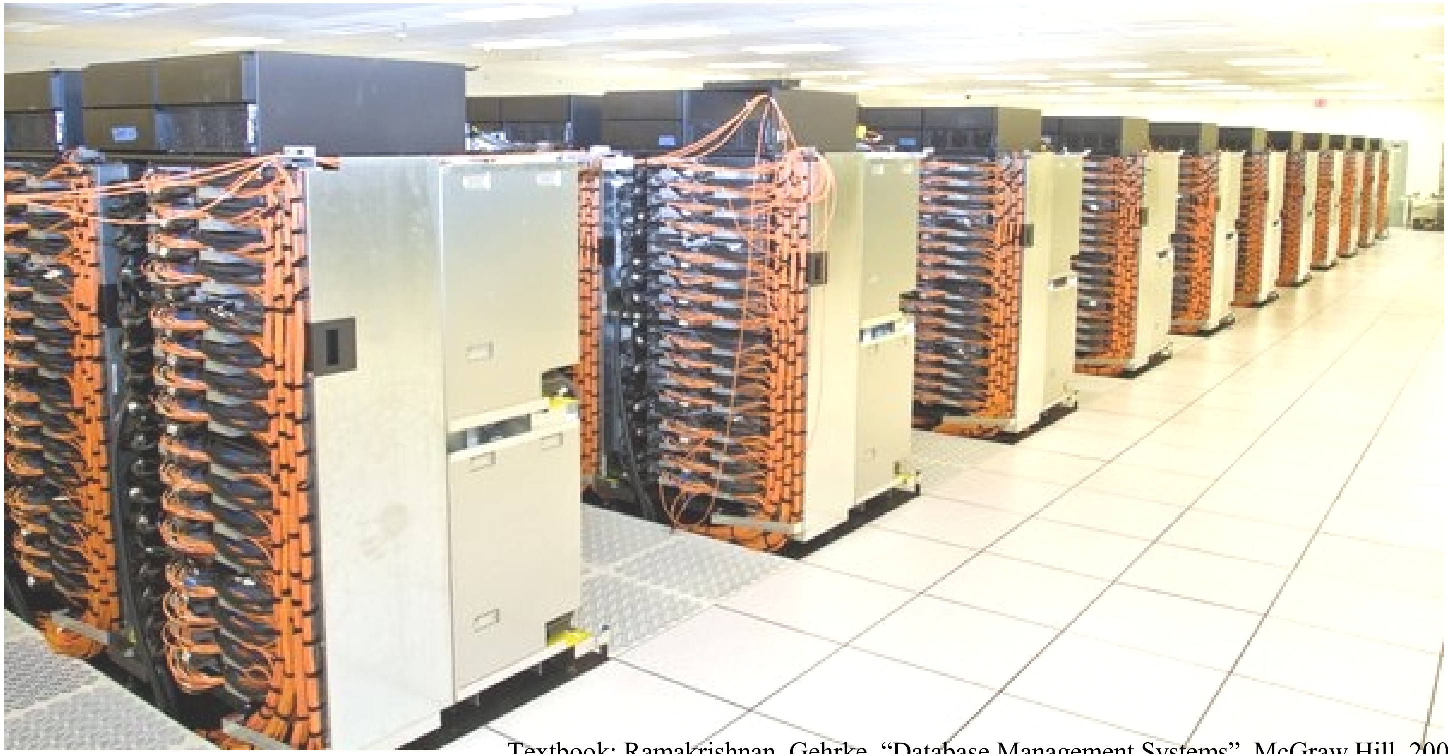
Databases

Cap. 7. SQL. Subqueries. Union and Difference



Textbook: Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

2017 UPT

Conf.Dr. Dan Pescaru

SQL Subqueries

1. Is a query within another query. Alternative name: **nested query**
2. Subqueries enable writing of queries that select data rows for criteria that are actually developed while the query is executing (at run time)

E.g.

```
SELECT prj_list FROM tables
    WHERE cond_including_subquery
```

Subquery categories (I)

1. There are three basic types of subqueries if we consider the return type:

   A. Subqueries that operate on lists by use of the IN operator or with a comparison operator using ANY or ALL modifiers. These subqueries can return a *set of values*, but the values must be from a *single column* of a table

Subquery categories (II)

B. Subqueries that use an comparison operator (=, <, >, <>) without modifiers – these subqueries must return only a *single, scalar value*

C. Subqueries that use the EXISTS operator to test the existence of data rows satisfying specified criteria. These subqueries can return a *set of values*

Uncorrelated vs. correlated subqueries

1. Uncorrelated subqueries - The subquery is independent from the main query. It can be resolved independently of the main query
2. Correlated subqueries - The subquery cannot be resolved independently of the main query (it depends of some values passed by the main query)

Example database (harbour)

- **Sailors table**

sid	name	rank	age
22	John	7	45
31	Horace	1	33
58	Andrei	8	54
71	John	9	55

- **Boats Table**

bid	name	color
101	Cleo	Blue
102	Gazelle	Red
103	Poseidon	Green

- **Reserves Table**

sid	bid	date
58	101	2014/10/03
22	102	2014/10/18
58	103	2014/11/23
22	103	2014/11/25

Subqueries – general rules

1. A subquery SELECT statement is very similar to a regular query
2. The SELECT clause of a subquery must contain only one expression, only one aggregate function, or only one column
3. The value(s) returned by a subquery must be join-compatible with the WHERE clause of the outer query
4. Subqueries cannot manipulate their results internally. This means that a subquery cannot include the ORDER BY clause

Subqueries – the IN operator

1. Subqueries that are introduced with the keyword IN take the general form:

   ```
   WHERE expression [NOT] IN (subquery)
   ```

2. E.g.: all sailors who haven’t reserved the boat 103

   ```
   SELECT  S.sname  FROM Sailors S
       WHERE  S.sid NOT IN (SELECT R.sid
                            FROM Reserves R
                            WHERE R.bid=103)
   ```

3. To understand semantics think of a nested loops evaluation: for each Sailors row, check the qualification by computing the subquery

Understanding the IN operator (I)

1. In order to understand how this query executes, we begin our examination with the lowest subquery
2. We will execute it independently of the outer queries

```
SELECT R.sid
        FROM Reserves R
        WHERE R.bid=103
```

3. The result is { 58, 22 }

Understanding the IN operator (II)

1. Now, let's substitute the result of subquery as the operand for the IN operator and execute the main query

```
SELECT  S.sname  FROM Sailors S
   WHERE  S.sid NOT IN { 58, 22 }
```

2. The result is { Horace, John }

Using IN to implement INTERSECT

1. Find all sailors that reserves both blue and green boats

```
SELECT DISTINCT s.sid, s.name
         FROM Sailors s, Boats b, Reserves r
         WHERE s.sid=r.sid AND r.bid=b.bid AND
         b.color='Blue' AND
            s.sid IN
                (SELECT s1.sid
                     FROM Sailors s1, Boats b1, Reserves r1
                     WHERE s1.sid=r1.sid AND r1.bid=b1.bid
                                   AND b1.color='Green')
```

Subqueries – comparison operators

1. The subquery must return a single scalar value. Most of the time it is an aggregation without group by (discussed in the next chapter)
2. This is also called a scalar subquery because a single column of a single row is returned by the subquery
3. If a subquery returns more than one value the query will fail to execute

Subqueries – comparison operators

1. Find all sailors that are older than the sailor which reserves the boat 103 on {2014-11-23}

```
SELECT * FROM Sailors WHERE
  age > (SELECT s.age
                   FROM Sailors s INNER JOIN
                           Reserves r ON s.sid=r.sid
                   WHERE r.bid=103  AND
                               r.date='2014-11-23')
```

Subqueries – ALL and ANY modifiers

1. The ALL and ANY keywords can modify a comparison operator to allow an outer query to accept multiple values from a subquery
2. The general form of the WHERE clause for this type of query is

```
WHERE <expression> <comp_op>
                        [ALL | ANY] (subquery)
```

Using ALL

1. The ALL keyword modifies the greater than (or less than) comparison operator to mean greater than (or less than) all values

E.g. Find all sailors that have the maximum rank

```
SELECT * FROM Sailors WHERE
    rank >= ALL ( SELECT rank
                           FROM Sailors)
```

Using ANY

1. Less restrictive than ALL, modifies the greater than (or less than) comparison operator to mean greater than (or less than) any (or some) values

E.g. Find the sailors with rank greater than some sailor called Horatio

```
SELECT * FROM Sailors WHERE
    rank > ANY ( SELECT rank FROM Sailors
                 WHERE name='Horatio')
```

IN, ANY, ALL

1. Equivalent operators

- IN is equivalent with =ANY
- NOT IN is not equivalent with <>ANY
- NOT IN is equivalent with <>ALL

The EXISTS operator

1. The WHERE clause of the outer query tests for the existence of rows returned by the inner query
2. The subquery does not actually produce any data; rather, it returns a value of TRUE or FALSE

E.g. Find all sailors that reserves at least one boat (=> *correlated* query!)

```
SELECT * FROM Sailors s WHERE
    EXISTS ( SELECT * FROM Reserves r
                WHERE r.sid=s.sid)
```

Using EXISTS to implement DIVISION

1. Find the names of sailors who have reserved all boats (multiple nested subqueries)

```
SELECT s.sid, s.name FROM Sailors s
    WHERE NOT EXISTS ( SELECT b.bid
             FROM Boats b
             WHERE NOT EXISTS (
                      SELECT r.bid
                               FROM Reserves r
                      WHERE r.bid = b.bid
                               AND r.sid = s.sid ))
```

FROM nested queries

1. Subqueries could be places inside the FROM clause. Useful for embedding aggregation functions
2. E.g. Form pairs of unreserved boats and sailors

```
SELECT s.*, b.*
FROM Sailors s,
(SELECT b1.bid, b1.name
    FROM Boats b1
    WHERE NOT EXISTS (
                  SELECT r.*
    FROM Reserves r
                      WHERE r.bid=b1.bid)) b;
```

UNION

1. Combines the results of two queries, and eliminates duplicate selected rows
2. Relations have to be union compatible
3. UNION ALL preserves duplicates

```
SELECT s1.*
    FROM Harbour1.Sailors s1
UNION [ALL]
SELECT s2.*
    FROM Harbour2.Sailors s1;
```

UNION for FULL JOIN

1. Union can be used to implement full join in system that do not support full join (e.g. MySQL)

```
SELECT * FROM T1
    LEFT JOIN T2 ON T1.id = T2.id
UNION
SELECT * FROM T1
    RIGHT JOIN T2 ON T1.id = T2.id;
```

SQL DIFFERENCE (I)

1. Implemented by the EXCEPT (or MINUS) operator
2. Relations have to be union compatible
3. It is not implemented by MySQL (can be implemented using NOT IN)

SQL DIFFERENCE (II)

1. E.g. Find all sailors which reserved a red boat but not a green one:

```
SELECT DISTINCT s.sid, s.name
    FROM Sailors s, Reserves r, Boats b
    WHERE s.sid=r.sid AND r.bid=b.bid AND
b.color="Red"
EXCEPT
SELECT DISTINCT s.sid, s.name
    FROM Sailors s, Reserves r, Boats b
    WHERE s.sid=r.sid AND r.bid=b.bid AND
b.color="Green"
```