CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

Lecture #2
Data Models and Data Languages

- A data model is a mathematical formalism for describing and representing data.
- A data model is accompanied by a data language that has two parts: a data definition language and a data manipulation language.
  - A data definition language (DDL) has a syntax for describing “database templates” in terms of the underlying data model.
  - A data manipulation language (DML) supports the following operations on data:
    - Insertion
    - Deletion
    - Update
    - Retrieval and extraction of data (query language).
The Relational Data Model

- Introduced by E.F. Codd in 1970

- It is based on the mathematical notion of relation

  - Informally, a relation is a table with columns and rows

  - Formally, a relation $R$ is a subset of a cartesian product of sets
    $R \subseteq D_1 \times \cdots \times D_k$
Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- **Relational Algebra**, which is a *procedural* language.
  - It is an *algebraic formalism* in which queries are expressed by applying a sequence of operations to relations.

- **Relational Calculus**, which is a *declarative* language.
  - It is a *logical formalism* in which queries are expressed as formulas of first-order logic.
The Five Basic Operations of Relational Algebra

- **Group I**: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product.

- **Group II**: Two special unary operations on relations:
  - Projection
  - Selection.

- Relational Algebra consists of all expressions obtained by combining these five basic operations in syntactically correct ways.
The Projection Operation

- **Projection** is a family of unary operations of the form
  \[ \pi_{\text{<attribute list>}erahv} (\text{<relation name>}) \]

- The intuitive description of the projection operation is as follows:
  - When projection is applied to a relation R, it removes all columns whose attributes do **not** appear in the <attribute list>.
  - The remaining columns may be re-arranged according to the order in the <attribute list>.
  - Any duplicate rows are also eliminated.
### The Projection Operation - Example

**SAVINGS**

<table>
<thead>
<tr>
<th>branch-name</th>
<th>acc-no</th>
<th>cust-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aptos</td>
<td>153125</td>
<td>Vianu</td>
<td>3,450</td>
</tr>
<tr>
<td>Santa Cruz</td>
<td>123658</td>
<td>Hull</td>
<td>2,817</td>
</tr>
<tr>
<td>San Jose</td>
<td>321456</td>
<td>Codd</td>
<td>9,234</td>
</tr>
<tr>
<td>San Jose</td>
<td>334789</td>
<td>Codd</td>
<td>875</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{cust-name}, \text{branch-name}}(\text{SAVINGS}) \]

<table>
<thead>
<tr>
<th>cust-name</th>
<th>branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>
The Selection Operation

- **Selection** is a family of unary operations of the form
  \[ \sigma_\Theta (R), \]
  where \( R \) is a relation and \( \Theta \) is a condition that can be applied as a test to each row of \( R \).

- When a selection operation is applied to \( R \), it returns the subset of \( R \) consisting of all rows that satisfy the condition \( \Theta \)

- **Question**: What is the precise definition of a “condition”?
The Selection Operation

- **Definition:** A condition in the selection operation is an expression built up from:
  - Comparison operators =, <, >, ≠, ≤, ≥ applied to operands that are constants or attribute names or component numbers.
  - These are the basic (atomic) clauses of the conditions.
  - The Boolean logic operators ∧, ∨, ¬ applied to basic clauses.

- **Examples:**
  - balance > 10,000
  - branch-name = “Aptos”
  - (branch-name = “Aptos”) ∧ (balance < 1,000)
Relational Algebra

Definition: A relational algebra expression is a string obtained from relation schemas using union, difference, cartesian product, projection, and selection.

Context-free grammar for relational algebra expressions:

\[ E := R, S, \ldots | (E_1 \lor E_2) | (E_1 - E_2) | (E_1 \times E_2) | \pi_X(E) | \sigma_\Theta(E), \]

where

- \( R, S, \ldots \) are relation schemas
- \( X \) is a list of attributes
- \( \Theta \) is a condition.
By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.

When used in combination, however, the five relational algebra operations can express interesting and, quite often, rather complex queries.

Derived relational algebra operations are operations on relations that are expressible via a relational algebra expression (built from the five basic operators).
Intersection

- **Input**: Two k-ary relations R and S, for some k.
- **Output**: The k-ary relation \( R \cap S \), where

\[
R \cap S = \{(a_1, \ldots, a_k): (a_1, \ldots, a_k) \text{ is in } R \text{ and } (a_1, \ldots, a_k) \text{ is in } S\}
\]

- **Fact**: \( R \cap S = R - (R - S) = S - (S - R) \)

Thus, intersection is a derived relational algebra operation.
**θ-Join and Beyond**

- **Definition**: A θ-Join is a relational algebra expression of the form
  \[ \sigma_\theta (R \times S) \].
  θ-joins are often combined with projection to express interesting queries.

- **Example**: FACULTY(name, dpt, salary), CHAIR(dpt, name)
  - Find the salaries of department chairs
    \[
    \text{C-SALARY}(\text{dpt, salary}) = \\
    \pi_{\text{F.dpt, F.Salary}} (\sigma_{\text{F.name} = \text{C.name} \land \text{F.dpt} = \text{C.dpt}} (\text{FACULTY} \times \text{CHAIR}))
    \]

- **Note**: The θ-Join in this example is an **equijoin**, since θ is a conjunction of equality basic clauses.
Example: FACULTY(name, dpt, salary), C-SALARY(dpt, salary)
Find the names of all faculty members of the EE department who earn a bigger salary than their department chair.

\[
\text{HIGHLY-PAID-IN-EE(}\text{Name}) = \\
\pi \text{ F.name} \left( \sigma \text{ F.dpt = "EE" } \land \text{ C.dpt = "EE" } \land \text{ F.salary > C.salary} \right) \\
\text{(FACULTY} \times \text{ C-SALARY))}
\]

Note: The \(\Theta\)-Join above is not an equijoin.
Natural Join

- **Fact:** The most FAQs against databases involve the natural join operation \( \bowtie \).

- **Motivating Example:** Given
  
  TEACHES(fac-name, course, term) and
  ENROLLS(stud-name, course, term),

  we want to obtain
  TAUGHT-BY(stud-name, course, term, fac-name)

  It turns out that TAUGHT-BY = ENROLSS \( \bowtie \) TEACHES
Natural Join

Given TEACHES(fac-name, course, term) and
    ENROLLS(stud-name, course, term):
To compute TAUGHT-BY(stud-name, course, term, fac-name)

1. ENROLLS × TEACHES
2. \( \sigma_{T.course = E.course \land T.term = E.term} (\text{ENROLLS} \times \text{TEACHES}) \)
3. \( \pi_{\text{stud-name, E.course, E.term, fac-name}} (\sigma_{T.course = E.course \land T.term = E.term} (\text{ENROLLS} \times \text{TEACHES})) \)

The result is ENROLLS \( \bowtie \) TEACHES.
Natural Join

- **Definition:** Let $A_1, \ldots, A_k$ be the common attributes of two relation schemas $R$ and $S$. Then
  
  $$R \bowtie S = \pi_{<\text{list}>} \left( \sigma_{R.A_1 = S.A_1 \land \ldots \land R.A_k = S.A_k} (R \times S) \right),$$
  
  where $<\text{list}>$ contains all attributes of $R \times S$, except for $S.A_1, \ldots, S.A_k$ (in other words, duplicate columns are eliminated).

- **Naïve Algorithm for** $R \bowtie S$:
  
  For every tuple in $R$, compare it with every tuple in $S$ as follows:
  - test if they agree on all common attributes of $R$ and $S$;
  - if they do, take the tuple in $R \times S$ formed by these two tuples,
  - remove all values of attributes of $S$ that also occur in $R$;
  - put the resulting tuple in $R \bowtie S$. 
Natural Join

Some Algebraic Laws for Natural Join

- $R \bowtie S = S \bowtie R$ (up to rearranging the columns)
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $(R \bowtie R) = R$
- If $A$ is an attribute of $R$, but not of $S$, then
  \[ \sigma_{A = c} (R \bowtie S) = \sigma_{A = c} (R) \bowtie S \]
- ...

“There are more properties of join than we have room to list”

D. Maier, *The Theory of Relational Databases*, 1983
Quotient (Division)

- **Motivating Example:**
  Given ENROLLS(stud-name,course) and TEACHES(fac-name,course), find the names of students who take every course taught by V. Vianu.

- **Other Motivating Examples:**
  - Find the names of customers who have an account in every branch of Wachovia in San Jose.
  - Find the names of Netflix customers who have rented every film in which Paul Newman starred.

- These and other similar queries can be answered using the Quotient (Division) operation.
Definition: Let R be a relation of arity r and let S be a relation of arity s, where r > s. The quotient (or division) $R \div S$ is the relation of arity $r - s$ consisting of all tuples $(a_1,\ldots,a_{r-s})$ such that for every tuple $(b_1,\ldots,b_s)$ in S, we have that $(a_1,\ldots,a_{r-s}, b_1,\ldots,b_s)$ is in R.

Example: Given ENROLLS(stud-name,course) and TEACHES(fac-name,course), find the names of students who take every course taught by V. Vianu.
- Find the courses taught by V. Vianu
  \[ \pi_{\text{course}} \left( \sigma_{\text{fac-name} = \text{`V. Vianu''}} (\text{TEACHES}) \right) \]
- The desired answer is given by the expression:
  \[ \text{ENROLLS} \div \pi_{\text{course}} \left( \sigma_{\text{fac-name} = \text{`V. Vianu''}} (\text{TEACHES}) \right) \]
Fact: The quotient operation is expressible in relational algebra.

Proof: For concreteness, assume that R has arity 5 and S has arity 2.

Key Idea: Use the difference operation

- \( R \div S = \pi_{1,2,3}(R) \) – “tuples in \( \pi_{1,2,3}(R) \) that do not make it to \( R \div S \)”
- Consider the relational algebra expression \( (\pi_{1,2,3}(R) \times S) - R \).

Intuitively, it is the set of all tuples that fail the test for membership in \( R \div S \). Hence,

- \( R \div S = \pi_{1,2,3}(R) - \pi_{1,2,3}( (\pi_{1,2,3}(R) \times S) - R )) \).
The Expressive Power of Relational Algebra

- When combined together, the five basic relational algebra operations can express interesting and complex queries.

- In particular, relational algebra can express:
  - The Intersection Operation
  - The Natural Join Operation
  - The family of $\Theta$-Join Operations
  - The Quotient Operation
  - ....
Relational Completeness

- **Definition** (Codd – 1972): A database query language $L$ is **relationally complete** if it is at least as expressive as relational algebra, i.e., every relational algebra expression $E$ has an equivalent expression $F$ in $L$.

- Relational completeness provides a **benchmark** for the expressive power of a database query language.

- Every commercial database query language should be at least as expressive as relational algebra.
  - Relational completeness is a **threshold** that every commercial database query language should **meet** or **exceed**.
Independence of the Basic Relational Algebra Operations

- **Question:** Are all five basic relational algebra operations really needed? Can one of them be expressed in terms of the other four?

- **Theorem:** Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

**Proof Idea:** For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other four operations.
Theorem: Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

Proof Sketch: (projection and cartesian product only)

- Property of projection:
  - It is the only operation whose output may have arity smaller than its input.
  - Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at least as big as the maximum arity of its arguments.

- Property of cartesian product:
  - It is the only operation whose output has arity bigger than its input.
  - Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at most as big as the maximum arity of its arguments.
Relational Algebra: Summary

- When combined with each other, the five basic relational algebra operations can express interesting and complex queries (natural join, quotient, ...)

- The five basic relational algebra operations are independent of each other: none can be expressed in terms of the other four.
  - **Question:** What does this imply for establishing the relational completeness of some other database query language?

- So, in conclusion, Codd’s choice of the five basic relational algebra operations has been very judicious.
The basic SQL construct is:

```sql
SELECT <attribute list>
FROM <relation list>
WHERE <condition>
```

More formally,

```sql
SELECT R_{i1}.A1, ..., R_{im}.Am
FROM R_1, ..., R_K
WHERE \Psi
```

Restrictions:

- \( R_1, ..., R_K \) are distinct relation names (no repetitions)
- Each \( R_{ij}.A_j \) is an attribute of \( R_{ij} \)
- \( \Psi \) is a condition with a precise (and rather complex) syntax.
# SQL vs. Relational Algebra

<table>
<thead>
<tr>
<th>SQL</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Projection $\pi$</td>
</tr>
<tr>
<td>FROM</td>
<td>Cartesian Product</td>
</tr>
<tr>
<td>WHERE</td>
<td>Selection $\sigma$</td>
</tr>
</tbody>
</table>

**Semantics of SQL via interpretation to Relational Algebra**

```
SELECT $R_{i1}$.A1, $\ldots$, $R_{im}$.Am
FROM $R_1$, $\ldots$, $R_K$
WHERE $\psi$
```

equals

```
$\pi_{R_{i1}.A1, \ldots, R_{im}.Am}(\sigma_\psi(R_1 \times \ldots \times R_K))$
```
SQL vs. Relational Algebra

Example: FACULTY(name, dpt, salary), CHAIR(dpt, name)
Find the salaries of department chairs
C-SALARY(dpt,salary) =

- Relational Algebra
  \[ \pi_{\text{F.dpt, F.Salary}}(\sigma_{\text{F.name = C.name \land F.dpt = C.dpt}} (\text{FACULTY} \times \text{CHAIR})) \]

- SQL
  ```
  SELECT FACULTY.dpt, FACULTY.salary 
  FROM FACULTY, CHAIR 
  WHERE FACULTY.name = CHAIR.name AND 
    FACULTY.dpt = CHAIR.dpt
  ```
Self-Joins

Recall that the relation names in the FROM list must be **distinct**.

- **Question**: How do we then compute $R \times R$ in SQL?

- **Question**: Why do we care to compute $R \times R$?

- **Answer**: Many interesting queries involve self-joins, which, in turn, require computing $R \times R$. 
Example: Given MANAGES(manager,employee), we want to compute 2-MANAGES(2manager,employee).

In relational algebra, we can reference columns by position number. So, 2-MANAGES is expressed by the relational algebra expression \( \pi_{1,4} (\sigma_{2 = 3} (\text{MANAGES} \times \text{MANAGES})) \).

However, SQL does not support referencing columns by position number. Instead, SQL supports an aliasing mechanism.
Aliases in SQL

- SQL allows us to give one or more new names to a given relation; these are aliases of the given relation.

- Rules for Aliases Creation
  - Aliases are created in the FROM list
    - FROM \(<rel.\ name1>\) AS \(<rel.\ name2>\), ...
    - The new names can be referenced in the SELECT list and in the WHERE clause.

- Example: Expressing \(R \times R\) in SQL:
  - SELECT *
  - FROM R as S, R as T
Aliases in SQL

Example: MANAGES(manager,employee)

2-MANAGES(2manager,employee) in SQL:

```
SELECT R.manager as 2manager, T.employee as employee
FROM MANAGES AS R, MANAGES AS T
WHERE R.employee = T.manager
```

Note: This example also illustrates how SQL allows for the renaming of attribute names in the SELECT list.

Note: Aliases in SQL are used not only out of necessity (self-joins), but also for convenience in order to create short nicknames for relations.
Sets vs. Multisets

**Informal Definition:**

- A **set** is a collection of distinct objects viewed as a new object.
- A **multiset (bag)** is a collection of (not necessarily distinct) objects viewed as a new object.
- The **multiplicity** $m(a,B)$ of an element $a$ of a multiset $B$ is the number of occurrences of that element in the multiset.

**Examples:**

- $B = \{1,3,3,6,6,6,7\}$ is a multiset, but not a set (here, $m(3,B)=2$)
- $PF(n)$ is the multiset representing the prime factorization of $n$
  - $PF(90) = \{2,3,3,5\}$ (so, $m(2,PF(90)) = 1$, $m(3,PF(90)) = 2$)
  - $PF(64) = \{2,2,2,2,2\}$ (so, $m(2,PF(64)) = 6$).
Operations on Multisets

- **Union** \( R \cup S \): \( m(t, R \cup S) = m(t, R) + m(t, S) \)
- **Difference** \( R – S \): \( m(t, R – S) = \max\{m(t, R) – m(t, S), 0\} \).
- **Intersection** \( R \cap S \): \( m(t, R \cap S) = \min\{m(t, R), m(t, S)\} \)
- **Cartesian Product:**
  If \( t \in R \) and \( t' \in S \), then \( m(tt', R \times S) = m(t, R)m(t', S) \).

**Note:**
- If \( R \) is a multiset, then \( \pi_X(R) \) is also a multiset.
- If \( R \) is a set, then \( \pi_X(R) \) may be a multiset.
- If \( R \) is a set, then \( \sigma_\Theta(R) \) is a set.
- If \( R \) is a multiset, then \( \sigma_\Theta(R) \) may be a set.
Setting the Record Straight: Multiset Semantics in SQL

- Relations are sets (i.e., they consist of distinct tuples)

- Relational algebra expressions take relations as arguments and return relations as values.
  - In particular, all duplicates are eliminated in relational algebra.

- The SELECT ... FROM ... WHERE construct of SQL does not eliminate duplicates, unless explicitly requested.
  - In general, the SELECT ... FROM ... WHERE construct takes multisets as arguments and returns multisets as values.
  - In particular, the SELECT ... FROM ... WHERE construct may return a multiset, even if the arguments are sets.
## Set Semantics vs. Multiset Semantics

### SAVINGS

<table>
<thead>
<tr>
<th>branch-name</th>
<th>acc-no</th>
<th>cust-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aptos</td>
<td>153125</td>
<td>Vianu</td>
<td>3,450</td>
</tr>
<tr>
<td>Santa Cruz</td>
<td>123658</td>
<td>Hull</td>
<td>2,817</td>
</tr>
<tr>
<td>San Jose</td>
<td>321456</td>
<td>Codd</td>
<td>9,234</td>
</tr>
<tr>
<td>San Jose</td>
<td>334789</td>
<td>Codd</td>
<td>875</td>
</tr>
</tbody>
</table>

### $\pi_{\text{cust-name,branch-name}}(\text{SAVINGS})$

<table>
<thead>
<tr>
<th>cust-name</th>
<th>branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>

### SELECT cust-name,branch-name FROM SAVINGS

<table>
<thead>
<tr>
<th>cust-name</th>
<th>Branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>
Eliminating Duplicates in SQL

- **Question:** Why are not duplicates eliminated in SQL?

- **Answer:**
  - **Efficiency** (duplicate elimination may take quadratic time).
  - **Necessity:** In certain cases, eliminating duplicates results in information loss and errors in computing averages.

- **Question:** What if we want to eliminate duplicates in SQL?

- **Answer:** Use the construct:
  
  ```sql
  SELECT DISTINCT <attribute list>
  FROM <relation list>
  WHERE <condition>
  ```
The Relational Completeness of SQL

- **Fact:** SQL is a relationally complete database query language
- **Reason:**
  - The `SELECT DISTINCT ... FROM ... WHERE ...` construct of SQL makes it possible to express cartesian product, projection, and selection.
  - In addition, SQL has explicit constructs for union and difference:
    - The union \( R \cup S \) of two relations \( R \) and \( S \) is expressed by
      \[
      (\text{SELECT} * \text{FROM} \ R) \text{ UNION } (\text{SELECT} * \text{FROM} \ S)
      \]
    - The difference \( R - S \) of two relations \( R \) and \( S \) is expressed by
      \[
      (\text{SELECT} * \text{FROM} \ R) \text{ EXCEPT } (\text{SELECT} * \text{FROM} \ S)
      \]
    - UNION and EXCEPT eliminates duplicates! (Set semantics)
    - UNION ALL and EXCEPT ALL does not. (Multiset semantics)
Relational Calculus

- In addition to relational algebra, Codd introduced relational calculus.
- Relational calculus is a declarative database query language based on first-order logic.
- Codd’s main technical result is that relational algebra and relational calculus have essentially the same expressive power.
- More precisely, relational algebra has the same expressive power as safe relational calculus.
- Relational calculus comes into two different flavors:
  - Tuple relational calculus (this is the version studied by Codd)
  - Domain relational calculus.
- Here, we will focus on domain relational calculus (which is the version considered in the AHV textbook). Refer to Codd’s 1972 paper for the precise definition of tuple relational calculus.