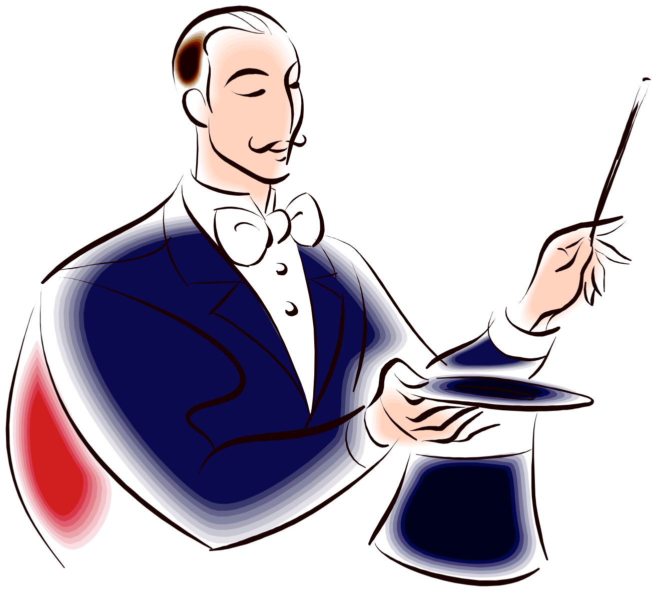
COSC344
Database Theory and Applications



Lecture 7
SQL – Data Manipulation Language (2)

Learning Objectives of this Lecture

- You should
  - understand the difference between subquery and correlated query and how they are executed.
  - understand the difference among ANY, SOME and ALL operators.
  - be able to insert multiple rows into a table using the result of a query.
  - be able to create tables from existing tables
- Source
  - Textbook: Chapter 6.4 and Chapter 7.1
  - Oracle documentation

Subqueries

- A subquery (nested query) in SQL is **a query within a query**
- The inner query is evaluated first and the result is used in the outer query.

Example: Find the names of all employees who have the same salary as Joyce English.

```
SELECT salary
FROM employee
WHERE fname='Joyce' AND lname='English';
```

```
SALARY
----------
  25000
```

```
SELECT fname, lname
FROM employee
WHERE salary =25000;
```

Restrictions on Subqueries

- For the comparison operators, the subquery must produce a **single** tuple.
- DISTINCT can be used.
- In general only a single column can be selected (EXISTS is the exception).
- IN can also be used. Handy if the subquery might produce more than one tuple.
- The general form is
  – <scalar expr> <operator> <subquery>
- Not recommended
  – <subquery> <operator> <scalar expr>
  – <subquery> <operator> <subquery>

Example Subquery Using IN

Example: Find the names of all employees who work for departments located in Houston

```
SELECT fname, lname
FROM employee
WHERE dno IN
  (SELECT dnumber
     FROM dept_locations
     WHERE dlocation='Houston');
```

Which Attributes

- Suppose dept_locations had used *dno* instead of *dnumber*.
- Would this query be ambiguous?

```
SELECT fname, lname
FROM employee
WHERE dno IN
  (SELECT dno
     FROM dept_locations
     WHERE dlocation='Houston');
```

Subqueries with HAVING

- Subqueries can be used in a HAVING clause.
- The subqueries can use aggregate functions as long as they obey the rules on the number of tuples produced.
- Be careful about GROUP BY or HAVING in a subquery

Example: For each salary, list the salary and number of employees making this salary.
Output only the tuples with salary less than the salary of John smith

```
SELECT salary, COUNT(salary)
FROM employee
GROUP BY salary
HAVING salary <
   (SELECT salary
      FROM employee
      WHERE fname='John' AND
            lname='Smith');
```

salary	COUNT(salary)
25000	3

Correlated Subqueries

- A correlated subquery exists when the WHERE clause of the inner query refers to a table in the FROM clause of the outer query.
- The subquery is executed once for each row of the outer query's table.

Example: Find the customers with orders on 03-10-1990

```
SELECT *
FROM customers outer
WHERE TO_DATE('03-10-1990',
        'DD-MM-YYYY') IN
      (SELECT odate
       FROM orders inner
       WHERE outer.cnum=inner.cnum);
```

Cnum	Cname	City	Rating	Snum
2001	Hoffman	London	100	1001
2003	Liu	San Jose	200	1002
2008	Cisneros	San Jose	300	1007
2007	Pereira	Rome	100	1004

How a Correlated Subquery is Performed

- Select a row from the table named in the outer query. This is the current row.
- Store the values of the current row in the alias named in the FROM clause of the outer query.
- Perform the subquery.
- Evaluate the predicate of the outer query using the results of the subquery. This determines if the current row is selected.
- Repeat the above for each row of the outer query's table.

EXISTS Operator

- A Boolean operator that takes a subquery as an argument
  - Yields TRUE if the subquery produces any output
  - Yields FALSE if the subquery produces no output
  - Cannot be unknown

Example: Retrieve the names of employees only if there is a department named Headquarters

```
SELECT fname, lname
FROM employee
WHERE EXISTS
      (SELECT *
       FROM department
       WHERE dname='Headquarters');
```

EXISTS and Correlated Subqueries (1)

- In a correlated subquery, the EXISTS clause is evaluated once for each row in the table of the outer query

Example: Retrieve the names of employees who have no dependents

```
SELECT fname, lname
FROM employee
WHERE NOT EXISTS
      (SELECT *
       FROM dependent
       WHERE ird=eird);
```

EXISTS and Correlated Subqueries (2)

Example: Find the salespeople who have multiple customers

```
SELECT DISTINCT snum
FROM customers outer
WHERE EXISTS
    (SELECT *
     FROM customers inner
     WHERE inner.snum=outer.snum AND
        inner.cnum <> outer.cnum);
```

What does this line do?

EXISTS and Joins

Example: list the name and city of the salespeople who have multiple customers

```
SELECT DISTINCT s.snum,sname,s.city
FROM salespeople s, customers couter
WHERE EXISTS
    ( SELECT *
      FROM customers cinner
      WHERE cinner.snum=couter.snum AND
        cinner.cnum <> couter.cnum) AND
        s.snum=couter.snum;
```

ANY/SOME Operator

- Take a subquery as an argument
- For a row to satisfy the condition specified in the outer query, the value in the attribute that introduces the subquery must satisfy **at least one** of the values in the list of values returned by the subquery
- Must be preceded by a comparison operator

Example: Find the salespeople with customers in their cities

```
SELECT *
  FROM salespeople s
  WHERE city = ANY
    (SELECT city
       FROM customers c
       WHERE s.snum=c.snum);
```

ANY Can be Misleading

- ANY is not necessarily intuitive

Example: Select the customers who have a greater rating than any customer in Rome.

```
SELECT *
   FROM customers
   WHERE rating > ANY
      (SELECT rating
          FROM customers
          WHERE city='Rome');
```

```
SELECT *
   FROM customers
   WHERE rating > ANY
      (SELECT MAX(rating)
          FROM customers
          WHERE city='Rome');
```

Which query is correct?

ALL Operator

- The predicate is true if every value selected by the subqueries satisfies the predicate.

Example: Select the customers who have a greater rating than *every* customer in Rome

```
SELECT *
FROM customers
WHERE rating > ALL
      (SELECT rating
       FROM customers
       WHERE city='Rome');
```

DELETE/UPDATE (Revisited)

```
DELETE FROM <table>
WHERE <condition>;

UPDATE <table>
SET <column>=<value> {,<column>=<value>}
WHERE <condition>;
```

Can include subqueries

```
    DELETE FROM department
    WHERE NOT mgrird IN
          (SELECT ird
           FROM employee);
```

INSERT With SELECT

- Insert multiple tuples into a relation using the result of a query

Example: Create a temporary table that has the employee last name, project name and hours per week for each employee working on a project

```
CREATE TABLE Works_On_Info
( Emp_name  VARCHAR2(15),
  Proj_name VARCHAR2(15),
  Hours_per_week NUMBER(3,1));
```

```
INSERT INTO Works_On_Info (Emp_name, Proj_name, Hours_per_week)
SELECT e.lname, p.pname, w.hours
FROM project p, works_on w, employee e
WHERE p.pnumber=w.pno AND w.eird=e.ird;
```

Creating Tables From Existing Tables

- CREATE TABLE allows to create a new table from existing data.
- Once created there is no inherent relationship between the new table and existing table(s)

```
CREATE TABLE Female_employee

AS (SELECT * FROM employee WHERE Sex=‘F’);


CREATE TABLE Works_On_Info

AS (SELECT e.lname, p.pname, w.hours

    FROM project p, works_on w, employee e

    WHERE p.pnumber=w.pno AND w.eird=e.ird);
```

Questions to Ponder

- What constitutes a good database design?
- When you turn your ER diagram into tables, is the resultant set of tables the most desirable?
- How can we decide whether a given set of tables is “good”?
- Is it possible for a set of tables to cause us grief later on?