MySQL

1. `Select` statement is used to specify which column we want as output or what we want as output
2. `From` statement is used to specify the table from where we need data
3. `Where` statement takes the condition on which we want our data to be filtered
4. `Order by` statement sorts the data in the required format
5. `Between` statement is used to specify the range
6. `In` is used to iterate over something , just like an list iteration
7. `Like` statement is just like regex
8. `Regexp` like regular expressions $-end |-or ^-start [gim] g or m or i, [a-h] range
9. `Is null` operator for finding our null rows
10. `Limit` how many values you want from the query
11. `Join` < Table name>`On` Joining tables

`Select` First_name, First_name + Last_name `AS` Full_name

`From` Customers

`Where` Loylty_points ≥100 and loyty_points≤1000

`Order by` First_name `Desc` or `Asc`, we can also order by a custom math expression

`Order by` loyolty_points*quantity `Desc` or `Asc`,

`Select` `*`

`From` Customers, Orders

`Where` order_date `Between` '21/09/21' `and` '30/09/21'

```
Select *
From Customers, Orders
Where states In ('VA', 'MA', 'CA')
```

```
Select *
From customers
Where First_name Like '%b'
```

```
Select *
From customers
Where First_name Regexp '[s-m]k'
```

```
Select *
From customers
Where First_name IS NULL
```

Inner Joins

```
Select Order_id, orders.customer_id, first name
From orders
Join customers
    on orders.customer_id == customers.customer_id
```

```
Select Order_id, o.customer_id, first name
From orders o
Join customers  c
```

`on` o.customer_id == c.customer_id

Joining Across Databases

`use` sql_inventory

`Select` Order_id, o.customer_id, first name

`From` sql_store.order_items oi

`Join` sql_inventory.products p

`on` oi.product_id == p.product_id

Self Joins

Suppose there is a database having employee id and manger id that he reports to but as manager is also an employee so we can join tables to know which employee has which manager except he is not the CEO

`select` e.employee_id, e.first_name `as` employee, m.first_name `as` manager

`from` employees e

`join` employees m

`on` e.reports_to = m.employee_id

Joining Multiple Tables

`select` o.order_id, o.order_date, c.first_name, c.last_name, os.name `as` status

`from` orders o

`join` customers c

`on` o.customer_id = c.customer_id

`Join` order_statuses os

```
    on o.status = os.order_status_id
```

Compound Join condition

If Values are duplicated in every column and have no primary key or composite primary key, we can use more than one column to uniquely identify other column values

```
select *
from order_items oi
join order_item_notes oin
    on oi.order_id = oin.order_id
    and oi.product_id = oin.product_id
```

Implicit Join syntax

Explicit join

```
select *
from orders o
join customers c
    on o.customer_id = c.customer_id
```

Implicit syntax

```
select *
from order o, customers c
where o.customer_id = c.customer_id
```

Outer Join

some of the customers dont have orders so we use this becuase inner will return where only the on condition is satisfied

It return all values even if the equality is satisfied or not

`Left Join`

`Right Join`

```
select c.customer_id, c.first_name, o.order_id
from customers c
left join orders o
    on c.customer_id = o.customer_id
order by c.customer_id
```

Outer Join between multiple tables

```
select c.customer_id, c.first_name, o.order_id, sh.name as shipper
from customers c
left join orders o
    on c.customer_id = o.customer_id
left join shippers sh
    on o.shipper_id = sh.shipper_id
order by c.customer_id
```

Avoid using right join as it gets messy, use left joins instead

Self Outer Join

```
select *
```

`from` employee e

`left join` employee m

`on` e.reports_to = m.employee_id

The Using clause

if column is same across all tables otherwise it wont work

`select` *

o.order_id

c.first_name

sh.name as shipper

`from` orders o

`join` customers c

`using` (customer_id)

`left join` shippers sh

`using` (shipper_id)

`select` *

`from` order_item oi

`join` order_item_notes oin

`using` (order_id, product_id)

earlier we used and condition when we didn't had unique column to use a combination of columns to uniquely identify something

Natural Joins

Not recommended

`select` o.order_id, c.first_name

`from` orders o

`Natural join` customers c

Cross Join

Joining **every record** in customer table to every record in product table

implicit syntax

`select` sh._name as shipper, p.name as product

`from` shipper sh products p

`order by` sh.name

explicit syntax

`select` c.first_name as customer, p.name as product

`from` customers c

`cross join` products p

`order by` c.first_name

Combining Records from multiple tables

`Union` keyword

`select` first_name

`from` customers

`union`

`select` name

`from` shippers

Column Attributes

insert, update and delete data

`Insert Into` customers (

first_name, last_name, birth_date)

`Values` (Default , 'john', 'smith', '15/03/2000')

Inserting Multiple rows

`Insert Into` shippers (name)

`Values` ('shipper1'),

('shipper2'),

('shipper3')

Creating a copy of a table

`Create` `Table` orders_archive as - write anything below this as subquery that will be copied

`select` * from orders

Updating a Single/Multiple row

`Update` invoices

`set` payment_total = 10, payment_date = '2019'

`where` invoice_id =1

Setting again to default

```
Update invoices
set payment_total = Default, payment_date = '2019'
where invoice_id =1
```

multiple rows

```
Update invoices
set
    payment_total = invoice_total*0.5,
    payment_date = due_date
where invoice_id in (3,4)
```

Deleting rows

```
delete from invoices
where invoice_id =1
 select *
from clients where name = 'myworks'
```

Restoring the database