SQL Part III

IOE 373 Lecture 06

Topics

- HAVING Clause
- LIKE and DISTINCT Operators
- Subqueries
  - Uncorrelated
  - Correlated

Example Data

- Example (From BBC Country Profile)
- Table named ***Countries***:

CountryName	Region	Area	Population	GDP
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
Algeria	Middle East	2400000	32900000	75012000000
Andorra	Europe	468	64000	
…	…	…	…	…
Yemen	Middle East	536869	21500000	12255000000
Zambia	Africa	752614	11000000	4950000000
Zimbabwe	Africa	390759	12900000	6192000000

Review – GROUP BY

- For each region, show the region name and average GDP of all countries in this region.

CountryName	Region	Area	Population	GDP
Afghanistan	South Asia	652225	26000000	
…	…	…	…	…
Zimbabwe	Africa	390759	12900000	6192000000

Review – GROUP BY

- For each region, show the region name and average GDP of all countries in this region.

```
SELECT Region, AVG(GDP) AS AvgGDP
 FROM Countries GROUP BY Region
```

CountryName	Region	Area	Population	GDP
Afghanistan	South Asia	652225	26000000	
…	…	…	…	…
Zimbabwe	Africa	390759	12900000	6192000000

Region	AvgGDP
South Asia	69990000
…	…
Africa	23999900

Review – GROUP BY

- Note: Attributes in **SELECT** clause outside of aggregate functions must appear in **GROUP BY** list
- For example

WRONG!

```
SELECT CountryName, Region, AVG(GDP)
   AS AvgGDP
FROM Countries GROUP BY Region
```

- Would create an error, because CountryName is not in the **GROUP BY** clause

HAVING Clause

- **HAVING** specifies additional conditions for an aggregate function or **GROUP BY** statement
- Example:
- Show the region name and average GDP of region with average GDP > 200,000,000,000 (200 Billion Dollars)

```
SELECT Region, AVG(GDP) FROM Countries
GROUP BY Region HAVING AVG(GDP) >
200000000000
```

Differences Between **HAVING** and **WHERE**

- **HAVING** clauses are applied AFTER the formation of groups
- Conditions in the **WHERE** clause are applied BEFORE forming groups
- You cannot use aggregate functions in **WHERE** clause

HAVING Clause

- You cannot use this:

WRONG!

```
SELECT Region, AVG(GDP) FROM Countries
GROUP BY Region WHERE AVG(GDP) > 20000000
```

- Because there is no such thing as **AVG**(GDP) BEFORE the aggregation!

HAVING Clause

- However, sometimes you can use both:

```
SELECT Region, AVG(GDP) FROM Countries
 WHERE region='Africa'
   OR region='Middle East'
    GROUP BY Region
```

- Will give you the same result as

```
SELECT Region, AVG(GDP) FROM Countries
 GROUP BY Region HAVING region='Africa'
   OR region='Middle East'
```

- Because no matter if you filter the region name before or after the aggregation, you will have the same result

LIKE Operator

- LIKE operator is used to find values in a field that match the pattern you specify
- For example, select all countries names start with "B"

```
SELECT CountryName FROM Countries
WHERE CountryName LIKE 'B*'
```

- Wildcard character:
- '*' match zero or more characters
- '?' match a single character
- '#' match a single digit (0-9)

LIKE Operator Match Patterns

Pattern	**Match**	**Match Example**
ti	String contains 'ti'	Indica**ti**on, **ti**dy, spaghet**ti**
a?t	Three letter word start with 'a', ends with 't'	act, att, aqt
a#a	One digit between two 'a's	a0a, a1a, a9a
[acef]	one character that is from a, c, e, or f	a, f, c, e
[a-z]	one character from a to z	a, b, c, d, e, g, y
[0-9]	any one digit from 0 to 9	0,1,9
19##[a-q]	Four digits starts with '19', followed by one character from a to q	1984a, 1900b, 1977c, 1999p, 1907q

Example of **LIKE** Operator

- Select countries with name containing the word ‛land’

```
SELECT CountryName FROM Countries
WHERE CountryName LIKE ‛*land*’
```

- Result:

CountryName
British Virgin Islands
Cayman Islands
Switzerland
Cocos Islands
...
Solomon Islands
Thailand
Turks and Caicos Islands
Virgin Islands
Christmas Island

DISTINCT Operator

- Some SQL results may contain duplicated rows. **DISTINCT** operator can be used to remove duplicates
- For example, if we want to see all the regions listed in the *Countries* table:

```
SELECT DISTINCT Region FROM Countries
```

- If you don't use **DISTINCT**, it will show you all 200+ records, most of them duplicates

More on Subqueries

- Uncorrelated (Regular) Subqueries
  - Just runs once and returns a value or a set of values which is then used by the outer query
- Correlated Subqueries (similar to recursion)
  - Runs for each row returned by the outer query (recursively.) The output of the whole query is based upon comparing the data returned by each row to all the other rows of the table.

Uncorrelated Subqueries

- You can use **IN** operator to nest sub-queries:

```
SELECT CountryName, Region FROM Countries
   WHERE Region IN
     (SELECT Region FROM Countries
        WHERE CountryName='Brazil' OR
        Name='Mexico')
```

- This query shows each country and its region in the same region as 'Brazil' or 'Mexico'.
- The **SELECT** query inside the (…) is called sub-query or nested query.
- This is an example of a **non-correlated subquery**: The subquery executes only once

Uncorrelated Subqueries

- You can use binary operator to test the values from sub-query:

```
SELECT CountryName FROM Countries
  WHERE Population >
      (SELECT Population FROM Countries
       WHERE CountryName='Russia')
```

- This shows each country name where the population is larger than 'Russia'

Uncorrelated Subqueries

- Use the operator **ALL** or **ANY** when the sub-query have multiple values

```
SELECT CountryName FROM Countries
  WHERE Population > ALL
     (SELECT Population FROM Countries
      WHERE Region='Europe')
```

- This shows each country that has a population greater than the population of ALL individual countries in Europe (NOT the combined population of all European countries!)

- Equivalent to

```
SELECT CountryName FROM Countries
  WHERE Population >
  (SELECT MAX(Population) FROM Countries
      WHERE Region='Europe')
```

Uncorrelated Subqueries

- ANY Operator

```
SELECT CountryName FROM Countries
   WHERE Population > ANY
        (SELECT Population FROM Countries
         WHERE Region='Europe'
```

- Shows you each country that has a population greater than the population of ANY country from Europe.
- Equivalent to

```
SELECT CountryName FROM Countries
   WHERE Population >
   (SELECT MIN(Population) FROM Countries
         WHERE Region='Europe')
```

Correlated subqueries

- Find the largest country in each region, show the region, the name (of the country) and the population (of the country)

```
SELECT Region, CountryName, Population
  FROM Countries AS x
  WHERE Population >= ALL
    (SELECT Population FROM Countries AS y
          WHERE y.Region=x.Region)
  ORDER BY Population DESC
```

- Use alias (**AS** …) to distinguish tables with the same name.

How does that work?

```
SELECT Region, CountryName, Population
  FROM Countries AS x
  WHERE Population >= ALL

     (SELECT Population FROM Countries AS y
      WHERE y.Region=x.Region)

ORDER BY Population DESC
```

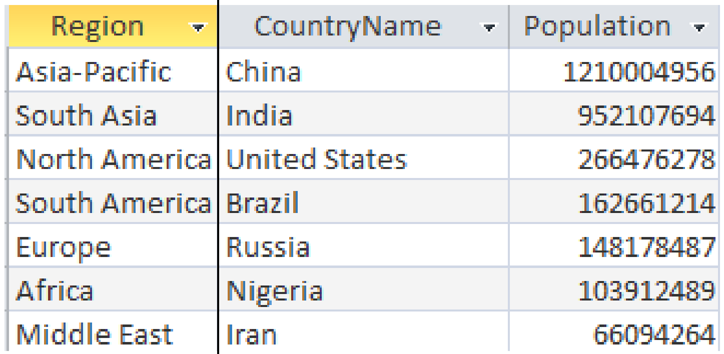
Recursion!!

The subquery is executed multiple times, once for every row of the outer query (e.g. in this case for every x.Region)

Region	Population
Europe	8023244
South Asia	22664136
North America	65647
Europe	3249136
Asia-Pacific	65628
Europe	72766
	10342899
	3463574
	103065
	18260863
	14436
	7676953

How does that work?

```
SELECT Region, CountryName, Population
   FROM Countries AS x
   WHERE Population >= ALL
      (SELECT Population FROM Countries AS y
        WHERE y.Region=x.Region)
ORDER BY Population DESC
```



Region	CountryName	Population
Asia-Pacific	China	1210004956
South Asia	India	952107694
North America	United States	266476278
South America	Brazil	162661214
Europe	Russia	148178487
Africa	Nigeria	103912489
Middle East	Iran	66094264

Region	Population
Europe	8023244
South Asia	22664136
North America	65647
Europe	3249136
Asia-Pacific	65628
Europe	72766
	10342899
	3463574
	103065
	18260863
	14436
	7676953

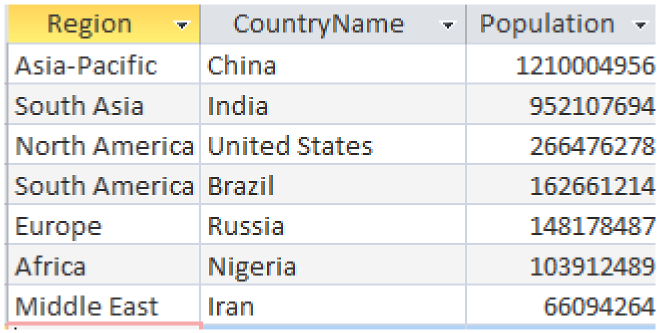
Same result (without the ALL clause)

```
SELECT Region, CountryName, Population
FROM Countries AS x
WHERE Population =
    (SELECT MAX(Population) FROM Countries
        AS y WHERE y.Region=x.Region)
ORDER BY Population DESC
```

How does it work?

```
SELECT Region, CountryName, Population
   FROM Countries AS x
   WHERE Population =
        (SELECT Max(Population) FROM Countries AS y
              WHERE y.Region=x.Region)
```

Region	Country Name	Population
South America	Brazil	162661214
South Asia	India	952107694
Middle East	Iran	66094264
Europe	Russia	148178487
Asia-Pacific	China	1210004956
North America	United States	266476278
Africa	Nigeria	103912489



Region	CountryName	Population
Asia-Pacific	China	1210004956
South Asia	India	952107694
North America	United States	266476278
South America	Brazil	162661214
Europe	Russia	148178487
Africa	Nigeria	103912489
Middle East	Iran	66094264

Alternative: 2 step Solution

- Step 1 – Create a table out of the first query (e.g. finding the maximum population by region)

```
SELECT Countries.Region,
    Max(Countries.Population) AS
    RegionMaxPopulationCountry INTO
    CountryMaxPopulationByRegion
FROM Countries
GROUP BY Countries.Region
```

Step 2 : Join Tables and find the country with the highest Population in each region

```
SELECT CountryMaxPopulationByRegion.Region,
        Countries.CountryName, Countries.Population
FROM CountryMaxPopulationByRegion
INNER JOIN Countries ON
        CountryMaxPopulationByRegion.Region=Countries.Region
WHERE Countries.Population=CountryMaxPopulati
        onByRegion.RegionMaxPopulationCountry
ORDER BY Countries.Population DESC;
```

Another Example of Nested SELECT (Correlated subquery)

- Some countries have populations more than three times that of any of their neighbors (in the same region). Give the countries and regions.

```
SELECT CountryName, Region FROM Countries AS
x
  WHERE Population > ALL
    (SELECT 3*Population FROM Countries AS y
      WHERE y.Region = x.Region
       AND y.Name <> x.Name)
```

- Don't forget to exclude the country itself from sub-query!

Name	Region
Brazil	South America
China	Asia-Pacific
India	South Asia