Practically work with MYSQL Tool

1-Creating the Databases

First open the file create database.sql in mysql workbench. when you click on yellow icon above it run the whole written code and you can see this on output terminal window. But if you only select specific line of code then select yellow icon it will only run that line. On side bar of schema when you refresh this you can see number of files on schema bar

1; The Select Statement:

First write USE sql_store and run this and it will select sql_store database that become bold on side bar.

```
USE sql_store;

SELECT *
FROM customers
WHERE customer_id = 1
ORDER BY first_name
```

In this the order of the statements is necessary like first comes SELECT and then others.

2- The SELECT Clause

(i)

```
SELECT
        first_name,
        last_name,
        points,
        (points + 10) * 100 AS 'discount factor'
FROM customers
```

(ii)

```
SELECT DISTINCT state
```

FROM customers

EXERCISE

Return all he products

--Name

--unit price

--new price (unit price * 1.1)

Solution

SELECT name, unit_price, unit_price * 1.1 AS new_ price

FROM products

3- The WHERE Clause

(i)

SELECT *

FROM customers

WHERE points > 3000

(ii)

SELECT *

FROM customers

WHERE state = 'VA'

(iii)

SELECT *

FROM customers

WHERE state <> 'VA'

(iv)

```
SELECT *
FROM customers
WHERE birth_date = '1990-01-01'
```

Exercise:

Get the orders placed this year

Solution:

```
SELECT *
FROM orders
WHERE order_date > '2017-01_01'
```

4- The AND, OR and NOT Operations

(i)

```
SELECT *
FROM customers
WHERE birth_date > '190-01-01' AND points > 1000
```

(ii)

```
SELECT *
FROM customers
WHERE birth_date > '190-01-01' OR points > 1000
```

(iii)

```
SELECT *
FROM customers
WHERE birth_date > '190-01-01' OR points > 1000 AND
        state = 'VA'
```

(iv)

```
SELECT *
```

```
FROM customers
WHERE birth_date > '190-01-01' OR
(points > 1000 AND state = 'VA')
```

The AND operator always evaluated first and the OR

(v)

```
SELECT *
FROM customers
WHERE NOT (birth_date > '1990-01-01' OR points > 1000)
```

(vi)

```
 SELECT *
FROM customers
WHERE birth_date <= '1990-01-01' AND points <= 1000
```

Exercise:

From the order_ items table, get the items

For order #6

Where the total price is greater than 30

Solution:

```
SELECT *
FROM order_items
WHERE order_id = 6 AND unit_price * quantity > 30
```

5- The IN Operators

(i)

```
SELECT *
```

```
FROM Customers
WHERE state IN ('VA', 'FL', 'GA')
```

(ii)

```
SELECT *
FROM Customers
WHERE state NOT IN ('VA','FL', 'GA')
```

EXERCISE:

Return Products with

Quantity in stock equals to 49, 38,72

Solution:

```
SELECT *
FROM Products
WHERE quantity_in_stock IN (49,38,72)
```

6- The Between Operators

(i)

```
SELECT *
FROM customers
WHERE points BETWEEN 1000 AND 3000
```

Exercise:

Return customers born

Between 1/1/1990 and 1/1/2000

Solution:

```
SELECT *
FROM customers
WHERE birth_date BETWEEN '1990-01-01' AND '2000-01-01'
```

7- The Like Operator

(i)

```
SELECT *
FROM customers
WHERE last_name LIKE  'b%'
```

(ii)

```
SELECT *
FROM customers
WHERE last_name LIKE  'brush%'
```

(iii)

```
SELECT *
FROM customers
WHERE last_name LIKE  '%b%'
```

(iv)

```
SELECT *
FROM customers
WHERE last_name LIKE  '%y'
```

(v)

```
SELECT *
FROM customers
WHERE last_name LIKE  '_____y'
```

Exercise:

Get the Customers whose

Addresses contains TRAIL or AVENUE

Phone number end with 9

Solution:

```
SELECT *
FROM customers
WHERE address LIKE '%trail%' OR
      address like '%avenue%'

SELECT *
FROM customers
WHERE phone LIKE '%9'
```

8- The REGEXP Operator

(i)

```
SELECT *
FROM customers
WHERE last_name REGEXP 'field'
```

(ii)

```
SELECT *
FROM customers
WHERE last_name REGEXP 'field$'
```

(iii)

```
SELECT *
FROM customers
```

```
WHERE last_name REGEXP 'field|mac'
```

(iv)

```
SELECT *
FROM customers
WHERE last_name REGEXP 'field|mac|rose'
```

(v)

```
SELECT *
FROM customers
WHERE last_name REGEXP '^field|mac|rose'
```

(vi)

```
SELECT *
FROM customers
WHERE last_name REGEXP 'field$|mac|rose'
```

(vii)

```
SELECT *
FROM customers
WHERE last_name REGEXP '[gim]e'
```

(viii)

```
SELECT *
FROM customers
WHERE last_name REGEXP '[a-h]e'
```

^ beginning

$ end

| logical

[acd]

[a-f]

Exercise:

Get the customers whose

First names are ELKA or AMBUR

Last name end with EY or ON

Last names start with MY or contains SE

Last names contain B followed by R or U

Solution:

```
SELECT *
FROM customers
WHERE first_name REGEXP 'elka|ambur'
```

```
SELECT *
FROM customers
WHERE last_name REGEXP 'ey$|on$'
```

```
SELECT *
FROM customers
WHERE last_name REGEXP '^my|se'
```

```
SELECT *
FROM customers
WHERE last_name REGEXP 'b[ru]'
```

9- The IS NULL Operator

(i)

```
SELECT *
FROM customers
WHERE phone IS NULL
```

(ii)

```
SELECT *
FROM customers
WHERE phone IS NOT NULL
```

Exercise:

Get the Orders that are not shipped yet

Solution:

```
SELECT *
FROM orders
WHERE shipper_id IS NULL
```

10- The ORDER BY Clause

(i)

```
SELECT *
FROM customers
ORDER BY first_name
```

(ii)

```
SELECT *
FROM customers
ORDER BY first_name DESC
```

(iii)

```
SELECT *
FROM customers
ORDER BY state, first_name
```

(iv)

```
SELECT first_name, last_name
FROM customers
ORDER BY birth_date
```

(v)

```
SELECT first_name, last_name, 10 AS points
FROM customers
ORDER BY  points, first_name
```

(vi)

```
SELECT first_name, last_name, 10 AS points
FROM customers
ORDER BY 1,2
```

Exercise:

Select all the items where order_id is 2 and sort them with total price in descending order

Solution:

```
SELECT *
FROM order_items
WHERE order_id = 2
ORDER BY quantity * unit_price DESC
```

11- The LIMIT Clause

(i)

```
SELECT *
FROM customers
LIMIT
```

(ii)

```
SELECT *
FROM customers
LIMIT 6, 3
```

It skips first six records and then select 7,8 and 9

Exercise:

Get top 3 loyal customers

Solution:

```
SELECT *
FROM customers
ORDER BY points DESC
LIMIT 3
```

If we look at order then LIMIT clause comes at the end

12- Inner Joins

(i)

```
SELECT *
FROM orders
JOIN customers
        ON orders.customer_id = customers.customer_id
```

(ii)

```
SELECT order_id,first_name, last_name
```

```
FROM orders
JOIN customers
        ON orders.customer_id = customers.customer_id
```

(iii)

```
SELECT orders.customer_id, order_id,first_name, last_name
FROM orders
JOIN customers
        ON orders.customer_id = customers.customer_id
```

(iv)

```
SELECT o.customer_id, order_id,first_name, last_name
FROM orders o
JOIN customers c
        ON o.customer_id = c.customer_id
```

Exercise:

Solution:

```
SELECT order_id, oi.product_id, quantity, oi.unit_price
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
```

13- Joining Across DataBases

(i)

```
SELECT *
FROM order_items oi
JOIN sql_inventory.products p
```

```
    ON oi.product_id = p.product_id
```

(ii)

```
USE sql_inventory;

SELECT *
FROM sql_store.order_items oi
JOIN sql_inventory.products p
    ON oi.product_id = p.product_id
```

14- SELF JOIN

(i)

```
USE sql_hr;

SELECT *
FROM employees e
JOIN employees m
    ON e.reports_to = m.employee_id
```

(ii)

```
SELECT
  e.employee_id,
  e.first_name,
  m.first_name
FROM employees e
JOIN employees m
    ON e.reports_to = m.employee_id
```

(iii)

```
SELECT
   e.employee_id,
   e.first_name,
   m.first_name AS manager
FROM employees e
JOIN employees m
        ON e.reports_to = m.employee_id
```

15- Joining Multiple Tables

```
USE sql_store;

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
```

Exercise:

For the invoice we have payment for the client using credit card

Solution:

```
USE sql_invoicing;

SELECT *
```

```
FROM payments p
JOIN clients c
        ON  p.client_id = c.client_id
JOIN payment_methods pm
        ON p.payment_method = pm.payment_method_id



SELECT
  p.date,
   p.invoice_id,
   p.amount,
   c.name,
   pm.name
FROM payments p
JOIN clients c
        ON  p.client_id = c.client_id
JOIN payment_methods pm
        ON p.payment_method = pm.payment_method_id
```

16- Compound JOIN Conditions

```
USE sql_store;


SELECT *
FROM order_items oi
JOIN order_item_notes oin
      ON oi.order_id = oin.order_id
    AND oi.product_id = oin.product_id
```

17- Implicit Join Syntax

(i)

```
SELECT *
FROM orders o, customers c
WHERE o.customer_id = c.customer_id
```

18- Outer Joins (LEFT AND RIGHT)

(i)

```
SELECT
   c.customer_id,
   c.first_name,
   o.order_id
FROM customers c
JOIN orders o
        ON c.customer_id = o.customer_id
ORDER BY c.customer_id
```

(ii)

```
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
```

(iii)

```
SELECT
```

```
    c.customer_id,
    c.first_name,
    o.order_id
FROM customers c
RIGHT JOIN orders o
        ON c.customer_id = o.customer_id
ORDER BY c.customer_id
```

(iv)

```
SELECT
    c.customer_id,
    c.first_name,
    o.order_id
FROM orders o
RIGHT JOIN customers c
        ON c.customer_id = o.customer_id
ORDER BY c.customer_id
```

Exercise:

Solution:

```
SELECT
    p.product_id,
    p.name,
    oi.quantity
FROM products p
LEFT JOIN order_items oi
        ON p.product_id = oi.product_id
```

19- Outer Joins Between Multiple Tables

(i)

```
SELECT
   c.customer_id,
   c.first_name,
   o.order_id
FROM customers c
RIGHT JOIN orders o
        ON c.customer_id = o.customer_id
JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
ORDER BY c.customer_id
```

(ii)

```
SELECT
   c.customer_id,
   c.first_name,
   o.order_id
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
```

(iii)

```
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
```

Exercise:

Solution1:

```
SELECT
    o.order_id,
    o.order_date,
    c.first_name AS customer,
    sh.name AS shipper
FROM orders o
JOIN customers c
        ON o.customer_id = c.customer_id
JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
```

Solution 2:

```
SELECT
    o.order_id,
```

```
    o.order_date,
    c.first_name AS customer,
    sh.name AS shipper
FROM orders o
JOIN customers c
        ON o.customer_id = c.customer_id
LEFT JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
```

(iii)

```
SELECT
    o.order_id,
    o.order_date,
    c.first_name AS customer,
    sh.name AS shipper,
    os.name AS status
FROM orders o
JOIN customers c
        ON o.customer_id = c.customer_id
LEFT JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
JOIN order_statuses os
        ON o.status = os.order_status_id
```

20- SELF OUTER JOINS

(i)

```
USE sql_hr;

SELECT
```

```
  e.employee_id,
    e.first_name,
    m.first_name AS manager
FROM employees e
JOIN employees m
        ON e.reports_to = m.employee_id
```

(ii)

```
USE sql_hr;

SELECT
    e.employee_id,
    e.first_name,
    m.first_name AS manager
FROM employees e
LEFT JOIN employees m
        ON e.reports_to = m.employee_id
```

21-The Using Clause

(i)

```
USE sql_store;

SELECT
    o. order_id,
    c.first_name
FROM orders o
JOIN customers c
        USING(customer_id)
JOIN shippers sh
```

```
    USING(shipper_id)
```

Exercise:

Solution:

```
USE sql_invoicing;
SELECT
   p.date,
   c.name AS client,
   p.amount,
   pm.name AS payment_method
FROM payments p
JOIN clients c USING(client_id)
JOIN payment_methods as pm
    ON p.payment_method = pm.payment_method_id
```

22- Natural JOINS

```
USE sql_store;
SELECT
   o.order_id,
   c.first_name
FROM orders o
NATURAL JOIN customers c
```

23- CROSS JOINS

```
USE sql_store;
SELECT
  c.first_name AS customer,
   p.name AS product
```

```
FROM customers c
CROSS JOIN products p
ORDER BY c.first_name
```

Exercise:

Do a cross join between shippers and products

Using the implicit Syntax

And then Using the explicit Join

Solution:

Implicit syntax

```
SELECT
    sh.name AS shipper,
    p.name AS product
FROM shippers sh, products p
ORDER BY sh.name
```

Explicit Syntax

```
SELECT
    sh.name AS shipper,
    p.name AS product
FROM shippers sh
CROSS JOIN products p
ORDER BY sh.name
```

24- Unions

(i)

```
USE sql_store;
SELECT *
```

```
FROM orders
WHERE order_date >= '2019-01-01'
```

(ii)

```
SELECT
   order_id,
   order_date,
   'ACTIVE'

FROM orders
WHERE order_date >= '2019-01-01'
```

(iii)

```
SELECT
   order_id,
   order_date,
   'ACTIVE' AS status

FROM orders
WHERE order_date >= '2019-01-01'
```

(iv)

```
SELECT
   order_id,
   order_date,
   'ARCHIVED' AS status

FROM orders
WHERE order_date < '2019-01-01'
```

(v)

```
SELECT
   order_id,
   order_date,
   'ACTIVE' AS status
FROM orders
WHERE order_date >= '2019-01-01'
UNION
SELECT
   order_id,
   order_date,
   'ARCHIVED' AS status
FROM orders
WHERE order_date < '2019-01-01'
```

(vi)

```
SELECT first_name
FROM customers
UNION
SELECT name
FROM shippers
```

(vi)

```
SELECT name
FROM shippers
UNION
SELECT first_name
FROM customers
```

Exercise:

Solution:

(i)

```
SELECT
   customer_id,
   first_name,
   points,
   'BRONZE' AS type
FROM customers
WHERE points < 2000
```

(ii)

```
SELECT
   customer_id,
   first_name,
   points,
   'BRONZE' AS type
FROM customers
WHERE points < 2000
UNION
SELECT
   customer_id,
   first_name,
   points,
   'SILVER' AS type
FROM customers
WHERE points BETWEEN 2000 AND 3000
UNION
SELECT
```

```
    customer_id,
    first_name,
    points,
    'GOLD' AS type
FROM customers
WHERE points > 3000
ORDER BY first_name
```

25- Column Attributes

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INT,
    hire_date DATE DEFAULT CURRENT_DATE,
    CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES
departments(department_id)
);
```

26- Inserting a Single Row

```
INSERT INTO customers  (
    last_name,
    first_name,
    birth_date,
    address,
    city,
    state)
```

```
VALUES (
   'SMITH',
   'JOHN',
   '1990-01-01',
   'address',
   'city',
   'CA')
```

27- INSERTING Into Multiple Rows

```
INSERT INTO shippers (name)
VALUES ('Shipper 1'),
          ('shipper 2'),
          ('shipper 3')
```

Exercise:

Insert three rows in product table

Solution:

```
INSERT INTO products (name, quantity_in_stock, unit_price)
VALUES ('Product1',10,1.95),
          ('Product2',11,1.95),
          ('Produc31',12,1.95)
```

28- INSERTING Hierarchical Rows

(i)

```
INSERT INTO orders (customer_id, order_date, status)
VALUES (1, '2019-01-02',1);
SELECT LAST_INSERT_ID()
```

(ii)

```
INSERT INTO orders (customer_id, order_date, status)
VALUES (1, '2019-01-02',1);
INSERT INTO order_items
VALUES
        (LAST_INSERT_ID(), 1, 1, 2.95),
        (LAST_INSERT_ID(), 2, 1, 3.95)
```

29- Creating a copy of Table

(i)

```
CREATE TABLE orders_archived AS
SELECT * FROM orders
```

(ii)

```
SELECT *
FROM orders
WHERE order_date < '2019-01-01'
```

Exercise:

Solution:

```
USE sql_invoicing;
CREATE TABLE invoices_archived AS

SELECT
   i.invoice_id,
   i.number,
   c.name AS client,
   i.invoice_total,
   i.payment_total,
```

```
    i.invoice_date,
    i.payment_date,
    i.due_date
FROM invoices i
JOIN clients c
        USING(client_id)
WHERE payment_date IS NOT NULL
```

30- Updating a Single Row

(i)

```
UPDATE invoices
SET payment_total = 10, payment_date = '2019-03-01'
WHERE invoice_id = 1
```

(ii)

```
UPDATE invoices
SET payment_total = DEFAULT, payment_date = NULL
WHERE invoice_id = 1
```

31- Updating Multiple Rows

```
UPDATE employees
SET salary = salary * 1.1
 WHERE department_id = 1
```

32- USING Subqueries in Update

(i)

```
SELECT client_id
FROM clients
```

```
WHERE name = 'Myworks'
```

(ii)

```
UPDATE invoices
SET
        payment_total = invoice_total * 0.5,
        payment_date = due_date
WHERE client_id =
                        (SELECT client_id
                        FROM clients
                        WHERE name = 'Myworks')
```

(iii)

```
UPDATE invoices
SET
        payment_total = invoice_total * 0.5,
        payment_date = due_date
WHERE client_id IN
                        (SELECT client_id
                        FROM clients
                        WHERE state IN ('CA','NY'))
```

33- Deleting Rows

```
DELETE FROM invoices
WHERE client_id =(
        SELECT *
        FROM clients
        WHERE name = 'Myworks'
```

)