Relational Model

- It is the most popular implementation model
  - Simplest, most uniform data structures
  - Most formal (algebra to describe operations)
- Introduced in 1970 (by E. F. Codd)
- Everything from real world is represented by relations (i.e. tables)
- Each table has multiple rows and columns
  - Row in a table "binds" values together (row = tuple)

Relations

- Attributes (=columns)
  - Domain: set of permitted values
  - Tuple order is not important
  - The two relations are exactly the same

<table>
<thead>
<tr>
<th>account-number</th>
<th>branch-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>Downtown</td>
<td>500</td>
</tr>
<tr>
<td>A-102</td>
<td>Perryridge</td>
<td>400</td>
</tr>
<tr>
<td>A-201</td>
<td>Brighton</td>
<td>900</td>
</tr>
<tr>
<td>A-215</td>
<td>Miami</td>
<td>700</td>
</tr>
<tr>
<td>A-217</td>
<td>Brighton</td>
<td>750</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
</tbody>
</table>

The account relation
The Mathematical Concept of Relation

- Let $D_1, D_2, ..., D_n$ be domains (not necessarily distinct)
- the Cartesian product of these $n$ sets
  
  $D_1 \times D_2 \times ... \times D_n$
  
  is the set of all possible ordered $n$-tuples
  
  $(v_1, v_2, ..., v_n)$ such that $v_1 \in D_1, v_2 \in D_2, ..., v_n \in D_n$

- Example: let $D_1 = \{\text{Nick}, \text{Susan}\}$ and $D_2 = \{\text{BS}, \text{MS}, \text{PhD}\}$
  
  $D_1 \times D_2 = \{(\text{Nick}, \text{BS}), (\text{Nick}, \text{MS}), (\text{Nick}, \text{PhD}),$
  
  $(\text{Susan}, \text{BS}), (\text{Susan}, \text{MS}), (\text{Susan}, \text{PhD})\}$

- A relation is any subset of the Cartesian product
  
  $R_1 = \{(\text{Nick}, \text{BS}), (\text{Nick}, \text{MS}), (\text{Susan}, \text{BS}), (\text{Susan}, \text{PhD})\}$
  
  $R_2 = \{}$

Relation Schema

- A relation schema specifies:
  
  - Name of relation
  - Names of attributes of the relation
  - The domain for each attribute

- Database schema = set of relation schemas, constraints (i.e. the logical design)

- Database instance = snapshot of the data in database

Relation Schema Examples

- Account-schema = (account-number, branch-name, balance)
- Branch-schema = (branch-name, branch-city, assets)
- Customer-schema = (customer-name, customer-street, customer-city)
  
  For simplicity assume customer-name unique
- Depositor-schema = (customer-name, account-number)
- Loan-schema = (loan-number, branch-name, amount)
- Borrower-schema = (customer-name, loan-number)

Keys

- Same definitions from Entity-Relationship model

  - Superkey
    
    - Set of one or more attributes that, taken collectively, uniquely identify a tuple within the relation
    
    - E.g., \{customer-name\}, \{customer-name, customer-city\}

  - Candidate key
    
    - Superkey for which no proper subset is superkey (i.e. minimal)
    
    - E.g., \{customer-name\}

  - Primary key
    
    - Candidate key chosen by database designer as principal means of identifying tuples within relation
Foreign Keys

- A relation \( r_1 \) may include among its attributes the primary key of another relation, \( r_2 \)
  - This attribute is called foreign key from \( r_1 \) referencing \( r_2 \)
  - \( r_1 \) is called the referencing relation
  - \( r_2 \) is called the referenced relation
- Example
  - loan-schema includes "branch-name" which is a primary key for branch-schema
  - therefore: branch-name is foreign key

Schema Diagram

Relational Algebra

- Procedural Query Language
- Fundamental Operators
  - Unary
    - Select
    - Project
    - Rename
  - Binary
    - Union
    - Set Difference
    - Cartesian-Product

Roadmap

- Relational Model
- Relational Schema
- Keys
- Schema Diagrams
- Relational Algebra
  - Fundamental operators
Select Operator

- select operator selects tuples that satisfy given predicate
  \( \sigma_{\text{predicate}} (\text{relation}) \)

- selection predicate:
  - comparisons: \( =, \neq, <, \leq, >, \geq \)
  - combinations: \( \land, \lor, \neg \)

Example:
- \( \sigma_{\text{amount} > 1200} (\text{loan}) \)

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-11</td>
<td>Round Hill</td>
<td>900</td>
</tr>
<tr>
<td>L-14</td>
<td>Downtown</td>
<td>1500</td>
</tr>
<tr>
<td>L-15</td>
<td>Perryridge</td>
<td>1500</td>
</tr>
<tr>
<td>L-16</td>
<td>Perryridge</td>
<td>1300</td>
</tr>
<tr>
<td>L-17</td>
<td>Downtown</td>
<td>1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Redwood</td>
<td>2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Mianus</td>
<td>500</td>
</tr>
</tbody>
</table>

Project Operator

- project operator returns relation with attributes left out
  \( \Pi_{\text{attribute-list}} (\text{relation}) \)

- attribute-list \( \subseteq \text{relation} \)

Example:
- \( \Pi_{\text{customer-name}} (\text{depositor}) \)

<table>
<thead>
<tr>
<th>customer-name</th>
<th>account-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hayes</td>
<td>A-101</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-201</td>
</tr>
<tr>
<td>Jones</td>
<td>A-217</td>
</tr>
<tr>
<td>Lindsey</td>
<td>A-222</td>
</tr>
<tr>
<td>Smith</td>
<td>A-215</td>
</tr>
<tr>
<td>Turner</td>
<td>A-305</td>
</tr>
</tbody>
</table>

Composition of Relational Operators

- Result of relational operator is a relation!
- Can arbitrary combine operators
- Relations are sets \( \rightarrow \) eliminate duplicate values

Example:
- \( \Pi_{\text{branch-name}} (\sigma_{\text{amount} > 1200} (\text{loan})) \)

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-14</td>
<td>Downtown</td>
<td>1500</td>
</tr>
<tr>
<td>L-15</td>
<td>Perryridge</td>
<td>1500</td>
</tr>
<tr>
<td>L-16</td>
<td>Redwood</td>
<td>1300</td>
</tr>
<tr>
<td>L-23</td>
<td>Redwood</td>
<td>2000</td>
</tr>
</tbody>
</table>

Union Operator

- Set operation: \( r \cup s \)
- Produces union of two sets

For union operation to be valid between \( r \) and \( s \):
- 1. both relations must be of same arity
- 2. domains of corresponding attributes must match

Commutative operation
- \( r \cup s = s \cup r \)
Set Difference Operator

- Set operation: $r - s$
- Allows us to find tuples that are in relation $r$, but not in $s$
- Relations must have same arity and matching attribute domains, as with the Union operator
- Non-commutative operation
  - $r - s \neq s - r$

Cartesian-Product Operator

- Cartesian product operator ($\times$) allows us to combine information from any two relations
- Combine attribute-lists:
  - Relation $r$, schema $R = (A, B, C)$
  - Relation $s$, schema $S = (C, D, E)$
  - $r \times s$, schema $= (r.A, r.B, r.C, s.C, s.D, s.E) = (A, B, r.C, s.C, D, E)$
- Cardinality of relation $r = \text{number of tuples in } r$
  - notation: $\text{cardinality}(r) = |r|$
- Cardinality of Cartesian product:
  - $|r \times s| = |r| \times |s|$

Rename Operator

- rename operator gives name to results
  - $\rho_{\text{new-name}}(\text{expression})$
- If we also want to rename attributes to $A_1, A_2, \ldots, A_n$
  - $\rho_{(A_1, A_2, \ldots, A_n)}(\text{expression})$

Relational Algebra (Formal Definition)

- A relational algebra expression is:
  - A relation in the database
  - A constant relation, e.g., $\{(A-101, \text{Pgh}, \$20), (A-203, \text{Oak}, \$5)\}$
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_{\text{predicate}}(E_1)$
  - $\Pi_{\text{attr-list}}(E_1)$
  - $\rho_{\text{new-name}}(E_1)$
- where $E_1$ and $E_2$ are also relational-algebra expressions
Relational Operators

- Fundamental Operators
  - Select
  - Project
  - Union
  - Set Difference
  - Cartesian Product
  - Rename Operator

- Additional Operators
  - Set Intersection
  - Natural Join
  - Division
  - Assignment

Extended Relational Operators

- Generalized Projection
- Aggregation
- Aggregation with Grouping
- Outer-Join

Set Intersection Operator

- Set operation: \( r \cap s \)
- Allows us to find tuples that are in both relations \( r \) and \( s \)
- Relations must have same arity and matching attribute domains, as with the Union operator
- \( r \cap s = r - (r - s) \)
- Question: is it commutative?

Natural-Join Operator

- Forms a cartesian product of its two argument relations
- Performs selection, forcing equality on attributes that appear on both relations
- Removes duplicate attributes
- If \( r, s \) have no common attributes then \( r \times s = r \bowtie s \)

Relational Database Example

- **Employee** (SSN, name, street, city)
- **Works** (SSN, comp-name, salary)
- **Company** (comp-name, comp-city, state)
- **Manages** (SSN, manager-SSN)
- Note: we will sometimes use the initials of the relations instead of their full names
Example Queries /1

- Find all employees that live in "Pittsburgh"
  \[ \sigma_{\text{city} = \text{"Pittsburgh"}} (\text{Employee}) \]

- Find the names of all employees that live in Pittsburgh
  \[ \Pi_{\text{name}} (\sigma_{\text{city} = \text{"Pittsburgh"}} (\text{Works})) \]

- Find the names of all the employees that work for "Blockbuster"
  \[ \Pi_{\text{name}} (\sigma_{\text{comp-name} = \text{"Blockbuster"}} (\text{Works})) \]

- OR, equivalently:
  \[ \Pi_{\text{name}} (\sigma_{\text{comp-name} = \text{"Blockbuster"}} (\text{Works})) \]

Example Queries /2

- Find the names and cities of all the employees that work for "Blockbuster"
  \[ \Pi_{\text{name,city}} (\sigma_{\text{comp-name} = \text{"Blockbuster"}} (\text{Works})) \]

- Find the names, salaries and cities of all employees that work for "Blockbuster" and make over $15,000
  \[ \Pi_{\text{name, salary, city}} (\sigma_{\text{comp-name} = \text{"Blockbuster"}} \land \text{salary} > 15000 (\text{Employee} \Join \text{Works})) \]

- Find the names of all employees who live in the same city as the company they work for
  \[ \Pi_{\text{name}} (\sigma_{\text{city} = \text{comp-city}} (\text{Employee} \Join \text{Works} \Join \text{Company})) \]

- We would not need the selection, if comp-city was named city.

\[ \Theta \in \text{join} = (\text{equi-join}) \]

\[ r \parallel_{r.A_i = s.A_j} s \]

\[ r \parallel_{r.A_i < s.A_j} s \]

\[ r \parallel_{r.A_i = s.A_j} (r \times s) \]

\[ r \parallel_{r.A_i > s.A_j} s \]

\[ r \parallel_{r.A_i \neq s.A_j} s \]

\[ r \parallel_{r.A_i < s.A_j} (r \times s) \]

- \[ \Theta \text{-Join (condition-join)} \]

- \[ \Theta \text{-Join (equi-join)} \]

- \[ \Theta \text{-Join (left-condition-join)} \]

- \[ \Theta \text{-Join (right-condition-join)} \]

- \[ \Theta \text{-Join (natural-join)} \]

\[ r \parallel_{r.A_i \geq s.A_j} s \]

\[ r \parallel_{r.A_i \leq s.A_j} s \]
Natural-Join

- Equi-join without duplicate columns
  \( r \bowtie_s s \)
- \( P = \text{list of attributes; } P = R \cap S \)
- \( r \bowtie_s s = \pi_{R \cup S}(r \bowtie_{r \cap s} r) \)
- \( r \bowtie_s s = ? \)

Division

- Let \( r(R) \) and \( s(S) \) be relations such as \( S \subseteq R \)
- The division of \( r \) by \( s \), denoted by \( r \div s \), is relation whose schema is \( Q = R - S \) and includes all \( t \) such as \( t[R] = t \) and \( t[S] = t \)

Division Usage

Query: "Retrieve the names of students who took all the classes that John took."

Division

- \( r \div s \)
- suitable for queries that include phrase "for all"

Definition:
- relation \( r \), schema \( R = (A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) \)
- relation \( s \), schema \( S = (B_1, B_2, ..., B_n) \)
- relation \( r \div s \), schema \( Q = R - S = (A_1, A_2, ..., A_n) \)
- Tuple \( t \) in \( r \div s \) if
  - \( t \) in \( \Pi_Q(r) \), and
  - for every tuple \( u \) in \( s \), \( tu \) is in \( r \)

Alternative definition
- \( r \div s = \Pi_Q(r) - \Pi_Q((\Pi_Q(r) \times s) - r) \)
Assignment Operator

- Assign parts of relational expression to temporary variables.
- Assignment operation $\leftarrow$
- Works like assignment in programming languages.

Example:
- $\text{tmp1 }\leftarrow \Pi_Q (r) \times s$
- $\text{tmp2 }\leftarrow \Pi_A (\text{tmp1} - r)$

Relational Operators

- Fundamental Operators
  - Select
  - Project
  - Union
  - Set Difference
  - Cartesian Product
  - Rename Operator

- Additional Operators
  - Set Intersection
  - Natural Join
  - Division
  - Assignment

Extended Operators

- Generalized Projection
- Aggregation
- Aggregation with Grouping
- Outer-Join

Generalized Projection

- Extend projection operator by allowing arithmetic functions in the projection list.

General form:
$$\Pi_{F_1, F_2, \ldots, F_n} (\text{expr})$$

Example:
- $\Pi_{\text{customer\_name}, \text{credit\_limit} - \text{balance}} (\text{credit-info})$
- $\Pi_{\text{customer\_name}, \text{credit\_limit} - \text{balance}} \text{ as available\_credit} (\text{credit-info})$

Aggregate Functions

- Take collection of values and return single result.

Examples:
- $\text{sum}(), \text{min}(), \text{max}(), \text{count}(), \text{avg}()$

Notation:
$$G_{\text{sum}()} (\text{employees})$$

- sets: no duplicates
- multisets: duplicates allowed
### Aggregate Function Example

<table>
<thead>
<tr>
<th>employee-name</th>
<th>branch-name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Perryridge</td>
<td>40K</td>
</tr>
<tr>
<td>Brown</td>
<td>Perryridge</td>
<td>50K</td>
</tr>
<tr>
<td>Gopal</td>
<td>Perryridge</td>
<td>25K</td>
</tr>
<tr>
<td>Johnson</td>
<td>Downtown</td>
<td>35K</td>
</tr>
<tr>
<td>Loreena</td>
<td>Downtown</td>
<td>50K</td>
</tr>
<tr>
<td>Peterson</td>
<td>Downtown</td>
<td>40K</td>
</tr>
<tr>
<td>Rao</td>
<td>Austin</td>
<td>30K</td>
</tr>
<tr>
<td>Sato</td>
<td>Austin</td>
<td>20K</td>
</tr>
</tbody>
</table>

\[ \overline{\text{avg}}(\text{salary}) \text{ (employees)} \]

- Sum of salary: 36.25K

### Aggregate Function Example – II

<table>
<thead>
<tr>
<th>employee-name</th>
<th>branch-name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Perryridge</td>
<td>40K</td>
</tr>
<tr>
<td>Brown</td>
<td>Perryridge</td>
<td>50K</td>
</tr>
<tr>
<td>Gopal</td>
<td>Perryridge</td>
<td>25K</td>
</tr>
<tr>
<td>Johnson</td>
<td>Downtown</td>
<td>35K</td>
</tr>
<tr>
<td>Loreena</td>
<td>Downtown</td>
<td>50K</td>
</tr>
<tr>
<td>Peterson</td>
<td>Downtown</td>
<td>40K</td>
</tr>
<tr>
<td>Sao</td>
<td>Austin</td>
<td>30K</td>
</tr>
<tr>
<td>Sato</td>
<td>Austin</td>
<td>20K</td>
</tr>
</tbody>
</table>

\[ \overline{\text{count-distinct}}(\text{branch-name}) \text{ (employees)} \]

- Count of branch-name: 3

### Aggregation with Grouping

- Example:
  - total employee salaries per Branch
  - partition relation employees into groups
  - apply aggregate per group

<table>
<thead>
<tr>
<th>employee-name</th>
<th>branch-name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Perryridge</td>
<td>40K</td>
</tr>
<tr>
<td>Brown</td>
<td>Perryridge</td>
<td>50K</td>
</tr>
<tr>
<td>Gopal</td>
<td>Perryridge</td>
<td>25K</td>
</tr>
<tr>
<td>Johnson</td>
<td>Downtown</td>
<td>35K</td>
</tr>
<tr>
<td>Loreena</td>
<td>Downtown</td>
<td>50K</td>
</tr>
<tr>
<td>Peterson</td>
<td>Downtown</td>
<td>40K</td>
</tr>
<tr>
<td>Rao</td>
<td>Austin</td>
<td>30K</td>
</tr>
<tr>
<td>Sato</td>
<td>Austin</td>
<td>20K</td>
</tr>
</tbody>
</table>

\[ \overline{\text{sum}}(\text{salary}) \text{ (employees)} \]

- branch-name: Perryridge, Downtown, Austin

- Sum of salary (branch-name):
  - Perryridge: 115K
  - Downtown: 125K
  - Austin: 50K

### Outer Join

- Join selects only tuples satisfying the join condition

- **Outer Join**
  - **Left outer join** \((r \bowtie s)\) also keeps every tuple in first or left relation
  - **Right outer join** \((r \bowtie s)\) also keeps every tuple in second or right relation
  - **Full outer join** \((r \bowtie s)\) also keeps every tuple

- Attributes of tuples with no matching tuples are set to NULL
### Outer Join

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>Null</td>
</tr>
<tr>
<td>s</td>
<td>b</td>
<td>g</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>r ⊙ s</td>
<td>b</td>
<td>g</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>Null</td>
</tr>
<tr>
<td>s</td>
<td>b</td>
<td>g</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>r ⊙ s</td>
<td>b</td>
<td>g</td>
<td>Null a</td>
<td></td>
</tr>
</tbody>
</table>

### Relational Database Example

- **Employee** (SSN, name, street, city)
- **Works** (SSN, comp-name, salary)
- **Company** (comp-name, comp-city, state)
- **Manages** (SSN, manager-SSN)

Note: we will sometimes use the initials of the relations instead of their full names

### Example Queries /1

- Find all employees that live in "Pittsburgh"
  \[\sigma_{city = \text{"Pittsburgh"}}(\text{Employee})\]

- Find the names of all employees that live in Pittsburgh
  \[\Pi_{name}(\sigma_{city = \text{"Pittsburgh"}}(\text{Works}))\]

- Find the names of all the employees that work for "Blockbuster"
  \[\Pi_{name}(\sigma_{comp-name = \text{"Blockbuster"}}(\text{Employee} \bowtie \text{Works}))\]

- OR, equivalently:
  \[\Pi_{name}(\text{Employee} \bowtie \sigma_{comp-name = \text{"Blockbuster"}}(\text{Works}))\]

### Example Queries /2

- Find the names and cities of all the employees that work for "Blockbuster"
  \[\Pi_{name, city}(\sigma_{comp-name = \text{"Blockbuster"}}(\text{Employee} \bowtie \text{Works}))\]

- Find the names, salaries and cities of all employees that work for "Blockbuster" and make over $15,000
  \[\Pi_{name, salary, city}(\sigma_{comp-name = \text{"Blockbuster"}} \land \text{salary} > 15000)(\text{E} \bowtie \text{W})\]

- Find the names of all employees who live in the same city as the company they work for
  \[\Pi_{name}(\sigma_{city = \text{comp-city}}(\text{Employee} \bowtie \text{Works} \bowtie \text{Company}))\]

  - We would not need the selection, if comp-city was named city.
Example Queries /3

- Find the names of all employees who live on the same street and city as their managers
  - Note: we will need to join employee (E), manager (M), and employee again to get the street, city info for the manager. For this we rename the second instance of employee as emp2

\[ \Pi E.name (\sigma \text{manager-SSN} = \text{emp2.SSN} \land E.city = \text{emp2.city} \land E.street = \text{emp2.street} (E M \rho_{\text{emp2}} E)) \]

Example Queries /4

- Find the names of all people who do not work for "First Bank"
  \[ \Pi \text{name (E W) - } \Pi \text{name (E M)} \]

- Note: the above assumes that all people are currently employed, i.e. have an entry in the Works relation. If we are not to assume this, the query should be written as follows:

\[ \Pi \text{name (E) - } \Pi \text{name (E M)} \]

Example Queries /5

- Get the average salary of all employees
  \[ G \text{avg(salary) (E W)} \]

- Get the average salary of all employees at "Blockbuster"
  \[ G \text{avg(salary) (E W)} \]

- Get the average salary of all employees per company:
  \[ \text{comp-name} G \text{avg(salary) (E W)} \]

Example Queries /6

- Find all the names of employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{name (E W)} \]

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{manager-SSN (E W)} \]

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{name (E M)} \]

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{name (E M)} \]

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{name (E M)} \]

- Find the names of the managers of all employees who live in Pittsburgh and make between $30K and $50K
  \[ \Pi \text{name (E M)} \]