Announcements

• Assignment 1

• Hints and Grading
Today’s Lecture

1. The Relational Model & Relational Algebra

2. Relational Algebra Pt. II
1. The Relational Model & Relational Algebra
What you will learn about in this section

1. The Relational Model

2. Relational Algebra: Basic Operators
Levels of abstraction

- **External Schema**: schema seen by apps
- **Conceptual Schema**: a.k.a. logical schema describes stored data in terms of data model
- **Physical Schema**: includes storage details, file organization, indexes

Classical picture. Remember it!
Motivation

The Relational model is **precise**, **implementable**, and we can operate on it (query/update, etc.)

Database maps internally into this **procedural language**.
The Relational Model: Schemata

• Relational Schema:

```
Students(sid: string, name: string, gpa: float)
```

Relation name

String, float, int, etc. are the **domains** of the attributes

Attributes
The Relational Model: Data

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>002</td>
<td>Joe</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
<td>3.8</td>
</tr>
<tr>
<td>004</td>
<td>Alice</td>
<td>3.5</td>
</tr>
</tbody>
</table>

The number of attributes is the **arity** of the relation.
The Relational Model: Data

### Student

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>002</td>
<td>Joe</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
<td>3.8</td>
</tr>
<tr>
<td>004</td>
<td>Alice</td>
<td>3.5</td>
</tr>
</tbody>
</table>

The number of tuples is the **cardinality** of the relation.

A **tuple or row** (or **record**) is a single entry in the table having the attributes specified by the schema.
A relational instance is a set of tuples all conforming to the same schema.
To Reiterate

• A *relational schema* describes the data that is contained in a *relational instance*

Let $R(f_1 : \text{Dom}_1, \ldots, f_m : \text{Dom}_m)$ be a *relational schema* then, an *instance* of $R$ is a subset of $\text{Dom}_1 \times \text{Dom}_2 \times \ldots \times \text{Dom}_n$

In this way, a *relational schema* $R$ is a *total function from attribute names to types*
A relational schema describes the data that is contained in a relational instance.

A relation R of arity t is a function: $R : \text{Dom}_1 \times \ldots \times \text{Dom}_t \rightarrow \{0,1\}$

I.e. returns whether or not a tuple of matching types is a member of it.

Then, the schema is simply the signature of the function.

Note here that order matters, attribute name doesn’t… We’ll (mostly) work with the other model (last slide) in which attribute name matters, order doesn’t!
A relational database

• A *relational database schema* is a set of relational schemata, one for each relation

• A *relational database instance* is a set of relational instances, one for each relation

Two conventions:
1. We call relational database instances as simply *databases*
2. We assume all instances are valid, i.e., satisfy the *domain constraints*
Remember the CMS

• Relation DB Schema
  • Students(sid: string, name: string, gpa: float)
  • Courses(cid: string, cname: string, credits: int)
  • Enrolled(sid: string, cid: string, grade: string)

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Bob</td>
<td>3.2</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>564</td>
<td>564-2</td>
<td>4</td>
</tr>
<tr>
<td>308</td>
<td>417</td>
<td>2</td>
</tr>
</tbody>
</table>

Note that the schemas impose effective domain/type constraints, i.e. Gpa can’t be “Apple”
Part of the Model: Querying

```
SELECT S.name
FROM Students S
WHERE S.gpa > 3.5;
```

“We don’t tell the system how or where to get the data—just what we want, i.e., Querying is declarative.”

To make this happen, we need to translate the declarative query into a series of operators... we’ll see this next!
Virtues of the model

• Physical independence (logical too), Declarative

• Simple, elegant clean: Everything is a relation
Relational Algebra
RDBMS Architecture

How does a SQL engine work?

1. **SQL Query**
   - Declarative query (from user)

2. **Relational Algebra (RA) Plan**
   - Translate to relational algebra expression

3. **Optimized RA Plan**
   - Find logically equivalent - but more efficient - RA expression

4. **Execution**
   - Execute each operator of the optimized plan!
RDBMS Architecture

How does a SQL engine work?

Relational Algebra (RA) Plan

Optimized RA Plan

Execution

Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!
Relational Algebra (RA)

• **Five basic operators:**
  1. Selection: $\sigma$
  2. Projection: $\Pi$
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: $-$

• **Derived or auxiliary operators:**
  • Intersection, complement
  • Joins (natural, equi-join, theta join, semi-join)
  • Renaming: $\rho$
  • Division
Keep in mind: RA operates on sets!

• RDBMSs use *multisets*, however in relational algebra formalism we will consider *sets*!

• Also: we will consider the *named perspective*, where every attribute must have a *unique name*
  • →attribute order does not matter...

Now on to the basic RA operators...
1. Selection ($\sigma$)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary}>40000}$ (Employee)
  - $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
- The condition $c$ can be $=, <, \leq, >, \geq, <>$
Another example:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{Salary > 40000}} \text{ (Employee)} \]
2. Projection ($\Pi$)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A_1,...,A_n}(R)$
- Example: project social-security number and names:
  - $\Pi_{\text{SSN, Name}}(\text{Employee})$
  - Output schema: Answer(\text{SSN, Name})

SQL:
```
SELECT DISTINCT sname, gpa
FROM Students;
```

RA:
$\Pi_{\text{name}, \text{gpa}}(\text{Students})$
Another example:

\( \Pi_{Name,Salary} (Employee) \)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>
Note that RA Operators are Compositional!

SELECT DISTINCT sname, gpa FROM Students WHERE gpa > 3.5;

\[ \Pi_{sname, gpa}(\sigma_{gpa > 3.5}(Students)) \]

\[ \sigma_{gpa > 3.5}(\Pi_{sname, gpa}(Students)) \]

How do we represent this query in RA?

Are these logically equivalent?
3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: R1 × R2
- Example:
  - Employee × Dependents
- Rare in practice; mainly used to express joins

SQL:
```
SELECT *
FROM Students, People;
```

RA:
```
Students × People
```
Another example:

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

**Students × People**

<table>
<thead>
<tr>
<th>ssn</th>
<th>pname</th>
<th>address</th>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>216 Rosse</td>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Renaming ($\rho$)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B_1, \ldots, B_n}(R)$

- **Note:** this is shorthand for the proper form (since names, not order matters!):
  - $\rho_{A_1 \rightarrow B_1, \ldots, A_n \rightarrow B_n}(R)$

SQL:

```sql
SELECT sid AS studId, sname AS name, gpa AS gradePtAvg
FROM Students;
```

RA:

$$\rho_{\text{studId}, \text{name}, \text{gradePtAvg}}(\text{Students})$$

We care about this operator because we are working in a named perspective
Another example:

\[ \rho_{\text{studId}, \text{name}, \text{gradePtAvg}}(\text{Students}) \]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|c|}
\hline
\text{studId} & \text{name} & \text{gradePtAvg} \\
\hline
001 & John & 3.4 \\
002 & Bob & 1.3 \\
\hline
\end{array}
\]
Natural Join (⋈)

- Notation: \( R_1 \bowtie R_2 \)

- Joins \( R_1 \) and \( R_2 \) on equality of all shared attributes
  - If \( R_1 \) has attribute set \( A \), and \( R_2 \) has attribute set \( B \), and they share attributes \( A \cap B = C \), can also be written: \( R_1 \bowtie_C R_2 \)

- Our first example of a derived RA operator:
  - Meaning: \( R_1 \bowtie R_2 = \Pi_{A \cup B} (\sigma_{C=D} (\rho_{C\rightarrow D} (R_1 \times R_2))) \)
  - Where:
    - The rename \( \rho_{C\rightarrow D} \) renames the shared attributes in one of the relations
    - The selection \( \sigma_{C=D} \) checks equality of the shared attributes
    - The projection \( \Pi_{A \cup B} \) eliminates the duplicate common attributes

SQL:

```
SELECT DISTINCT ssid, S.name, gpa, ssn, address
FROM Students S,
     People P
WHERE S.name = P.name;
```

RA:

```
Students(sid,name,gpa)
People(ssn,name,address)
```

```
Students \bowtie People
```
Another example:

**Students S**

<table>
<thead>
<tr>
<th>sid</th>
<th>S.name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
</tr>
</tbody>
</table>

**People P**

<table>
<thead>
<tr>
<th>ssn</th>
<th>P.name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>5423341</td>
<td>Bob</td>
<td>217 Rosse</td>
</tr>
</tbody>
</table>

Students \(\bowtie\) People
Natural Join

• Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$ ?

• Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$ ?

• Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$ ?
Example: Converting SQL Query -> RA

\[
\Pi_{gpa, address}(\sigma_{gpa>3.5}(S \bowtie P))
\]

\[
\begin{align*}
\text{Students}(\text{sid, sname, gpa}) \\
\text{People}(\text{ssn, sname, address})
\end{align*}
\]

\[
\begin{align*}
\text{SELECT DISTINCT} \\
gpa, \text{address} \\
\text{FROM Students S,} \\
\text{People P} \\
\text{WHERE gpa > 3.5 AND} \\
\text{sname = pname;}
\end{align*}
\]
Logical Equivalence of RA Plans

• Given relations R(A,B) and S(B,C):
  
  • Here, projection & selection commute:
    • \( \sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R)) \)

  • What about here?
    • \( \sigma_{A=5}(\Pi_B(R)) \neq \Pi_B(\sigma_{A=5}(R)) \)
1. Union (∪) and 2. Difference (−)

- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees
- R1 – R2
- Example:
  - AllEmployees -- RetiredEmployees
What about Intersection (\(\cap\))?

- It is a derived operator
- \(R_1 \cap R_2 = R_1 - (R_1 - R_2)\)
- Also expressed as a join!
- Example
  - UnionizedEmployees \(\cap\) RetiredEmployees
RA Expressions Can Get Complex!

\[ \Pi_{\text{name}} \]

buyer-ssn=ssn

\[ \Pi_{\text{ssn}} \]

\[ \sigma_{\text{name}=\text{fred}} \]

\[ \Pi_{\text{pid}} \]

\[ \sigma_{\text{name}=\text{gizmo}} \]
RA has Limitations!

• Cannot compute “transitive closure”

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Mary</td>
<td>Father</td>
</tr>
<tr>
<td>Mary</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>Mary</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>Nancy</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

• Find all direct and indirect relatives of Fred

• Cannot express in RA !!!
  • Need to write C program, use a graph engine, or modern SQL...