones con el id

```
SELECT id, nombre, email, (id+27) FROM usuarios;
```

Resultado:

```
1   Jared   jared@gmail.com     28
4   Jared   jared2@gmail.com    31
5   Jared3  jared3@gmail.com    32
6   Jared4  jared4@gmail.com    33
```

alias AS

con AS le damos un nombre a ese nuevo campo, simplificando el llamado de ciertas columnas

```
SELECT email, (7+7) AS 'OPERACIONES' FROM usuarios;
```

ORDER BY

ordena los resultados por orden en de campo de manera ASC ***acendente*** o DESC ***descendente***

```
SELECT id, nombre, email, (id+27) AS suma FROM usuarios ORDER BY suma DESC;
```

Resultado:

```
6   Jared4  jared4@gmail.com    33
5   Jared3  jared3@gmail.com    32
4   Jared   jared2@gmail.com    31
1   Jared   jared@gmail.com     28
```

funciones matematicas con SQL

Libreria de funciones 1. ABS(7) ***basoluto*** 2. CEIL(7.35) ***redondeo*** 3. floor() 4. exp() 5. in()
6. mod()
7. power()
8. sqrt() 9. rand() ***aleatorios*** 10. round(7.97) ***redonde por abajo*** 11. round(id,2) ***1.00 agrega decimales*** 12. truncate(7.36598,3) ***7.36*** quita decimales

funciones para caracteres

1. upper(nombre) ***uppercase***
2. lower(nombre) ***lowercase***
3. concat(nombre, ' ', apellido) ***concatena***
4. UPPER(CONCAT(nombre, ' ', apellido)) ***funcion dentro de otra funcion***
5. LENGTH(CONCAT(nombre, ' ', apellido)) ***cantidad de caracteres***
6. trim() ***quita espacion de adelante y por detras de cada cadena***

funciones para fechas

1. curdate() ***fecha actual***
2. curtime() ***hora actual***
3. sysdate() ***fecha del sistema***
4. datediff(fecha, curdate) ***diferencia entre fechas***
5. dayname(fecha) ***nombre del dia del registro***
6. dayofmonth(fecha) ***dia del mes***
7. dayofweek(fecha) ***dia de la semana***
8. dayofyear(fecha) ***dia del año***
9. month(fecha) ***mes de esa fecha***
10. year(fecha) ***año de la fecha***
11. day(fecha) ***dia de la fecha***
12. hour(fecha) ***hora de la fecha***
13. minute(fecha) ***minuto de la fecha***
14. second(fecha) ***sugundo de la fecha***
15. date_format(fecha, '%d-%m-%Y') ***formatear la fecha, los mismos codigos de php***

funciones generales

1. ISNULL(apellido) ***comprueba si es nullo el campo, true o flase***
2. STRCMP('hola','hola') ***0*** / ('hola','ho1la') ***1 comprueba si son iguales o diferentes***
3. SELECT VERSION() FROM usuarios; ***me dice la version de SQL, debe ser dentro de la consulta SELECT*** **mi versión 8.0.22**
4. SELECT USER() FROM usuarios; ***root@localhost*** **Me devuelve el usuario root**
5. SELECT DISTINCT USER() FROM usuarios; ***Me muestra los usuarios distintos, si son el mismo solo muestra 1***

6. SELECT IFNULL(apellido, ‘Este campo esta vacio’)FROM usuarios; ***comprueba si esta vacio el campo esta vacio, y le agrega otro valor***

WHERE

consultas con WHERE es una condicional 1. SELECT * FROM usuarios WHERE email = ‘admin@admin.com (mailto:admin@admin.com)’; el WHERE lo que hace es solicitar o condicionar la consulta en un dato especifico que debe cumplirse

```
SELECT * FROM usuarios WHERE email = 'jared@gmail.com';
```

1 Jared Latorre jared@gmail.com (mailto:jared@gmail.com) 321654 2019-05-01

Ejemplos

1. mostrar nombre y apellido de todos los usuarios registrados en 2019

   ```
   SELECT nombre, apellido FROM usuarios WHERE YEAR(fecha) = 2019;
   ```

2. mostrar nombre y apellido de todos los usuarios ***QUE NO*** se registraron en 2019

   ```
   SELECT nombre, apellido FROM usuarios WHERE YEAR(fecha) != 2019;
   ```

3. mostrar nombre y apellido de todos los usuarios ***QUE NO*** se registraron en 2019 ***y/o que SEA NULL***

   ```
   SELECT nombre, apellido FROM usuarios WHERE YEAR(fecha) != 2019 OR
   ISNULL(fecha);

   SELECT nombre, apellido FROM usuarios WHERE YEAR(fecha) != 2019 OR NOT
   ISNULL(fecha);
   ```

WHERE y comodines %

3. mostra el email de los usuarios donde el apellido contenga la letra y que la contraseña sea igual a 1234

   ```
   SELECT email FROM usuarios WHERE apellido LIKE '%a%' AND password = '1234';
   ```

Ejemplos de cosulta SELECT

4. sacar todos los registros de la tabla usuarios, cuyo año sea ***PAR***

> La única diferencia entre números pares e impares que existe es que los pares son múltiplos del número 2 y los impares no. En otras palabras; Al ser divididos entre 2, todos los números pares darán como resto 0.

```
SELECT * FROM usuarios WHERE (YEAR(fecha) % 2 = 0);
```

1. sacar todos los registros de la tabla usuarios, cuyo año sea ***IMPAR***

```
SELECT * FROM usuarios WHERE (YEAR(fecha) % 2 != 0);
```

5. sacar todos los registros cuyo nombre tenga mas de 5 letras y ademas que se hayan registrado antes del 2020 y mostrar en mayusculas

```
SELECT UPPER(nombre), apellido FROM usuarios WHERE (LENGTH(nombre) > 5) AND
YEAR(fecha) < 2022;
```

LIMIT y ORDER BY

ordenamos por nombre de forma descendente y llamamos solo 3 registros

```
SELECT id, nombre FROM usuarios ORDER BY nombre desc LIMIT 3;
```

1. limit 4,3; ***desde el registro 4, 3 registros más***

```
5   Jared5
6   Jared6
7   Jared7
```

UPDATE y DELETE

actualizar registros

```
UPDATE usuarios SET fecha=CURDATE(), apellido='cero' WHERE id=1;
```

Auto Tarea

mirar como se cambias muchos registros y cambiarles el dominio del correo con like

eliminar registros

```
DELETE FROM usuarios WHERE id=8;

DELETE FROM usuarios WHERE email = 'admin@admin.com';
```

agrupamiento

consultas de agrupamiento

llenemos tablas para empezar hacer consultas de agrupamiento

codigo final

tabla categorias

```
id  nombre
1   MARVEL
2   DC COMICS
3   X-MEN
```

tabla de usuarios

```
id  nombre  apellido    email   password    fecha
1   Wizard  Deejay  wizard@gmail.com    123456  1984-01-08
2   Jared   Kenneth kenja@dumy.com  123456  2019-05-07
3   admon   admon   admon@dumy.com  123456  2021-05-31
```

lista de posts

```
id  usuario_id  catogeria_id titulo descripcion fecha
1   1   1   ironman tony stark  2021-05-31
2   1   1   spiderman   piter parker    2021-05-31
3   1   1   american captain    steven grant rogers 2021-05-31
4   1   1   wonder woman    lilli aspell    2021-05-31
5   2   2   batman  bruce wein  2021-05-31
6   2   2   flash   jay garrick 2021-05-31
7   2   2   superman    clar ken    2021-05-31
8   2   2   joker   jack naiper 2021-05-31
9   3   3   wolverin    logan   2021-05-31
10  3   3   magneto poder de manejar el hierro a voluntad   2021-05-31
11  3   3   profesor x  charlis francis xavier  2021-05-31
12  3   3   mystique    jenifer lawrens 2021-05-31
13  1   3   fenix   jean grey   2021-05-31
14  1   3   storm   Ororo Iqadi Munroe  2021-05-31
15  1   3   scarlet witch   wanda maximoff  2021-05-31
```

consultas de agrupamiento

lo que hace es argupar los titulos que se repiten y me muestra su cantidad, para eso es que sirven las consultas de agrupamiento

```
SELECT COUNT(titulo) FROM posts GROUP BY categoria_id;
```

consulta mas especifica

```
SELECT COUNT(titulo) as 'cantidad de posts', categoria_id AS categoria FROM posts
GROUP BY categoria_id;
```

consulta

```
cantidad de posts   categoria
4                   1
4                   2
7                   3
```

condiciones en consultas de agrupamiento HAVING

having es lo mismo que el WHERE pero para grupos

```
SELECT COUNT(titulo) AS 'Numero de posts', categoria_id FROM posts GROUP BY
categoria_id HAVING COUNT(titulo) >= 3;
```

solo trae la consulta agrupada que cumple la condicion

```
cantidad de posts   categoria
7                   3
```

funciones de agrupamiento

1. AVG ***promedio***
2. COUNT ***cuenta***
3. MAX ***valor maximo del grupo***
4. MIN ***valor minimo del grupo***
5. SUM ***suma todo el contenido del grupo*** ejemplos:

   ```
   SELECT AVG(id) 'promedio de posts' FROM posts;
   ```

   resultande del promedio

   ```
   promedio de posts
   8.0000
   ```

   sumemos todos los id’s

   ```
   SELECT SUM(id) AS 'suma de id' FROM posts;
   ```

subconsultas

- basicamente consiste en hacer consultas dentro de otras.
- consiste en utilizar los resultados de la subconsulta para operar en la consulta principal
- jugando con las claves ajenas / foraneas.

Operador IN

esta consulta busca los id que estan en los usuarios y coinciden con los registrados en la tabla posts

```
SELECT * FROM usuarios WHERE id IN (SELECT usuario_id FROM posts);
```

Code..

```
id  nombre  apellido    email   password    fecha
1   Wizard  Deejay  wizard@gmail.com    123456  1984-01-08
2   Jared   Kenneth kenja@dumy.com  123456  2019-05-07
3   super   super   super@super.com super   2021-06-01
5   chanchito   feliz   chanchito@chanchito.com feliz   2021-06-01
```

Operador NOT IN

esta consulta busca lo que no coincida en la consulta

```
SELECT * FROM usuarios WHERE id NOT IN (SELECT usuario_id FROM posts);
```

Code..

```
id  nombre  apellido    email   password    fecha
4   admin   admin   admin@admin.com admin   2021-06-01
6   lalala  lalala  lalala@lalala.com   lalala  2021-06-01
```

ejercicios

1. Sacar los usuarios que tengan algun posts que en su titulo hable de man

```
SELECT nombre, apellido FROM usuarios WHERE id IN (SELECT usuario_id FROM
posts WHERE titulo LIKE '%man');

nombre  apellido
Wizard  Deejay
Jared   Kenneth
```

2. sacar todas los posts de la categoria accion utilizando su nombre

```
SELECT titulo FROM posts WHERE categoria_id
IN (SELECT id FROM categorias WHERE nombre='accion')
```

3. mostrar las categorias con mas de tres entradas

```
SELECT * FROM categorias WHERE id IN (SELECT categoria_id FROM posts GROUP BY
categoria_id HAVING COUNT(categoria_id) >=5);

id  nombre
2   dccomic
3   xmen
```

4. mostrar los usuarios que crearon un posts un martes

```
SELECT * FROM usuarios WHERE id IN
(SELECT usuario_id FROM posts WHERE dayofweek(fecha)=3);
```

resultado: coincide con hoy que es martes

```
5   chanchito   feliz   chanchito@chanchito.com     2021-06-01
```

5. mostrar el nombre de él usuario que tenga mas posts

subconsulta

```
SELECT usuario_id, COUNT(id) FROM posts GROUP BY usuario_id ORDER BY COUNT(id)
DESC;
```

resultado

```
usuario_id  COUT(id)
1       7
3       4
2       4
5       1
```

Subconsulta completa

```
SELECT nombre FROM usuarios WHERE id IN
(SELECT usuario_id, COUNT(id) FROM posts GROUP BY usuario_id ORDER BY COUNT(id)
DESC LIMIT 1 );
```

1. mostrar las categorias sin posts

con posts

```
SELECT * FROM categorias WHERE id IN
(SELECT categoria_id FROM posts);
```

sin posts

```
SELECT * FROM categorias WHERE id NOT IN
(SELECT categoria_id FROM posts);
```

consultas multitablas y joins

consiste en hacer una consulta de varias tablas

1. vamos a mostrar todos los datos de los posts, con el nombre de usuario y el nombre la categoria.

2. para hacer las consultas multitabla debemos tener en cuenta

   ```
   SELECT campo1,campo2,campo3
   FROM tabla1, tabla2, tabla3
   WHERE e.usuario_id = u.id AND e.categoria_id = c.id
   ```

   WHERE condiciona la consulta.

Debemos hacer la condicion comparando los id de las llaves foraneas, con el fin de que coincidan los datos almacenados en las tablas

asi tenemos la tabla actualmente

id, usuario_id, categoria_id, titulo, descripcion, fecha

```
1   1   1   ironman tony stark  2021-05-31
2   1   1   spiderman   piter parker    2021-05-31
3   1   1   american captain    steven grant rogers 2021-05-31
4   1   2   wonder woman    lilli aspell    2021-05-31
5   2   2   batman  bruce wein  2021-05-31
6   2   2   flash   jay garrick 2021-05-31
7   2   2   superman    clar ken    2021-05-31
8   2   2   joker   jack naiper 2021-05-31
9   3   3   wolverin    logan   2021-05-31
10  3   3   magneto poder de manejar el hierro a voluntad   2021-05-31
11  3   3   profesor x  charlis francis xavier  2021-05-31
12  3   3   mystique    jenifer lawrens 2021-05-31
13  1   3   fenix   jean grey   2021-05-31
14  1   3   storm   Ororo Iqadi Munroe  2021-05-31
15  1   3   scarlet witch   wanda maximoff  2021-05-31
16  5   3   chanchi feliz   Hola mundo desde SQL subconsulta    2021-06-01
```

y de esta forma no podemos saber que esta pasando en la tabla, ya que no es presentable

vamos a mostrar todos los datos

que campos tenemos?

```
desc usuarios; # id nombre apellido email password fecha
desc posts; # id usuario_id categoria_id titulo descripcion fecha
desc categorias; # id nombre
```

traigamos datos

```
select * from posts, usuarios, categorias;
```

consulta de las tablas como quedo

consulta haciendo referencia a los campos de las tablas

```
SELECT usuarios.id , usuarios.nombre as 'usuario', posts.titulo as 'super heroe',
posts.descripcion as 'nombre real',  categorias.nombre as 'compañia', posts.fecha
FROM posts, usuarios, categorias
WHERE posts.usuario_id = usuarios.id AND posts.categoria_id = categorias.id ORDER
BY usuarios.id;
```

consulta haciendo referencia a las tablas con alias

```
SELECT u.id , u.nombre as 'User', p.titulo as 'Hero', p.descripcion as 'Secret
person',  c.nombre as 'Company', p.fecha as 'Date'
FROM posts p, usuarios u, categorias c
WHERE p.usuario_id = u.id AND p.categoria_id = c.id ORDER BY u.id;
```

1. u.id ***alias que hace referencia a la tabla usuarios.id*** 2. post p ***asignacion de un alias***
3. p.usuario_id = u.id ***comparamos id que conincida limitando los datos que traemos***
4. u.nombre as 'User ***renombrando campos***

ejemplos de consulta multitabla

1. mostrar el nombre de las categorias y al lado cuantos posts tienen

```
SELECT c.nombre, COUNT(p.id)
FROM categorias c, posts p
WHERE p.categoria_id = c.id GROUP BY p.categoria_id;
```

- traemos el nombre de la categoria - contamos el id de los post - condicionamos: que me compare el id de los posts contra el id de la tabla categoria - agrupamos por los posts que conincidan con los id de categoria_id, agrupando el mismo id de la llave foranea

2. mostrar el email de los usuarios y al lado cuantos posts tiene

mi consulta

```
SELECT u.email, count(p.id) AS 'total de posts' FROM usuarios u, posts p
WHERE u.id = p.usuario_id GROUP BY u.email;
```

la consulta del curso

```
SELECT u.email, count(titulo) FROM usuarios u, posts p WHERE p.usuario_id =
u.id group by p.usuario_id;
```

inner join

el inner join los que hace es una combinacion interna entre dos tablas

```
SELECT p.id, p.titulo, u.nombre AS 'User', c.nombre AS 'categoria'
FROM posts p
INNER JOIN usuarios u ON p.usuario_id = u.id
INNER JOIN categorias c ON p.categoria_id = c.id;
```

Lo que hace el inner join es traer la información de manera mas rapida para tablas con muchos registros, con tablas pequeñas es imperseptible

```
SELECT p.id, p.titulo, u.nombre AS 'User'
FROM posts p
INNER JOIN usuarios u ON p.usuario_id = u.id;
```

union de tablas

Left Join

- inner join cruza las tablas y muestra los registros que contengan datos y coincidan con ON y la condicion

  ```
  SELECT categorias.nombre, COUNT(posts.id)
  FROM categorias
  INNER JOIN posts ON posts.categoria_id = categorias.id
  GROUP BY posts.categoria_id;
  ```

- Left join me trae lo que este en la tabla de la izquierda, asi no tenga registros,
- manteniendo las filas de la izquierda, asi tenga registros en 0 o null

  ```
  SELECT categorias.nombre, COUNT(posts.id)
  FROM categorias
  LEFT JOIN posts ON posts.categoria_id = categorias.id
  GROUP BY posts.categoria_id;
  ```

Rigth Join

- mantiene todas las filas de la tabla de la derecha tengan registros en 0 o null

  ```
  SELECT c.nombre, COUNT(p.id)
  FROM posts p
  RIGTH JOIN categorias c ON p.categoria_id = c.id
  GROUP BY p.categoria_id;
  ```

Vistas en MySQL

- es almacenar una consulta en un nombre simplificando volver hacer toda la sentencia
- una consulta almacenada en la base de datos que se utiliza como tabla virtual
- No almacena datos si no que utiliza asociaciones y los datos originales de las tablas, de forma que siempre se mantiene actualizada
- las consultas virtuales se visualizan mostrando las tablas, ya que no existe comando para ver las vistas almacenadas

creacion de una vista

```
CREATE VIEW posts_por_categorias AS
SELECT c.nombre, COUNT(p.id)
FROM posts p
RIGHT JOIN categorias c ON p.categoria_id = c.id
GROUP BY p.categoria_id;
```

consultar las vistas creadas y las tablas

```
SHOW tables;
```

llamar la vista

```
SELECT * FROM posts_por_categorias;
```

haciendo consulta y llamando un dato específico

```
SELECT * FROM posts_por_categorias WHERE nombre='marvel';
```

borrar vistas

```
DROP VIEW posts_por_categorias
```