The Relational Model
Today’s Lecture

1. The Relational Model & Relational Algebra

2. Relational Algebra Pt. II  [Optional: may skip]
1. The Relational Model & Relational Algebra
What you will learn about in this section

1. The Relational Model
2. Relational Algebra: Basic Operators
3. Execution
4. ACTIVITY: From SQL to RA & Back
Motivation

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

Database maps internally into this **procedural language**.
A Little History

• Relational model due to Edgar “Ted” Codd, a mathematician at IBM in 1970
  • A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6): 377–387

• IBM didn’t want to use relational model (take money from IMS)
  • *Apparently used in the moon landing*...
The Relational Model: Schemata

• Relational Schema:

Students(sid: string, name: string, gpa: float)
An **attribute** (or **column**) is a typed data entry present in each tuple in the relation.

The number of attributes is the **arity** of the relation.

### The Relational Model: Data

<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 Relational Model: Data

A **tuple** or **row** (or record) is a single entry in the table having the attributes specified by the schema.

<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.
The Relational Model: Data

<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>

Recall: In practice, DBMSs relax the set requirement, and use multisets.

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*
One More Time

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

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

  *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!*

Lecture 16 > Section 1 > The Relational Model
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>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>564</td>
<td>A</td>
</tr>
</tbody>
</table>

Note that the schemas impose effective domain/type constraints, i.e. Gpa can’t be “Apple”
2nd Part of the Model: Querying

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

“Find names of all students with 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!

Actually, I showed how to do this translation for a much richer language!
Virtues of the model

• Physical independence (logical too), Declarative

• Simple, elegant clean: Everything is a relation

• Why did it take multiple years?
  • Doubted it could be done efficiently.
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 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

We’ll look at these first!

And also at one example of a derived operator (natural join) and a special operator (renaming)
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
  - \(\sigma_{\text{Salary} > 40000}(\text{Employee})\)
  - \(\sigma_{\text{name} = \text{“Smith”}}(\text{Employee})\)
- The condition c can be =, <, \(\leq\), >, \(\geq\), \(<>\)

**SQL:**
```
SELECT * 
FROM Students 
WHERE gpa > 3.5;
```

**RA:**
```
\(\sigma_{gpa > 3.5}(\text{Students})\)
```
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>

\[ \sigma_{\text{Salary} > 40000} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>
2. Projection ($\Pi$)

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

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

RA:
```
\Pi_{\text{sname}, \text{gpa}}(\text{Students})
```
Another example:

\[ \Pi_{\text{Name,Salary}} \text{(Employee)} \]

<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!

Students(sid, sname, gpa)

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

$\Pi_{\text{sname}, gpa}(\sigma_{gpa>3.5}(\text{Students}))$

$\sigma_{gpa>3.5}(\Pi_{\text{sname}, gpa}(\text{Students}))$

How do we represent this query in RA?

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

- Each tuple in \( R_1 \) with each tuple in \( R_2 \)
- Notation: \( R_1 \times R_2 \)
- Example:
  - Employee \( \times \) Dependents
- Rare in practice; mainly used to express joins

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

**RA:**
```
Students(sid,sname,gpa)
People(ssn,pname,address)
```

\( Students \times People \)
### Another example:

**People**

<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>

**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>

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

RA:
```
\rho_{studId,name,gradePtAvg}(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>studId</th>
<th>name</th>
<th>gradePtAvg</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>
Natural Join ($\bowtie$)

- **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:**

\[
\text{SELECT DISTINCT ssid, S.name, gpa, ssn, address}
\text{FROM Students S, People P}
\text{WHERE S.name = P.name;}
\]

**RA:**

\[
\text{Students} \bowtie \text{People}
\]
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

<table>
<thead>
<tr>
<th>sid</th>
<th>S.name</th>
<th>gpa</th>
<th>ssn</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>John</td>
<td>3.4</td>
<td>1234545</td>
<td>216 Rosse</td>
</tr>
<tr>
<td>002</td>
<td>Bob</td>
<td>1.3</td>
<td>5423341</td>
<td>216 Rosse</td>
</tr>
</tbody>
</table>
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 SFW Query -> RA

Students(sid,sname,gpa)
People(ssn,sname,address)

SELECT DISTINCT
  gpa,
  address
FROM Students S,
    People P
WHERE gpa > 3.5 AND
  sname = pname;

How do we represent this query in RA?

Π_{gpa,address}(σ_{gpa>3.5}(S \bowtie P))
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?
    • $\sigma_{A=5}(\Pi_B(R)) ?= \Pi_B(\sigma_{A=5}(R))$

We’ll look at this in more depth later in the lecture...
RDBMS Architecture

How does a SQL engine work?

We saw how we can transform declarative SQL queries into precise, compositional RA plans.
RDBMS Architecture

How does a SQL engine work?

SQL Query → Relational Algebra (RA) Plan → Optimized RA Plan → Execution

We’ll look at how to then optimize these plans later in this lecture.
RDBMS Architecture

How is the RA “plan” executed?

We already know how to execute all the basic operators!
RA Plan Execution

• Natural Join / Join:
  • We saw how to use memory & IO cost considerations to pick the correct algorithm to execute a join with (BNLJ, SMJ, HJ...).

• Selection:
  • We saw how to use indexes to aid selection
  • Can always fall back on scan / binary search as well

• Projection:
  • The main operation here is finding distinct values of the project tuples; we briefly discussed how to do this with e.g. hashing or sorting

We already know how to execute all the basic operators!
Activity-16-1.ipynb
2. Adv. Relational Algebra
What you will learn about in this section

1. Set Operations in RA
2. Fancier RA
3. Extensions & Limitations
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

We’ll look at these

And also at some of these derived operators
1. Union (\(\cup\)) and 2. Difference (\(-\))

- R1 \(\cup\) R2
  - Example:
    - ActiveEmployees \(\cup\) 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
Fancier RA
Theta Join ($\bowtie_\theta$)

- A join that involves a predicate
- $R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2)$
- Here $\theta$ can be any condition

Note that natural join is a theta join + a projection.
Equi-join (⋈<sub>A=B</sub>)

- A theta join where θ is an equality
- R1 ⋈<sub>A=B</sub> R2 = σ<sub>A=B</sub> (R1 x R2)
- Example:
  - Employee ⋈<sub>SSN=SSN</sub> Dependents

Most common join in practice!
Semijoin (⋉)

• $R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie S)$
• Where $A_1, \ldots, A_n$ are the attributes in $R$
• Example:
  • Employee $\bowtie$ Dependents

SQL:

```sql
SELECT DISTINCT sid, sname, gpa
FROM Students, People
WHERE sname = pname;
```

RA:

$Students \bowtie People$
Semijoins in Distributed Databases

- Semijoins are often used to compute natural joins in distributed databases

\[
\text{Employee} \bowtie_{\text{ssn} = \text{ssn}} (\sigma_{\text{age} > 71} (\text{Dependents}))
\]

\[
\begin{align*}
R &= \text{Employee} \bowtie T \\
T &= \Pi_{\text{SSN}} \sigma_{\text{age} > 71} (\text{Dependents}) \\
\text{Answer} &= R \bowtie \text{Dependents}
\end{align*}
\]

Send less data to reduce network bandwidth!
RA Expressions Can Get Complex!

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

\( \sigma_{\text{name}=\text{fred}} \)

\( \sigma_{\text{name}=\text{gizmo}} \)

\( \Pi_{\text{pid}} \)

\( \Pi_{\text{ssn}} \)

\( \Pi_{\text{pid}} \)

\( \Pi_{\text{ssn}} \)

\( \sigma_{\text{buyer-ssn}=\text{ssn}} \)

\( \sigma_{\text{seller-ssn}=\text{ssn}} \)

Person  Purchase  Person  Product
Multisets
Recall that SQL uses Multisets

<table>
<thead>
<tr>
<th>Multiset X</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tuple</strong></td>
</tr>
<tr>
<td>(1, a)</td>
</tr>
<tr>
<td>(1, a)</td>
</tr>
<tr>
<td>(1, b)</td>
</tr>
<tr>
<td>(2, c)</td>
</tr>
<tr>
<td>(2, c)</td>
</tr>
<tr>
<td>(2, c)</td>
</tr>
<tr>
<td>(1, d)</td>
</tr>
<tr>
<td>(1, d)</td>
</tr>
</tbody>
</table>

Equivalent Representations of a **Multiset**

\[ \lambda(X) = \text{“Count of tuple in } X \text{”} \]

(Items not listed have implicit count 0)

<table>
<thead>
<tr>
<th>Multiset X</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tuple</strong></td>
</tr>
<tr>
<td>(1, a)</td>
</tr>
<tr>
<td>(1, b)</td>
</tr>
<tr>
<td>(2, c)</td>
</tr>
<tr>
<td>(1, d)</td>
</tr>
</tbody>
</table>

Note: In a set all counts are \{0,1\}. 
Generalizing Set Operations to Multiset Operations

For sets, this is intersection

$$\lambda(Z) = \min(\lambda(X), \lambda(Y))$$
Generalizing Set Operations to Multiset Operations

Multiset \( X \)  
<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>2</td>
</tr>
<tr>
<td>(1, b)</td>
<td>0</td>
</tr>
<tr>
<td>(2, c)</td>
<td>3</td>
</tr>
<tr>
<td>(1, d)</td>
<td>0</td>
</tr>
</tbody>
</table>

Multiset \( Y \)  
<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(Y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>5</td>
</tr>
<tr>
<td>(1, b)</td>
<td>1</td>
</tr>
<tr>
<td>(2, c)</td>
<td>2</td>
</tr>
<tr>
<td>(1, d)</td>
<td>2</td>
</tr>
</tbody>
</table>

\( \cup \)  

Multiset \( Z \)  
<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(Z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>7</td>
</tr>
<tr>
<td>(1, b)</td>
<td>1</td>
</tr>
<tr>
<td>(2, c)</td>
<td>5</td>
</tr>
<tr>
<td>(1, d)</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \lambda(Z) = \lambda(X) + \lambda(Y) \]

For sets, this is union
Operations on Multisets

All RA operations need to be defined carefully on bags

- $\sigma_C(R)$: preserve the number of occurrences
- $\Pi_A(R)$: no duplicate elimination
- Cross-product, join: no duplicate elimination

This is important- relational engines work on multisets, not sets!
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...