SQL

CS 564- Fall 2015

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

MOTIVATION

- The most widely used database language
- Used to query and manipulate data

- SQL stands for **S**tructured **Q**uery **L**anguage
  - many SQL standards: SQL-92, SQL:1999, SQL:2011
  - vendors support different subsets
  - we will discuss the common functionality

Basic SQL Query

optional

SELECT [DISTINCT] attributes
FROM one or more tables
WHERE conditions on the tables

conditions of the form: Attr1 **op** Attr2

Example Database

City (<u>ID</u>, Name, CountryCode, District, Population)

CountryLanguage (<u>CountryCode</u>, Language, IsOfficial, Percentage)

Country (<u>Code</u>, Name, Continent, Region, SurfaceArea, IndepYear, Population, LifeExpectancy, GNP, GNPOld, LocalName, GovernmentForm, HeadOfState, Capital, Code2)

EXAMPLE

What is the population of USA?

```
SELECT Population
FROM Country
WHERE Code = 'USA';
```

SEMANTICS

1. Think of a *tuple variable* ranging over each tuple of the relation mentioned in **FROM**
2. Check if the current tuple satisfies the **WHERE** clause
3. If so, compute the attributes or expressions of the **SELECT** clause using this tuple

* In Select Clauses

When there is one relation in the **FROM** clause, * in the **SELECT** clause stands for "all attributes of this relation"

```
SELECT *
FROM City
WHERE Population >= '1000000'
AND CountryCode = 'USA';
```

Renaming Attributes

If we want the output schema to have different attribute names, we can use **AS** <new name> to rename an attribute

```
SELECT Name AS LargeUSACity
FROM City
WHERE Population >= '1000000'
AND CountryCode = 'USA';
```

Arithmetic Expressions

We can use any arithmetic expression (that makes sense) in the **SELECT** clause

```
SELECT Name,
 (Population/ 1000000) AS PopulationInMillion
FROM City
WHERE Population >= '1000000’ ;
```

What Can We Use in Where Clauses?

- attribute names of the relation(s) used in the **FROM** clause
- comparison operators: =, <>, <, >, <=, >=
- arithmetic operations
- **AND**, **OR**, **NOT** to combine conditions
- operations on strings (e.g. concatenation)
- pattern matching: s **LIKE** p
- special stuff for comparing dates and times

PATTERN MATCHING

s **LIKE** p: pattern matching on strings

- % = any sequence of characters
- _ = any single character

```
SELECT Name, GovernmentForm
FROM Country
WHERE GovernmentForm LIKE '%Monarchy%';
```

Using Distinct

- The default semantics of SQL is bag semantics
- The use of **DISTINCT** in the **SELECT** clause removes all duplicate tuples in the result, and returns a set

```
SELECT DISTINCT GovernmentForm
FROM Country;
```

ORDER BY

The use of **ORDER BY** orders the tuples by the attribute we specify in decreasing (**DESC**) or increasing (**ASC**) order

```
SELECT Name, (Population / 1000000) AS
PopulationInMillion
FROM City
WHERE Population >= '5000000’
ORDER BY PopulationInMillion DESC;
```

LIMIT

The use of **LIMIT <**number**>** limits the output to be only the specified number of tuples

- can be used with **ORDER BY** to get a maximum or minimum value!

```
SELECT Name, (Population / 1000000) AS
PopulationInMillion
FROM City
ORDER BY PopulationInMillion DESC
LIMIT 2;
```

Multiple Relations

- We often want to combine data from more than one relation
- We can address several relations in one query by listing them all in the **FROM** clause
- If two attributes from different relations have the same name, we can distinguish them by writing <relation>**.**<attribute>

Example

What is the name of countries that speak Greek?

```
SELECT Name
FROM Country, CountryLanguage
WHERE Code = CountryCode
   AND Language = 'Greek';
```

This is BAD style!!

EXAMPLE: GOOD STYLE

```
SELECT Country.Name
FROM Country, CountryLanguage
WHERE Country.Code=CountryLanguage.CountryCode
AND CountryLanguage.Language = 'Greek';
```

```
SELECT C.Name
FROM Country C, CountryLanguage L
WHERE C.Code = L.CountryCode
AND L.Language = 'Greek';
```

VARIABLES

Variables are necessary when we want to use two copies of the same relation!

```
SELECT C.Name
FROM Country C, CountryLanguage L1,
CountryLanguage L2
WHERE  C.Code = L1.CountryCode
   AND C.Code = L2.CountryCode
   AND L1.Language = 'Greek’
   AND L2.Language = 'English';
```

SEMANTICS

1. Start with the cross product of all the relations in the **FROM** clause
2. Apply the conditions from the **WHERE** clause
3. Project onto the list of attributes and expressions in the **SELECT** clause
4. If **DISTINCT** is specified, eliminate duplicate rows

Semantics of SQL: Nested Loop

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

```
answer := {}
for x1 in R1 do
    for x2 in R2 do
        …..
          for xn in Rn do
              if Conditions
                  then answer := answer ∪ {(a1,…,ak)}
return answer
```

SEMANTICS OF SQL

- The query processor will almost never evaluate the query this way
- SQL is a declarative language
- The DBMS the system figures out what is the most efficient to compute it (optimization)

SEMANTICS OF SQL: RA

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

$$\pi_{a_1,a_2,\ldots,a_k}(\sigma_{Conditions}(R_1 \times R_2 \times \cdots \times R_n))$$

MORE SQL

- Union, intersection, and difference of relations can be expressed:
  - (subquery) **UNION** (subquery)
  - (subquery) **INTERSECT** (subquery)
  - (subquery) **EXCEPT** (subquery)
- Duplicates with union, except, intersect
  - default: eliminate duplicates!
  - use **ALL** to keep duplicates

DUPLICATES

- When doing projection:
  - easier to avoid eliminating duplicates
  - tuple-at-a-time processing
- When doing intersection, union or difference:
  - more efficient to sort the relations first
  - at that point you may as well eliminate the duplicates anyway

Nested Queries

NESTED QUERIES

A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places:

- in **FROM** clauses
- in **WHERE** clauses

```
SELECT C.Name
FROM Country C
WHERE C.code =
   (SELECT C.CountryCode
    FROM City C
    WHERE C.name = 'Berlin');
```

Can you rewrite this query without a subquery (**unnesting**)?

Nested Queries

Find all countries in Europe with population more than 50 million

```
SELECT C.Name
FROM (SELECT Name, Continent
      FROM Country
      WHERE Population >50000000) AS C
WHERE C.Continent = 'Europe' ;
```

Can you unnest this query?

SET-COMPARISON OPERATOR: IN

Find all countries in Europe that have **some** city with population more than 5 million

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe’
AND C.Code IN (SELECT CountryCode
               FROM City
               WHERE Population > 5000000);
```

SET-COMPARISON OPERATOR: EXISTS

Find all countries in Europe that have **some** city with population more than 5 million

correlated subquery

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe’
AND EXISTS (SELECT *
            FROM City T
            WHERE T.Population > 5000000
            AND T.CountryCode = C.Code);
```

SET-COMPARISON OPERATOR: ANY

Find all countries in Europe that have **some** city with population more than 5 million

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe’
AND 5000000 <= ANY (SELECT T.Population
         FROM City T
         WHERE T.CountryCode = C.Code);
```

SET-COMPARISON OPERATORS

Find all countries in Europe that have **all** cities with population less than 1 million

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe’
AND NOT EXISTS (SELECT *
           FROM City T
           WHERE T.Population > 1000000
           AND T.CountryCode = C.Code);
```

SET-COMPARISON OPERATORS: ALL

Find all countries in Europe that have **all** cities with population less than 1 million

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe’
AND 1000000 > ALL (SELECT T.Population
        FROM City T
        WHERE T.CountryCode = C.Code);
```

AGGREGATION

AGGREGATION

- **SUM**, **AVG**, **COUNT**, **MIN**, **MAX** can be applied to a column in a **SELECT** clause to produce that aggregation on the column
- **COUNT**(*) counts the number of tuples

```
SELECT AVG(Population)
FROM Country
WHERE Continent = ‘Europe’;
```

Aggregation: Eliminate Duplicates

- **COUNT**(DISTINCT) to remove duplicate tuples before counting!

```
SELECT COUNT (DISTINCT Language)
FROM CountryLanguage ;
```

GROUP BY

- We may follow a SELECT-FROM-WHERE expression by **GROUP BY** and a list of attributes
- The relation is then grouped according to the values of those attributes, and any aggregation is applied only within each group

```
SELECT GovernmentForm, COUNT(Code)
FROM Country
GROUP BY GovernmentForm ;
```

RESTRICTIONS

If any aggregation is used, then each element of the **SELECT** list must be either:

- aggregated, or
- an attribute on the **GROUP BY** list

Group By + Having

- The **HAVING** <condition> can follow a **GROUP BY** clause
- The condition
  - applies to each group, and groups not satisfying the condition are removed
  - can refer only to attributes of relations in the **FROM** clause, as long as the attribute makes sense within a group

EXAMPLE

```
SELECT Language, COUNT(CountryCode) AS N
FROM CountryLanguage
WHERE Percentage >= 50
GROUP BY Language
HAVING N > 2
ORDER BY N DESC ;
```

PUTTING IT ALL TOGETHER!

```
SELECT [DISTINCT] S
FROM R, S, T ,…
WHERE C1
GROUP BY attributes
HAVING C2
ORDER BY attribute ASC/DESC
LIMIT N ;
```

CONCEPTUAL EVALUATION

1. Compute the **FROM-WHERE** part, obtain a table with all attributes in $R_1,…,R_n$
2. Group by the attributes in the **GROUP BY**
3. Compute the aggregates and keep only groups satisfying condition C2
4. Compute aggregates in S
5. Order by the attributes specified