Distributed Query Processing

Advanced Topics in Database Management (INFSCI 2711)

Some materials are from Database System Concepts,
Siberschatz, Korth and Sudarshan

Vladimir Zadorozhny, DINS, University of Pittsburgh
Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)
Basic Query Processing Architecture

Find the names of customers having accounts in Brooklyn

```
select customer-name
from branch, account, depositor
where branch-name = "Brooklyn"
and branch.branch-ID = account.branch-ID
and account.customer-ID = depositor.customer-ID)
```

```
\(\Pi_{\text{customer-name}}(\sigma_{\text{branch-city} = \text{"Brooklyn"}}(\text{branch} \Join (\text{account} \Join \text{depositor}))))
```

Diagram:
- **query** → **parser** → **internal representation** → **optimizer** → **execution plan** → **output**
- **data** → **evaluator**
- **Catalog (metadata repository)**
- **Statistics about data**

**Branch**:
- (branch-ID, branch-name)

**Account**:
- (acc-number, branch-ID, customer-ID)

**Depositor**:
- (customer-ID, customer-name, customer-addr)
Basic operations:

- **Selection** ($\sigma$) Selects a subset of rows from relation.
- **Projection** ($\pi$) Deletes unwanted columns from relation.
- **Union** ($\cup$) Tuples in reln. 1 and in reln. 2.
- **Set-difference** (___) Tuples in reln. 1, but not in reln. 2.
- **Cross-product** ($\times$) Allows us to combine two relations.

Additional operations:

- Intersection, join, renaming: Not core operations, but (very!) useful.
- Since each operation returns a relation, operations can be composed! (Algebra is “closed”.)
### Sailors Database

#### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>53</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

**Figure 5.1** An Instance $S_3$ of Sailors

#### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
</tr>
</tbody>
</table>

**Figure 5.2** An Instance $R_2$ of Reserves

#### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>
Find names of sailors who’ve reserved boat #103

- Solution 1: $\pi_{\text{name}}((\sigma_{\text{bid}=103} \text{Reserves}) \bowtie \text{Sailors})$

- Solution 2: $\rho (\text{Temp1}, \sigma_{\text{bid}=103} \text{Reserves})$
  
  $\rho (\text{Temp2}, \text{Temp1} \bowtie \text{Sailors})$
  
  $\pi_{\text{name}}(\text{Temp2})$

- Solution 3: $\pi_{\text{name}}(\sigma_{\text{bid}=103} (\text{Reserves} \bowtie \text{Sailors}))$
Find names of sailors who’ve reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

\[ \pi_{sname}((\sigma_{\text{color} = 'red'} \ Boats) \bowtie_{\bowtie} \ Reserves \bowtie_{\bowtie} \ Sailors) \]

- A more efficient solution:

\[ \pi_{sname}(\pi_{sid}(\pi_{bid}((\sigma_{\text{color} = 'red'} \ Boats) \bowtie_{\bowtie} \ Res) \bowtie_{\bowtie} \ Sailors)) \]

A query optimizer can find this, given the first solution!
Optimization

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
An evaluation plan defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.
Cost difference between evaluation plans for a query can be enormous
- E.g. seconds vs. days in some cases

Steps in **cost-based query optimization**
1. Generate logically equivalent expressions using equivalence rules
2. Annotate resultant expressions to get alternative query plans
3. Choose the cheapest plan based on estimated cost

Estimation of plan cost based on:
- Statistical information about relations. Examples:
  - number of tuples, number of distinct values for an attribute
- Statistics estimation for intermediate results
  - to compute cost of complex expressions
- Cost formulae for algorithms, computed using statistics
Pictorial Depiction of Equivalence Rules

Rule 5

Rule 6a

Rule 7a

If $\theta$ only has attributes from E1
Multiple Transformations (Cont.)

(a) Initial expression tree

(b) Tree after multiple transformations
Join Ordering Example

- For all relations $r_1$, $r_2$, and $r_3$,
  
  $$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$
  
  (Join Associativity)

- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose

  $$(r_1 \bowtie r_2) \bowtie r_3$$

  so that we compute and store a smaller temporary relation.
Materialization

- **Materialized evaluation**: evaluate one operation at a time, starting at the lowest-level. Use intermediate results materialized into temporary relations to evaluate next-level operations.

- E.g., in figure below, compute and store

\[ \sigma_{\text{balance} < 2500}(\text{account}) \]

then compute and store its join with *customer*, and finally compute the projections on *customer-name*.
Materialization (Cont.)

- Materialized evaluation is always applicable
- Cost of writing results to disk and reading them back can be quite high
  - Our cost formulas for operations ignore cost of writing results to disk, so
    - Overall cost = Sum of costs of individual operations + cost of writing intermediate results to disk
Pipelining

- **Pipelined evaluation**: evaluate several operations simultaneously, passing the results of one operation on to the next.
- E.g., in previous expression tree, don’t store result of \( \sigma_{balance<2500}(\text{account}) \)
  - instead, pass tuples directly to the join. Similarly, don’t store result of join, pass tuples directly to projection.
- Much cheaper than materialization: no need to store a temporary relation to disk.
- Pipelining may not always be possible – e.g., sort.
- For pipelining to be effective, evaluation algorithms should generate output tuples even as tuples are received for inputs to the operation.
- Pipelines can be executed in two ways: **demand driven** (lazy, pull) and **producer driven** (eager, push).
Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites
Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network.
  - The potential gain in performance from having several sites process parts of the query in parallel.
Distributed Data Storage

- Assume relational data model
- Replication
  - System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
- Fragmentation
  - Relation is partitioned into several fragments stored in distinct sites
  - Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.
Data Replication

- A relation or fragment of a relation is replicated if it is stored redundantly in two or more sites.
- Full replication of a relation is the case where the relation is stored at all sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.
Data Replication (Cont.)

- Advantages of Replication
  - **Availability**: failure of site containing relation $r$ does not result in unavailability of $r$ is replicas exist.
  - **Parallelism**: queries on $r$ may be processed by several nodes in parallel.
  - **Reduced data transfer**: relation $r$ is available locally at each site containing a replica of $r$.

- Disadvantages of Replication
  - Increased cost of updates: each replica of relation $r$ must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.

  - One solution: choose one copy as primary copy and apply concurrency control operations on primary copy.
Data Fragmentation

- Division of relation r into fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation r.

- **Horizontal fragmentation**: each tuple of r is assigned to one or more fragments

- **Vertical fragmentation**: the schema for relation r is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

- Example: relation account with following schema
  - $Account = (branch\_name, account\_number, balance)$
Horizontal Fragmentation of \textit{account} Relation

<table>
<thead>
<tr>
<th>branch_name</th>
<th>account_number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillside</td>
<td>A-305</td>
<td>500</td>
</tr>
<tr>
<td>Hillside</td>
<td>A-226</td>
<td>336</td>
</tr>
<tr>
<td>Hillside</td>
<td>A-155</td>
<td>62</td>
</tr>
</tbody>
</table>

\[ \text{account}_1 = \sigma_{\text{branch\_name} = \text{"Hillside"}}(\text{account}) \]

<table>
<thead>
<tr>
<th>branch_name</th>
<th>account_number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valleyview</td>
<td>A-177</td>
<td>205</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-402</td>
<td>10000</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-408</td>
<td>1123</td>
</tr>
<tr>
<td>Valleyview</td>
<td>A-639</td>
<td>750</td>
</tr>
</tbody>
</table>

\[ \text{account}_2 = \sigma_{\text{branch\_name} = \text{"Valleyview"}}(\text{account}) \]
Vertical Fragmentation of `employee_info` Relation

<table>
<thead>
<tr>
<th><code>branch_name</code></th>
<th><code>customer_name</code></th>
<th><code>tuple_id</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hillside</td>
<td>Lowman</td>
<td>1</td>
</tr>
<tr>
<td>Hillside</td>
<td>Camp</td>
<td>2</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Camp</td>
<td>3</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Kahn</td>
<td>4</td>
</tr>
<tr>
<td>Hillside</td>
<td>Kahn</td>
<td>5</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Kahn</td>
<td>6</td>
</tr>
<tr>
<td>Valleyview</td>
<td>Green</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ deposit_1 = \Pi_{\text{branch\_name}, \text{customer\_name}, \text{tuple\_id}}(\text{employee\_info}) \]

<table>
<thead>
<tr>
<th><code>account_number</code></th>
<th><code>balance</code></th>
<th><code>tuple_id</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>A-305</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>A-226</td>
<td>336</td>
<td>2</td>
</tr>
<tr>
<td>A-177</td>
<td>205</td>
<td>3</td>
</tr>
<tr>
<td>A-402</td>
<td>10000</td>
<td>4</td>
</tr>
<tr>
<td>A-155</td>
<td>62</td>
<td>5</td>
</tr>
<tr>
<td>A-408</td>
<td>1123</td>
<td>6</td>
</tr>
<tr>
<td>A-639</td>
<td>750</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ deposit_2 = \Pi_{\text{account\_number}, \text{balance}, \text{tuple\_id}}(\text{employee\_info}) \]
Advantages of Fragmentation

- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed

- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a relation

- Vertical and horizontal fragmentation can be mixed.
  - Fragments may be successively fragmented to an arbitrary depth.
Data Transparency

- **Data transparency**: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system.

- Consider transparency issues in relation to:
  - Fragmentation transparency
  - Replication transparency
  - Location transparency
Query Transformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation \( r \) from its fragments.
  - Replace relation \( r \) by the expression to construct relation \( r \) from its fragments.
- Consider the horizontal fragmentation of the \( account \) relation into:
  \[
  \begin{align*}
  account_1 &= \sigma_{branch\_name = \text{“Hillside”}}(account) \\
  account_2 &= \sigma_{branch\_name = \text{“Valleyview”}}(account)
  \end{align*}
  \]
- The query \( \sigma_{branch\_name = \text{“Hillside”}}(account) \) becomes:
  \[
  \sigma_{branch\_name = \text{“Hillside”}}(account_1 \cup account_2)
  \]
  which is optimized into:
  \[
  \sigma_{branch\_name = \text{“Hillside”}}(account_1) \cup \sigma_{branch\_name = \text{“Hillside”}}(account_2)
  \]
Since $account_1$ has only tuples pertaining to the Hillside branch, we can eliminate the selection operation.

Apply the definition of $account_2$ to obtain

$$\sigma \ branch\_name = \text{“Hillside”} \ (\sigma \ branch\_name = \text{“Valleyview”} \ (account))$$

This expression is the empty set regardless of the contents of the $account$ relation.

Final strategy is for the Hillside site to return $account_1$ as the result of the query.
Simple Join Processing

- Consider the following relational algebra expression in which the three relations are neither replicated nor fragmented:
  \[ \text{account} \Join \text{depositor} \Join \text{branch} \]
- \text{account} is stored at site \( S_1 \)
- \text{depositor} at \( S_2 \)
- \text{branch} at \( S_3 \)
- For a query issued at site \( S_i \), the system needs to produce the result at site \( S_i \)
Possible Query Processing Strategies

- Ship copies of all three relations to site $S_1$ and choose a strategy for processing the entire result locally at site $S_1$.
- Ship a copy of the account relation to site $S_2$ and compute $temp_1 = account \Join depositor$ at $S_2$. Ship $temp_1$ from $S_2$ to $S_3$, and compute $temp_2 = temp_1 \Join branch$ at $S_3$. Ship the result $temp_2$ to $S_1$.
- Devise similar strategies, exchanging the roles $S_1$, $S_2$, $S_3$.
- Must consider following factors:
  - amount of data being shipped
  - cost of transmitting a data block between sites
  - relative processing speed at each site
Semijoin Strategy

- Let \( r_1 \) be a relation with schema \( R_1 \) stores at site \( S_1 \)
  - Let \( r_2 \) be a relation with schema \( R_2 \) stores at site \( S_2 \)
- Evaluate the expression \( r_1 \bowtie r_2 \) and obtain the result at \( S_1 \).
  1. Compute \( temp_1 \leftarrow \Pi_{R_1 \cap R_2} (r_1) \) at \( S_1 \).
  2. Ship \( temp_1 \) from \( S_1 \) to \( S_2 \).
  3. Compute \( temp_2 \leftarrow r_2 \bowtie temp_1 \) at \( S_2 \)
  4. Ship \( temp_2 \) from \( S_2 \) to \( S_1 \).
  5. Compute \( r_1 \bowtie temp_2 \) at \( S_1 \). This is the same as \( r_1 \bowtie r_2 \).
Formal Definition

- The **semijoin** of $r_1$ with $r_2$, is denoted by:
  \[ r_1 \bowtie_r r_2 \]
- It is defined by:
  \[ \Pi_{R_1} (r_1 \bowtie_r r_2) \]
- Thus, $r_1 \bowtie_r r_2$ selects those tuples of $r_1$ that contributed to $r_1 \bowtie_r r_2$.
- In step 3 above, $temp_2 = r_2 \bowtie_r r_1$.
- For joins of several relations, the above strategy can be extended to a series of semijoin steps.
Join Strategies that Exploit Parallelism

- Consider \( r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 \) where relation \( r_i \) is stored at site \( S_i \). The result must be presented at site \( S_1 \).
- \( r_1 \) is shipped to \( S_2 \) and \( r_1 \bowtie r_2 \) is computed at \( S_2 \); simultaneously \( r_3 \) is shipped to \( S_4 \) and \( r_3 \bowtie r_4 \) is computed at \( S_4 \).
- \( S_2 \) ships tuples of \( (r_1 \bowtie r_2) \) to \( S_1 \) as they produced; \( S_4 \) ships tuples of \( (r_3 \bowtie r_4) \) to \( S_1 \).
- Once tuples of \( (r_1 \bowtie r_2) \) and \( (r_3 \bowtie r_4) \) arrive at \( S_1 \) \( (r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4) \) is computed in parallel with the computation of \( (r_1 \bowtie r_2) \) at \( S_2 \) and the computation of \( (r_3 \bowtie r_4) \) at \( S_4 \).
Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms.

Data models may differ (hierarchical, relational, etc.).

Transaction commit protocols may be incompatible.

Concurrency control may be based on different techniques (locking, timestamping, etc.).

System-level details almost certainly are totally incompatible.

A **multidatabase system** is a software layer on top of existing database systems, which is designed to manipulate information in heterogeneous databases.

- Creates an illusion of logical database integration without any physical database integration.
Advantages

- Preservation of investment in existing
  - hardware
  - system software
  - applications
- Local autonomy and administrative control
- Allows use of special-purpose DBMSs
- Step towards a unified homogeneous DBMS
  - Full integration into a homogeneous DBMS faces
    - Technical difficulties and cost of conversion
    - Organizational/political difficulties
      - Organizations do not want to give up control on their data
      - Local databases wish to retain a great deal of autonomy
Unified View of Data

- Agreement on a common data model
  - Typically the relational model
- Agreement on a common conceptual schema
  - Different names for same relation/attribute
  - Same relation/attribute name means different things
- Agreement on a single representation of shared data
  - E.g. data types, precision,
  - Character sets
    - ASCII vs EBCDIC
    - Sort order variations
- Agreement on units of measure
- Variations in names
  - E.g. Köln vs Cologne, Mumbai vs Bombay
Several issues in query processing in a heterogeneous database

Schema translation
- Write a **wrapper** for each data source to translate data to a global schema
- Wrappers must also translate updates on global schema to updates on local schema

Limited query capabilities
- Some data sources allow only restricted forms of selections
  - E.g. web forms, flat file data sources
- Queries have to be broken up and processed partly at the source and partly at a different site

Removal of duplicate information when sites have overlapping information
- Decide which sites to execute query

Global query optimization
Mediator systems are systems that integrate multiple heterogeneous data sources by providing an integrated global view, and providing query facilities on global view.

- Unlike full fledged multidatabase systems, mediators generally do not bother about transaction processing.
- But the terms mediator and multidatabase are sometimes used interchangeably.
- The term virtual database is also used to refer to mediator/multidatabase systems.
- Manual navigation over multilevel links: inefficient

Find the top selling book on C++ at Amazon?

Objective: database-like declarative queries:

```sql
select bookTitle
from Amazon
where bookTopic = "C++" and
bookSalesRank > all ( select bookSalesRank
from Amazon
where bookTopic = "C++" )
```
Approximate Joins

Q1: relate Precipitation and Population in different states

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MONTH</th>
<th>COUNTRY</th>
<th>PRECIPITATION</th>
<th>STATE</th>
<th>Year</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1807</td>
<td>1</td>
<td>UNITED STATES OF AMERICA</td>
<td>1092</td>
<td>South Carolina</td>
<td>1900</td>
<td>1830</td>
</tr>
<tr>
<td>1807</td>
<td>2</td>
<td>UNITED STATES OF AMERICA</td>
<td>660</td>
<td>South Carolina</td>
<td>1901</td>
<td>1907</td>
</tr>
<tr>
<td>1807</td>
<td>3</td>
<td>UNITED STATES OF AMERICA</td>
<td>178</td>
<td>South Carolina</td>
<td>1902</td>
<td>1935</td>
</tr>
<tr>
<td>1807</td>
<td>4</td>
<td>UNITED STATES OF AMERICA</td>
<td>483</td>
<td>South Carolina</td>
<td>1903</td>
<td>1957</td>
</tr>
<tr>
<td>1807</td>
<td>5</td>
<td>UNITED STATES OF AMERICA</td>
<td>940</td>
<td>South Carolina</td>
<td>1904</td>
<td>1978</td>
</tr>
<tr>
<td>1807</td>
<td>6</td>
<td>UNITED STATES OF AMERICA</td>
<td>991</td>
<td>South Carolina</td>
<td>1905</td>
<td>2012</td>
</tr>
<tr>
<td>1807</td>
<td>7</td>
<td>UNITED STATES OF AMERICA</td>
<td>940</td>
<td>South Carolina</td>
<td>1906</td>
<td>2045</td>
</tr>
<tr>
<td>1807</td>
<td>8</td>
<td>UNITED STATES OF AMERICA</td>
<td>1422</td>
<td>South Carolina</td>
<td>1907</td>
<td>2045</td>
</tr>
<tr>
<td>1807</td>
<td>9</td>
<td>UNITED STATES OF AMERICA</td>
<td>1143</td>
<td>South Carolina</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State

<table>
<thead>
<tr>
<th>State</th>
<th>Year</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Florida SC

1.00 0.06

1.00 0.00 1.00
### Approximate Joins (cont.)

#### Q1: relate Precipitation and Population in different areas

<table>
<thead>
<tr>
<th>City</th>
<th>Year</th>
<th>Precipitation</th>
<th>Area</th>
<th>Year</th>
<th>Population</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pittsburgh</td>
<td>2012</td>
<td>700</td>
<td>Allegheny County</td>
<td>2011</td>
<td>306,211</td>
<td>s1</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>2012</td>
<td>700</td>
<td>Pennsylvania</td>
<td>2012</td>
<td>1,548,000</td>
<td>s2</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>2012</td>
<td>800</td>
<td>Pennsylvania</td>
<td>2012</td>
<td>1,548,000</td>
<td>s3</td>
</tr>
</tbody>
</table>

#### Q2: relate Precipitation and Temperature in different areas

<table>
<thead>
<tr>
<th>City</th>
<th>Year</th>
<th>Precipitation</th>
<th>Area</th>
<th>Year</th>
<th>Temperature</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pittsburgh</td>
<td>2012</td>
<td>700</td>
<td>Allegheny County</td>
<td>2011</td>
<td>60</td>
<td>s1</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>2012</td>
<td>800</td>
<td>Pennsylvania</td>
<td>2012</td>
<td>70</td>
<td>s2</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>2012</td>
<td>700</td>
<td>Allegheny County</td>
<td>2011</td>
<td>60</td>
<td>s3</td>
</tr>
</tbody>
</table>
Q1: relate Precipitation and Population in different areas?

Q2: relate Precipitation and Temperature in different areas?
We will learn more about Web query processing later