**CRUD**

**CREATE DATABASE** holamundo; **=>** Creamos una db

**SHOW DATABASES**; **=>** Muestra todas las db dentro de sys

**CREATE TABLE** **tabla** ( **=>** Creamos una **tabla**

id **INT** **NOT NULL AUTO\_INCREMENT**, **=>** Creamos columna id

tipo **VARCHAR(**50**) NOT NULL**, **=>** || || tipo

estado **VARCHAR(**25**) NOT NULL**,

**PRIMARY KEY (**id**) =>** Le asignamos un identificador único

);

**USE** **tabla**; **=>** Accedemos a la **tabla** para utilizarla

**INSTERT INTO** **tabla** (tipo, estado) **VALUES (** ‘valorDeTipo’, ‘valorDeEstado’**);**

* Agregamos una fila con valos valores que le pasamos en **VALUES**

|  |  |
| --- | --- |
| **ALTER TABLE** **tabla** **MODIFY COLUMN** id **INT** **AUTO\_INCREMENT**; | Modificamos la **tabla**. A la columna id le decimos que su valor sea un **INT** auto incremental |

**SHOW CREATE TABLE** **tabla**; => Muestra la **tabla**

**SELECT** **\* FROM** **tabla**; => Seleccionamos todo de la **tabla**.

**SELECT** **\* FROM** **tabla** **WHERE** id=1; => || donde la id sea 1.

**SELECT** **\* FROM** **tabla** **WHERE** estado = ‘x’; => los que tengan x estado.

**SELECT** **\* FROM** **tabla** **WHERE** estado = ‘x’ **AND** tipo ‘y’; => operador **AND**.

**⚠ Evitá usar** \* **en producción; mejor seleccionar solo las columnas necesarias.**

**DROP TABLE** **tabla**; => Eli**min**amos la **tabla**.

**DELETE FROM** **tabla** **WHERE** estado = ‘x’; => **☓** Error, es necesario un id (PK) so

**DELETE FROM** **tabla** **WHERE** estado = ‘x’ **AND** id=’2’; => **√** Eli**min**amos el registro.

**UPDATE** **tabla** **SET** estado = ‘nuevoEstado’ **WHERE** tipo = ‘valorTipo’; **☓** **Error, falta id**

**UPDATE** **tabla** **SET** estado = ‘nuevoEstado’ **WHERE** tipo = ‘valorTipo’ **AND** id = 2; **√**

**CONDICIONES**

|  |  |
| --- | --- |
| **CREAR TABLA CON COLUMNAS** | **AGREGAR FILAS** |
| **CREATE TABLE** users (  id **INT** **NOT NULL** **AUTO\_INCREMENT**,  name **VARCHAR**(50) **NOT NULL**,  age **INT** **NOT NULL**,  email **VARCHAR**(100) **NOT NULL**,  **PRIMARY KEY** (id)  ); | **INSTERT INTO** users (name, age, email)  **VALUES**  ('Ana', 30, 'ana@example.com'),  ('Carl’, 25, 'carlos@gmail.com'),  ('Marta', 98, 'marta@gmail.com'),  ('Mirta', 66, 'mirta@gmail.ar'),  ('Dan', 32, 'dan@gmail.com'); |

**SELECT** **\* FROM** users;

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **age** | **email** |
| 1 | Ana | 30 | ana@example.com |
| 2 | Carl | 25 | carlos@gmail.com |
| 3 | Marta | 98 | marta@gmail.com |
| 4 | Mirta | 66 | mirta@gmail.ar |
| 5 | Dan | 32 | dan@gmail.com |

**SELECT** **\* FROM** users **LIMIT** 1; => Nos devuelve el primer registro que encuentre

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **age** | **email** |
| 1 | Ana | 30 | ana@example.com |

**SELECT** **\* FROM** users **WHERE** **id > 2** **LIMIT** **3**; => Devuelve los primeros **3** registros que tengan un **id mayor a 2**.

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **age** | **email** |
| 3 | Marta | 98 | marta@gmail.com |
| 4 | Mirta | 66 | mirta@gmail.ar |
| 5 | Dan | 32 | dan@gmail.com |

**SELECT** **\* FROM** users =>=> Para pedir los ultimos tres registros <=<=

**ORDER BY** id **DESC** => Ordena los registros por id de mayor a menor

**LIMIT** 3; => Devuelve los 3 primeros que encuentra.

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **age** | **email** |
| 5 | Dan | 32 | dan@gmail.com |
| 4 | Mirta | 66 | mirta@gmail.ar |
| 3 | Marta | 08 | marta@gmail.com |

**SELECT** **\* FROM** user **WHERE** email = ‘ana@example.com’;

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **age** | **email** |
| 1 | Ana | 30 | ana@example.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SELECT** **\* FROM** users  **WHERE** (email **LIKE** ‘m%’ **OR** email **LIKE** ‘d%’)  **AND** (id > 3); | | | => Devuelve los registros en los cuales el valor de la columna email comienza con m o con d, y además si su id es > 3 | |
| **id** | **name** | **age** | | **email** |
| 3 | Marta | 98 | | marta@gmail.com |
| 4 | Mirta | 66 | | mirta@gmail.ar |
| 5 | Dan | 32 | | dan@gmail.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SELECT** **\* FROM** users  **WHERE** (email **LIKE** 'd%' **OR** email **LIKE** '%ar')  **AND** (id % 2 = 0); | | | => Devuelve los registros en los cuales el valor de la columna email comienza con ‘d’ o ter**min**a con ‘ar’, y además si su id es par; | |
| **id** | **name** | **age** | | **email** |
| 4 | Mirta | 66 | | mirta@gmail.ar |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SELECT** **\* FROM** users  **WHERE** (email **LIKE** '%ar' **OR** email **LIKE** 'a%')  **AND** (id < 5 )  **ORDER BY** id **DESC**; | | | => Devuelve los registros en los cuales el valor de la columna email ter**min**a con ‘ar’ o comienza con’a’, y además si su id es menor a 5, ordenado descendentemente. | |
| **id** | **name** | **age** | | **Email** |
| 4 | Mirta | 66 | | mirta@gmail.ar |
| 1 | Ana | 30 | | ana@example.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SELECT** **\* FROM** users  **WHERE** (age BETWEEN 30 **AND** 70)  **AND** (email **LIKE** '%.c%'); | | | => Devuelve los registros en los cuales el valor de la columna age este entre 30 y 70, y el valor de la columna email  contenga ‘.c’ | |
| **id** | **name** | **age** | | **Email** |
| 1 | Ana | 30 | | ana@example.com |
| 5 | Dan | 32 | | dan@gmail.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SELECT** **\* FROM** users  **WHERE** NOT (age BETWEEN 30 **AND** 70)  **AND** NOT(email **LIKE** 'c%'); | | | => Devuelve los registros en los cuales el valor de la columna age NO este entre 30 y 70, y el valor de la columna email NO comience con ‘c’. | |
| **id** | **name** | **age** | | **Email** |
| 3 | Marta | 98 | | marta@gmail.com |

|  |  |
| --- | --- |
| **SELECT** **max**(age), **min**(age) **FROM** users; | => Devuelve dos valores, el del mayor y menor valor dentro de la columan age de todos los registro de users. |
| **Max(age)** | **Min(age)** |
| 98 | 25 |

|  |  |
| --- | --- |
| **SELECT** **max**(age) **AS** mayorEdad,  **min**(age) **AS** menorEdad  from users; | => Lo mismo pero le asignamos un valor al nombre de la columna |
| **mayorEdad** | **menorEdad** |
| 98 | 25 |

|  |  |
| --- | --- |
| **SELECT** id, name **FROM** users; | => Seleccionamos solo las columnas que nos **INT**eresan |
| **id** | **name** |
| 1 | Ana |
| 2 | Carl |
| 3 | Marta |
| 4 | Mirta |
| 5 | Dan |

|  |  |
| --- | --- |
| **SELECT** id **AS** identifier,  Name **AS** username  **FROM** users; | => Le damos un nuevo alias a las columnas |
| **identifier** | **userName** |
| 1 | Ana |
| 2 | Carl |
| 3 | Marta |
| 4 | Mirta |
| 5 | Dan |

**JOINS**

|  |  |
| --- | --- |
| **CREAR TABLA CON COLUMNAS** | **AGREGAR FILAS** |
| **CREATE TABLE** products (  id **INT** **NOT NULL** **AUTO\_INCREMENT**,  name **VARCHAR**(50) **NOT NULL**,  created\_by **INT** **NOT NULL**,  brand **VARCHAR**(25) **NOT NULL**,  **PRIMARY KEY** (id),  **FOREIGN KEY**(created\_by) **REFERENCES** users (id)  ); | **INSTERT INTO** products  (name, brand, created\_by)  **VALUES**  ('1100', 'nokia', 1),  ('Galaxy S22', 'samsung', 3),  ('Redmi N 11', 'xiaomi', 4),  ('P30 Lite', 'huawei', 1),  ('Moto G100', 'motorola', 5),  ('Galaxy A54', 'samsung', 2),  ('Redmi 12C', 'xiaomi', 3),  ('P50 Pro', 'huawei', 4),  ('Moto E22', 'motorola', 2); |

La columna  **created\_by**  tiene asignada una **FOREIGN KEY** que hace referencia a la id de un usuario, es decir users (id). Esto quiere decir que la columna created\_by va a contener solamente valores que pertenezcan a la id de algún usuario. Si yo hiciera: ('Moto E22', 'motorola', **6**); daría error ya que solo hay 5 usuarios, por ende la id máxima es 5.

*"En esta* ***tabla****, el campo created\_by solo puede tener valores que ya existan en el campo id de la* ***tabla*** *users. No se puede insertar ni actualizar un producto si created\_by apunta a un usuario que no existe."*

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **name** | **brand** | **created\_by** |
| 1 | 1100 | nokia | 1 |
| 2 | Galaxy S22 | samsung | 3 |
| 3 | Redmi N 11 | xiaomi | 4 |
| 4 | P30 Lite | huawei | 1 |
| 5 | Moto G100 | motorola | 5 |
| 6 | Galaxy A54 | samsung | 2 |
| 7 | Redmi 12C | xiaomi | 3 |
| 8 | P50 Pro | huawei | 4 |
| 9 | Moto E22 | motorola | 2 |

|  |  |
| --- | --- |
| **LEFT JOIN** | Hago una unión de las dos **tabla**s, siendo la **tabla** **users** la que ‘manda’ ya que de ella vendrán todos los registros.  En cambio de la **tabla** **products** traemos solo los registros que tengan relación con los traídos de **users**.  En este ejercicio agregué esta fila a **users** ('Ray', 15, 'ray@gmail.com'); para que exista un usuario que no tiene asociado ningún registro dentro de **products**. |
| **SELECT**  u.id **AS** user\_id,  u.email **AS** user\_email,  p.name **AS** prod\_name  **FROM** users u  LEFT JOIN products p  **ON** u.id = p.created\_by; | |  |  |  | | --- | --- | --- | | **user\_id** | **user\_email** | **prod\_name** | | 1 | ana@example.com | 1100 | | 1 | ana@example.com | P30 Lite | | 2 | carlos@gmail.com | Galaxy A54 | | 2 | carlos@gmail.com | Moto E22 | | 3 | marta@gmail.com | Galaxy S22 | | 3 | marta@gmail.com | Redmi 12C | | 4 | mirta@gmail.ar | Redmi N 11 | | 4 | mirta@gmail.ar | P50 Pro | | 5 | dan@gmail.com | Moto G100 | | 6 | ray@gmail.com | ***null*** | |

Al hacer el **LEFT JOIN** llamamos a todos los registros de la **tabla** de la izquierda, luego le decimos que si se cumple una coincidencia entre dos valores (uno en una **tabla** y otro en otra) también traiga el registro que cumple con esa coincidencia. En este caso la coincidencia seria que **user id = product created\_by;**

Si alguno de los productos tiene un valor de **created\_by** que coincide con la **id** de alguno de los usuarios, trae ese registro. Si no tiene coincidencia no lo trae. Pero no trae el que está en la lista de la derecha, ya que como mencioné antes, de la **tabla** izquierda se traen **TODOS** los valores.

En el caso de ‘Ray’ podemos ver que la consulta lo trae, ya que está dentro de la **tabla** de la izquierda. Pero al no coincidir con ninguno de los valores de **created\_by** de ninguno de los productos, devuelve como resultado ***null.*** Esto se podría interpretar como que Ana creó 2 productos, el Nokia 1100 y el P30 Lite. Dan creó el Moto G100, y Ray no creó ningún producto.

|  |  |
| --- | --- |
| **RIGHT JOIN** | Hago una unión de las dos **tabla**s, siendo la **tabla** **products** la que ‘manda’ ya que de ella vendrán todos los registros.  En cambio de la **tabla** **users** traemos solo los registros que tengan relación con los traídos de **products**. |
| **SELECT**  u.id user\_id,  u.email **AS** u\_email,  p.name **AS** p\_name,  p.id **AS** id\_prod  **FROM** users u  **RIGHT JOIN** products p  **ON** u.id = p.created\_by; | |  |  |  |  | | --- | --- | --- | --- | | **user\_id** | **u\_email** | **p\_name** | **id\_prod** | | 1 | ana@example.com | 1100 | 1 | | 3 | marta@gmail.com | Galaxy S22 | 2 | | 4 | mirta@gmail.ar | Redmi N 11 | 3 | | 1 | ana@example.com | P30 Lite | 4 | | 5 | dan@gmail.com | Moto G100 | 5 | | 2 | carlos@gmail.com | Galaxy A54 | 6 | | 3 | marta@gmail.com | Redmi 12C | 7 | | 4 | mirta@gmail.ar | P50 Pro | 8 | | 2 | carlos@gmail.com | Moto E22 | 9 | |

Acá estamos haciendo lo mismo pero al revez, en este caso la **tabla** de la derecha **products** es la que ‘manda’, entonces va a traer todos los registros de esta. En cambio de la **tabla** **users** se va a traer solo los registros que cumplan con la coincidencia que se le está pasando u.**id =** p.**created\_by;**

Por eso se aprecian dos cosas importantes:

1. El orden de los elementos está dado por la id de **products**, no por la id de los **users**.
2. Los usuarios que **NO** tienen asociado ningún producto (qué en el ejemplo anterior tenían valor ***null***) **NO** se muestran en la lista.

|  |  |
| --- | --- |
| **INNER JOIN** | Nos va a traer **users** y **products** siempre y cuando estos puedan ser asociados. |
| **SELECT**  u.id user\_id,  u.email **AS** u\_email,  p.name **AS** p\_name,  p.id **AS** id\_prod  **FROM** users u  **INNER JOIN** products p  **ON** u.id = p.created\_by; | |  |  |  |  | | --- | --- | --- | --- | | **user\_id** | **u\_email** | **p\_name** | **id\_prod** | | 1 | ana@example.com | 1100 | 1 | | 1 | ana@example.com | P30 Lite | 4 | | 2 | carlos@gmail.com | Galaxy A54 | 6 | | 2 | carlos@gmail.com | Moto E22 | 9 | | 3 | marta@gmail.com | Galaxy S22 | 2 | | 3 | marta@gmail.com | Redmi 12C | 7 | | 4 | mirta@gmail.ar | Redmi N 11 | 3 | | 4 | mirta@gmail.ar | P50 Pro | 8 | | 5 | dan@gmail.com | Moto G100 | 5 | |

|  |  |
| --- | --- |
| **CROSS JOIN** | En este caso se van a cruzar todos los registros de una **tabla** con todos los registros de la otra.  **Hay que tener cuidado porque devuelve muchos registros.** |
| **SELECT**  u.id user\_id,  u.email **AS** u\_email,  p.name **AS** p\_name,  p.id **AS** id\_prod  **FROM** users u  **CROSS JOIN** products p  **ON** u.id = p.created\_by; | |  |  |  |  | | --- | --- | --- | --- | | **user\_id** | **u\_email** | **p\_name** | **id\_prod** | | 6 | ray@gmail.com | 1100 | 1 | | 5 | dan@gmail.com | 1100 | 1 | | 4 | mirta@gmail.ar | 1100 | 1 | | 3 | marta@gmail.com | 1100 | 1 | | 2 | carlos@gmail.com | 1100 | 1 | | 1 | ana@example.com | 1100 | 1 | | 6 | ray@gmail.com | Galaxy S22 | 2 | | … | … | … | 2 | |

**GROUP BY**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SELECT**  **count**(id) **AS** cantidad,  brand **AS** marca  **FROM** products  **GROUP BY** brand  **ORDER BY** **count**(id) **DESC**; | |  |  | | --- | --- | | **cantidad** | **marca** | | 2 | samsung | | 2 | xiaomi | | 2 | huawei | | 2 | motorola | | 1 | nokia | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SELECT**  **count**(p.id) **AS** cantidad,  u.name **AS** user\_name  **FROM** products p  **LEFT JOIN** users u  **ON** u.id = p.created\_by  **GROUP BY** p.created\_by  **HAVING** **count**(p.id) < 2; => A  -- O podría ser :  **HAVING** **count**(p.id) > 1; => B | A   |  |  | | --- | --- | | **cantidad** | **user\_name** | | 1 | Dan |   B   |  |  | | --- | --- | | **cantidad** | **user\_name** | | 2 | Ana | | 2 | Carl | | 2 | Marta | | 2 | Mirta | |

Aunque el SQL se escribe en cierto orden, **MySQL lo ejecuta INTernamente así**:

**1. FROM + JOIN**

Se construye la **tabla** temporal que une products con users, usando **LEFT JOIN**.  
Esto significa: para cada producto, se busca el usuario cuyo id coincida con created\_by.  
Si no hay coincidencia, se rellena con NULL en los campos del usuario.

En este paso, ya tenemos una **tabla** con filas como:  
p.id | p.name | p.created\_by | u.name

**2. GROUP BY** p.**created\_by**

Se agrupan todas las filas según el valor de p.created\_by.

Ejemplo:

* Todos los productos creados por el usuario 1 → van a un grupo.
* Los del usuario 2 → a otro grupo, etc.

Ahora, en vez de trabajar con filas individuales, **trabajás con grupos de filas**.

**3. COUNT(**p.**id)**

En este contexto, COUNT(p.id) **cuenta cuántos productos hay en cada grupo** (por usuario).

Es decir:

* Si el usuario 1 creó 3 productos, el conteo en ese grupo será 3.
* Si el usuario 2 no creó productos, ese grupo tendrá 0 (aunque no se verá si p.id es NULL, porque COUNT(p.id) **no cuenta NULLs**).

**4. HAVING count(**p.**id) > 1**

Este filtro se aplica **después de agrupar**.  
Se eli**min**an todos los grupos donde el conteo de productos es 1 o menos.

**5. SELECT count(**p.**id),** u.**name**

Finalmente, se seleccionan:

* Cuántos productos tiene el grupo (usuario)
* El nombre del usuario (u.name) — puede ser NULL si el **LEFT JOIN** no encontró coincidencia