UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - SQL: Queries, Subqueries, and Aggregation

illinois.edu

Announcements

- HW1 is due tonight
- MP1 will be released tonight

Review

- How did we extend the operands of relational algebra?
- How did we extend the operations?
- Aggregation operations require which other operator?
- What does SQL stand for?

Review

- What clause (part) of a SQL query corresponds to the projection operator?
- What clause corresponds to selection?
- What clause specifies the relation(s) we are operating on?

RA -> SQL

$$\pi_{a,b,c}(\sigma_{a=b}(A \times B))$$

RA -> SQL

$$\pi_{a,b,c}(\sigma_{a=b}(A \times B))$$

```
SELECT a, b, c
FROM A,B
WHERE a=b;
```

SQL->RA

```
SELECT (a+b)*c AS value
FROM A,B,C
WHERE (a <> b OR b=c) AND c=d;
```

SQL->RA

```
SELECT (a+b)*c AS value
FROM A,B,C
WHERE (a <> b OR b=c) AND c=d;
```

$$\pi_{(a+b)*c \rightarrow value}\left(\sigma_{(a \neq b \ OR \ b=c) AND \ c=d}(A \times B \times C)\right)$$

Writing queries

- Write a query to find the title of all songs with 5 or more letters.

Writing queries

```
SELECT songTitle
FROM Song
WHERE songTitle LIKE '_ _ _ _ _ %';
```

Where were we…

- We learned SELECT, FROM, and WHERE clauses
- We learned some things we can do in SELECT and WHERE clauses
  - renaming attributes, evaluating expressions, pattern matching, etc.
- Let's learn another clause

Sorting

- We can get result of our query sorted using ORDER BY
- Can specify ASC or DESC if we would like the list sorted in ascending or descending order
  - Ascending is the default

Example

```
SELECT *
FROM Song
ORDER BY length DESC;
```

SongTitle	AlbumTitle	Length
Siva	Gish	4:21
Feel The Pain	Without a Sound	4:18
Lithium	Nevermind	4:17
Breed	Nevermind	3:03

Combining relations

- We can specify more than one table in the FROM clause
- This will join tuples of both tables
  - similar to a Cartesian product of two relations
- Can use dot operator (period) to refer to specific attributes

Example

```
SELECT Album.albumTitle, bandName, length
FROM Song, Album
WHERE
    Song.albumTitle=Album.albumTitle AND
    Album.yearReleased>=1980 AND
    bandName LIKE 'N%i%l%';
```

Album.albumTitle	bandName	length
In the Aeroplane Over the Sea	Neutral Milk Hotel	4:26
High Violet	National	3:25

Combining relations

- We can specify more than one table in the FROM clause
  - We can very easily rename the relations by specifying the name
  - Technically these are “tuple variables”

Example

```
SELECT A1.albumTitle, A2.albumTitle
FROM Album A1, Album A2
Where A1.albumTitle<>A2.albumTitle
```

A1.albumTitle	A1.albumTitle
In the Aeroplane Over the Sea	High Violet
In the Aeroplane Over the Sea	In the Airplane Over the Sea
High Violet	In the Aeroplane Over the Sea
High Violet	Oh, Inverted World
Oh, Inverted World	High Violet
Oh, Inverted World	In the Airplane Over the Sea

Joins

- Specified in the FROM clause
  - CROSS JOIN => cross product
  - JOIN ON => theta join
  - NATURAL JOIN => natural join
  - OUTER JOIN => outer joins
    - can be combined with "LEFT" and "RIGHT" keywords, along with "ON" and "NATURAL" to produce many variations

Example

```
SELECT albumTitle, bandName, length
FROM Song NATURAL JOIN Song
WHERE
    Album.yearReleased>=1980 AND
    bandName LIKE 'N%i%l%';
```

Album.albumTitle	bandName	length
In the Aeroplane Over the Sea	Neutral Milk Hotel	4:26
High Violet	National	3:25

Example

name
Ke$ha
Madonna
Prince

```
SELECT Musician.firstName as name
FROM
    Musician LEFT OUTER JOIN Band  ON
        Band.name=Musician.firstName AND
        Musician.lastName  NOT LIKE '_%'
WHERE Band.genre='Pop'
ORDER BY Musician.firstName
```

Set Operations

- Can be specified between two queries
  - UNION
  - INTERCETION
  - EXCEPT (difference)

Example

name
Wilco
Cults

```
(SELECT bandName as name FROM Band)
    INTERSECT
(
    (SELECT albumName as name FROM Album)
        UNION
    (Select songName as name FROM Song)
);
```

Subqueries

- There are ways we can build queries using other queries as components
- Technically, we've already been using them
  - Joins are subqueries
  - Set operations used subqueries

Example

```
SELECT bandName
FROM
    (Band NATURAL JOIN Song)
        EXCEPT
    (Band NATURAL JOIN
        (SELECT * FROM Song WHERE Song>3:00)
    );
```

Subqueries

- We can also use subqueries in WHERE clause
  - Use subquery to get a single scalar value for a comparison
  - Use subquery for comparing against an entire relation

Example

```
SELECT city
FROM Band
WHERE BandName=
    (SELECT Band
    FROM Album
    WHERE title="Strange Mercy");
```

Example

```
SELECT city
FROM Band
WHERE BandName=
    ('St. Vincent');
```

Relation Conditions

- We have several keywords to compare a tuple against an entire relation
  - EXISTS - the subquery returned a tuple
  - IN / NOT in - our tuple appears in the subquery
  - (condition) ALL - our tuple must satisfy the condition for all tuples in the subquery
  - (condition) ANY - our tuple must satisfy the condition for some tuple in the subquery

Example

```
SELECT city
FROM Band NATURAL JOIN Musician
WHERE firstName,lastName IN(
    SELECT firstName,lastName
    FROM Musician
    WHERE instrument LIKE '%bass%');
```

EXAMPLE

```
SELECT bandName
FROM Band
WHERE dateFormed < ALL (
    SELECT dateReleased
    FROM Album
    WHERE albumTitle LIKE 'A%');
```

EXAMPLE

```
SELECT albumTitle
FROM Album NATURAL JOIN Song
WHERE songTitle < ANY(
    SELECT bandName
    FROM Band
    WHERE bandName IN (
        SELECT albumTitle FROM album));
```

Correlated Subquery

- Each inner query can be reevaluated for each tuple in the outer query
  - Can make the inner query dependent on the value of the attributes from the outer tuple

Example

```
SELECT A1.albumTitle
FROM Album A1
WHERE price >= ALL (
    SELECT price
    FROM Album
    WHERE dateReleased = A1.dateReleased
);
```

Duplicate Elimination

- We can eliminate duplicates using the DISTINCT keyword to SELECT clause
- Two notes:
  - Duplicate elimination is expensive!
  - Duplicate elimination is automatically performed by set operations unless ALL keyword is specified

Example

```
SELECT DISTINCT firstName
FROM Musician
```

Example

```
(SELECT bandName as name FROM Band)
    UNION ALL
(SELECT albumTitle as name FROM
Album)
    UNION ALL
(SELECT songTitle as name FROM Song)
```

Aggregation

- We can aggregate on values in the SELECT clause
- Aggregation keywords:
  - COUNT, SUM, AVG, MIN, MAX
- COUNT will return total number of occurrences unless we use DISTINCT keyword

Example

```
SELECT COUNT(firstName)
FROM Musician
```

```
SELECT COUNT(DISTINCT firstName)
FROM Musician
```

Grouping

- Grouping in SQL is its own clause
- GROUP BY
  - occurs after the WHERE clause

EXAMPLE

```
SELECT bandName, SUM(price)
FROM Album
GROUP BY bandName;
```

One last clause…

- HAVING allows us to create conditions based on aggregation
- Example: HAVING MIN(Salary)<125,000

Example

```
SELECT albumName,SUM(length)
FROM Album NATURAL JOIN Song
GROUP BY albumName
WHERE Price<100
HAVING COUNT(songTitle)<5
```

Writing SQL Queries

1. Find all distinct pairs of songs from the same album
   - ('Yesterday', 'Another Girl') is not distinct from ('Another Girl', 'Yesterday')
2. Find the total number of songs written by bands from London

Writing SQL Queries 1

```
SELECT DISTINCT S1.name,S2.name
FROM Song S1, Song S2
WHERE S1.name<S2.name
```

Writing SQL Queries 2

```
SELECT COUNT(DISTINCT songTitle)
FROM Song NATURAL JOIN Album
    NATURAL JOIN Band
WHERE city='London'
```

More Queries

1. Find the total price for all albums that have at three or more songs with at least two words in the title written by bands from London
2. Find an alphabetical list of all the 80's pop songs written by drummers or bassists with the same name as their band (e.g. Phil Collins and Sting)

More queries

3. Find all pairs of artists who were in bands together for at least 3 decades and wrote more than 50 songs.
4. Find all albums released in the 70's having 12 or more songs that were written by bands in which two musicians have played in other bands.

Big picture

- After today, we should have
  - a theoretical understanding of queries
  - a practical understanding of queries
- Given a good database schema, we can answer complex questions
- Designing a database schema will be our next topic