SQL (Part 2)

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems, 3rd edition, Chapter 6.2, 6.3, 6.4

Important Notices

- Lab2 assignment was posted on Piazza under Resources→Lab2 on Sunday, April 14.
  - Lab2 will be discussed at Lab Sections.
  - Due **Sunday, April 28**, by 11:59pm.
    - Late Lab Assignments will not be accepted.
    - Be sure that you post the correct file!
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas will be used for both Lab submission and grading.
  - Load file for Lab2 will be posted this week.
    - Helps with testing, but we won't post query solutions.
  - Lab1 solution has been posted on Piazza.
- The **second** Gradiance Assignment, "CMPS 182 Spring 2019 #2", has been assigned, and is due **Friday, April 26**, by 11:59pm.

Meaning of an SQL Query with Multiple Relations in the FROM Clause

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY < list of attributes [ASC | DESC] >]

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

- Let Result begin as an empty multiset of tuples.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if t_1, …, t_n satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, …, c_m components of *t* into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Return Result.

Database Schema for Some Examples

- Let us assume we have the following database schema with five relation schemas, with Primary Keys underlined.

  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#)

Cartesian Product 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	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman

Join in SQL

```
SELECT *
FROM Movies, StarsIn
WHERE title = movieTitle AND year = movieYear;
```

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

Disambiguating Attributes

```
SELECT *
FROM Movies, StarsIn
WHERE title = 'Pretty Woman' AND movieYear <= 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

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Disambiguating Attributes

```
SELECT *
FROM Movies, StarsIn
WHERE title = 'Pretty Woman' AND movieYear <= 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	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

But what if the first attribute of StarsIn was <u>also</u> called "title"?

Disambiguating Attributes (cont'd)

```
SELECT *
FROM Movies, StarsIn
WHERE Movies.title = 'Pretty Woman' AND movieYear <= 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	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Tuple Variables

What if the movieTitle attribute of StarsIn was called title?

```
SELECT *
FROM Movies, StarsIn
WHERE movietitle = title;
```

```
SELECT *
FROM Movies m, StarsIn s
WHERE m.title = s.title;
```

m and s are tuple variables.

- m binds to a tuple (row) in the Movies relation.
- s binds to a tuple (row) in StarsIn relation.
- *Could also write Movies.title = StarsIn.title, and not bother having the tuple variables m and s.*
- But we'll frequently use tuple variables for clarity and brevity.

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

Database Schema for Some Examples

- Let us assume we have the following database schema with five relation schemas, with primary keys underlined.

  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#)

Some Example Join Queries
(We’ll Write Answers on the Board)

1. Who were the male stars in the 2009 movie Avatar?
2. Which stars appeared in movies produced by MGM in 2016?
3. Who is the president of MGM?
4. Which movies are longer than the 2009 movie Avatar?

SQL Join Expressions

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

- JOIN
  - ON
  - CROSS JOIN
- NATURAL JOIN
- Outer Joins
  - (FULL) OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- All of these can appear in the FROM clause
  - We'll briefly discuss all, except OUTER JOIN, briefly now.
    - NATURAL JOIN will reappear later in the course.
  - OUTER JOIN will be discussed later in the course.

JOIN … ON …

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 **JOIN** S **ON** R.B=S.D **AND** R.A=S.E;
  - Selects only tuples from R and S where R.B=S.D and R.A=S.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 R.B=S.D AND R.A=S.E;
```

CROSS JOIN

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;
```

NATURAL JOIN

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;
```

  - Note that attribute C only appears once!

Set and Bag Operations in SQL

- Set Union, Set Intersection, Set Difference
- Bag Union, Bag Intersection, Bag Difference

- Other set/bag operations
  - IN, NOT IN, op ANY, op ALL, EXISTS, NOT EXISTS
  - More on these operations later in this lecture

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

Set Union

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

- Input to union must be *union-compatible*.
  - R and S must have attributes of the same types in the same order.
- Output of Union has the same schema as R or S.
- Meaning: Output consists of the **set** of all tuples from R and from S.
  - UNION could (should??) have been called UNION DISTINCT

```
(SELECT *
FROM R)
UNION
(SELECT *
FROM S);
```

```
(SELECT *
FROM R
WHERE A > 10)
UNION
(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 must have attributes of the same types in the same order.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the collection of all tuples from R and from S, including duplicate tuples.
  - *Subtlety: Attributes/column names may be different; R's are used.*

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

```
(SELECT *
FROM R
WHERE A > 10)
UNION ALL
(SELECT *
FROM S
WHERE B < 300);
```

Are These Two Queries Always Equivalent?

That is, Do They Always Have Same Result?

Assume that R(A,B,C) and S(A,B,C) are Union-Compatible

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

UNION ALL

(SELECT DISTINCT *
FROM S
WHERE B < 300);
```

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

UNION

(SELECT *
FROM S
WHERE B < 300);
```

Intersection; Difference (Except)

- 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>
  - Find all tuples that are in the results of both Query1 and Query2.

- Set Difference, Bag Difference
  - <Query1> **EXCEPT** <Query2>, <Query1> **EXCEPT ALL** <Query2>
  - Find all tuples that are in the result of Query1, but not in the result of Query2.

Operator Precedence

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

Order of operations originally was: UNION, INTERSECT and EXCEPT have the same priority, and are executed left-to-right.

But this changed in the SQL standard, and has changed in most implementations! Now, INTERSECT has a higher priority than UNION and EXCEPT (just as * has a higher priority than + and -) , so:

Query1 UNION Query2 INTERSECT Query3

would be executed as:

Query1 UNION (Query2 INTERSECT Query3)

not as:

(Query1 UNION Query2) INTERSECT 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.
  - Note that queries with UNION, INTERSECT, and EXCEPT have two subqueries.
- A subquery can return a constant (scalar value) which can be compared against another constant in the WHERE clause, or can be used in a boolean expression.
- A subquery can also return a relation.
- A subquery returning a relation can appear in the FROM clause, followed by a tuple variable that refers to the tuples in the result of the subquery
  - … just as a tuple variable can be used to refer to a table.

Subqueries that Return Scalar Values

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

- Find names of executives who produced the movie ‘Star Wars’.
  - Careful: Don’t write query the second way--possible runtime error!

```
SELECT e.name
FROM Movies m, MovieExec e
WHERE m.title=‘Star Wars’ AND m.producerC# = e.cert#;
```

Outer query

```
SELECT e.name
FROM MovieExec e
WHERE e.cert# = (SELECT m.producerC#
                 FROM Movies m
                 WHERE m.title = ‘Star Wars’);
```

Inner query

Subqueries that Return Relations

```
SELECT e.name
FROM MovieExec e
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM Movies m
                   WHERE m.title = ‘Star Wars’);
```

- **IN, NOT IN**

Is this query **equivalent** to the one above?

```
SELECT e.name
FROM MovieExec e, Movies m
WHERE m.title = ‘Star Wars’
   AND m.producerC# = e.cert#;
```

Putting a Subquery that Returns a Relation into the FROM Clause

```
SELECT e.name
FROM MovieExec e,
                        (SELECT m.producerC#
                         FROM Movies m
                         WHERE m.title = ‘Star Wars’) p
 WHERE e.cert# = p.producerC# ;
```

Is this query equivalent to the one above?

```
SELECT e.name
FROM MovieExec e, Movies m
WHERE e.cert# = m.producerC#
    AND m.title = ‘Star Wars’;
```

Subqueries with Subqueries

What does this query do? **(Assume that MovieExec.name is UNIQUE.)**

```
SELECT e.name
FROM MovieExec e
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM Movies m
                   WHERE (m.title, m.year) IN (SELECT s.movieTitle, s.movieYear
                                                FROM StarsIn s
                                                WHERE s.starName = 'Harrison Ford')
                   );
```

Is this query equivalent? (Do tuple variables make queries clearer?)

```
SELECT e.name
FROM MovieExec e, Movies m, StarsIn s
WHERE e.cert# = m.producerC# AND m.title = s.movieTitle
   AND m.year = s.movieYear AND s.starName = 'Harrison Ford';
```

Correlated Subqueries

- In all the examples so far, the inner query has been <u>independent</u> of the outer query.
- An inner query can also <u>depend on</u> attributes in the outer query; that's called **correlation**.
- Find the movie titles that have been used for two or more movies.

```
SELECT DISTINCT m.title
FROM Movies m
WHERE m.year < ANY (SELECT m2.year
                    FROM Movies m2
                    WHERE m2.title = m.title);
```

DISTINCT needed because title may been been used for three or more movies!

Correlation via tuple variable m

Checks that year is less than at least one of the answers returned by the subquery.
< ANY, <= ANY, > ANY, >= ANY, <> ANY, = ANY
< ALL, <= ALL, > ALL, >= ALL, <> ALL, = ALL

Correlated Subqueries (cont'd)

```
SELECT s.starName
FROM StarsIn s
WHERE EXISTS (SELECT *
              FROM StarsIn c
              WHERE s.movieTitle <> c.movieTitle AND
                    s.movieYear = c.movieYear AND
                    s.starName = c.starName);
```

Checks that the subquery returns a non-empty result.
Can write EXISTS or NOT EXISTS.

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.

Set Comparison Operators (cont'd)

- *x op* ANY Q, x *op* ALL Q in WHERE clause
  - x is a scalar expression
  - Q is a SQL query
  - *op* is one of { <, <=, >, >=, <>, = }.
- x *op* ANY Q
  - What does this mean?
  - SOME can be used instead of ANY
- x op ALL Q
  - What does this mean?

Subqueries in the FROM Clause

- Find the names of all movie executives who produced movies that Harrison Ford acted in.

```
SELECT e.name
FROM MovieExec e, (SELECT m.producerC#
                   FROM Movies m, StarsIn s
                   WHERE m.title = s.movieTitle AND
                         m.year = s.movieYear AND
                         s.starName = ‘Harrison Ford’) p
WHERE e.cert# = p.producerC#;
```

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

Ski Activity Examples

Customers

cid	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

cid	slopeid	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

Example 1

- Find the names of customers who did some activity on 01/07/13.

  Please try to write query yourselves.

Example 1

- Find the id and name of each customer who did some activity on 01/07/13.

```
SELECT c.cid, c.cname
FROM Customers c, Activities a
WHERE a.day = DATE ‘01/07/13’ AND a.cid = c.cid;
```

Wrong!

```
SELECT c.cid, c.cname
FROM Customers c
WHERE c.cid IN (SELECT a.cid
                FROM Activities a
                WHERE a.day = DATE ‘01/07/13’);
```

Example 2

- Find the id and name of each customer who did not do any activity on 01/07/13.

```
SELECT c.cid, c.cname
FROM Customers c
WHERE c.cid NOT IN (SELECT a.cid
                    FROM Activities a
                    WHERE a.day = DATE ‘01/07/13’);
```

Example 3

- Find the id and name of each customer who went on the slope “Olympic Lady” on 01/07/13.

```
SELECT c.cid, c.cname
FROM   Customers c
WHERE c.cid IN (SELECT a.cid
                FROM Activities a
                WHERE a.day = DATE ‘01/07/13’ AND a.slopeid IN ( SELECT s.slopeid
                                                                 FROM Slopes s
                                                                 WHERE s.name=‘Olympic Lady’)
                 );
```

Could you also rewrite this as a join, with SELECT DISTINCT?

Example 3 (Equivalent?)

- Find the id and name of each customer who went on the slope “Olympic Lady” on 01/07/13.

```
SELECT DISTINCT c.cid, c.cname
FROM   Customers c, Activities a, Slopes s
WHERE c.cid = a.cid
   AND a.day = DATE ‘01/07/13’
   AND s.name=‘Olympic Lady’
   AND a.slopeid = s.slopeid;
```

Is this equivalent to the query on previous slide?

Example 4

- Determine the colors of all slopes that Cho went on.

Please try to write query yourselves.

Example 4

- Determine the colors of all slopes that Cho went on.

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

  - Would this be correct, if cname **is UNIQUE**?
  - Would this be correct, if cname **is not UNIQUE**?
  - How might you fix this, if cname is not UNIQUE?

Example 5

- Find the id and name of each customer who did some activity on the day 01/07/13.
  - Note: This is same problem as Example 1, but a different query.

```
SELECT c.cid, c.cname
FROM Customers c
WHERE EXISTS (SELECT *
                        FROM Activities a
                        WHERE a.cid = c.cid AND a.day = DATE ‘01/07/13’);
```

Example 6

- Find the names of all customers who did some activity.

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

Example 7

- Find the names of all customers who are skiers and whose age is greater than the age of every snowboarder.

```
SELECT c.cname
FROM Customers c
WHERE c.type='skier' AND c.age  > ALL (SELECT c2.age
                                       FROM Customers c2
                                       WHERE c2.type = 'snowboard');
```

What happens if this subquery returns an empty set?

- Find the names of the oldest customers.

  Please try to write this final query yourselves.

Practice Homework 3: Schema

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.)

Practice Homework 3: Queries

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 the names of 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.