COMP2411 – Lab 5

Outline

- **JOINING TABLES**
- **SQL SUBQUERIES**
- **CREATE VIEWS**
- **PARAMETERIZED SQL**

Join Two Tables

- Example:

  Find Allen's name from the EMP table and location of Allen's department from the DEPT table.

```
SQL > SELECT ENAME, LOC
      2 FROM EMP, DEPT
      3 WHERE ENAME = 'ALLEN'
      4 AND EMP.DEPTNO = DEPT.DEPTNO;
```

Using Table Labels To Abbreviate Table Names

- SQL allows you to define a temporary label in the FROM clause by placing the label after the table name separated by a blank. You may then use these labels in place of the full table names within the query.
- Example:

  Issue the same query as the last example but use temporary labels to abbreviate the table names.

```
SQL > SELECT DNAME, E.*
      2 FROM EMP E, DEPT D
      3 WHERE E.DEPTNO = D.DEPTNO
      4 AND LOC = 'CHICAGO'
      5 ORDER BY E.DEPTNO;
```

Joining A Table To Itself

- In some case, a query may refer to a same table more than one time for different purposes, which means you may need to join a table to itself. In order to discriminate the different use as well as make the query simple, you can assign different labels for the same table.
- Example:

  For each employee whose salary exceeds his manager's salary, list the employees' names and salary and the manager's name and salary.

```
SQL > SELECT WORKER.ENAME, WORKER.SAL,
              MANAGER.ENAME,  MANAGER.SAL
    2 FROM EMP WORKER, EMP MANAGER
    3 WHERE WORKER.MGR = MANAGER.EMPNO
    4 AND WORKER.SAL > MANAGER.SAL;
```

Outline

- **JOINING TWO OR MORE TABLES**
- **SQL SUBQUERIES**
- **CREATE VIEWS**
- **PARAMETERIZED SQL**

SQL Sub-queries

- SQL is powerful to support complex queries by using sub-queries contained in the WHERE clause.
- The sub-queries "dynamically" build a search-condition from values stored in the database.
- You can use one or more Sub-queries within one query; also, a sub-query can return one or more values of one column or multiple columns.

Sub-query Return Only One Value

- Example:

  List the name and job of employees who have the same job as JONES.

```
SQL > SELECT ENAME, JOB
      2 FROM EMP
      3 WHERE JOB =
      4 ( SELECT JOB FROM EMP WHERE
        ENAME = 'JONES' );
```

- Note that sub-queries must be enclosed in parentheses but need not be indented.

Sub-query Return A Set Of Values

- If a sub-query may return more than one values, you must attach the word ANY or ALL to the comparison operator (=,!=,>,>=,<,<=) preceding the sub-query to clarify the meaning of your query.
- Example 1:

  Find the employees that earn more than ALL employees in department 30.

```
SQL > SELECT SAL, JOB, ENAME, DEPTNO
  2 FROM EMP
  3 WHERE SAL > ALL
 4 (SELECT SAL FROM EMP
 5 WHERE DEPTNO = 30)
 6 ORDER BY SAL DESC;
```

Sub-query Return A Set Of Values (cont.)

- Example 2:

  Find all the employees in department 10 that have a job that is NOT the same as anyone in department 10.

```
SQL > SELECT ENAME, JOB FROM EMP
  2 WHERE DEPTNO = 10
  3 AND JOB NOT IN
  4 (SELECT JOB FROM EMP
  5 WHERE DEPTNO = 30);
```

Sub-query Return More Than One Column

- Example:

  List the name, job title, and salary of employees who have the same job and salary as Ford.

```
SQL > SELECT ENAME, JOB, SAL
      2 FROM EMP
      3 WHERE (JOB,SAL)=
     4 (SELECT JOB, SAL FROM EMP
     5 WHERE ENAME = 'FORD');
```

Compound Queries with Multiple Sub-queries

- Example:

  Find all the employees in department 10 that have a job that is the same as anyone in the SALES department.

```
SQL > SELECT ENAME, JOB FROM EMP
      2 WHERE DEPTNO = 10
      3 AND JOB IN
      4 (SELECT JOB FROM EMP
      5 WHERE DEPTNO IN
     6 (SELECT DEPTNO FROM DEPT
     7 WHERE DNAME = 'SALES'));
```

Refer to table EMP

Refer to table DEPT

Synchronizing A Repeating Sub-query With A Main Query

- In all the previous examples, the sub-query was executed only once and the resulting value was substituted into the WHERE clause of the main query.
- The following example shows a sub-query that is *executed repeatedly*, once for each row considered for selection (called the *candidate row*) by the main query.

Synchronizing A Repeating Sub-query With A Main Query (cont.)

- Example:

  Find all the employees that earn more than the average salary of employees in their department.

```
SQL > SELECT DEPTNO, ENAME, SAL FROM EMP X
    2 WHERE SAL >
    3 (SELECT AVG(SAL) FROM EMP
    4 WHERE X.DEPTNO = DEPTNO)
    5 ORDER BY DEPTNO;
```

- The processing steps of the example is that:

  Step1: The query points to a candidate row in the table EMP, then the main query tells the sub-query which department average to compute.

  Step2: The sub-query computes the average salary for the candidate employee's department.

  Step3: The main query compares the average salary with the candidate employee's salary.

  The query execute step1-3 repeatedly, until scanning all the rows in EMP.

Outline

- **JOINING TWO OR MORE TABLES**
- **SQL SUBQUERIES**
- **CREATE VIEWS**
- **PARAMETERIZED SQL**

CREATE VIEWS

- A view is a window into the data in one or more tables. The view itself contains no data and takes up no space.
- Syntax:

```
CREATE VIEW view_name
[(column_name[,column_name]…)]
AS SELECT_STATEMENT;
```

CREATE VIEWS

- Example:

  Create a view with DEPTNO, DNAME, ENAME, JOB from the DEPT and EMP tables.

```
SQL > CREATE VIEW DEPT_EMP
      2  AS SELECT DEPT.DEPTNO, DNAME,
         ENAME, JOB
      3 FROM DEPT, EMP
      4 WHERE DEPT.DEPTNO = EMP.DEPTNO;
SQL > SELECT * FROM DEPT_EMP;
```

Outline

- **JOINING TWO OR MORE TALBES**
- **SQL SUBQUERIES**
- **CREATE VIEWS**
- **PARAMETERIZED SQL**

Why Use Parameterized SQL?

- Suppose you want to write a query to list the employees with various jobs (each time only return employees of one job), not just those whose job is SALESMAN. How can you do that?
- Solution1: You could do that by editing a different CHAR value into the WHERE clause each time you run the command.

  But there is a smarter way!
- Solution 2: You can write a 'query template' for these similar queries (called parameterized SQL), by using substitution variables in place of the values you want to specify for different queries.

What is Substitution Variable?

- A substitution variable is a user variable name preceded by one or two ampersands (&). When SQL*Plus encounters a substitution variable in a command, SQL*Plus executes the command as though it contained the value of the substitution variable, rather than the variable itself.

Where and How to Use Substitution Variables?

- You can use substitution variables anywhere in SQL and SQL*Plus commands, except as the first word entered at the command prompt.
- When SQL*Plus encounters an undefined substitution variable in a command, SQL*Plus prompts you for the value, then reads your response from the keyboard.
- If a terminal is not available (for example, you run the command file in batch mode), SQL*Plus uses the redirected file.

Example Of Parameterized SQL

- Create a command file named STATS, to be used to calculate a subgroup statistic (the maximum value) on a numeric column:

```
SQL> CLEAR BUFFER
   buffer cleared
SQL> INPUT
      1 SELECT &GROUP_COL,
      2 MAX(&NUMBER_COL) MAXIMUM
      3 FROM &TABLE
      4 GROUP BY &GROUP_COL
      5
SQL> SAVE STATS
   Created file STATS.sql
SQL>
```

Example Of Parameterized SQL (cont.)

- Now run the command file STATS and respond as shown below to the prompts for values:

```
SQL> @STATS
Enter value for group_col: JOB
old 1: SELECT &GROUP_COL,
new 1: SELECT JOB,
Enter value for number_col: SAL
old 2: MAX(&NUMBER_COL) MAXIMUM
new 2: MAX(SAL) MAXIMUM
Enter value for table: EMP
old 3: FROM &TABLE
new 3: FROM EMP
Enter value for group_col: JOB
old 4: GROUP BY &GROUP_COL
new 4: GROUP BY JOB
```