Relational Databases and SQL
CS102 - April 11 & 13

Relational database management systems have been around for more than 40 years, are a $30+ billion per year industry, and show no sign of slowing down. Why so successful? Simple model, high-level expressive query language, reliable and scalable systems. Even today's 'NoSQL' systems are starting to look more and more like RDBMSs.

Popular commercial proprietary systems: Oracle, Microsoft SQL Server, IBM DB2, others
Popular open-source systems: MySQL, SQLite, PostgreSQL, others

Basic concepts

- Relation (table)
- Attribute (column)
- Tuple (row)
- Types and domains

Main differences from spreadsheets

- Feels more "row-oriented"
- Order not significant (can change on reopen)
- Each table has regular structure
- Data elements always values, not formulas

Creating and loading data

- Environment/system-dependent, but can nearly always start with CSV file or similar

Querying

- A query is executed over one or more tables, returns a table as its result
- Examples
    - Find all cities with temp between 20 and 30; return city, state, and temp
        - Find average temp for each state; return state and average temp
    - Same, but for each region instead of each state
    - Find all regions that have both coastal and non-coastal states; return region and number of coastal/non-coastal states
    - Find all pairs of cities that are near each other, i.e., lat and lng are both less than 1.0 apart; return city pairs
    - Find the southernmost city

The SQL language

- Also more than 40 years old, one of oldest languages still in use (others: Fortran, C)
- Supported by all RDBMSs, standardized across products (more or less...)
- Interactive or embedded in programs
- Also can be used to modify the database

Basic Select statement

```
Select columns
From tables
Where condition
```

Find all coastal states; return state

```
Select state
From Regions
Where coastal = 'Y'
```

Find all cities with temp between 20 and 30; return city, state, and temp

```
Select city, state, temp
From CityTemps
Where temp >= 20 And temp <= 30
```

Ordering

Previous query sorted by temp - add “order by temp”, then “order by temp desc”
Previous query sorted by state, then temp, then city reversed - add “order by state, temp, city desc”

Multiple tables in From clause ('joins')

Find all cities with temp between 20 and 30 in a coastal state; return city

```
Select city
From CityTemps, Regions
Where temp >= 20 And temp <= 30 and coastal = 'Y'
      And CityTemps.state = Regions.state
```

Find all cities in the Northeast; return city and coastal

```
Select city, coastal
From CityTemps, Regions
Where region = 'Northeast' And CityTemps.state = Regions.state
```

Previous query but also return state and temp

```
Select city, CityTemps.state, temp, coastal
From CityTemps, Regions
Where region = 'Northeast' And CityTemps.state = Regions.state
```

Select *

Previous query but return all attributes

```
Select *
From CityTemps, Regions
Where region = 'Northeast' And CityTemps.state = Regions.state
```

Second query (all cities with temp between 20 and 30) but return all attributes

```
Select *
From CityTemps
Where temp >= 20 And temp <= 30
```

Aggregation and grouping

Find average temp for all cities

```
Select avg(temp) From CityTemps
```

Find average temp of cities with latitude under 35

```
Select avg(temp) From CityTemps where lat < 35
```

Find minimum, maximum, and average temp of cities with latitude under 35

```
Select min(temp), max(temp), avg(temp) From CityTemps where lat < 35
```

Find number of cities with latitude under 35

```
Select count(*) From CityTemps where lat < 35
```

Rename result column - add 'as toasty' after count(*)

Find minimum and maximum temp of cities in the Pacific

```
Select min(temp), max(temp)
From CityTemps, Regions
Where region = 'Pacific' And CityTemps.state = Regions.state
```

Find average temp for each state

```
Select state, avg(temp)
From CityTemps
Group By state
```

Same query ordered by descending average temp - add “order by avg(temp) desc”

Find average temp for each region, sorted by descending average temp

```
Select region, avg(temp)
From CityTemps, Regions
Where CityTemps.state = Regions.state
Group By region
Order by avg(temp) desc
```

Same query, but count coastal states only - add “coastal = 'Y'“ to Where clause

Average temp for each region/coastal combination - add “coastal” to Select and Group By, remove “coastal = 'Y'“ from Where clause

Table variables, duplicates

Find all regions that have both coastal and non-coastal states; return region

```
Select R1.region
From Regions R1, Regions R2
Where R1.coastal = 'Y' And R2.coastal = 'N' And R1.region=R2.region
```

Remove duplicates - add “Distinct” after Select

Find all pairs of cities that are near each other, i.e., lat and lng are both less than 1.0 apart; return city pairs

```
Select C1.city, C2.city
From CityTemps C1, CityTemps C2
Where abs(C1.lat - C2.lat) < 1 And abs(C1.lng - C2.lng) < 1
And C1.city <> C2.city
```

Remove duplicate-pairs - change "C1.city <> C2.city" to "C1.city < C2.city"

Subqueries in Where clause

Find the southernmost city

```
Select city
From CityTemps C1
Where Not Exists (Select * From CityTemps C2 Where C2.lat < C1.lat)
```

Find regions with no coastal states

```
Select Distinct region
From Regions R1
Where Not Exists (Select * From Regions R2 Where coastal = 'Y'
                  And R1.region = R2.region)
```

Find regions that have at least one city with lat greater than 45

```
Select Distinct region
From Regions R
Where Exists (Select * From CityTemps C
              Where lat > 45 And C.state = R.state)
```

Same query without using subquery

```
Select Distinct region
From Regions R, CityTemps C
Where lat > 45 And C.state = R.state
```

Find all cities whose temp is more than twice the average

```
Select city
From CityTemps
Where temp > (Select avg(temp)*2 From CityTemps)
```

Data modification

Raise all temperatures by 2 degrees

```
Update CityTemps Set temp = temp + 2
```

Add additional 2 degrees for non-coastal states

```
Update CityTemps Set temp = temp + 2
Where state In (Select state From Regions Where coastal = 'N')
```

Delete all cities in California

```
Delete From CityTemps Where state = 'California'
```

Delete all states from the Regions table that have no cities in CityTemps

```
Delete From Regions Where state Not In (select state from CityTemps)
```

Also `Insert` commands for inserting one row, or all rows in the result of a query

Advanced: Having clause

Find all states with at least three cities

```
Select state
From CityTemps
Group By state
Having count(*) >= 3
```

Same query without Having clause

```
Select Distinct C1.state
From CityTemps C1, CityTemps C2, CityTemps C3
Where C1.state = C2.state And C2.state = C3.state
And C1.city <> C2.city And C2.city <> C3.city And C1.city <> C3.city
```

But difficult for higher threshold

Find all states with minimum temperature below 10

```
Select state
From CityTemps
Group By state
Having min(temp) < 10
```

Same query without Group By / Having

```
Select Distinct state From CityTemps Where temp < 10
```

But doesn't work for average temperature below 10

Advanced: Subqueries in From and Select clauses

Find all regions that have both coastal and non-coastal states; return region and number of coastal and non-coastal states

```
Select Distinct R1.region,
       (Select count(*) From Regions R3
        Where R3.region = R1.region And R3.coastal = 'Y') as numcoastal,
       (Select count(*) From Regions R3
        Where R3.region = R1.region And R3.coastal = 'N') as numnot
From Regions R1, Regions R2
Where R1.coastal = 'Y' And R2.coastal = 'N' And R1.region=R2.region
```

Same query using subquery in From clause instead of Select clause

```
Select C.region, numcoastal, numnot
From (Select region, count(*) as numcoastal
      From Regions Where coastal = 'Y' Group By region) C,
     (Select region, count(*) as numnot
      From Regions Where coastal = 'N' Group By region) NC
Where C.region = NC.region And numcoastal > 0 And numnot > 0
```

Some other features

- **Set operators:** `Union, Intersect, Except`
- **Keys:** Designated column of a table that must have a unique value in each row. (Or a designated set of columns that must have a unique set of values in each row.)
- **Null values:** Value for an attribute in a row may have special value 'null', usually to denote unknown or undefined. Not included in numeric aggregations; not =, <>, <, or > than any value.

  `Select * From CityTemps Where temp <= 10 or temp > 10`

  will not return all rows of table in presence of null values