Database Management Systems:
Fundamentals and Introduction to SQL

MySQL Queries

UNION

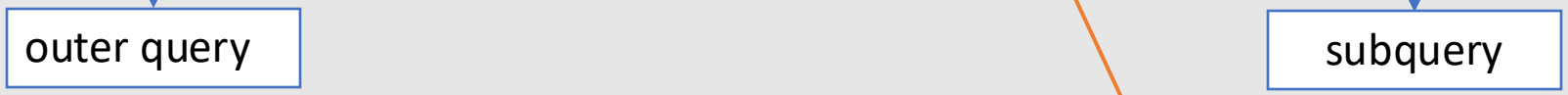
- combine rows retrieved by two or more SELECT statements into one result set
- **UNION [DISTINCT] –** REMOVE DUPLICATES
- **UNION ALL** – keep all records

Subquery

Subquery

- one query nested within another
- perform a comparison between values selected by the inner query against values selected by the outer query

- `SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);`



Note: Since the WHERE clause is testing for equality, the subquery must return a single row with a single column.

Subquery

- allow queries that are structured so that it is possible to isolate each part of a statement
- provide alternative ways to perform operations that would otherwise require complex joins and unions
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Subquery

- min() and max() cannot be used in WHERE clause
- return the record with max MILES in driver_log

- possible solution:
- 1. User-defined variable
  - SET @max = (SELECT max(…) FROM …);
  - SELECT …. WHERE col = @max;
- 2. JOIN

  create tmp table containing min or max;

  Join with original table
- 3. **Using subquery**

  WHERE col = (SELECT MAX() ……)

Subquery

- comparison with Join (*find which artist is missing in collection)*

```
mysql> SELECT artist.* FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+------+-------+
| a_id | name  |
+------+-------+
|    2 | Monet |
+------+-------+
```

```
mysql> SELECT * FROM artist
    -> WHERE a_id NOT IN (SELECT a_id FROM painting);

+------+-------+
| a_id | name  |
+------+-------+
|    2 | Monet |
+------+-------+
```

Subquery VS Join

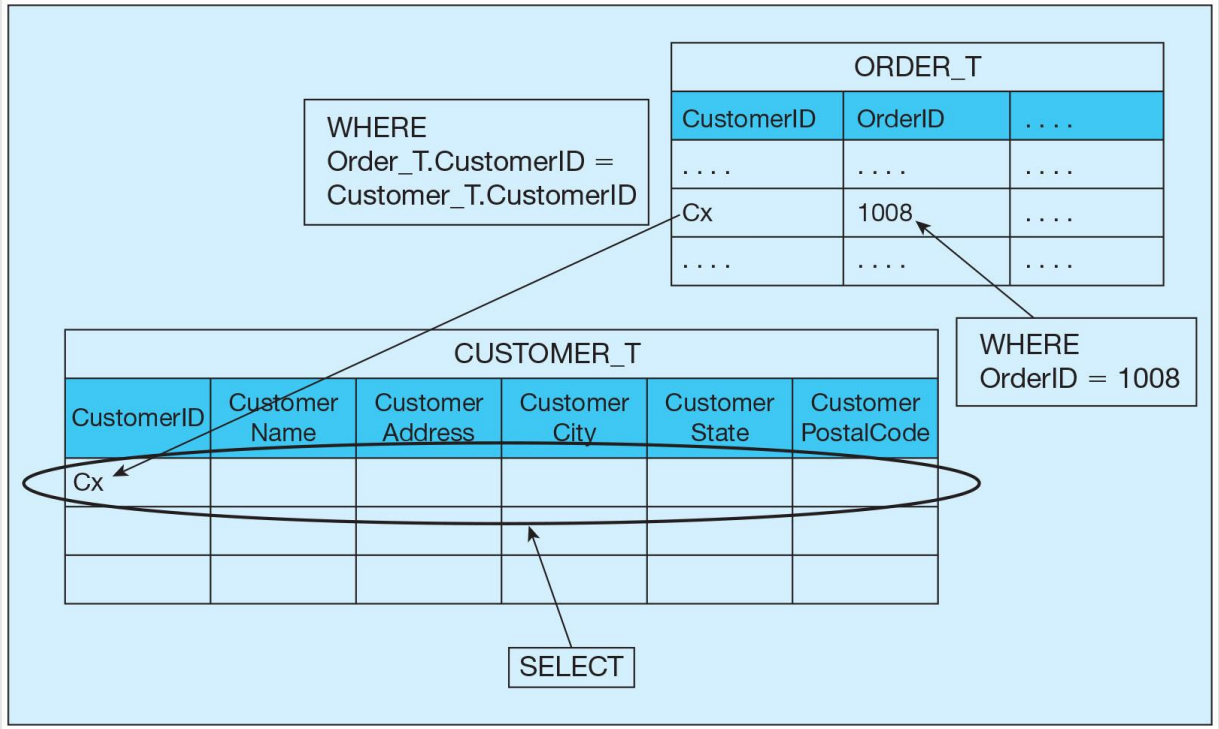
Q: What are the name and address of the customer who placed order number 1008?

```
SELECT CustomerName, CustomerAddress, CustomerCity,
   CustomerState, CustomerPostalCode
FROM Customer_T, Order_T
WHERE Customer_T.CustomerID = Order_T. CustomerID
   AND OrderID = 1008;
```

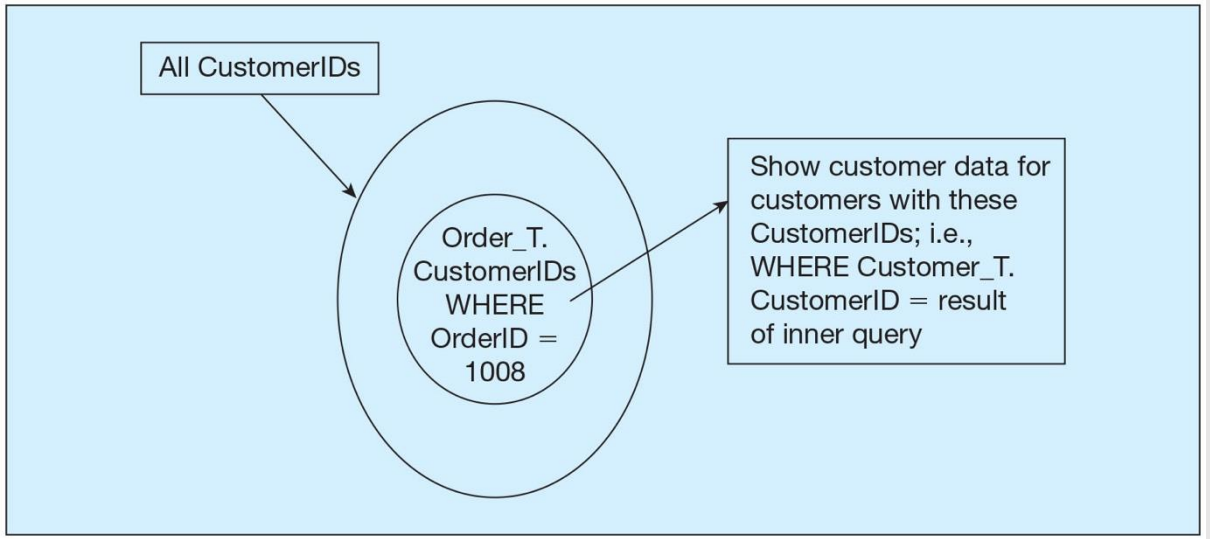
```
SELECT CustomerName, CustomerAddress, CustomerCity, CustomerState,
CustomerPostalCode
   FROM Customer_T
      WHERE Customer_T.CustomerID =
         (SELECT Order_T.CustomerID
            FROM Order_T
               WHERE OrderID = 1008);
```

Subquery VS Join: Graphical Depiction

a) Join query approach



b) Subquery approach



Subquery with Any, Some

- ANY keyword, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns”

- `SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);`

- **IN** is an alias for "**= ANY**" with subquery
- `SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);`
- `SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);`

- SOME is an alias for ANY
- `SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);`
- `SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);`

Subquery with All

- return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns

- `SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);`

Correlated Subqueries

- A correlated subquery is a subquery that contains a reference to a table that appears in the **outer query**
- SELECT * FROM t1

```
WHERE column1 = ANY (SELECT column1 FROM t2
                     WHERE t2.column2 = t1.column2);
```

- Inside to outside
- SELECT column1 FROM t1 AS x

```
WHERE column1 = (SELECT column1 FROM t2 AS x
                 WHERE x.column1 = (SELECT column1 FROM t3
                                    WHERE x.column2 = t3.column1));
```

Which table?

Correlated Subqueries

Q: Find all the employees who earn more than the average salary in their department.

```
SELECT name, salary, department_id
FROM employees outer
WHERE salary > (
      SELECT AVG(salary)
      FROM employees
      WHERE department_id = outer.department_id
      GROUP BY department_id);
```

Subqueries with EXISTS or NOT EXISTS

- If a subquery returns any rows at all,

  EXISTS subquery is TRUE, and NOT EXISTS subquery is FALSE.

- a [NOT] EXISTS subquery almost always **contains correlations**

- vs. IN
  - IN: compare the record with each record in the subquery list
  - EXISTS: do the comparison once by matching the record using the correlation

Subquery in SELECT and FROM

- *Q: return total miles per driver AND percentage of each driver's mileage*

```
SELECT @total := SUM(miles) AS 'total miles'
FROM driver_log;

SELECT name,
SUM(miles) AS 'miles/driver',
(SUM(miles)*100)/@total AS 'percent of total miles'
FROM driver_log
GROUP BY name;
```

Subquery in SELECT

```
SELECT title,
        (SELECT name FROM artist
                WHERE artist.a_id=painting.a_id) AS name
FROM painting;
#return the painting and artist name
```

```
SELECT title,
(SELECT name FROM artist WHERE artist.a_id=painting.a_id) AS name,
(SELECT name FROM states WHERE states.abbrev = painting.state) AS state
FROM painting;
#return the painting title, artist name and artist state
```

Subquery in FROM

- any table referred to in the FROM clause **must have a name**, even a "derived" table produced from a subquery

```
mysql> SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4;
+-------+------------+
| name  | birth      |
+-------+------------+
| Ralph | 1973-11-02 |
| Sybil | 1970-04-13 |
| Nancy | 1969-09-30 |
| Aaron | 1968-09-17 |
+-------+------------+
```

```
mysql> SELECT * FROM
    -> (SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4) AS t
    -> ORDER BY birth;
+-------+------------+
| name  | birth      |
+-------+------------+
| Aaron | 1968-09-17 |
| Nancy | 1969-09-30 |
| Sybil | 1970-04-13 |
| Ralph | 1973-11-02 |
+-------+------------+
```

Query Efficiency Considerations

- Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store it as a view

```
CREATE VIEW view_name
    AS
    SELECT
```

Advantages of Dynamic Views

- Assist with data security
- Enhance programming productivity
- Contain most current base table data
- Use little storage space
- Provide customized view for user

Guidelines for Better Query Design

- Understand how indexes are used in query processing
- Write simple queries
- Break complex queries into multiple simple parts
- Don’t combine a query with itself (if possible, avoid self-joins)
- Create temporary tables for groups of queries
- Retrieve only the data you need

UNION– In-class Exercise

**mail.sql*

find four users ('user@host') who sent the highest number of emails and four users who received the highest number of emails, then sort the result of the union by the user name

Subquery – in-class exercise

Use **subquery** to solve the following queries *artist.sql

- *Which paintings did Van Gogh paint?*
- *Who painted the Mona Lisa?*
- *For which artists did you purchase paintings in Kentucky or Indiana?*

**mail.sql*

Q: how many emails were sent by a particular user (a user is defined as a combination of name and host: name@host) and how many emails they received, sorting the result by total mails they sent in descending order