7BDIN006W
Big Data Theory and Practice

Lecture 4

Big Data and Relational Model. SQL: Sorting Data, Subqueries, Joins

UNIVERSITY OF WESTMINSTER

Sorting Data

Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
  - ASC: ascending order, default
  - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT    ename, job, deptno, hiredate
FROM      emp
ORDER BY hiredate ;
```

Sorting

- **Sorting in descending order:**

```
SELECT    ename, job, deptno, hiredate
FROM      emp
ORDER BY hiredate DESC ;
```

1

- **Sorting by column alias:**

```
SELECT empno, ename, sal*12 annsal
FROM   emp
ORDER BY annsal ;
```

2

- **Sorting by multiple columns:**

```
SELECT ename, deptno, sal
FROM   emp
ORDER BY deptno, sal DESC;
```

3

Using Subqueries to Solve Queries

Using a Subquery to Solve a Problem

Who has a salary greater than Allen's?

Main query:

Which employees have salaries greater than Allen's salary?

Subquery:

What is Allen's salary?

Subquery Syntax

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT       select_list
                    FROM          table);
```

- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.

Using a Subquery

```
SELECT ename, sal
FROM   emp
WHERE  sal >
                (SELECT sal
                 FROM   emp
                 WHERE  ename = 'ALLEN');
```

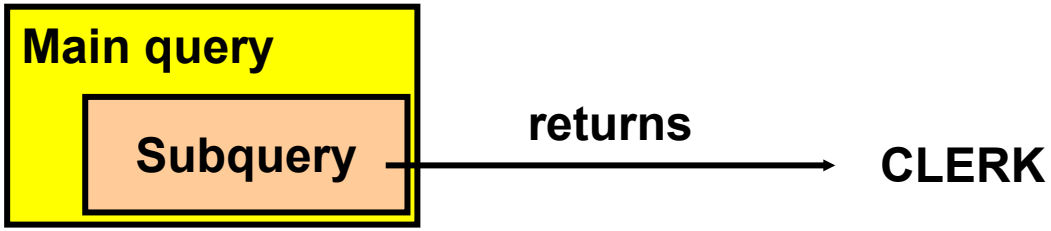
Allen's Salary

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The `ORDER BY` clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

Types of Subqueries

- **Single-row subquery**



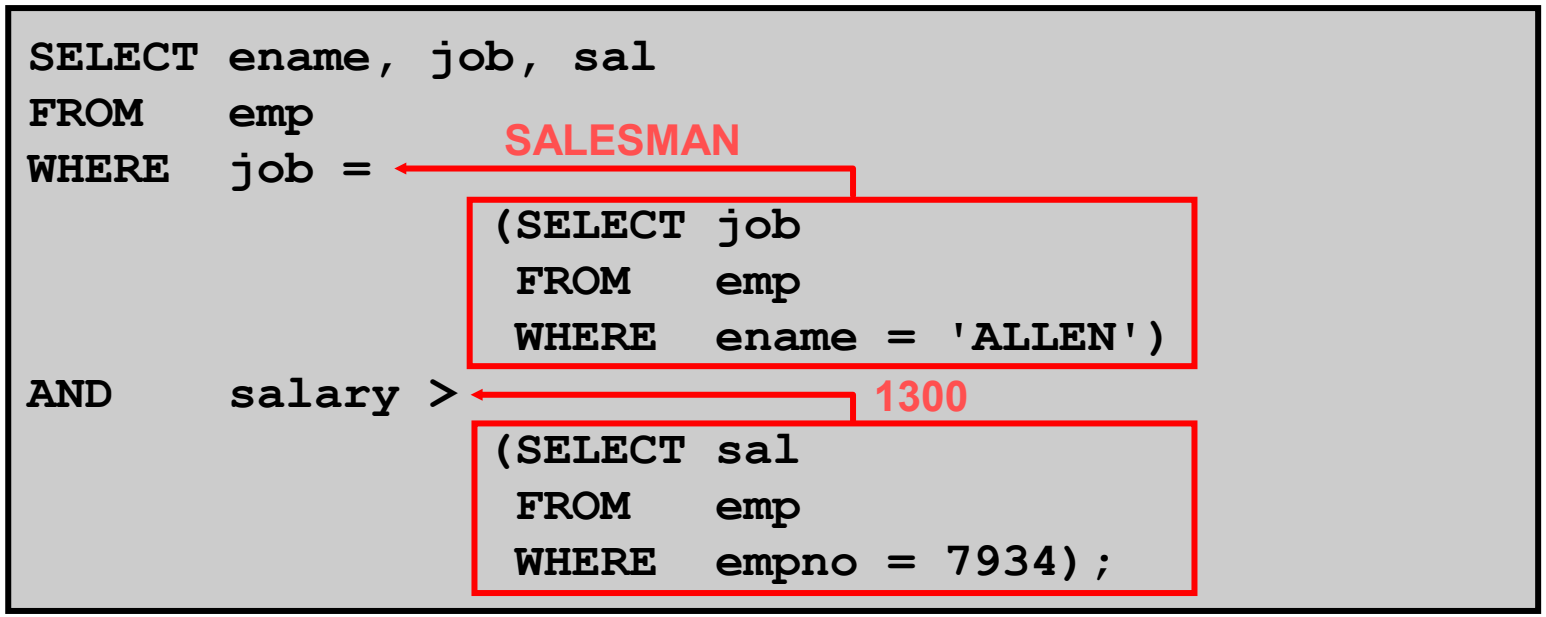
- **Multiple-row subquery**

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Executing Single-Row Subqueries



Will This Statement Return Rows?

```
SELECT ename, job
FROM   emp
WHERE  job =
                (SELECT job
                 FROM   emp
                 WHERE  ename = 'BING');
```

```
no rows selected
```

Subquery returns no values.

What Is Wrong with This Statement?

```
SELECT empno, ename
FROM   emp
WHERE  job =
                (SELECT   job
                 FROM     emp
                 WHERE    deptno = 20)
```

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

Single-row operator with multiple-row subquery

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ϑ ANY | ϑ SOME	Compare value to each value returned by the subquery
ϑ ALL	Compare value to every value returned by the subquery
where ϑ can be any of the 6 comparison operators, i.e. < | <= | = | >= | > and <> or !=	

Using the IN or =ANY Operator in Multiple-Row Subqueries

```
SELECT empno, ename
FROM   emp
WHERE  job =
                    (SELECT   job
                     FROM     emp
                     WHERE    deptno = 20)
```

Above returns an error as the subquery returns multiple rows

```
SELECT empno, ename
FROM   emp
WHERE  job IN
                   (SELECT   job
                    FROM     emp
                    WHERE    deptno = 20)
```

```
SELECT empno, ename
FROM   emp
WHERE  job = ANY
                   (SELECT   job
                    FROM     emp
                    WHERE    deptno = 20)
```

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT empno, ename, job, sal
FROM   emp
WHERE  sal < ANY
                    (SELECT sal
                     FROM   emp
                     WHERE  job = 'SALESMAN')
AND    job <> 'SALESMAN';
```

1250, 1600, 1500

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT empno, ename, job, sal
FROM   emp
WHERE  sal < ALL
                    (SELECT sal
                     FROM   emp
                     WHERE  job = 'SALESMAN')
AND    job <> 'SALESMAN';
```

1250, 1600, 1500

Executing a Multi-Column Single-Row Subqueries

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

(ANALYST, 3000)

Executing a Multi-Column Single-Row Subqueries

```
SELECT ename, job, sal
FROM   emp
WHERE (job, sal) =ANY
                      (SELECT job, sal
                       FROM   emp
                       WHERE  deptno = 30);
```

only the =ANY and the IN can be used

```
SELECT ename, job, sal
FROM   emp
WHERE (job, sal) <= ANY
                      (SELECT job, sal
                       FROM   emp
                       WHERE  deptno = 30);
```

Null Values in a Subquery

```
SELECT ename
FROM   emp
WHERE  empno NOT IN
                    (SELECT mgr
                     FROM   emp
                     WHERE job = 'PRESIDENT');
no rows selected
```

Nested Subqueries

```
SELECT empno, ename, job, sal
FROM   emp
WHERE  sal < ALL
               (SELECT sal
                FROM   emp
                WHERE  job = 'SALESMAN'
                  AND  deptno =ANY
                                (SELECT deptno
                                 FROM dept
                                 WHERE loc = 'CHICAGO'))
AND    job <> 'SALESMAN';
```

Displaying Data from Multiple Tables

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Cross joins
- Natural joins
- `USING` clause
- Full (or two-sided) outer joins
- Arbitrary join conditions for outer joins

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT   table1.column, table2.column
FROM     table1 [, table2] |
         [CROSS JOIN table2] |
         [JOIN table2
           ON (table1.column_name = table2.column_name)]|
         [JOIN table2
           ON (condition_involving table1 and table2)]|
         [NATURAL JOIN table2] |
         [JOIN table2 USING (column_name)] |
         [LEFT|RIGHT|FULL OUTER JOIN table2
           ON (table1.column_name = table2.column_name)];
```

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition.

Creating Cross Joins

- The CROSS JOIN clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

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

```
SELECT emp.*, dname
FROM   emp
         CROSS JOIN dept
WHERE  emp.deptno = dept.deptno;
```

```
SELECT emp.*, dname
FROM   emp
         CROSS JOIN dept
WHERE  emp.deptno = dept.deptno
  and  loc = 'CHICAGO';
```

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.
- Do not use aliases on columns that are identified in the `USING` clause and listed elsewhere in the SQL statement.

Using Table Aliases

- Use table aliases to simplify queries.
- Use table aliases to improve performance.

```
SELECT e.empno, e.ename, d.loc, d.deptno
FROM   emp e
          CROSS JOIN dept d
WHERE  e.deptno = d.deptno
  and  loc = 'CHICAGO';
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Retrieving Records with the ON Clause

```
SELECT empno, ename, e.deptno, d.deptno, loc
FROM   emp e
         JOIN dept d
              ON e.deptno = d.deptno);
```

Self-Joins Using the ON Clause

You want to find the details of employees along with those of their managers

EMP (WORKER)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-Dec-1980	800	-	20
7499	ALLEN	SALESMAN	7698	20-Feb-1981	1600	800	30
7521	WARD	SALESMAN	7698	22-Feb-1981	1250	500	30
7566	JONES	MANAGER	7839	02-Apr-1981			

...

EMP (MANAGER)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-Dec-1980	800	-	20
7499	ALLEN	SALESMAN	7698	20-Feb-1981	1600	800	30
7521	WARD	SALESMAN	7698	22-Feb-1981	1250	500	30
7566	JONES	MANAGER	7839	02-Apr-1981	2975	-	20

...

MGR in the *WORKER* table is equal to EMPNO in the *MANAGER* table.

Self-Joins Using the ON Clause

```
SELECT e.empno emp_empno, e.ename emp_ename, e.mgr emp_mgrno,
       m.empno man_empno, m.ename man_ename
FROM   emp e
       JOIN emp m
            ON (e.mgr = m.empno);
```

Applying Additional Conditions to a Join

```
SELECT e.empno, e.ename, d.loc, d.deptno
FROM   emp e
          JOIN dept d
              ON (e.deptno = d.deptno)
                 and loc = 'CHICAGO';
```

```
SELECT e.empno, e.ename, d.loc, d.deptno
FROM   emp e
          JOIN dept d
              ON (e.deptno = d.deptno)
WHERE loc = 'CHICAGO';
```

```
SELECT e.empno, e.ename, d.loc, d.deptno
FROM   emp e
          JOIN dept d
              ON (e.deptno = d.deptno
                    and 'CHICAGO');
```

Retrieving Records with Nonequijoins

```
SELECT e.ename, sal, grade
FROM   emp e
         JOIN salgrade
              ON sal BETWEEN
                        losal AND hisal;
```

Creating Three–Way Joins with the ON Clause

```
SELECT empno, ename, e.deptno, dname, grade
FROM   emp e
       JOIN dept d
            ON d.deptno = e.deptno
       JOIN salgrade
            ON (sal BETWEEN
                   losal AND hisal);
```