Chapter 9: Subqueries

Introduction to Subqueries

Definition

A subquery is a SELECT statement embedded within another SQL statement (SELECT, INSERT, UPDATE, DELETE, MERGE, CREATE TABLE, CREATE VIEW).

Key Characteristics

- **Syntactically autonomous**: Can usually run independently
- **Correlated subqueries**: Depend on data from the outer (parent) query
- **May return**:
  - Single value (scalar)
  - Single column
  - Multiple columns/rows
- **Used in clauses like**: WHERE, FROM, SELECT, HAVING, etc.

Query Hierarchy

- **Top-level query**: The main/outermost query
- **Parent query**: Any query that contains a subquery
- **Nested subqueries**: Subqueries can be nested within other subqueries

Valid Subquery Components

- Data from one or more tables/views
- Scalar/aggregate functions
- GROUP BY, HAVING, joins, etc.

Important Note

No need for key relationships - subqueries often reference different tables unrelated to the parent query.

Use Cases for Subqueries

1. **Complex multistage queries**
   Use subquery results to drive subsequent filtering or calculations

2. **Creating populated tables**
   Use in CREATE TABLE AS SELECT to build and populate a table
3. **Manipulating large datasets**
   Use in INSERT/UPDATE to move or transform large amounts of data
4. **Creating named views**
   Define view objects with subqueries
5. **Defining dynamic views (inline views)**
   Replace a table in the FROM clause using a subquery
6. **Scalar subqueries**
   Return a single value for use in expressions (e.g., SELECT emp_name, (SELECT MAX(salary) FROM employees) AS max_sal)

Purpose of Subqueries

- Used to solve complex data problems
- Ideal for comparisons (above average, top/bottom values)
- Common in combination with aggregate functions

Example: SALES_DATA Table

Table Creation

```sql
CREATE TABLE SALES_DATA (
    ID NUMBER,
    REP VARCHAR2(10),
    YEAR NUMBER(4),
    TOTAL_SALES NUMBER
);
```

Sample Data

```sql
INSERT INTO SALES_DATA VALUES (1, 'Joe', 2018, 249);
INSERT INTO SALES_DATA VALUES (2, 'Joe', 2017, 178);
INSERT INTO SALES_DATA VALUES (3, 'Joe', 2016, 483);
-- Similar inserts for Ann, Moe, Lyn
COMMIT;
```

Subquery Example Use Case

Task: Find 2018 reps whose sales > average 2017 sales.

Step-by-step approach:

1. Find average 2017 sales:

```sql
SELECT AVG(TOTAL_SALES) AVG_SALES
FROM SALES_DATA
WHERE YEAR = 2017;
-- Result: 478.75
```

2. Manual filtering:

```sql
SELECT REP
FROM SALES_DATA
WHERE YEAR = 2018
AND TOTAL_SALES > 478.75;
```

3. Using a subquery:

```sql
SELECT REP
FROM SALES_DATA
WHERE YEAR = 2018
AND TOTAL_SALES > (
    SELECT AVG(TOTAL_SALES)
    FROM SALES_DATA
    WHERE YEAR = 2017
);
```

Types of Subqueries

1. Single-row Subqueries

- Returns exactly one row
- Can include multiple columns
- Use =, <, >, etc.

- If more than one row is returned → **ORA-01427 error**
- Must ensure single row return using:
  - Primary key filters
  - Aggregate functions (e.g., MIN, MAX)
  - ROWNUM (e.g., WHERE ROWNUM < 2)

Example:

```sql
SELECT employee_id
FROM employees
WHERE ship_id = (
    SELECT ship_id
    FROM employees
    WHERE last_name = 'Smith' AND first_name = 'Al'
);
```

2. Multiple-row Subqueries

- Returns zero or more rows
- Must use operators like IN, ANY, ALL, SOME, NOT IN
- Using = with multiple rows causes errors

Example using IN:

```sql
SELECT ship_id, last_name
FROM employees
WHERE ship_id IN (
    SELECT ship_id
    FROM employees
    WHERE last_name = 'Smith'
);
```

Key operators:

Operator	Use
IN	Match any value in subquery
NOT IN	Match none (fails with NULLs!)
ANY	True if any subquery value matches condition
ALL	True if all values satisfy the condition
SOME	Same as ANY

Note: Greater-than/less-than operators (>, <, etc.) require ANY, ALL, or SOME with multirow subqueries.

⚠️ NOT IN and NULLs

- If a subquery used with NOT IN returns even one NULL, no rows are returned
- Fix: Exclude NULLs manually:

```sql
SELECT *
FROM parent
WHERE house NOT IN (
    SELECT house
    FROM kid
    WHERE house IS NOT NULL
);
```

3. Multiple-column Subqueries

- Returns more than one column
- Requires matching structure in the parent query

4. Correlated Subqueries

- References columns from outer query
- Executes once per outer row
- Used in SELECT, UPDATE, and DELETE
- **Cannot run independently**

5. Scalar Subqueries

- Returns a single value (one column, one row)
- Can be used almost anywhere an expression is allowed

- Can also be correlated

Exam Tip

Subquery types are **not mutually exclusive** - a single subquery may belong to multiple categories (e.g., scalar + correlated).

Correlated Subqueries in Detail

What is a Correlated Subquery?

- A correlated subquery is **dependent on the parent query**
- It **references columns from the parent query**, making it **unable to run independently**
- It executes **once for every row** considered by the parent query

Example: SHIP_CABINS

Objective: Find ship cabins where SQ_FT (square footage) is greater than the average for that cabin's ROOM_STYLE.

```sql
SELECT A.SHIP_CABIN_ID, A.ROOM_STYLE, A.ROOM_NUMBER, A.SQ_FT
FROM SHIP_CABINS A
WHERE A.SQ_FT > (
    SELECT AVG(SQ_FT)
    FROM SHIP_CABINS
    WHERE ROOM_STYLE = A.ROOM_STYLE
)
ORDER BY A.ROOM_NUMBER;
```

Key Details:

- ROOM_STYLE = A.ROOM_STYLE in the subquery links it to the parent (A is the alias)
- The subquery runs once per row to calculate the average SQ_FT for that specific ROOM_STYLE

Why Use Correlated Subqueries?

Without them, you'd need:

1. One query to get averages
2. A separate query to compare each cabin against those averages

Correlated subqueries let you combine this into one query.

Validation Query:

```sql
SELECT ROOM_STYLE, AVG(SQ_FT)
FROM SHIP_CABINS
GROUP BY ROOM_STYLE;
```

Other Notes:

- Can be used in SELECT, UPDATE, and DELETE statements
- Table aliases are only necessary if there's a column name conflict
- **Performance Warning**: They may be slower due to running once per parent row
- **Powerful**: Can solve problems that other queries can't easily handle

Correlated Subqueries in UPDATE and DELETE

UPDATE with Correlated Subquery

- Correlated subqueries can appear in:
  - SET clause
  - WHERE clause
- **Purpose**: Modify only the specific rows that meet a condition evaluated per row of the outer query
- **Syntax essentials**:

```sql
UPDATE table_alias
SET column = (SELECT ...)
WHERE column = (SELECT ... WHERE outer_table.col = inner_table.col);
```

- **Alias Usage**: Always alias the outer table and refer to it inside the subquery to create correlation

Example: Update TERMS_OF_DISCOUNT to '10 PCT' for the highest TOTAL_PRICE invoice per quarter:

```sql
UPDATE INVOICES INV
SET TERMS_OF_DISCOUNT = '10 PCT'
WHERE TOTAL_PRICE = (
    SELECT MAX(TOTAL_PRICE)
    FROM INVOICES
    WHERE TO_CHAR(INVOICE_DATE, 'RRRR-Q') = TO_CHAR(INV.INVOICE_DATE, 'RRRR-Q')
);
```

Key Notes:

- Correlation happens by comparing outer and inner INVOICE_DATE converted to 'YYYY-Q' format
- Aggregate MAX() ensures subquery returns one row—so no GROUP BY needed

UPDATE using EXISTS with Correlated Subqueries

- **Purpose**: Conditionally update rows only if certain conditions are met based on related table data

Example: Update PORTS capacity to the number of ships at that port:

```sql
UPDATE PORTS P
SET CAPACITY = (
    SELECT COUNT(*)
    FROM SHIPS
    WHERE HOME_PORT_ID = P.PORT_ID
)
WHERE EXISTS (
    SELECT *
    FROM SHIPS
    WHERE HOME_PORT_ID = P.PORT_ID
);
```

Notes:

- Two correlated subqueries (lines 2-4 and 5-7)
- EXISTS only checks for existence—efficient for filtering

DELETE with Correlated Subquery

- Used to delete rows based on a per-row condition

Example: Delete cabin rows with the smallest balcony per ROOM_TYPE + ROOM_STYLE:

```sql
DELETE FROM SHIP_CABINS S1
WHERE S1.BALCONY_SQ_FT = (
    SELECT MIN(BALCONY_SQ_FT)
    FROM SHIP_CABINS S2
    WHERE S1.ROOM_TYPE = S2.ROOM_TYPE
    AND S1.ROOM_STYLE = S2.ROOM_STYLE
);
```

Key: Correlation is via two columns (ROOM_TYPE, ROOM_STYLE).

EXISTS and NOT EXISTS Operators

- **EXISTS** → Returns TRUE if the subquery returns any rows
- **NOT EXISTS** → Returns TRUE if subquery returns no rows

Syntax:

```sql
SELECT ...
FROM table1
WHERE EXISTS (
    SELECT *
    FROM table2
    WHERE table2.col = table1.col
);
```

Example: Get ports that have ships docked there:

```sql
SELECT PORT_ID, PORT_NAME
FROM PORTS P1
WHERE EXISTS (
    SELECT *
    FROM SHIPS S1
    WHERE P1.PORT_ID = S1.HOME_PORT_ID
);
```

NOTES:

- EXISTS doesn't compare values—just checks for row existence
- Efficient for semi-joins (exists in related table)

Exam Watch Pointers:

- EXISTS subquery syntax: WHERE EXISTS (subquery)—no comparison needed
- COUNT(*) subqueries always return a single row (0 if none match)
- Be careful with EXISTS on large tables—subquery still runs even if no values are returned

WITH Clause (Subquery Factoring Clause)

Purpose

Assigns a name to a subquery block for reuse in the same query.

Syntax

```sql
WITH name1 AS (subquery1),
     name2 AS (subquery2)
SELECT ...
FROM name1
WHERE ...;
```

Key Points

- Acts like an inline view or temporary table only within the query
- Subqueries defined with WITH must come before the main SELECT
- The names (e.g., PORT_BOOKINGS, DENSEST_PORT) are not database objects
- Subquery names can't be used inside their own definition
- Use WITH to organize complex queries and avoid repeating subqueries

Additional Subquery Use Cases

- Within INSERT, CREATE TABLE, UPDATE, DELETE, SELECT
- To create inline views (FROM clause)
- To perform comparisons using aggregate results
- As an expression alternative
- For nesting (up to 255 levels)

- As correlated subqueries (row-by-row logic)