Foundation of Relational Databases

Yanlei Diao
Databases and DBMS’s

- A **database** is a large, integrated collection of data
- A **database management system** (DBMS) is a software system designed to store and manage a large amount of data
  - *Declarative interface* to define, add/update, and query data
  - *Efficient querying*
  - *Concurrent users*
  - *Crash recovery*
  - *Access control*…
What Type of Data is stored?

- All critical business data!
  - Banking
  - Ticketing
  - Retail
  - Electronic commerce
  - Insurance
  - Healthcare
  - Enterprise HR
  - Government
  - Telecommunications
  - Social networks

ORACLE CUSTOMERS INCLUDE
- 10 of the 10 top aerospace and defense companies
- 20 of the 20 top airlines
- 20 of the 20 top automotive companies
- 20 of the 20 top banks
- 9 of the 10 top consumer goods companies
- 9 of the 10 top engineering and construction companies
- 20 of the 20 top governments
- 20 of the 20 top high tech companies
- 20 of the 20 top insurers
- 20 of the 20 top manufacturers
- 20 of the 20 top oil and gas companies
- 20 of the 20 top pharmaceutical companies
- 20 of the 20 top retailers
- 10 of the 10 top SaaS providers
- 20 of the 20 top supply chains
- 20 of the 20 top telecommunications companies
- 20 of the 20 top universities
- 10 of the 10 top utilities
Early DBMS’s

- Many small data items, many queries and updates
  - Banking, airline reservations

- 1960s Navigational DBMS
  - Tree / graph-based data model
  - Manual navigation to find what you want
  - Support for “search” = “programming”

- 1973 Turing Award Winner
  - Charles William Bachman
  - “The Programmer as Navigator”
  - The network data model
Relational DBMS’s

- Relational model (1970)
  - Data independence: hides details of physical storage from users
  - Declarative query language: say what you want, not how to compute it
  - Mathematical foundation: what queries mean, possible implementations

- 1981 Turing Award Winner
  - Edgar F. (“Ted”) Codd
  - Mathematically-inclined researcher
  - Legitimized DBMS’s as a theoretically respectable research field in CS
Relational DBMS

- **Query optimization (1970’s till now)**
  - Earliest: System R at IBM, INGRES at UC Berkeley
  - Queries can be efficiently executed despite data independence and declarative queries!

- **2014 Turing Award Winner**
  - Michael Stonebraker (Berkeley / MIT)
  - “For fundamental contributions to modern database systems”
Evolution of DBMS's

INGRES
UC Berkeley, Stonebraker et al

System R
IBM San Jose, Gray, Selinger et al

Informix
Postgres

Sybase
MS SQL Server

IBM DB2
Oracle

MySQL
The Picture Today (Gartner 2015)
Foundation of Relational Databases
Foundation of Relational Databases

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory
Relational Model

- A relational database is a set of *relations*.
- Each relation has:
  - **Schema**: specifies name of relation, name and type (domain) of each attribute.
  - **Instance**: a table with rows (*tuples*) and columns (*attributes, fields*).
    - *cardinality* = #rows, *degree / arity* = #columns.
- A relation is a *set* of tuples (in theory).
  - All rows must be distinct, no duplicates.
### Example Instance of Students Relation

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = 3, degree = 5
- All rows are distinct.
- Some columns of two rows can be the same.
Creating Relations in SQL

- Create the Students relation

- Specify domain constraints:
  - type of each field
  - later enforced by the DBMS upon tuple insertion or update.

```
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL);
```

```
CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2));
```
Adding Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2);
```

- Powerful variants of these commands are available; more later!
Integrity Constraints

- **Integrity Constraints** (IC’s): condition that must be true for any instance of the database.
  - Domain constraint
  - Primary key constraint
  - Foreign key constraint
  - Specified when the schema is defined.

- DBMS enforces ICs.
  - Stored data is faithful to real-world meaning.
  - Avoids data entry errors, too!
Primary Key Constraints

- **Key** of a relation: *minimum* set of attributes that uniquely identify each entity.
  1. No two tuples can have same values in all key fields.
  2. This is not true for any subset of the key.
     - Part 2 false? A *superkey*.
     - If more than 1 key for a relation, *candidate keys*.
     - One of candidate keys is chosen to be the *primary key*.

- E.g., Students(sid, name, login, age, gpa)
Primary and Candidate Keys in SQL

- Specify candidate keys using \texttt{UNIQUE}.
- Choose one candidate key as the \textit{primary key}.

“For a given student and course, there is a single grade.”

“... and no two students in a course receive the same grade.”

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
cid  CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid));
```

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
cid  CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
UNIQUE (cid, grade) );
```
Foreign Keys

- **Foreign key**: set of fields used to `refer` to the primary key of another relation.
  - Like a `logical pointer`.
  - *sid* is a foreign key referring to Students.
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
    (sid CHAR(20), cid CHAR(20), grade CHAR(2),
     PRIMARY KEY (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students );
```

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>53666</td>
<td>Carnatic101</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
</tr>
</tbody>
</table>
Referential Integrity

- **Referential integrity**: any foreign key value must have a matching primary key value in referenced reln.
  - E.g., every *sid* value in Enrolled must appear in Students.
  - No dangling references.

- Can you name a data model without referential integrity?
Enforcing Referential Integrity

- What if an Enrolled tuple with a non-existent student id is inserted?
  - Reject it!

- What if a Students tuple is deleted?
  - CASCADE: delete all Enrolled tuples that refer to it.
  - NO ACTION: disallow if the Students tuple is referred to.
  - SET DEFAULT: set the foreign key to a default sid.
  - SET NULL: set the foreign key to a special value null, denoting `unknown` or `inapplicable`.

- Updates to sid in Students are treated similarly.
Referential Integrity in SQL

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
    REFERENCES Students (sid)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);
Where do IC’s Come From?

- Based upon real-world business logic.
- Can check violation in a database instance, but can **NEVER** infer an IC by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - E.g., *name* of the Students relation.
Outline

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory
Relational Query Languages

- **Relational model** allows simple, powerful querying of data.

- **Relational query languages**
  - *Declarative*: say “what you want”, not “how to get it”
  - *Formal mathematical foundation*
  - *Query optimization*
Formal Relational Query Languages

- Two mathematical languages form the basis for the “real” one, SQL, and for implementation:
  - **Relational Algebra**: operational, useful for representing execution plans.
  - **Relational Calculus**: declarative, useful for defining query semantics.
What is “Relational Algebra”

- **Relational algebra:**
  - Operands are relations.
  - Operators each take 1 or 2 relations and produce a relation.

- **Closure property:** relational algebra is closed under the relational model.
  - Relational operators can be arbitrarily composed!
Relational Algebra

Basic operations:

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

Additional operations:

- **Join** (\(\Join\)), **Intersection** (\(\cap\)), **Division** (\(/\)), **Renaming** (\(\rho\))
- Can be derived from basic operators. Not essential, but useful!
## Example Instances

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S1</strong></td>
<td><strong>R1</strong></td>
</tr>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>S2</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>
**Projection**

- Retain only attributes in the *projection list*; delete others.
- *Schema of result* contains exactly the fields in projection list.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{name}, \text{rating}}(S2)
\]
Projection (contd.)

- Projection operator has to eliminate *duplicates*!
  - SQL (real) systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)

**$S_2$**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

$\pi_{age}(S_2)$

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>55.5</td>
</tr>
</tbody>
</table>
Selection

- Select rows that satisfy the *selection condition*; discard others.

*Schema of result* identical to schema of input.

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[ \sigma \text{rating} > 8^{(S2)} \]
Selection (contd.)

- Composition: result relation of an operator can be the input to another operator.

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{rating} > 8}^{(S2)} \]

\[ \pi_{\text{aname, rating}}(\sigma_{\text{rating} > 8}^{(S2)}) \]
Set operations:
- Union (∪)
- Intersection (∩)
- Set difference (−)

Two input relations must be union-compatible:
- Same number of fields.
- Corresponding fields have the same type.

What is the schema of result?

<table>
<thead>
<tr>
<th>S1</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
### Example Set Operations

**$S_1$**

<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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**$S_2$**

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>guppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**$S_1 \cup S_2$**

<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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>28</td>
<td>guppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

*Duplicate elimination*: remove tuples that have same values in all attributes.
Example Set Operations

$S1$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

$S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

$S1 − S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</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>
</tbody>
</table>

$S1 \cap S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Duplicates?
**Cross (Cartesian) Product**

- **S1 \times R1:** each row of S1 is paired with each row of R1.

<table>
<thead>
<tr>
<th>S1</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Cross-Product (contd.)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(sid)</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
<td>(sid)</td>
<td>bid</td>
<td>day</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

- **Result schema** inherits all fields of S1 and R1.
  - **Conflict**: Both S1 and R1 have a field called *sid*.
- **Renaming operator**: $\rho (C(1 \rightarrow sid_1, 5 \rightarrow sid_2), S1 \times R1)$
Joins

- **Condition (theta) Join**: \( R \bowtie_c S = \sigma_c (R \times S) \)

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\( S_1 \bowtie S_1.sid < R_1.sid R_1 \)

- Result schema same as that of cross-product.
- But often fewer tuples, more efficient for computation.
**Joins**

- **Equi-Join**: A special case of condition join where the condition $\theta$ contains only *equalities*

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\[ S \bowtie_{sid} R \]

- **Result schema** contains only one copy of fields for which equality is specified.

- **Natural Join** ($R \bowtie S$): equijoin on all common fields
Example Schema

- **Sailors** *(sid: integer, sname: string, rating: integer, age: integer)*
- **Boats** *(bid: integer, color: string)*
- **Reserves** *(sid: integer, bid: integer, day: date)*
Find names of sailors who’ve reserved boat #103

- **Sailors** *(sid: integer, sname: string, rating: integer, age: integer)*
- **Boats** *(bid: integer, color: string)*
- **Reserves** *(sid: integer, bid: integer, day: date)*

1) How many relations do we need?
2) How do we compare two relations?
3) Where do we place the selection?
4) Which attributes are retained for output?
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: 
  \( \rho (\text{Temp1, } \sigma_{\text{bid} = 103} \text{Reserves}) \)
  
  \( \rho (\text{Temp2, Temp1} \bowtie \text{Sailors}) \)
  
  \( \pi_{\text{name}} (\text{Temp2}) \)

- Solution 3: 
  \( \pi_{\text{name}}(\sigma_{\text{bid} = 103}(\text{Reserves} \bowtie \text{Sailors})) \)

Algebraic equivalence!
Find names of sailors who’ve reserved a red boat

- **Boats** (*bid*: integer, *color*: string)
- **Reserves** (*sid*: integer, *bid*: integer, *day*: date)

- Boat color is only available in Boats; so need an extra join:

\[
\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)
\]
Find sailors who ’ve reserved a red or a green boat

- Can identify all red or green boats, then find sailors who ’ve reserved one of these boats:

\[ \rho \left( Tempboats, (\sigma_{\text{color} = 'red' \lor \text{color} = 'green'} Boats) \right) \]

\[ \pi_{\text{name}} (Tempboats \bowtie\bowtie Reserves \bowtie\bowtie Sailors) \]

- Can also define Tempboats using union. How?
Find sailors who’ve reserved a red and a green boat

- Will a single selection work?
- Instead, intersect sailors who’ve reserved red boats and sailors who’ve reserved green boats.

\[ \rho (\text{Tempred}, \pi_{\text{sid}} ((\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Reserves})) \]

\[ \rho (\text{Tempgreen}, \pi_{\text{sid}} ((\sigma_{\text{color} = 'green'} \text{Boats}) \bowtie \text{Reserves})) \]

\[ \pi_{\text{name}}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors}) \]
Find the names of sailors who’ve reserved all boats

- **Sailors** *(sid: integer, sname: string, rating: integer, age: integer)*
- **Boats** *(bid: integer, color: string)*
- **Reserves** *(sid: integer, bid: integer, day: date)*

**Relational algebra**

- **Selection** *(σ)* Selects a subset of rows from a relation.
- **Projection** *(π)* Retains a subset of columns in a relation.
- **Cross-product** *(×)* Allows us to combine two relations.
- **Set-difference** *(−)* Tuples in reln. 1, but not in reln. 2.
- **Union** *(∪)* Tuples in reln. 1 or in reln. 2.
- **Join** *(⋈)*, **Intersection** *(∩)*, **Division** *(÷)*, **Renaming** *(ρ)*
Outline

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory
Relational Calculus

- **Query** has the form:

\[ \{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1, x_2, \ldots, x_n \rangle) \} \]

  - domain variables, or constants
  - formula

- **Answer** includes all tuples \( \langle x_1, x_2, \ldots, x_n \rangle \) that make the formula \( p(\langle x_1, x_2, \ldots, x_n \rangle) \) true.
Formulas

- Formula is recursively defined:
  - Atomic formulas:
    - getting tuples from relations, or
    - making comparisons of values
  - Logical connectives: ¬, ∧, ∨
  - Quantifiers: ∃, ∀
Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to bind $X$.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

$$\left\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1, x_2, \ldots, x_n \rangle) \right\}$$

- There is an important restriction: the variables $x_1, \ldots, x_n$ that appear to the left of ‘$|$’ must be the only free variables in the formula $p(...)$.
Find names of sailors rated > 7 who have reserved boat #103

**Relational Algebra:**
\[ \pi_{sname}( (\sigma_{bid=103} \text{Reserves}) \bowtie (\sigma_{rating>7} \text{Sailors}) ) \]

**Relational Calculus:**
\[ \{ X_{sname} \mid \exists X_{sid}, X_{rating}, X_{age} \text{Sailors}(X_{sid}, X_{sname}, X_{rating}, X_{age}) \land X_{rating} > 7 \land \exists X_{bid}, X_{day} \text{Reserves}(X_{sid}, X_{bid}, X_{day}) \land X_{bid}=103 \} \]

- Where is the join?
  - Use \( \exists \) to find a tuple in Reserves that ‘joins with’ the Sailors tuple under consideration.
Find names of sailors who’ve reserved all boats

\{X_{\text{sname}} \mid \exists X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{sid}}, X_{\text{sname}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\
\quad \forall \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
\quad \quad (\exists X_{\text{day}} \langle X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves}) \} \}

- To find sailors who’ve reserved all red boats:

\{X_{\text{sname}} \mid \exists X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{sid}}, X_{\text{sname}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\
\quad \forall \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
\quad \quad (X_{\text{color}} \neq \text{'red'} \lor \exists X_{\text{day}} \langle X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves}) \} \}

\[
p \rightarrow q \equiv \neg p \lor q
\]
Find names of sailors who’ve reserved all boats

\{ X_{\text{name}} \mid \exists X_{\text{id}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{id}}, X_{\text{name}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\
\forall \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
(\exists X_{\text{day}} \langle X_{\text{id}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves}) \}

\forall x \in R \ F(x) \equiv \neg \exists x \in R \ \neg F(x)

\{ X_{\text{name}} \mid \exists X_{\text{id}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{id}}, X_{\text{name}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\
\neg \exists \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
(\neg \exists X_{\text{day}} \langle X_{\text{id}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves}) \}

How do we write it in Relational Algebra?
Find the names of sailors who’ve reserved all boats

- Step 1: find all sailors such that there exists a boat that he has not reserved (called formula F).

\[ \rho(S_{neg}, \pi_{sid}((\pi_{sid}\text{Reserves}) \times (\pi_{bid}\text{Boats}) - (\pi_{sid,bid}\text{Reserves}))) \]

- Step 2: find sailors for which F is not true and retrieve their names

\[ \pi_{sname}((\pi_{sid}\text{Reserves} - S_{neg}) \bowtie\bowtie\text{Sailors}) \]

‘-‘: the only way to express negation in relational algebra!