Database Systems Lab

Lab # 07

Subqueries

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2005

Semester Fall 2021

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

Contents

First let's outline the basic characteristics of a subquery.

- A subquery is a query (SELECT statement) inside a query
- A subquery is normally expressed inside parentheses
- The first query in the SQL statement is known as the outer query
- The query inside the SQL statement is known as the inner query
- The inner query is executed first
- The output of an inner query is used as the input for the outer query
- The entire SQL statement is sometimes referred to as a nested query

A subquery can return one value or multiple values. To be precise, the subquery can return:

- *One single value (one column and one row)*. This subquery is used anywhere a single value is expected, as in the right side of a comparison expression. Obviously, when you assign a value to an attribute, that value is a single value, not a list of values. Therefore, the subquery must return only one value (One column, one row). If the query returns multiple values, the DBMS will generate an error.

- *A list of values (one column and multiple rows).* This type of subquery is used anywhere a list of values is expected, such as when using the IN clause. This type of subquery is used frequently in combination with the IN operator in a WHERE conditional expression.

- *A virtual table (multicolumn, multirow set of values).* This type of subquery can be used anywhere a table is expected, such as when using the FROM clause.

It's important to note that a subquery can return no values at all; it is a NULL. In such cases, the output of the outer query may result in an error or a null empty set depending where the subquery is used (in a comparison, an expression, or a table set).

In the following sections, you will learn how to write subqueries within the SELECT statement to retrieve data from the database.

1. SELECT Subqueries

The most common type of subquery uses an inner SELECT subquery on the right side of a WHERE comparison expression. For example,

Task 1 To find the prices of all tickets with a price greater than or equal to the average ticket price, you write the following query:

```
SELECT      TICKET_NO, TICKET_TYPE, TICKET_PRICE
FROM TICKET
WHERE       TICKET_PRICE >= (SELECT AVG(TICKET_PRICE) FROM TICKET);
```

The output of the query is shown in Figure 69.

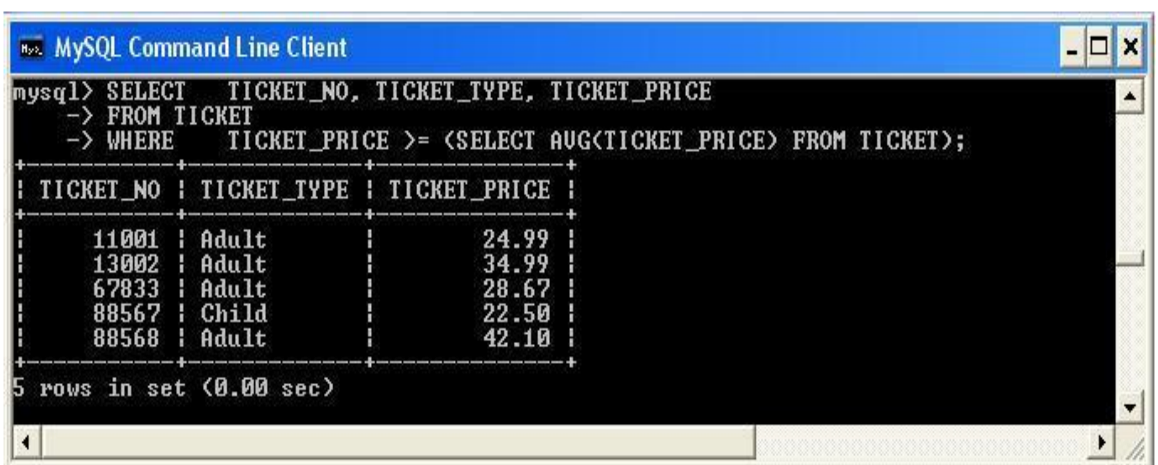


Figure 69 Example of SELECT Subquery

Note that this type of query, when used in a >, <, =, >=, or <= conditional expression, requires a subquery that returns only one single value (one column, one row). The value generated by the subquery must be of a “comparable” data type; if the attribute to the left of the comparison symbol is a character type, the subquery must return a character string. Also, if the query returns more than a single value, the DBMS will generate an error.

2. IN Subqueries

The following query displays all employees who work in a Theme Park that has the word ‘Fairy’ in its name. As there are a number of different Theme Parks that match this criteria you need to compare the PARK_CODE not to one park code (single value), but to a list of park codes. When you want to compare a single attribute to a list of values, you use the IN operator. When the PARK_CODE values are not known beforehand but they can be derived using a query, you must use an IN subquery. The following example lists all employees who have worked in such a Theme Park.

```
SELECT      DISTINCT EMP_NUM, EMP_LNAME, EMP_FNAME, PARK_NAME
FROM        EMPLOYEE NATURAL JOIN HOURS NATURAL JOIN
ATTRACTION NATURAL JOIN THEMEPARK
WHERE       PARK_CODE IN (SELECT THEMEPARK.PARK_CODE FROM
THEMEPARK WHERE       PARK_NAME LIKE ‘%Fairy%’);
```

The result of that query is shown in Figure 71.

Figure 71 Employees who work in a Theme Park LIKE 'Fairy'.

Task 2 Enter and execute the above query and compare your output with that shown in Figure 71.

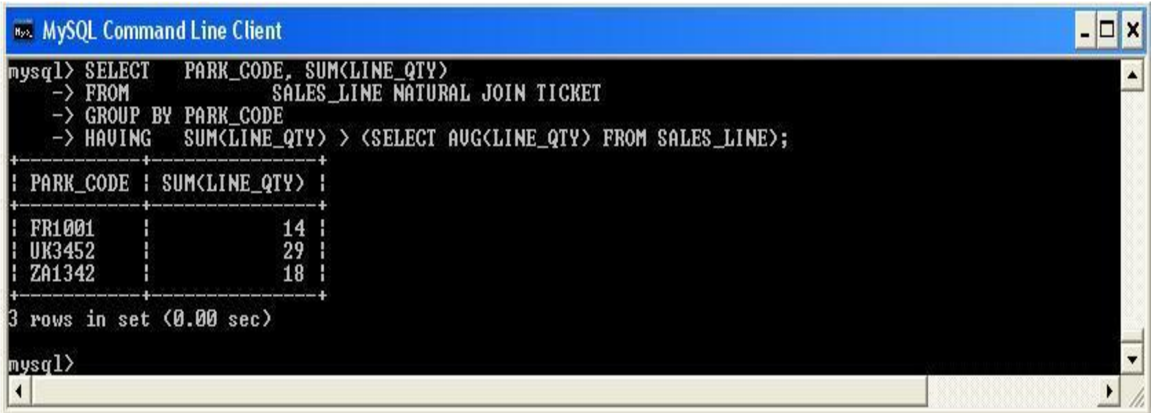
3. HAVING Subqueries

A subquery can also be used with a HAVING clause. Remember that the HAVING clause is used to restrict the output of a GROUP BY query by applying a conditional criteria to the grouped rows. For example, to list all PARK_CODEs where the total quantity of tickets sold is greater than the average quantity sold, you would write the following query:

Task 3

```
SELECT      PARK_CODE, SUM(LINE_QTY)
FROM        SALES_LINE NATURAL JOIN TICKET
GROUP BY PARK_CODE
HAVING  SUM(LINE_QTY) > (SELECT AVG(LINE_QTY) FROM SALES_LINE);
```

The result of that query is shown in Figure 72.



```
MySQL Command Line Client
mysql> SELECT   PARK_CODE, SUM(LINE_QTY)
    -> FROM           SALES_LINE NATURAL JOIN TICKET
    -> GROUP BY PARK_CODE
    -> HAVING   SUM(LINE_QTY) > (SELECT AVG(LINE_QTY) FROM SALES_LINE);
+-----------+---------------+
| PARK_CODE | SUM(LINE_QTY) |
+-----------+---------------+
| FR1001    |            14 |
| UK3452    |            29 |
| ZA1342    |            18 |
+-----------+---------------+
3 rows in set (0.00 sec)

mysql>
```

Figure 72 PARK_CODES where tickets are selling above average.

Using Group by on Multiple Columns

Group by x means **put all those with the same value for X in the one group)**.

`Group By x, y` *means* **put all those with the same values for both X and Y in the one group**.

To illustrate using an example, let's say we have the following table, to do with who is attending what subject at a university:

```
Table: Subject_Selection

+---------+----------+----------+
| Subject | Semester | Attendee |
+---------+----------+----------+
| ITB001  |        1 | John     |
| ITB001  |        1 | Bob      |
| ITB001  |        1 | Mickey   |
| ITB001  |        2 | Jenny    |
| ITB001  |        2 | James    |
| MKB114  |        1 | John     |
| MKB114  |        1 | Erica    |
+---------+----------+----------+
```

When you use a `group by` on the subject column only; say:

```
select Subject, Count(*)
from Subject_Selection
group by Subject
```

You will get something like:

```
+---------+-------+
| Subject | Count |
+---------+-------+
| ITB001  |     5 |
| MKB114  |     2 |
+---------+-------+
```

Because there are 5 entries for ITB001, and 2 for MKB114

If we were to `group by` two columns:

```
select Subject, Semester, Count(*)
from Subject_Selection
group by Subject, Semester
```

we would get this:

```
+---------+----------+-------+
| Subject | Semester | Count |
+---------+----------+-------+
| ITB001  |        1 |     3 |
| ITB001  |        2 |     2 |
| MKB114  |        1 |     2 |
+---------+----------+-------+
```

This is because, when we group by two columns, it is saying **"Group them so that all of those with the same Subject and Semester are in the same group, and then calculate all the aggregate functions** (Count, Sum, Average, etc.) **for each of those groups"**. In this example, this is demonstrated by the fact that, when we count them, there are **three** people doing ITB001 in semester 1, and **two** doing it in semester 2. Both of the people doing MKB114 are in semester 1, so there is no row for semester 2 (no data fits into the group "MKB114, Semester 2")

4. Multirow Subquery operator ALL.

So far, you have learned that you must use an IN subquery when you need to compare a value to a list of values. But the IN subquery uses an equality operator; that is, it selects only those rows that match (are equal to) at least one of the values in the list. What happens if you need to do an inequality comparison (> or <) of one value to a list of values? For example, to find the ticket_numbers and corresponding park_codes of the tickets that are priced higher than the highest-priced 'Child' ticket you could write the following query.

Task 4

```
SELECT TICKET_NO, PARK_CODE, TICKET_PRICE
FROM TICKET
WHERE TICKET_PRICE > ALL (SELECT TICKET_PRICE FROM TICKET
   WHERE TICKET_TYPE = 'CHILD');
```

The output of that query is shown in Figure 74.

Figure 74 Example of ALL.

This query is a typical example of a nested query. The use of the ALL operator allows you to compare a single value (TICKET_PRICE) with a list of values returned by the nested query, using a comparison operator other than equals. For a row to appear in the result set, it has to meet the criterion TICKET_PRICE > ALL of the individual values returned by the nested query.

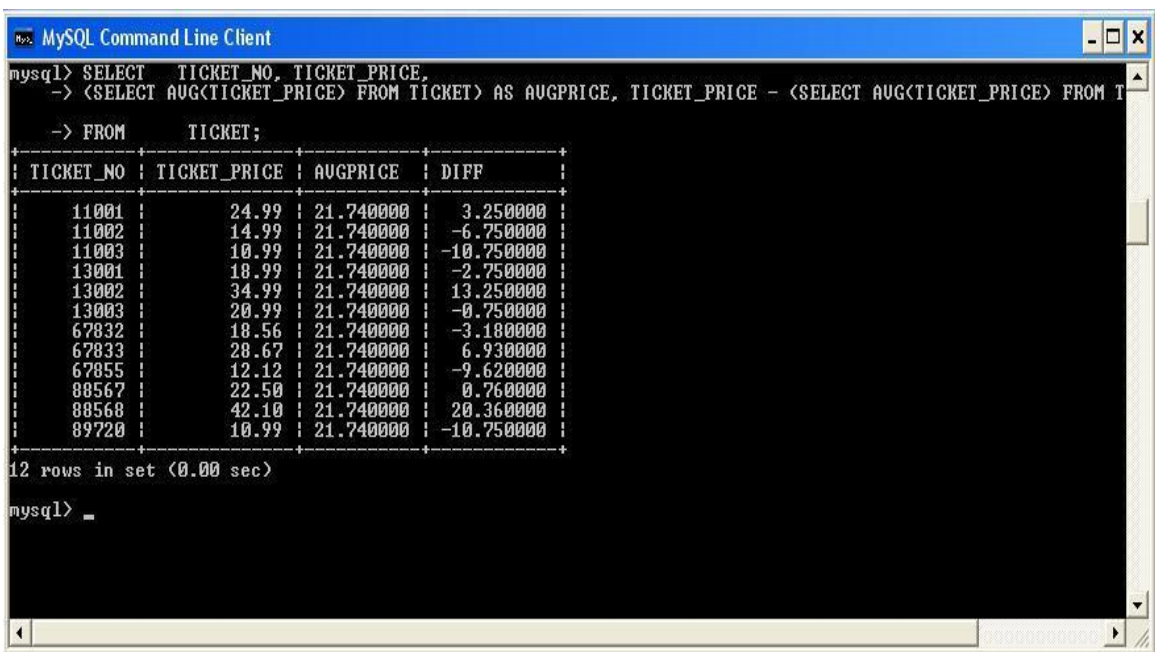
5. Attribute list Subqueries

The SELECT statement uses the attribute list to indicate what columns to project in the resulting set. Those columns can be attributes of base tables or computed attributes or the result of an aggregate function. The attribute list can also include a subquery expression, also known as an inline subquery. A subquery in the attribute list must return one single value; otherwise, an error code is raised. For example, a simple inline query can be used to list the difference between each tickets' price and the average ticket price:

Task 5

```
SELECT      TICKET_NO, TICKET_PRICE,
        (SELECT AVG(TICKET_PRICE) FROM TICKET) AS AVGPRICE,
        TICKET_PRICE - (SELECT AVG(TICKET_PRICE) FROM TICKET) AS DIFF
FROM TICKET;
```

The output for this query is shown in Figure 75.




```
mysql> SELECT    TICKET_NO, TICKET_PRICE,
    -> (SELECT AVG(TICKET_PRICE) FROM TICKET) AS AVGPRICE, TICKET_PRICE - (SELECT AVG(TICKET_PRICE) FROM T

    -> FROM      TICKET;
```

TICKET_NO	TICKET_PRICE	AVGPRICE	DIFF
11001	24.99	21.740000	3.250000
11002	14.99	21.740000	-6.750000
11003	10.99	21.740000	-10.750000
13001	18.99	21.740000	-2.750000
13002	34.99	21.740000	13.250000
13003	20.99	21.740000	-0.750000
67832	18.56	21.740000	-3.180000
67833	28.67	21.740000	6.930000
67855	12.12	21.740000	-9.620000
88567	22.50	21.740000	0.760000
88568	42.10	21.740000	20.360000
89720	10.99	21.740000	-10.750000

```
12 rows in set (0.00 sec)

mysql> _
```

Figure 75 Displaying the difference in ticket prices.

This inline query output returns one single value (the average ticket's price) and that the value is the same in every row. Note also that the query used the full expression instead of the column aliases when computing the difference. In fact, if you try to use the alias in the difference expression, you will get an error message. The column alias cannot be used in computations in the attribute list when the alias is defined in the same attribute list.

6. Correlated Subqueries

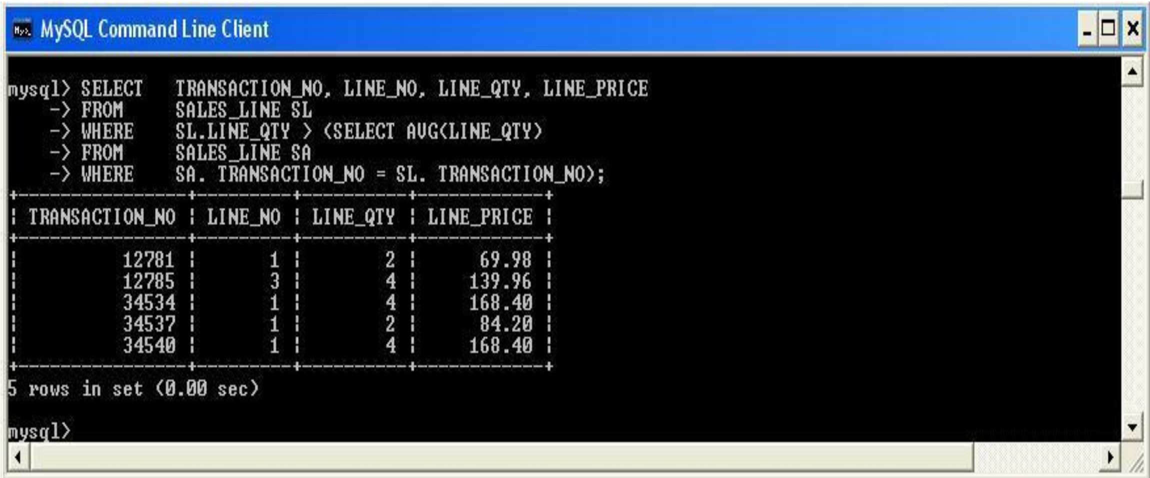
A correlated subquery is a subquery that executes once for each row in the outer query. The relational DBMS uses the same sequence to produce correlated subquery results:

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of the subqueries you have seen so far. The query is called a *correlated* subquery because the inner query is *related* to the outer query because the inner query references a column of the outer subquery. For example, suppose you want to know all the ticket sales in which the quantity sold value is greater than the average quantity sold value for *that* ticket (as opposed to the average for *all tickets*). The following correlated query completes the preceding two-step process:

Task 6

```
SELECT      TRANSACTION_NO, LINE_NO, LINE_QTY, LINE_PRICE
FROM        SALES_LINE SL
WHERE       SL.LINE_QTY > (SELECT AVG(LINE_QTY)
FROM        SALES_LINE SA
WHERE       SA. TRANSACTION_NO = SL. TRANSACTION_NO);
```



```
mysql> SELECT   TRANSACTION_NO, LINE_NO, LINE_QTY, LINE_PRICE
    -> FROM     SALES_LINE SL
    -> WHERE    SL.LINE_QTY > (SELECT AVG(LINE_QTY)
    -> FROM     SALES_LINE SA
    -> WHERE    SA. TRANSACTION_NO = SL. TRANSACTION_NO);
+----------------+---------+----------+------------+
| TRANSACTION_NO | LINE_NO | LINE_QTY | LINE_PRICE |
+----------------+---------+----------+------------+
|          12781 |       1 |        2 |      69.98 |
|          12785 |       3 |        4 |     139.96 |
|          34534 |       1 |        4 |     168.40 |
|          34537 |       1 |        2 |      84.20 |
|          34540 |       1 |        4 |     168.40 |
+----------------+---------+----------+------------+
5 rows in set (0.00 sec)

mysql>
```

Figure 76 Example of a correlated subquery

As you examine the output shown in figure 76, note that the SALES_LINE table is used more than once; so you must use table aliases.

7. SQL Query order of execution

The SQL order of execution defines the order in which the clauses of a query are evaluated. Some of the most common query challenges people run into could be easily avoided with a clearer understanding of the SQL order of execution, sometimes called the SQL order of operations. Understanding SQL query order can help you diagnose why a query won't run, and even more frequently will help you optimize your queries to run faster.

This order is:

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

1. FROM and JOINs

The FROM clause, and subsequent JOINs are first executed to determine the total working set of data that is being queried. This includes subqueries in this clause, and can cause temporary tables to be created under the hood containing all the columns and rows of the tables being joined.

2. WHERE

Once we have the total working set of data, the first-pass WHERE constraints are applied to the individual rows, and rows that do not satisfy the constraint are discarded. Each of the constraints can only access columns directly from the tables requested in the FROM clause. Aliases in the SELECT part of the query are not accessible in most databases since they may include expressions dependent on parts of the query that have not yet executed.

3. GROUP BY

The remaining rows after the WHERE constraints are applied are then grouped based on common values in the column specified in the GROUP BY clause. As a result of the grouping, there will only be as many rows as there are unique values in that column. Implicitly, this means that you should only need to use this when you have aggregate functions in your query.

4. HAVING

If the query has a GROUP BY clause, then the constraints in the HAVING clause are then applied to the grouped rows, discard the grouped rows that don't satisfy the constraint. Like the WHERE clause, aliases are also not accessible from this step in most databases.

5. SELECT

Any expressions in the SELECT part of the query are finally computed.

6. ORDER BY

If an order is specified by the ORDER BY clause, the rows are then sorted by the specified data in either ascending or descending order. Since all the expressions in the SELECT part of the query have been computed, you can reference aliases in this clause.

7. LIMIT / OFFSET

Finally, the rows that fall outside the range specified by the LIMIT and OFFSET are discarded, leaving the final set of rows to be returned from the query.

Exercises

E 5.1 Write a query that displays the first name, last name of all employees who earn more than the average hourly rate. Do not display duplicate rows. Your output should match that shown in Figure E-5.1

EMP_FNAME	EMP_LNAME
Enrica	Denver
Mirrelle	Namowa

2 rows in set (0.001 sec)

Figure 70 Output for E-5.1

E.5.2 Write a query to display an employee's first name, last name and date worked which lists the difference between the number of hours an employee has worked on an attraction and the average hours worked on that attraction. Label this column 'DIFFERENCE' and the average hours column 'AVERAGE'.

EMP_FNAME	EMP_LNAME	DATE_WORKED	AVERAGE	DIFFERENCE
Emma	Calderdale	2007-05-18	4.9091	1.0909
Emma	Calderdale	2007-05-20	4.9091	1.0909
Marshel	Ricardo	2007-05-18	4.9091	1.0909
Arif	Arshad	2007-05-23	4.9091	-1.9091
Arif	Arshad	2007-05-21	4.9091	1.0909
Arif	Arshad	2007-05-22	4.9091	-1.9091
Enrica	Denver	2007-05-21	4.9091	1.0909
Enrica	Denver	2007-05-22	4.9091	1.0909
Mirrelle	Namowa	2007-05-18	4.9091	-1.9091
Mirrelle	Namowa	2007-05-18	4.9091	-1.9091
Mirrelle	Namowa	2007-05-19	4.9091	1.0909

E 5.3 Write a query to find the attract capacity, with a attract capacity less than or equal to the average attract capacity.

References:

https://www.sisense.com/blog/sql-query-order-of-operations/

https://sqlbolt.com/lesson/select_queries_order_of_execution