CptS 451- Introduction to Database Systems

SQL as a Query Language – part2

(DMS ch-5)

Instructor: Sakire Arslan Ay

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Nested Queries

- Also called **subqueries**. Embedded inside an **outer** query.
- Similar to function calls in programming languages.
- **Example**: Who is in Sally's department?

```
SELECT E1.ename
FROM   Emp E1, Emp E2
WHERE  E2.ename = 'Sally' AND E1.dno=E2.dno;
```

OR:

```
SELECT ename
FROM  Emp
WHERE Emp.dno IN
      ( SELECT dno
        FROM Emp
        WHERE ename = 'Sally');
```

subquery

names are scoped

- Semantics:
  - The nested query returns a relation containing dno for which Sally works
  - For each tuple in Emp, evaluate the nested query and check if Emp.dno appears in the set of dno's returned by nested query.

Subqueries - Conditions Involving Relations

- Usually nested queries produce a relation as an answer.
- Conditions involving relations :

1.	**s IN R**	s is equal to *one* of the values in R (s is a list of attributes, R is a relation)
2.	**s> ALL R**	s is greater than *every* value in unary relation R (s is a single attribute, R is a relation)
3.	**s > ANY R,** **s > SOME R**	s is greater than *at least 1* element in unary relation R. –*any* and *some* are synonyms in SQL (s is a single attribute, R is a relation)
4.	**EXISTS R**	R is *not empty (*R is a relation)
	Other operators (<, = , <=, >=, <>) could be used instead of >	
	EXISTS can be negated.	

Example 1

- Find the employees with the highest salary.

```
SELECT ename
FROM    emp
WHERE sal >= ALL
          (SELECT MAX(sal)
           FROM  Emp);
```

- < ALL, <= ALL, >= ALL, = ALL, <> ALL also permitted

Example 2

- Who makes more than someone in the Hardware department?

```
SELECT ename
FROM Emp
WHERE sal > ANY
                (SELECT sal
                FROM Emp,  Dept
                WHERE Emp.dno = Dept.dno AND  Dept.dname = 'Hardware');
```

- "< ANY, <= ANY, >= ANY, > ANY = ANY, <> ANY" are permitted

Nested Queries Producing One Value

- Sometimes subqueries produce a single value

```
SELECT   ename
FROM     Emp
WHERE  Emp.dno =
                    (SELECT dno
                     FROM  Dept
                     WHERE (dname='Production'));
```

- Assume there is only one department called "Production," then the subquery returns one value.
- If it returns more, it's a run-time error.

Subqueries-Testing Empty Relations

- "EXISTS" checks for nonempty set
- Similarly, "NOT EXISTS" can be used.
- "Find employees who make more money than *some* manager"

```
SELECT ename
FROM    Emp as E1
WHERE EXISTS
            (SELECT ssn
             FROM Emp, Dept
             WHERE (Emp.ename = Dept.mgr)
                        AND (E1.sal > Emp.sal));
```

E1(ename,dno,salary)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	32K
Tom	333	56K
Mary	333	65K

Emp(ename,dno,salary)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	32K
Tom	333	56K
Mary	333	65K

Dept(dno,dname, mgr)

dno	dname	mgr
111	Sells	Alice
222	Toys	Lisa
333	Electronics	Mary

Result

E1.ename
Jack
Alice
Tom
Mary

Subqueries-Testing Empty Relations (cont)

- The nested query uses attributes name of E1 defined in outer query. These two queries are called *correlated*.
  - Semantics: for each assignment of a value to some term in the subquery that comes from a tuple variable outside, the subquery needs to be executed

Subqueries-Testing Empty Relations


```
SELECT ename
FROM   Emp as E1
WHERE EXISTS
            (SELECT ename
             FROM Emp, Dept
             WHERE (Emp.ename = Dept.mgr)
                        AND (E1.sal > Emp.sal));
```

- Evaluate exists condition on (Jack,111,81K)
  - Subquery returns: Alice , Lisa , Mary
  - exists returns true
- Evaluate exists condition on (Alice,111,70K)
  - Subquery returns: Lisa , Mary
  - exists returns true
- Evaluate exists condition on (Lisa,222,32K)
  - Subquery returns: empty relation
  - exists returns false
- Evaluate exists condition on (Tom,333,56K)
  - Subquery returns: Lisa
  - exists returns true
- Evaluate exists condition on (Mary,333,65K)
  - Subquery returns: Lisa
  - exists returns true

Emp x Dept

Emp.ename	Emp.dno	Emp.sal	Dept.dno	Dept.dname	Dept.mgr
Jack	111	81K	111	Sells	Alice
Jack	111	81K	222	Toys	Lisa
Jack	111	81K	333	Electronics	Mary
Alice	111	70K	111	Sells	Alice
Alice	111	70K	222	Toys	Lisa
Alice	111	70K	333	Electronics	Mary
Lisa	222	32K	111	Sells	Alice
Lisa	222	32K	222	Toys	Lisa
Lisa	222	32K	333	Electronics	Mary
Jack	333	81K	111	Sells	Alice
Jack	333	81K	222	Toys	Lisa
Jack	333	81K	333	Electronics	Mary
Tom	333	56K	111	Sells	Alice
Tom	333	56K	222	Toys	Lisa
Tom	333	56K	333	Electronics	Mary
Mary	333	65K	111	Sells	Alice
Mary	333	65K	222	Toys	Lisa
Mary	333	65K	333	Electronics	Mary

E1(ename,dno,salary)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	32K
Tom	333	56K
Mary	333	65K

Result

E1.ename
Jack
Alice
Tom
Mary

More on Subqueries

- **Subqueries in the FROM statement**
  - **Example:** "List the department dno, total salary, and number of employees for all departments with more than 1 employee."

```
SELECT dno, SUM(sal), COUNT(ssn)
FROM   Emp
GROUP BY dno
HAVING COUNT(ssn)>1;
```

```
SELECT Temp.dno, Temp.totalSal, Temp.numEmp
FROM   (
       SELECT dno, SUM(sal) as totalSal , COUNT(ename) as numEmp
       FROM   Emp
       GROUP BY dno
 ) AS Temp
WHERE Temp.numEmp > 1;
```

Both queries return the same result.

More on Subqueries

- **Subqueries to aggregate in multiple stages**
  - **Example:** "Find the max total salary amount among all departments."

```
SELECT MAX(Temp.totalSal)
FROM   (
       SELECT dno, SUM(sal) as totalSal
       FROM   Emp
       GROUP BY dno
 ) AS Temp;
```

More on Subqueries

- **Joining Subqueries**
  - **Example:** "Find the employees in the departments whose total salary amount is more than $200K"

```
SELECT Emp.ename
FROM   (
       SELECT dno, SUM(sal) as totalSal
       FROM   Emp
       GROUP BY dno
 ) AS Temp , Emp
WHERE Temp.dno=Emp.dno  AND Temp.totalSal>200000;
```

```
SELECT Emp.ename
FROM   Emp
WHERE Emp.dno IN (
       SELECT dno
       FROM   Emp
       GROUP BY dno
       HAVING SUM(sal) >200000);
```

More Subquery Examples

Query 1: *"Find the employees who work on some project."*

"Find the employees who doesn't work on any projects."

Query 2: *"Find the employee with highest salary"*.

Query 3: *"Find the employees whose salary is greater than overall average salary. "*

"Find the employees whose avg salary is greater than the average salary in the department they work in."

Query 4: *"Find the employees that work on all projects."*

Query 5: *"Find the department names that has the max total salary amount among all departments."*

More Subquery Examples

Suppliers(sid, sname, address)

Parts(pid, pname, color)

Catalog(sid, pid, cost)

Catalog(sid) is a FK referencing Suppliers(sid)

Catalog(pid) is a FK referencing Parts(pid)

Query 1: *"Find the distinct "sids" of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part)"*.

Query 2: *"Find the distinct "sids" of suppliers who supply only red parts"*.

Joins

- Joins are expressed implicitly using SELECT-FROM-WHERE clause.
- Alternatively, joins can be expressed using join expressions.
- Different vendors might have different implementations.

Joins


```
SELECT (column_list)
FROM  table_name
  [INNER | {LEFT |RIGHT | FULL } OUTER] JOIN table_name
    ON <condition>
WHERE …
```

- Explicit join semantics needed unless it is an INNER join
- (INNER is default)

Example Schema


```
CREATE TABLE Sailors (
        sid INTEGER  PRIMARY KEY,
        sname CHAR(20),
        rating INTEGER,
        age INT)

CREATE TABLE Boats (
        bid INTEGER  PRIMARY KEY,
        bname CHAR (20),
        color CHAR(10) )

 CREATE TABLE Reserves (
        sid INTEGER,
        bid INTEGER,
        day DATE,
        PRIMARY KEY (sid, bid, day),
        FOREIGN KEY (sid) REFERENCES Sailors,
        FOREIGN KEY (bid) REFERENCES Boats)
)
```

Inner Join

- Only the rows that match the search conditions are returned.

```
SELECT *
FROM Sailors  INNER JOIN Reserves
ON Sailors.sid = Reserves.sid
```

- Returns only those sailors who have reserved boats
- How many attributes will the result have?

- Equivalent to:

```
SELECT *
FROM Sailors, Reserves
WHERE Sailors.sid = Reserves.sid
```

NATURAL (Inner) Join

- SQL-92 also allows:

```
SELECT *
FROM Sailors NATURAL JOIN Reserves;
```

- "NATURAL" means equi-join for each pair of attributes with the same name
- Drops the duplicate attributes.
- How many attributes will the result have?


```
SELECT *
FROM Sailors INNER JOIN Reserves
ON Sailors.sid = Reserves.sid
```

Sailors(sid,sname,rating,age)

sid	sname	rating	age
22	Dustin	7	45
31	Lubber	8	55
95	Bob	3	60

Reserves(sid,bid,day)

sid	bid	day
22	101	10/10/2012
95	103	11/12/2012

Result

Sailors.sid	sname	rating	age	Reserves.sid	bid	day
22	Dustin	7	45	22	101	10/10/2012
95	Bob	3	60	95	103	11/12/2012

Left Outer Join

- Left Outer Join returns all matched rows, plus all unmatched rows from the table on the left of the join clause
  - use NULL in fields of non-matching tuples

```
SELECT *
FROM Sailors LEFT OUTER JOIN Reserves
ON Sailors.sid = Reserves.sid
```

- Returns all sailors & information on whether or not they have reserved boats


```
SELECT *
FROM Sailors LEFT OUTER JOIN Reserves
ON Sailors.sid = Reserves.sid
```

Sailors(sid,sname,rating,age)

sid	sname	rating	age
22	Dustin	7	45
31	Lubber	8	55
95	Bob	3	60

Reserves(sid,bid,day)

sid	bid	day
22	101	10/10/2012
95	103	11/12/2012

Result

Sailors.sid	sname	rating	age	Reserves.sid	bid	day
22	Dustin	7	45	22	101	10/10/2012
31	Lubber	8	55	NULL	NULL	NULL
95	Bob	3	60	95	103	11/12/2012

Right Outer Join

- Right Outer Join returns all matched rows, plus all unmatched rows from the table on the right of the join clause

```
SELECT *
FROM Reserves RIGHT OUTER JOIN Boats
ON Reserves.bid = Boats.bid
```

- Returns all boats & information on which ones are reserved.


```
SELECT *
FROM Reserves RIGHT OUTER JOIN Boats
ON Reserves.bid = Boats.bid
```

Reserves(sid,bid,day)

sid	bid	day
22	101	10/10/2012
95	103	11/12/2012

Boats(bid,bname,color)

bid	bname	color
101	Interlake1	Blue
102	Interlake2	Red
103	Clipper	Green
104	Marine	Red

Result

sid	Reserves.bid	day	Boats.bid	bname	color
22	101	10/10/2012	101	Interlake1	Blue
NULL	NULL	NULL	102	Interlake2	Red
95	103	11/12/2012	103	Clipper	Green
NULL	NULL	NULL	104	Marine	Red

Full Outer Join

- Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT *
FROM Reserves FULL OUTER JOIN Boats
ON Reserves.bid = Boats.bid
```

How about?


```
SELECT *
FROM Sailors FULL OUTER JOIN Boats
ON Sailors.sname = Boats.bname
```

Sailors(sid,sname,rating,age)

sid	sname	rating	age
22	Dustin	7	45
31	Lubber	8	55
95	Bob	3	60

Boats(bid,bname,color)

bid	bname	color
101	Interlake1	Blue
102	Interlake2	Red
103	Clipper	Green
104	Marine	Red

How about?


```
SELECT *
FROM Sailors
RIGHT OUTER JOIN Reserves ON Sailors.sid = Reserves.sid
LEFT OUTER JOIN Boats ON Reserves.bid = Boats.bid
```

Sailors(sid,sname,rating,age)

sid	sname	rating	age
22	Dustin	7	45
31	Lubber	8	55
95	Bob	3	60

Reserves(sid,bid,day)

sid	bid	day
22	101	10/10/2012
95	103	11/12/2012

Boats(bid,bname,color)

bid	bname	color
101	Interlake1	Blue
102	Interlake2	Red
103	Clipper	Green
104	Marine	Red

Join Summary

- R INNER JOIN S ON <condition>
- R FULL OUTER JOIN S ON <condition>
- R LEFT OUTER JOIN S ON <condition>
- R RIGHT OUTER JOIN S ON <condition>
- R NATURAL JOIN S
- R NATURAL FULL OUTER JOIN S
- R NATURAL LEFT OUTER JOIN S
- R NATURAL RIGHT OUTER JOIN S

Again: Different vendors might have different implementations.