SQL

Instructor: Wang-Chiew Tan

Reference: A First Course in Database Systems, 3rd edition, Chapter 6.2

Meaning of an SQL query with multiple relations in the FROM clause

```
SELECT [DISTINCT] c_1, c_2, ..., c_m
FROM R_1, R_2, ..., R_n
[WHERE condition]
[ORDER BY < list of attributes>] [DESC]
```

Suppose we now have more than 1 relation in the FROM clause.

- · Let Result denote an empty collection.
- For every tuple t_1 from R_1 , t_2 from R_2 , ..., t_n from R_n
 - if t₁, ..., t_n satisfy condition (i.e., condition evaluates to true), then add the resulting tuple that consists of c₁, c₂, ..., c_m components of t into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- If ORDER BY st of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Return Result.

Database schema for our running examples

• Let us assume we have the following database schema with five relation schemas.

Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)

Studio(name, address, presC#)

Products and Joins in SQL

SELECT *

FROM Movies, StarsIn;

| Movies | Title | Year | Length | Genre | studioName | producerC# |
|---------|-----------------|----------|-----------------|-------|------------|------------|
| | Pretty Woman | 1990 | 119 | true | Disney | 999 |
| | Monster's Inc. | 1990 | 121 | true | Dreamworks | 223 |
| | Jurassic Park | 1998 | 145 | NULL | Disney | 675 |
| StarsIn | movieTitle | movieYea | r starName | | | |
| | Pretty Woman | 1990 | Julia Roberts | | | |
| | Monster's Inc. | 1990 | John Goodman | | | |
| | | | | | | |

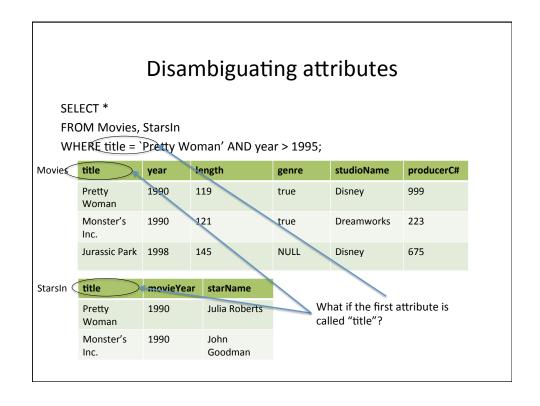
Products and Joins in SQL (cont'd)

SELECT *

FROM Movies, StarsIn

WHERE title = movietitle;

| Movies | title | year | length | genre | studioName | producerC# |
|---------|-----------------|----------|-----------------|-------|------------|------------|
| | Pretty Woman | 1990 | 119 | true | Disney | 999 |
| | Monster's Inc. | 1990 | 121 | true | Dreamworks | 223 |
| | Jurassic Park | 1998 | 145 | NULL | Disney | 675 |
| StarsIn | movieTitle | movieYea | r starName | | | |
| | Pretty Woman | 1990 | Julia Roberts | | | |
| | Monster's Inc. | 1990 | John Goodman | | | |



Disambiguating attributes (cont'd)

SELECT *

FROM Movies, StarsIn

WHERE **StarsIn.title** = `Pretty Woman' AND year > 1995;

| Movies | title | year | length | genre | studioName | producerC# |
|---------|-----------------|----------|-----------------|-------|------------|------------|
| | Pretty Woman | 1990 | 119 | true | Disney | 999 |
| | Monster's Inc. | 1990 | 121 | true | Dreamworks | 223 |
| | Jurassic Park | 1998 | 145 | NULL | Disney | 675 |
| StarsIn | title | movieYea | r starName | | | |
| | Pretty Woman | 1990 | Julia Roberts | | | |
| | Monster's Inc. | 1990 | John Goodman | | | |

Tuple variables

SELECT *

FROM Movies m, StarsIn s

WHERE title = movietitle;

- m and s are tuple variables.
- m binds to a tuple in the Movies relation.
- s binds to a tuple in StarsIn relation.

Self Joins

SELECT *

FROM StarsIn s1, StarsIn s2

WHERE s1.movieYear = s2.movieYear;

StarsIn

| title | movieYear | starName |
|-----------------|-----------|-----------------|
| Pretty Woman | 1990 | Julia Roberts |
| Monster's Inc. | 1990 | John Goodman |

SQL Join Expressions

Reference:

A First Course in Database Systems, 3rd edition, Chapter 6.3.6 – 6.3.8

R(A,B,C) and S(C,D,E)

- R CROSS JOIN S;
 - Product of the two relations R and S.
 - Schema of resulting relation: (R.A,R.B,R.C,S.C,S.D,S.E).
 - Equivalent to:

SELECT *

FROM R, S;

- R JOIN S ON B=D AND A=E;
 - Selects only tuples from R and S where B=D and A=E.
 - Schema of the resulting relation: (R.A,R.B,R.C,S.C,S.D,S.E);
 - Equivalent to:

SELECT *

FROM R, S

WHERE B=D AND A=E;

SQL Join (cont'd)

R(A,B,C) and S(C,D,E) SELECT R.A, R.B, R.C, S.D, S.E FROM R JOIN S ON B=D AND A=E;

- Selects only tuples from R and S where B=D and A=E.
- Schema of the resulting relation: (A,B,C,D,E).

Natural Joins

R(A,B,C) and S(C,D,E)

- R NATURAL JOIN S;
 - Schema of the resulting relation: (A,B,C,D,E)
 - Equivalent to:

SELECT R.A, R.B, R.C, S.D, S.E FROM R,S WHERE R.C = S.C;

Outerjoins

R(A,B,C) and S(C,D,E)

- R NATURAL FULL OUTER JOIN S;
 - Schema of the resulting relation: (A,B,C,D,E).
- R NATURAL LEFT OUTER JOIN S;
 - Schema of the resulting relation: (A,B,C,D,E).
- R NATURAL RIGHT OUTER JOIN S;
 - Schema of the resulting relation: (A,B,C,D,E).
- R FULL OUTER JOIN S ON B = D;
 - Schema of the resulting relation: (A,B,C,D,E).
- R FULL LEFT OUTER JOIN ON B=D;
- R FULL RIGHT OUTER JOIN ON B=D;

Set and Bag Operations in SQL

Reference:

A First Course in Database Systems, 3rd edition, Chapter 6.2.5, 6.4.1, 6.4.2

- Set Union, Set Intersection, Set Difference
- Bag Union, Bag Intersection, Bag Difference
- Other set/bag operations
 - IN, op ANY, op ALL, EXISTS, NOT IN, NOT EXISTS
 - More on these later.

Set Union

R(A,B,C), S(A,B,C)

- Input to union must be union-compatible.
 - R and S have the same set of attributes and the corresponding attributes are of the same type.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the set of all tuples of R and S.

(SELECT * (SELECT * FROM R) FROM R

UNION WHERE A > 10)

(SELECT * UNION

FROM S); (SELECT * FROM S

WHERE B < 300);

Bag Union

R(A,B,C), S(A,B,C)

- Input to union must be *union-compatible*.
 - R and S have the same set of attributes and the corresponding attributes are of the same type.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the collection of all tuples of R and S, including duplicate tuples.

(SELECT * (SELECT * FROM R FROM R) WHERE A > 10)

UNION ALL (SELECT * (SELECT * FROM S); FROM S

WHERE B < 300);

Any difference between these two queries?

(SELECT DISTINCT * (SELECT * FROM R FROM R WHERE A > 10) WHERE A > 10)

UNION ALL
(SELECT DISTINCT * (SELECT * FROM S FROM S

WHERE B < 300); WHERE B < 300);

Intersection, Difference

- Like union, intersection and difference are binary operators.
 - Input to intersection/difference operator consists of two relations R and S and they must be union-compatible.
 - Output has the same type as R or S.
- Set Intersection, Bag Intersection
 - <Query1> INTERSECT <Query2>, <Query1> INTERSECT ALL <Query2>
 - Meaning: Find all tuples that are common to both R and S.
- Set Difference, Bag Difference
 - <Query1> EXCEPT <Query2>, <Query1> EXCEPT ALL <Query2>
 - Find all tuples in R but not in S.
 - EXCEPT: takes the set of Q1 minus the set of Q2.
 - EXCEPT ALL: takes the bag of Q1 minus the bag of Q2.

Difference

 <Query1> EXCEPT <Query2> EXCEPT <Query3>; means (<Query1> EXCEPT <Query2>) EXCEPT <Query3>;

Subqueries

Reference: A First Course in Database Systems,

3rd edition, Chapter 6.3.

- A subquery is a query that is embedded in another query.
- Queries with UNION, INTERSECT, and EXCEPT have two subqueries.
- Subqueries can return used as part of a boolean expression.
 - A subquery returns a constant (or scalar value) which can be compared against another constant in the WHERE clause.
- Subqueries can return relations.
- Subqueries can appear in the FROM clause, followed by a tuple variable, which will represent a tuple in the result of the subquery.

Subqueries that return scalar values

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth)

• Find all names of executives who produced the movie 'Star Wars'.

SELECT name
FROM Movies, MovieExec

SELECT name

WHERE title='Star Wars' AND producerC# = cert#;

Subqueries that return scalar values

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth)

Find all names of executives who produced the movie 'Star Wars'.

FROM Movies, MovieExec
WHERE title='Star Wars' AND producerC# = cert#;

SELECT name
FROM MovieExec
WHERE cert# = (SELECT producerC#
FROM Movies

WHERE title = 'Star Wars');

Subqueries that return relations

SELECT name
FROM MovieExec
WHERE cert# IN (SELECT producerC#
FROM Movies
WHERE title = 'Star Wars');

• IN, NOT IN

Is this query equivalent to the one above?

SELECT name FROM MovieExec, Movies WHERE cert# = producerC# AND title = 'Star Wars';

Subqueries with subqueries

SELECT name

FROM MovieExec

WHERE cert# IN (SELECT producerC#

FROM Movies

WHERE (title, year) IN (SELECT movieTitle, movieYear

FROM StarsIn

WHERE starName = 'Harrison Ford');

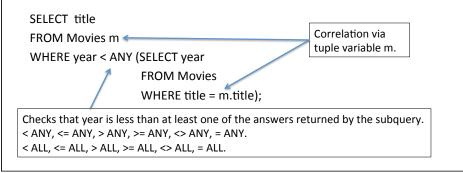
SELECT name

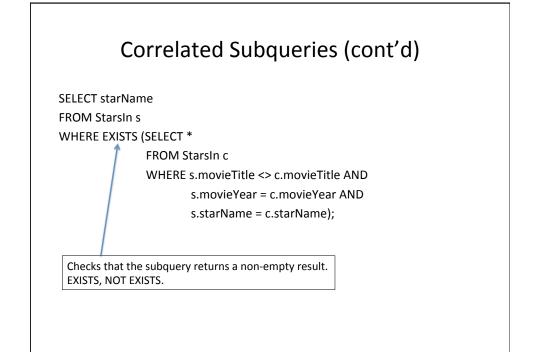
FROM MovieExec, Movies, StarsIn

WHERE cert# = producerC# AND title = movieTitle AND year = movieYear AND starName = 'Harrison Ford';

Correlated Subqueries

- In all the examples so far, the inner query is independent of the outer query.
- An inner query can also depend on the outer query.
- Find all titles of movies that have been used for two or more movies.





Set Comparison Operators

- x IN Q
 - Returns true if x occurs in the collection Q.
- x NOT IN Q
 - Returns true if x does not occur in the collection Q.
- EXISTS Q
 - Returns true if Q is a non-empty collection.
- NOT EXISTS Q
 - Returns true if Q is an empty collection.
- NOT EXISTS (Q1 EXCEPT Q2)
 - Returns true if (Q1-Q2) is an empty collection.

Set Comparison Operators (cont'd)

- *op* ANY, *op* ALL, where *op* is one of { <, <=, >, >=, <>, = }.
- x op ANY Q
 - Returns true if there exists an element y in the result of Q such that x op y is true.
- x op ALL Q
 - Returns true if *for every* element y in Q, we have that x op y is true.

Subqueries in the FROM clause

• Find the names of all movie executives who produce movies that Harrison Ford acted in.

SELECT name

FROM MovieExec m, (SELECT producerC#

FROM Movies, StarsIn

WHERE title = movieTitle AND

year = movieYear AND

starName = 'Harrison Ford') p

WHERE m.cert# = p. producerC#;

Movies(title, year, length, genre, studioName, producerC#) StarsIn(movieTitle, movieYear, starName) MovieExec(name, address, cert#, netWorth)

More examples

Customers

| <u>sid</u> | cname | level | type | age |
|------------|-------|----------|-----------|-----|
| 36 | Cho | Beginner | snowboard | 18 |
| 34 | Luke | Inter | snowboard | 25 |
| 87 | Ice | Advanced | ski | 20 |
| 39 | Paul | Beginner | ski | 33 |

Activities

| <u>sid</u> | <u>slopeid</u> | day |
|------------|----------------|----------|
| 36 | s3 | 01/05/13 |
| 36 | s1 | 01/06/13 |
| 36 | s1 | 01/07/13 |
| 87 | s2 | 01/07/13 |
| 87 | s1 | 01/07/13 |
| 34 | s2 | 01/05/13 |

Slopes

| slopeid | name | color |
|---------|--------------|-------|
| s1 | Mountain Run | blue |
| s2 | Olympic Lady | black |
| s3 | Magic Carpet | green |
| s4 | KT-22 | black |

• Find the names of customers who went on some slope on 01/07/13.

SELECT c.cname FROM Customers c, Activities a WHERE a.day='01/07/13' AND a.sid = c.sid;

SELECT cname
FROM Customers c
WHERE c.sid IN (SELECT a.sid
FROM Activities a
WHERE a.day='01/07/13');

Example 2

• Find the names of customers who did not go on any slope on 01/07/13.

SELECT cname
FROM Customers c
WHERE c.sid NOT IN (SELECT a.sid
FROM Activities a
WHERE a.day='01/07/13');

• Find all names of customers who went on the slope "Olympic Lady" on 01/07/13.

SELECT c.cname

FROM Customers c

WHERE c.id IN (SELECT a.id

FROM Activities a

WHERE a.day='01/07/13' AND a.slopeid IN (SELECT s.slopeid

FROM Slopes s

WHERE s.name='Olympic lady');

Example 4

• Determine the colors of all slopes that Cho went.

SELECT s.color
FROM Activities a, Slopes s
WHERE a.slope-id = s.slope-id AND
a.id = (SELECT id
FROM Customers
WHERE cname='Cho');

• Find the names of customers who have been on some slope on the day 01/07/13.

SELECT c.cname
FROM Customers c
WHERE EXISTS (SELECT *
FROM Activities a
WHERE a.sid = c.sid AND a.day='01/07/13');

Example 6

• Find the names of all customers who went on some slope.

SELECT c.cname
FROM Customers c
WHERE c.sid = ANY (SELECT sid
FROM Activities a);

 Find the names of all skiers whose age is greater than every snowboarder.

SELECT c.name FROM Customers c

What happens if this subquery returns an empty set?

WHERE c.type='skier' AND c.age >ALL (SELECT c.age

FROM Customers c

WHERE c.type = 'snowboard');

· Find the names of the oldest customers.

SELECT c.cname

FROM Customer c

WHERE c.age >= ALL (SELECT c.age

FROM Customers c);

Practice homework 3

The example database records information on bars, customers, beers, and the associations among them.

- Beers(name, manf): stores information about beers, including the manufacturer of each beer.
- Bars(name, city, addr, license, phone): stores information about bars including their city, street address, phone number and their operating license
- Drinkers(name, city, addr, phone): stores information about drinkers, including their city, street address and phone number.
- Likes (drinker, beer): indicates which drinker likes which beers (note that a drinker may like many beers and many drinkers may like the same beer).
- Sells (bar, beer, price): indicates the price of each beer sold at each bar (note that each bar can sell many beers and many bars can sell the same beer, at possibly different prices).
- Frequents (drinker, bar): indicates which drinker frequents which bars (note that each drinker may frequent many bars and many drinkers may frequent the same bar).

| 1. | Find the names of all beers, and their prices, served by the bar 'Blue Angel'. |
|----|---|
| 2. | Find the name and phone number of every drinker who likes the beer 'Budweiser'. |
| 3. | Find all bars frequented by both 'Vince' and 'Herb'. |
| 4. | Find all bars in 'Chicago' (and display all attributes) for which we know either the address (i.e., addr in our schema) or the phone number but not both. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | END |
| | |
| | |
| | |
| | |