COMP353 Databases

More on SQL Queries

SQL Queries: Review

- SQL query has a form

  SELECT . . .

  FROM . . .

  WHERE . . . ;

- The **SELECT** clause says which **A**ttribute(s) we are interested in
- The **FROM** clause says which **R**elation(s) we refer to
- The **WHERE** clause says which **T**uple(s) we refer to

Case Insensitivity

- SQL is *case insensitive*
- So, keyword FROM maybe written as:
  - FROM or
  - From or
  - FrOm
- Only in ***strings,*** SQL distinguishes between uppercase and the lowercase letters
  - So, the following are different strings:
    - ’FROM’
    - ’From’
    - ’FrOm’

Select Clause

- In place of * in the **SELECT** clause, we can put any attribute “we wish” to project on
- In the **SELECT** clause, we can also do **renaming**

```
SELECT title AS name, length AS duration
FROM Movie;
```

➔ the structure of the query output “*appears as*”: (name, duration)

Select Clause

- We can also use a *formula* in place of an attribute

```
SELECT title AS name, length/60 AS lengthInHours
FROM Movie;
```

➔ the structure of the output: (name, lengthInHours)

Select Clause

- SQL even allows using a constant as an item in the **SELECT** clause, as shown below.

```
SELECT title AS name, length/60 AS length, ’hrs.’ AS inHours
FROM Movie;
```

```
➔The structure of the output: (name,                length,   inHours)
                              Gone with the wind   1.98      hrs.
                              King Kong            0.75      hrs.
```

- Why? To put some “useful” words into the output that SQL displays

Comparison of Strings

- Two strings are equal if they have the same sequence of characters/symbols
- Strings are compared alphabetically
  - ’**fo**dder’ < ’**foo**’
  - ’**bar**’ < ’**bar**gain’
- **WHERE** R.A = T.B **AND** **s LIKE p**
  - **s** is an attribute of type *string* and **p** is a pattern; e.g. WHERE title LIKE ’Gone%’
  - “*Ordinary*” characters in **p** matches ordinary characters in **s**
  - What about “*Special*” characters in **p**: % **,** _
  - “%” in **p** matches **any** sequence of zero or more characters in **s**
  - “_” in **p** matches any **one** character in **s**

Comparison of Strings

- Suppose we remember a movie “Star *something*”, and we do remember that “the something” has four letters

```
SELECT title
FROM Movie
WHERE title LIKE ’Star _ _ _ _’;
```

```
SELECT title
FROM Movie
WHERE title LIKE ’Star%’;
```

Comparison of Strings

What if the pattern p includes ’, %, or _?

- Find all movies with a possessive (’s) in their title

```
LIKE ’%’s%’
```

```
SELECT title
FROM Movie
WHERE title LIKE ’%’’s%’;
```

  - The convention is that two apostrophes ’’ in a string represent **one** single apostrophe (’), and not the end of string

Comparison of Strings

- What if **p** involves the *special characters* % or _?
  - We should "escape" their special meaning using "some" **escape character**
  - **SQL** allows using *any* character as escape character
- s **LIKE** 'x%%x%' **ESCAPE** 'x';
  - Here, x is the escape character → x% means the character %, and not its usual meaning (the special character)
  - The pattern 'x%%x%' matches strings: %whatever%

Ordering the Ouput

- We may wish the output of a query to be displayed in some order. This could be done using the SQL clause:
  - **ORDER BY** <list of attributes>
- E.g., List Disney movies in 1990 by their length, shortest first, and then by the alphabetical order of the titles:

```
SELECT *
FROM Movie
WHERE studioName = 'Disney' AND year = 1990
ORDER BY length, title;
```

  - Default ordering is ASCending, unless we use the **DESC** keyword
  - Ties are broken by the “next” attribute in the **ORDER BY** list.

Products and Joins

- SQL has a simple way to couple relations in a query
  - How? Simply list each relation in the **FROM** clause
- All the relations in the **FROM** clause are coupled through **Cartesian product**
- Then we can put conditions in the **WHERE** clause in order to get a desired kind of **join**

Join (Example, Recall)

- Relation schemas:

  Movie (title, year, length, filmType)

  Owns (title, year, studioName)

- Query:

  Find titles and lengths of all movies produced by Disney

- Query in SQL:

```
SELECT Movie.title, Movie.length
FROM Movie, Owns
WHERE Movie.title = Owns.title AND Movie.year = Owns.year
            AND Owns.studioName = ’Disney’;
```

Union, Intersection, and Difference

- We can apply the common set operations of **union, intersection,** and **difference** to relations **R** and **S,** if they are *compatible*.
- When the output of two or more SQL queries are compatible, we may "combine" the queries using:
  - **UNION**
  - **INTERSECT**
  - **EXCEPT** (or **MINUS** in Oracle)

Union, Intersection, and Difference

- Relation schemas:

  Movie (title, year, length, filmType)

  StarsIn (title, year, starName)

- Query:

  Find titles and years of movies that appeared in either **Movie** or **StarsIn** relations

- Query in SQL:

```
SELECT title, year
FROM Movie
      UNION
SELECT title, year
FROM StarsIn;
```

Union, Intersection, and Difference

- Relation schemas:

  Star(name, address, gender, birthdate**)**

  Exec(name, address, cert#, netWorth)

- Query:

  Find names and addresses of all female movie stars who are also movie executives with a net worth of over $10,000,000

- Query in SQL:

```
SELECT name, address
FROM Star
WHERE gender = 'F'
    INTERSECT
SELECT name, address
FROM Exec
WHERE netWorth > 10000000;
```

Union, Intersection, and Difference

- Relation schemas:

  Star (name, address, gender, birthdate**)**

  Exec (name, address, cert#, netWorth)

- Query:

  Find names and addresses of movie stars who are **not** movie executives

- Query in SQL:

```
SELECT name, address
 FROM Star
      EXCEPT                  //or MINUS in Oracle//
SELECT name, address
 FROM Exec;
```

Duplicate Elimination

Note that in SQL:

- The **union**, **intersection**, and **difference** operations normally eliminate duplicates (the set semantics)
- To retain duplicates, hence preventing duplicate elimination, we must use the keyword **ALL** after the operator **UNION**, **INTERSECT**, and **EXCEPT**
  - *R* **UNION** **ALL** *S (the only bag operation supported in Oracle)*
  - *R* **INTERSECT** **ALL** *S*
  - *R* **EXCEPT** **ALL** *S*

Retaining Duplicates

- ***R* UNION ALL *S***
  - The bag of elements that are in **R**, **S**, or in both. If **R** is a bag in which tuple *t* appears ***n*** times, and **S** is a bag in which *t* appears ***m*** times, then the number of occurrences of tuple *t* in bag **R ∪ S** is ***n* + *m***
- ***R* INTERSECT ALL *S***
  - The bag of elements that are in both **R** and **S.** If **R** is a bag in which tuple *t* appears ***n*** times, and **S** is a bag in which *t* appears ***m*** times, then the number of occurrences of *t* in bag **R ∩ S** is *min*(***n*,*m***)
- ***R* EXCEPT ALL *S***
  - The bag of elements that are in **R** but not in **S.** If **R** is a bag in which tuple *t* appears ***n*** times, and **S** is a bag in which *t* appears ***m*** times, then the number of occurrences of *t* in bag **R – S** is *max*(**0, *n* - *m***)

Retaining Duplicates in Union

- Relation schemas:

  Movie (title, year, length, filmType)

  StarsIn (title, year, starName)

- Query:

  List the title and year of every movie that appears in **Movie** or **StarsIn**

- Query in SQL:

```
SELECT title, year
FROM Movie
        UNION ALL
 SELECT title, year
 FROM StarsIn;
```