7BDIN006W
Big Data Theory and Practice

Lecture 5

Big Data and Relational Model. SQL: Displaying Data from Multiple Tables, Aggregate / Group Functions

UNIVERSITY OF WESTMINSTER

Displaying Data from Multiple Tables
(Continued)

Creating Joins with the `USING` Clause

- If several columns have the same names but the data types do not match, the `NATURAL JOIN` clause can be modified with the `USING` clause to specify the columns that should be used for an equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

Retrieving Records with the USING Clause

```
SELECT *
FROM   emp
         JOIN dept
            USING (deptno) ;
```

```
SELECT emp.empno, ename, dept.deptno, dname, loc
FROM   emp
         JOIN dept
              USING (deptno) ;
```

The above statement will result in an error as when the using statement does not allow for common columns to be qualified.

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

```
SELECT *
FROM   emp
         NATURAL JOIN dept ;
```

`INNER` Versus `OUTER` Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) tables is called a left (or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

LEFT OUTER JOIN

```
SELECT e.ename, e.deptno, d.deptno
FROM   emp  e
          LEFT OUTER JOIN dept d
               ON e.deptno = d.deptno;
```

RIGHT OUTER JOIN

```
SELECT e.ename, e.deptno, d.deptno
FROM   emp  e
          RIGHT OUTER JOIN dept d
               ON e.deptno = d.deptno;
```

FULL OUTER JOIN

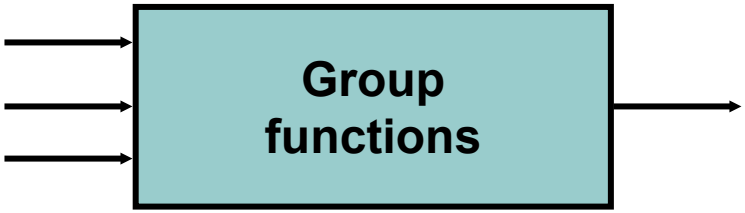
```
SELECT e.ename, e.deptno, d.deptno
FROM   emp  e
          FULL OUTER JOIN dept d
               ON e.deptno = d.deptno;
```

Reporting Aggregated Data Using Aggregate / Group Functions

Types of Aggregate / Group Functions

Group functions operate on sets of rows to give one result per group.

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **SUM**
- ***STDDEV****
- ***VARIANCE****



*** aggregate function that is not specified in the SQL standard**

Aggregate / Group Functions: Syntax

```
SELECT     group_function(column), ...      (3)
FROM       table                            (1)
[WHERE     condition]                       (2)
```

The numbers indicate the order in which the various clauses will be evaluated.

Using Aggregate / Group Functions

You can use `AVG` and `SUM` only for numeric data.

```
SELECT AVG(sal), MAX(sal),
       MIN(sal), SUM(sal)
FROM   emp
WHERE  job LIKE '%MAN%';
```

The `MIN` and `MAX` functions can be used for numeric, character, and date data types.

```
SELECT MIN(hiredate), MAX(hiredate)
FROM   emp;
```

Any null values present are removed prior to the application of an Aggregate / Group Function.

```
SELECT MIN(comm), sum(comm)
FROM   emp;
```

Using the COUNT Function

COUNT(*) returns the **number of rows in a table**:

1

```
SELECT COUNT(*)
FROM   emp
WHERE  deptno = 30;
```

COUNT(*expr*) returns the **number of rows with non-null values for the *expr***:

2

```
SELECT COUNT(comm)
FROM   emp
WHERE  deptno = 30;
```

Using the `DISTINCT` Keyword

- `COUNT(DISTINCT expr)` returns the number of distinct non-null values of the *`expr`*.
- To display the number of distinct department values in the `EMP` table:

```
SELECT COUNT(DISTINCT deptno)
FROM   emp;
```

- To display the total number of distinct job descriptions in the `EMP` table:

```
SELECT COUNT(DISTINCT job)
FROM   emp;
```

Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT     column, group_function          (5)
FROM       table                           (1)
[WHERE     condition]                      (2)
[GROUP BY  group_by_expression]            (3)
[ORDER BY  column];                        (4)
```

The numbers indicate the order in which the various clauses will be evaluated.

You can divide rows in a table into smaller groups by using the GROUP BY clause.

Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT    deptno, AVG(sal)
FROM      emp
GROUP BY deptno;
```

The GROUP BY column does not have to be in the SELECT list.

```
SELECT AVG(sal)
FROM      emp
GROUP BY deptno;
```

Multiple columns can be used in the GROUP BY clause.

```
SELECT    deptno, job, SUM(sal)
FROM      emp
GROUP BY deptno, job;
```

Using the `GROUP BY` Clause

When an aggregate function and/or a `GROUP BY` clause is used in a query, the query will be evaluated as follows:

a. **The query rows will be grouped according to the grouping columns**
b. **The aggregate function(s) will be applied on each of the groups**
   NB: If there is no GROUP BY clause is specified then all the rows of the query rows will be considered as a single group
c. **The results can be considered as an (internal/system) table whose**
   - **schema (header) will comprise all the grouping columns and aggregate functions used in the query,**
   - **instance consists of one row for each group (i.e. combination of grouping columns values).**
d. **The `SELECT` clause will return rows from the generated internal/system table, i.e. the `SELECT` clause must contain only aggregate functions and grouping columns**

Using the GROUP BY Clause

```
SELECT    deptno, job, SUM(sal)
FROM      emp
WHERE sal > 1250
GROUP BY deptno, job
ORDER BY deptno, job;
```

DEPTNO	JOB	SAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	MANAGER	2975
30	SALESMAN	1600
30	SALESMAN	1500

The total salary per department and job description of those earning more than 1250

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	MANAGER	2975
30	MANAGER	2975
30	SALESMAN	3100

Illegal Queries Using Group Functions

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause:

```
SELECT deptno, COUNT(ename)
FROM   emp;
```

```
SELECT deptno, COUNT(ename)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Column missing in the `GROUP BY` clause

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT   deptno, AVG(sal)
FROM     emp
WHERE    AVG(sal) > 1250
GROUP BY deptno;
```

```
WHERE  AVG(sal) > 1250
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Cannot use the `WHERE` clause to restrict groups

Restricting Group Results with the HAVING Clause

To restrict the rows returned after the application of aggregate function(s) and/or a GROUP BY clause in a query a HAVING clause needs to be used.

The syntax is as follows:

```
SELECT     column, group_function          (6)
FROM       table                           (1)
[WHERE     condition]                      (2)
[GROUP BY  group_by_expression]            (3)
[HAVING    group_condition]                (4)
[ORDER BY  column];                        (5)
```

The numbers indicate the order in which the various clauses will be evaluated.

Using the HAVING Clause

```
SELECT    deptno, job, SUM(sal)
FROM      emp
WHERE     job like '%MAN%'
GROUP BY deptno, job
HAVING    sum(sal) > 2500
ORDER BY deptno, job;
```

DEPTNO	JOB	SAL
10	MANAGER	2450
20	MANAGER	2975
30	MANAGER	2975
30	SALESMAN	1500
30	SALESMAN	1250
30	SALESMAN	1600
30	SALESMAN	1250

The total salary per department and job description of those whose job description contains the string "MAN"

DEPTNO	JOB	SUM(SAL)
10	MANAGER	2450
20	MANAGER	2975
30	MANAGER	2975
30	SALESMAN	5600

DEPTNO	JOB	SUM(SAL)
20	MANAGER	2975
30	MANAGER	2975
30	SALESMAN	5600

Using the HAVING Clause

```
SELECT   deptno, MAX(sal)
FROM     emp
GROUP BY deptno
HAVING   MAX(sal)>1200 ;
```

```
SELECT   job, SUM(sal) PAYROLL
FROM     emp
WHERE    job NOT LIKE '%MAN%'
GROUP BY job
HAVING   SUM(sal) > 3000
ORDER BY SUM(sal);
```

Using the HAVING Clause

The HAVING clause can be used without a GROUP BY

```
SELECT    MAX(sal)
FROM      emp
HAVING    MAX(sal)>1200 ;
```

An aggregate function returns a null values if all rows that it is applied to contain **only** nulls

```
SELECT    job, SUM(comm) COMMISION
FROM      emp
WHERE     job not in ('ANALYST', 'SALESMAN')
GROUP BY job
HAVING    SUM(comm) > 3000
ORDER BY SUM(comm);
```

Nesting Group Functions

Display the maximum average salary:

```
SELECT   MAX(AVG(sal))
FROM     emp
GROUP BY deptno;
```

```
SELECT   MAX(avg_sal)
FROM     (SELECT AVG(sal) avg_sal
          FROM emp
          GROUP BY deptno);
```

Not all implementation of SQL support nesting

Using Group Functions in a Subquery

- The Oracle server executes subqueries first.
- The Oracle server returns results into the `HAVING` clause of the main query.

```
SELECT ename, job, sal
FROM   emp
WHERE  sal =
                 (SELECT MIN(sal)
                  FROM   emp);
```

800

The HAVING Clause with Subqueries

```
SELECT    deptno, MIN(sal)
FROM      emp
GROUP BY  deptno
HAVING    MIN(sal) >
                        (SELECT MIN(sal)
                         FROM   emp
                         WHERE deptno = 30);
```

1100

What Is Wrong with This Statement?

```
SELECT empno, ename
FROM   emp
WHERE  sal =
                (SELECT   MIN(sal)
                 FROM     emp
                 GROUP BY deptno);
```

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Single-row operator with multiple-row subquery