SQL SUBQUERIES

INDEPENDENT SUBQUERY -

-- SCALAR SUBQUERY

```
-- Find the movies with highest rating by using subqueries
SELECT * FROM sql_cx_live.movies
WHERE score = (SELECT MAX(score) FROM  sql_cx_live.movies)


-- Find the movie with highest profit(vs order by)
USE sql_cx_live;
SELECT * FROM sql_cx_live.movies
WHERE (gross - budget) = (SELECT MAX(gross - budget) FROM movies)

-- or
SELECT * FROM movies
ORDER BY (gross - budge) DESC LIMIT 1
```

```
-- Find how many movies have a rating > the avg of all the movie ratings (Find the count of above average movies)

SELECT COUNT(*) FROM sql_cx_live.movies
WHERE score > (SELECT AVG(score) FROM sql_cx_live.movies)

-- Find the highest rated movie of 2000
SELECT * FROM sql_cx_live.movies
WHERE year = 2000
AND score = (SELECT MAX(score) FROM sql_cx_live.movies WHERE year = 2000)

-- Find the highest rated movie among all movies whose  number of votes are > the dataset avg votes

SELECT * FROM sql_cx_live.movies
WHERE  score = (SELECT MAX(score) FROM sql_cx_live.movies
                WHERE votes> (SELECT AVG(votes)
                              FROM sql_cx_live.movies))
CREATE DATABASE zomato
SELECT * FROM zomato.users;
```

```
SELECT * FROM zomato.orders;

SELECT * FROM zomato.order_details;

SELECT * FROM zomato.food;

SELECT * FROM zomato.menu;

SELECT * FROM zomato.delivery_partner;


-- ROW SUBQUERY

-- Find all users who never ordered

USE zomato;

SELECT * FROM users

WHERE user_id NOT IN (SELECT DISTINCT(user_id) FROM orders)


-- Find all the movies made by top 3 directors(in terms of total gross income)

-- SELECT * FROM sql_cx_live.movies

-- WHERE director IN (SELECT director

-- FROM sql_cx_live.movies

-- GROUP BY director

-- ORDER BY SUM(gross) DESC LIMIT 3)

-- GROUP BY star

-- HAVING avg_rating > 8.5 AND  total_vote  25000
```

```
-- -- SELECT * FROM sql_cx_live.movies
SELECT * FROM sql_cx_live.movies
WHERE star IN (SELECT star FROM sql_cx_live.movies
WHERE votes > 25000
GROUP BY star
HAVING AVG(score) > 8.5)
```

- Find the most profitable movie of each year

```
SELECT * FROM sql_cx_live.movies
WHERE (year, gross-budget) IN (SELECT year, MAX(gross - budget)
FROM sql_cx_live.movies
GROUP BY year
ORDER by MAX(gross - budget) DESC)
```

```
-- Find the highest rated movie of each genre votes cutoff of 25000
SELECT * FROM sql_cx_live.movies
WHERE (genre, score) IN (SELECT genre, MAX(score)
                         FROM sql_cx_live.movies
                         WHERE votes > 25000
                         GROUP BY genre)
AND votes>25000
-- because of limit we cant  use  in subquery,  have to use common table expression
SELECT * FROM sql_cx_live.movies
WHERE (star, director, gross) IN (SELECT * FROM top_duos)

-CORRELATED SUBQUERY
-- Inner query is depend on outside query for execution

-- Find all the movies that have a rating higher than the average rating of movies in the same genre.[Animation]
SELECT * FROM movies m1
WHERE score > (SELECT AVG(score)
FROM sql_cx_live.movies m2
WHERE m2.genre = m1.genre)
```

```
-- Find the favorite food of each customer.
WITH fav_food AS (SELECT t2.user_id, name, f_name, COUNT(*) AS
'frequency'
FROM zomato.users t1
JOIN zomato.orders t2 ON t1.user_id = t2.user_id
JOIN zomato.order_details t3 ON t2.order_id = t3.order_id
JOIN zomato.food t4 ON t3.f_id = t4.f_id
GROUP BY t2.user_id, t3.f_id)

SELECT * FROM fav_food
WHERE frequency = (SELECT MAX(frequency)
FROM fav_food f2
WHERE f2.user_id = f1.user_id)


--  SUBQUERY WITH SELECT
-- Get the percentage of votes for each movie compared to the total
number of votes.
SELECT name, (votes/(SELECT SUM(votes) FROM
sql_cx_live.movies))*100
FROM sql_cx_live.movies
```

```
-- Display all movie names ,genre, score and avg(score) of genre
-- here used correlated subquery
SELECT name, genre, score,
(SELECT AVG(score) FROM sql_cx_live.movies m2 WHERE m2.genre = m1.genre)
FROM sql_cx_live.movies m1


--  SUBQUERY WITH FROM
-- Display average rating of all the restaurants
SELECT r_name , avg_rating
FROM (SELECT r_id, AVG(restaurant_rating) AS 'avg_rating'
FROM  zomato.orders
GROUP BY r_id) t1 JOIN zomato.restaurants t2
ON t1.r_id = t2.r_id

-- Find genres having avg score > avg score of all the movies
SELECT genre, AVG(score)
FROM sql_cx_live.movies
GROUP BY genre
HAVING AVG(score) > (SELECT AVG(score) FROM sql_cx_live.movies)
```

-- SUBQUERY IN INSERT

-- Populate a already created loyal_customers table with records of only those customers who have ordered food more than 3 times.

```
USE zomato;

INSERT INTO loyal_users

(user_id, name)

SELECT t1.user_id, name, COUNT(*)

FROM orders t1

JOIN users t2 ON t1.user_id = t2.user_id

GROUP BY user_id

HAVING COUNT(*) > 3
```

SUBQUERY IN UPDATE

-- Populate the money col of loyal_cutomer table using the orders table. Provide a 10% app money to all customers based on their order value.

```
UPDATE zomato.loyal_users

SET money = (

SELECT SUM(amount)*0.1

FROM zomato.orders

WHERE orders.user_id = loyal_users.user_id)
```

```
-- SUBQUERY IN UPDATE

-- Delete all the customers record who have never ordered.


DELETE FROM zomato.users

WHERE user_id IN (SELECT user_id FROM zomato.users

WHERE user_id NOT IN (SELECT DISTINCT(user_id) FROM
zomato.orders))
```