Handout #3
Nested and Multi-Table Queries

How the Select Query is Evaluated

Recall the general form of a SELECT query:

```
SELECT <attribute or functions-applied-to-attributes-list>
FROM <table list>
WHERE <condition>
GROUP BY <grouping attribute(s)>
HAVING <group condition>
ORDER BY <attribute list>
;
```

Here’s a summary describing how it is evaluated conceptually. The order of items is important.

- Apply the FROM clause to identify the tables involved, and to create any joined tables.
- For each row of the participating tables or joins evaluate the <condition> of the WHERE clause. Include only those rows for which the condition evaluates to TRUE.
- Apply the GROUP BY clause to group together those of the included rows that have identical values of the <grouping attribute(s)>.
- Include only those groups that satisfy the <group condition> of the HAVING clause. The group (or aggregate) functions (SUM, AVG, COUNT, etc.) are applied to each group formed by the GROUP BY clause separately.
- Evaluate the ORDER BY clause by sorting the included rows in the specified order (ASC(default) or DESC) of the values in <attribute list>.
- For each of included rows, select only the columns and computed columns that appear in the SELECT clause.

The group functions are applied to each group formed by the GROUP BY clause separately. When the GROUP BY clause is absent, they are applied to the entire set.
Nested Queries

Recall an example from the previous class: list the descriptions of all parts that are included in order number 12491.

```
SELECT  PART_DESCRIPTION
FROM    PART
WHERE   PART_NUMBER IN
    (SELECT  PART_NUMBER
       FROM    ORDER_LINE
       WHERE   ORDER_NUMBER  = '12491')
```

It contains a nested subquery used to generate a set of part numbers that are included in order number 12491.

Subqueries are usually used to generate a list of values or a single value - result of a group function that is used as a part of a condition in a WHERE or a HAVING clause. Thus, a SELECT clause of the subquery must contain a single column, single computer column or a single aggregate function.

Subqueries that produce a list are used with the IN operator that tests membership in a list, or the EXISTS operator that tests whether a list is empty. Subqueries that produce a single value are used with relational operators (\(=\), \(>\), etc).

Here are some examples. Describe the result of each of the queries below. Each of these queries can be evaluated from the innermost out:

- SELECT FIRST, LAST
  FROM CUSTOMER
  WHERE CUSTOMER_NUMBER IN
      (SELECT CUSTOMER_NUMBER
       FROM ORDERS)

- SELECT FIRST, LAST
  FROM SALES_REP
  WHERE SLSREP_NUMBER IN
      (SELECT SLSREP_NUMBER
       FROM CUSTOMER
       GROUP BY SLSREP_NUMBER
       HAVING COUNT(*) > 3
      )
• SELECT FIRST, LAST
  FROM SALES_REP
  WHERE TOTAL_COMMISSION <
    (SELECT AVG(TOTAL_COMMISSION)
     FROM SALES_REP
    )

• An example that uses the EXISTS operator will have to wait until we examine correlated subqueries.

Here are some practice problems:

• List part descriptions of those parts that are not included in any order.

• List the first and the last names of all customers that have ordered a part that is not a Cornpopper or a Trademill.

Multi-Table Queries

Example: Display names of the sales representatives and customers who have the same zipcode. Here we have to disambiguate the column names that exist in both tables.

```
SELECT SALES_REP.LAST || ',' || SALES_REP.FIRST,
       CUSTOMER.LAST || ',' || CUSTOMER.FIRST
FROM SALES_REP, CUSTOMER
WHERE SALES_REP.ZIP_CODE = CUSTOMER.ZIP_CODE
```

Syntax: To refer to column named <column> in a table called <table> use <table>.<column>.

Aliasing refers to renaming a table within a query. The example above can be modified as follows:

```
SELECT S.LAST || ',' || S.FIRST, C.LAST || ',' || C.FIRST
FROM SALES_REP S, CUSTOMER C
WHERE S.ZIP_CODE = C.ZIP_CODE
```

Correlated subqueries are those subqueries that refer to a column from an outer query. Unlike the uncorrelated subqueries the inner correlated queries cannot be evaluated separately of the outer query. They are executed once for each row of the outer query.

Example: List all parts that are not included in any order in the descending order of the unit price.
SELECT PART_DESCRIPTION
FROM PART P
WHERE NOT EXISTS
    (SELECT *
     FROM ORDER_LINE O
     WHERE P.PART_NUMBER = O.PART_NUMBER)
ORDER BY UNIT_PRICE DESC;

Joins

Many queries need to access data from more than one table. To combine (or join) two or more tables in a query:

**equi-join** - connects tuples based on the equality of the attributes of rows in two tables. Most often it involves the equality between a *foreign key* of one table to a *primary key* of another.

**Example**: *print the names of the sales representatives that work with customers that live in Lansing.*

SLSREP_NUMBER is the foreign key in the CUSTOMER table and the primary key of the SALES_REP table.

```
SELECT S.LAST, S.FIRST
FROM CUSTOMER C, SALES_REP S
WHERE C.CITY = 'Lansing' AND
    C.SLSREP_NUMBER = S.SLSREP_NUMBER;
```

Relational operators other than = can also be used in a join condition of a **non-equijoin**.

**Practice problem:**

- Print the name, order number, quoted and unit prices of those parts and orders in which the quoted price is greater than the unit price of the part.

In the query above C.SLSREP_NUMBER = S.SLSREP_NUMBER is the *join condition*, it "ties together" the rows of the two tables.

When the **WHERE** clause is missing or does not contain a join condition, the result is a **cross join** (a.k.a. Cartesian product) of the two tables: a set of rows that includes *every possible row combination* from the two tables. In other words, each row from one table is conjoined with each row of the other.
**Example**: what will be produced by the following query? How many rows will be selected given that the PART table has 10 rows the ORDERS table has 7 rows and all parts have a non-zero price?

```sql
SELECT *
FROM PART, ORDERS
WHERE UNIT_PRICE >0;
```

A join can be performed over more than 2 tables:

**Example**: Find the names of the sales representatives that processed orders number 12498 and number 12500:

```sql
SELECT S.LAST, S.FIRST
FROM ORDERS O, CUSTOMER C, SALES_REP S
WHERE O.ORDER_NUMBER IN ('12498', '12500') AND
O.CUSTOMER_NUMBER = C.CUSTOMER_NUMBER AND
C.SLSREP_NUMBER = S.SLSREP_NUMBER;
```

**Practice problem**:

- print the names of the customers to whom parts are being offered below the unit price.
- print the warehouse numbers that store all parts ordered by Mary Nelson.

A table can also be joined with itself. In such case aliasing the table as shown below is necessary.

**Example**: Print a list of all pairs of parts for which the sum of the unit prices is greater than $100.

```sql
SELECT P1.PART_DESCRIPTION, P2.PART_DESCRIPTION, P1.UNIT_PRICE + P2.UNIT_PRICE
FROM PART P1, PART P2
WHERE P1.UNIT_PRICE + P2.UNIT_PRICE > 100 AND
P1.UNIT_PRICE + P2.UNIT_PRICE < 200
;
```

What will be produced by this query?

**Practice problems**

- How to eliminate duplicate pairs in the previous query?
• Create a query to display pairs of all sales representatives who have the same commission rate.