Filtering rows

```
SELECT * FROM flipcart.orders t1
JOIN flipcart.users t2
ON t1.user_id = t2.user_id
WHERE t2.city = 'PUNE' And t2.name = 'Sonia'
```

What is Subquery

In SQL, a subquery is a query within another query. It is a SELECT statement that is nested inside another SELECT, INSERT, UPDATE or DELETE statement. The subquery is executed first and its result is then used as a parameter or condition for the query.

Note- The topic is slightly difficult and needs a lot of practice

Name	rating	genre	year	released	score	votes	director
The Shining	R	Drama	June 13, 1980 (US)	1980	8.4	927000	Stanley
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

```
SELECT MAX(score) FROM cx-lives.movies

SELECT * FROM cx-lives.movies
WHERE score = 9.3
```

↳ this query will fail in future if any movie added in data. bcz this query is no dynamic (Hard coded)

dynamic query ↴

```
SELECT * FROM cx-lives.movies
WHERE score = [( SELECT MAX (score) FROM cx-lives.movies)]
```

↳ outer query

↳ inner query

* first inner query run then outer query run on the output of inner query

Types of subqueries

Based on:

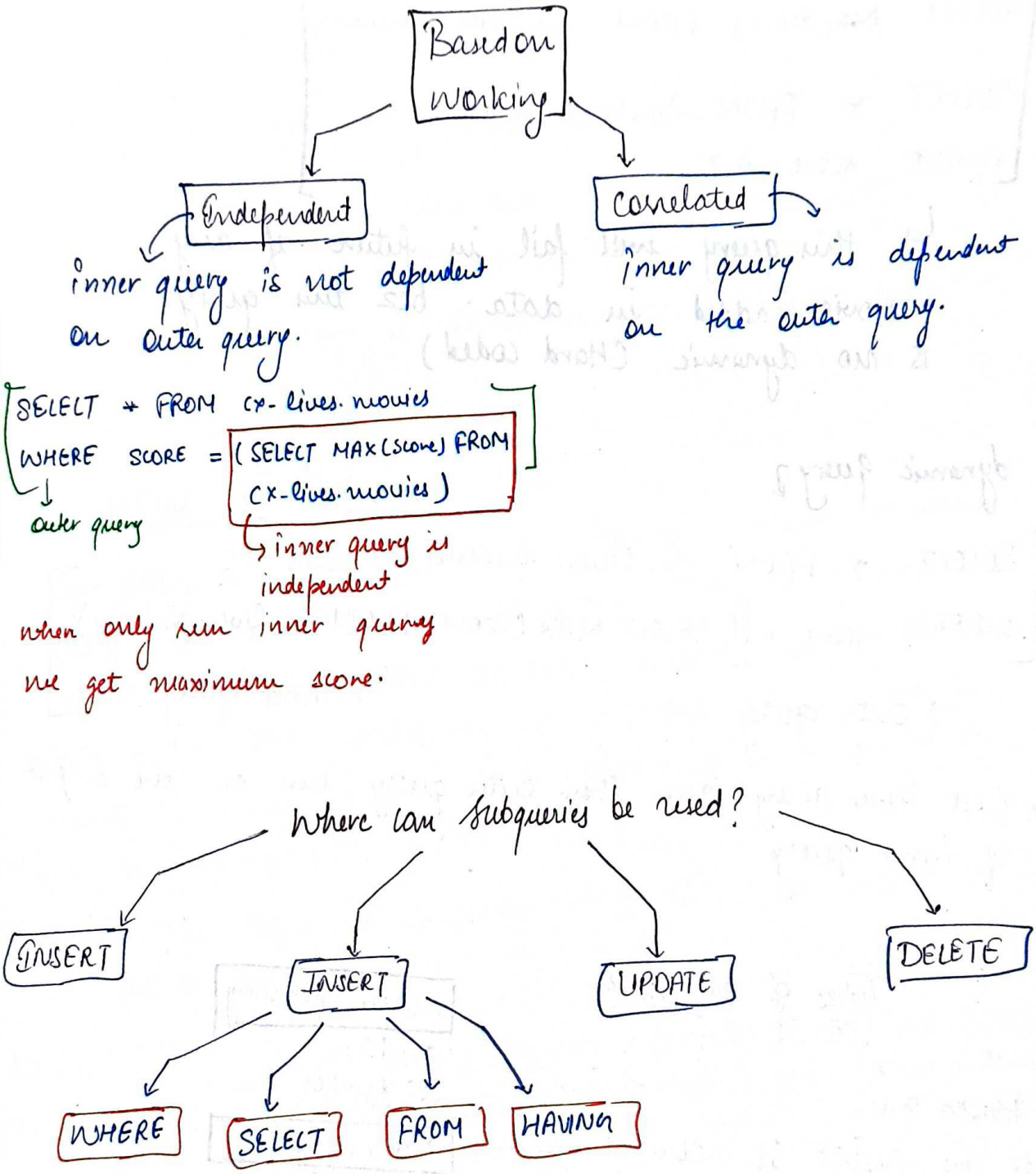
1. The result it returns
2. Based on working

- Scaler Subquery
  - → (9.3)
  - → horror
- Row subquery

genres
horror
action
romance

- Table subquery

genres	avg rating
horror	6.5
romance	7
action	6.8


Based on working
Independent
inner query is not dependent on outer query.
Correlated
inner query is dependent on the outer query.
SELECT * FROM cx-lives.movies
WHERE SCORE = (SELECT MAX(score) FROM cx-lives.movies)
outer query
inner query is independent
when only one inner query we get maximum score.
Where can subqueries be used?
INSERT
INSERT
UPDATE
DELETE
WHERE
SELECT
FROM
HAVING

Independent Subquery – Scalar Subquery

1. Find the movie with highest profit (Vs order by)

```
SELECT * FROM subquery.movies
WHERE (gross-budget) = (SELECT MAX(gross-budget) FROM
                        subquery.movies)
```

↳ This query is faster when ORDER method ignore indexing method.

We can also write

```
SELECT * FROM movies
ORDER BY (gross-budget) DESC LIMIT 1;
```

→ This one faster than upper query because ORDER use indexing. Faster in large dataset.

2. Find how many movies have a rating greater than the avg of all the movies ratings (Find the count of above average movies)

```
USE subquery;
```

→ database name

→ Once you run this code then no need to write every time.

```
SELECT COUNT(*) FROM movies
WHERE score > (SELECT AVG(score) FROM movies)
```

3. Find the highest rated movie of 2000

```
USE subquery;
SELECT * FROM movies
WHERE year = 2000 AND score = (SELECT MAX(score)
                               FROM movies
                               WHERE year = 2000)
```

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

```
SELECT * FROM movies
WHERE score = (SELECT MAX(score) FROM movies
               WHERE votes > (SELECT AVG(votes) FROM
                              movies));
```

Independent Subquery – Row Subquery (One col Multi Row)

1) Find the all users who never ordered

```
USE subquery;
SELECT * FROM users
WHERE user_id NOT IN (SELECT DISTINCT user_id FROM orders);
```

↳ Output one col Multi row

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

```
USE subquery;
SELECT * FROM movies
WHERE director IN (SELECT director, SUM(gross) FROM movies
                   GROUP BY director ORDER BY (gross) DESC LIMIT 3);
```

↳ this query work on other but not working in My SQL benchwork.

Alternate 2

```
WITH top_directors AS ( SELECT director
                        FROM movies
                        GROUP BY director
                        ORDER BY SUM(gross) DESC
```

LIMIT 3))

```
SELECT * FROM movies
WHERE director IN (SELECT * FROM top-directors)
```

Why we use subqueries?

→ Bcz SQL is Non-procedural progroming language. We cannot create flows and variable in SQL. like other language (Python, Java).

Independent Subquery - Table Subquery (Multi col Multi Row)

1. Find the most profitable movie of each year.

USE subquery;

```
SELECT * FROM movies
WHERE (year, gross-budget) IN (SELECT year, MAX(gross-budget)
                               FROM movies
                               GROUP BY year)
```

IN use when you compare or check bern two tables and row

2. Find the highest rated movie of each genre votes cutoff of 25000.

```
SELECT * FROM movies
WHERE (genre, score) IN (SELECT genre, MAX(score) FROM movies
                         WHERE votes > 25000
                         GROUP BY genre) AND votes > 25000
```

Correlated Subquery

1. Find all the movies that have a rating higher than the average rating of movies in the same genre.

```
USE subquery;
SELECT * FROM movies m1
WHERE score > [ ( SELECT AVG(score) FROM movies m2
                  WHERE m2.genre = m1.genre) ]
```

↳ finding same genre of avg score.

2. Find the favourite food of each customer.

```
WITH fav_food AS (
    SELECT name, f_name, COUNT(*) FROM users t1
    JOIN orders t2 ON t1.user_id = t2.user_id
    JOIN order_details t3 ON t2.order_id =
                             t3.order_id
    JOIN food t4 ON t3.f_id = t4.f_id
    GROUP BY t2.user_id, t3.f_id
)
SELECT * FROM fav_food f1
WHERE frequency = (SELECT MAX(frequency)
                   FROM fav_food f2
                   WHERE f2.users_id = f1.user_id
```

Usage with SELECT

1. Get the percentage of votes for each movie compared to the total number of votes

USE subquery;

```
SELECT Name,
(Votes / (SELECT SUM(Votes) FROM movies)) * 100
FROM movies
```

2.

```
SELECT name, genre, score,
(SELECT AVG(Score) FROM movies m2 WHERE m2.genre = m1.genre
FROM movies m2
```

Usage with FROM

1. Display average rating of all restaurants.

```
SELECT r_name, avg_rating
FROM (SELECT r_id, AVG (restaurant_rating) AS 'avg_rating'
      FROM Orders
      GROUP BY r_id) t1
JOIN restaurant t2
ON t1.r_id = t2.r_id
```

Usage with Having

1. Find genres having avg score > of all the movies avg score.

```
SELECT genre, AVG(score)
FROM movies
GROUP BY genre
HAVING AVG(score) > (SELECT AVG(score) FROM movies)
```

Subquery In INSERT

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

```
SELECT * FROM zomato.loyal_users;
USE zomato;

INSERT INTO loyal_users
(user_id, name)
SELECT t1.user_id, name
FROM order 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_customer table using the order table. Provide a 10% app money to all customer based on their order value.

```
UPDATE loyal_users
SET money = (SELECT SUM(amount) *0.1
             FROM orders
             WHERE orders.user_id = loyal_user.user_id
```

Subquery DELETE

→ Delete all the customers record who have never ordered

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

```
DELETE FROM users
WHERE user_id IN (SELECT user_id FROM users
                  WHERE userid NOTIN (SELECT
                  DISTINCT (user_id) FROM orders))
```