Chapter 3: Relational Model

- Structure of Relational Databases
- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus
- Extended Relational-Algebra-Operations
- Modification of the Database
- Views
Basic Structure

- Given sets $A_1, A_2, \ldots, A_n$ a relation $r$ is a subset of $A_1 \times A_2 \times \ldots \times A_n$

  Thus a relation is a set of $n$-tuples $(a_1, a_2, \ldots, a_n)$ where $a_i \in A_i$

- Example: If

  \begin{align*}
  \text{customer-name} & = \{\text{Jones, Smith, Curry, Lindsay}\} \\
  \text{customer-street} & = \{\text{Main, North, Park}\} \\
  \text{customer-city} & = \{\text{Harrison, Rye, Pittsfield}\}
  \end{align*}

  Then $r = \{(\text{Jones, Main, Harrison}), (\text{Smith, North, Rye}), (\text{Curry, North, Rye}), (\text{Lindsay, Park, Pittsfield})\}$ is a relation over $\text{customer-name} \times \text{customer-street} \times \text{customer-city}$
Relation Schema

- $A_1, A_2, ..., A_n$ are attributes
- $R = (A_1, A_2, ..., A_n)$ is a relation schema

Customer-schema = (customer-name, customer-street, customer-city)

- $r(R)$ is a relation on the relation schema $R$

customer (Customer-schema)
The current values (*relation instance*) of a relation are specified by a table.

An element $t$ of $r$ is a *tuple*; represented by a *row* in a table.

<table>
<thead>
<tr>
<th>customer-name</th>
<th>customer-street</th>
<th>customer-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>Smith</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>Curry</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>Lindsay</td>
<td>Park</td>
<td>Pittsfield</td>
</tr>
</tbody>
</table>

*customer*
Keys

- Let \( K \subseteq R \)

- \( K \) is a superkey of \( R \) if values for \( K \) are sufficient to identify a unique tuple of each possible relation \( r(R) \). By “possible \( r \)” we mean a relation \( r \) that could exist in the enterprise we are modeling.
  Example: \( \{ \text{customer-name, customer-street} \} \) and \( \{ \text{customer-name} \} \) are both superkeys of \textit{Customer}, if no two customers can possibly have the same name.

- \( K \) is a candidate key if \( K \) is minimal
  Example: \( \{ \text{customer-name} \} \) is a candidate key for \textit{Customer}, since it is a superkey (assuming no two customers can possibly have the same name), and no subset of it is a superkey.
Determining Keys from E-R Sets

- **Strong entity set.** The primary key of the entity set becomes the primary key of the relation.

- **Weak entity set.** The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set.

- **Relationship set.** The union of the primary keys of the related entity sets becomes a super key of the relation.
  For binary many-to-many relationship sets, above super key is also the primary key.
  For binary many-to-one relationship sets, the primary key of the “many” entity set becomes the relation’s primary key.
  For one-to-one relationship sets, the relation’s primary key can be that of either entity set.
Query Languages

- Language in which user requests information from the database.
- Categories of languages:
  - Procedural
  - Non-procedural
- “Pure” languages:
  - Relational Algebra
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Pure languages form underlying basis of query languages that people use.
Relational Algebra

- Procedural language
- Six basic operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename
- The operators take two or more relations as inputs and give a new relation as a result.
Select Operation

- Notation: $\sigma_P(r)$
- Defined as:

$$\sigma_P(r) = \{ t \mid t \in r \text{ and } P(t) \}$$

Where $P$ is a formula in propositional calculus, dealing with terms of the form:

- $\langle\text{attribute}\rangle = \langle\text{attribute}\rangle \text{ or } \langle\text{constant}\rangle$
- $\neq$
- $>$
- $>$
- $<$
- $\leq$

“connected by”: $\wedge$ (and), $\vee$ (or), $\neg$ (not)
Select Operation – Example

- Relation $r$:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
</tr>
</tbody>
</table>

- $\sigma_{A=B \land D > 5}(r)$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
</tr>
</tbody>
</table>
Project Operation

- Notation:

\[ \Pi_{A_1, A_2, ..., A_k} (r) \]

where \( A_1, A_2 \) are attribute names and \( r \) is a relation name.

- The result is defined as the relation of \( k \) columns obtained by erasing the columns that are not listed.

- Duplicate rows removed from result, since relations are sets.
Project Operation – Example

• Relation $r$:

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>10</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>20</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>30</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>40</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

• $\Pi_{A,C}(r)$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

| $\alpha$ | 1   |
| $\beta$  | 1   |
| $\beta$  | 2   |
Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$$

- For $r \cup s$ to be valid,
  1. $r$, $s$ must have the same arity (same number of attributes)
  2. The attribute domains must be compatible (e.g., 2nd column of $r$ deals with the same type of values as does the 2nd column of $s$)
### Union Operation – Example

- **Relations** $r$, $s$:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

- $r \cup s$:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>
Set Difference Operation

- Notation: $r - s$
- Defined as:
  \[
  r - s = \{ t \mid t \in r \text{ and } t \notin s \}
  \]
- Set differences must be taken between compatible relations.
  - $r$ and $s$ must have the same arity
  - attribute domains of $r$ and $s$ must be compatible
Set Difference Operation – Example

- Relations $r$, $s$:

  \[
  \begin{array}{|c|c|}
  \hline
  A & B \\
  \hline
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \hline
  \end{array}
  \quad \begin{array}{|c|c|}
  \hline
  A & B \\
  \hline
  \alpha & 2 \\
  \beta & 3 \\
  \hline
  \end{array}
  \]

- $r - s$

  \[
  \begin{array}{|c|c|}
  \hline
  A & B \\
  \hline
  \alpha & 1 \\
  \beta & 1 \\
  \hline
  \end{array}
  \]
Cartesian-Product Operation

- Notation: $r \times s$
- Defined as:
  
  $$r \times s = \{ t \, q \mid t \in r \text{ and } q \in s \}$$

- Assume that attributes of $r(R)$ and $s(S)$ are disjoint. (That is, $R \cap S = \emptyset$).

- If attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming must be used.
## Cartesian-Product Operation – Example

- Relations $r$, $s$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\gamma$</td>
<td>10</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

- $r \times s$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\beta$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\beta$</td>
<td>20</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\gamma$</td>
<td>10</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\alpha$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\beta$</td>
<td>10</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\beta$</td>
<td>20</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\gamma$</td>
<td>10</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$
  - Notation: $r \Join s$
  - Let $r$ and $s$ be relations on schemas $R$ and $S$ respectively. The result is a relation on schema $R \cup S$ which is obtained by considering each pair of tuples $t_r$ from $r$ and $t_s$ from $s$.
  - If $t_r$ and $t_s$ have the same value on each of the attributes in $R \cap S$, a tuple $t$ is added to the result, where
    - $t$ has the same value as $t_r$ on $r$
    - $t$ has the same value as $t_s$ on $s$
Composition of Operations (Cont.)

Example:

\[ R = (A, B, C, D) \]
\[ S = (E, B, D) \]

- Result schema = \( (A, B, C, D, E) \)

- \( r \bowtie s \) is defined as:

\[ \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s)) \]
**Natural Join Operation – Example**

- Relations \( r, s \):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \alpha )</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>( \gamma )</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \gamma )</td>
<td>4</td>
<td>( \beta )</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \gamma )</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \delta )</td>
<td>2</td>
<td>( \beta )</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

\( r \)

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>( \alpha )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>( \beta )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>( \gamma )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>( \delta )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>( \epsilon )</td>
<td></td>
</tr>
</tbody>
</table>

\( s \)

- \( r \Join s \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \alpha )</td>
<td>a</td>
<td>( \alpha )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \alpha )</td>
<td>a</td>
<td>( \gamma )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \gamma )</td>
<td>a</td>
<td>( \alpha )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \gamma )</td>
<td>a</td>
<td>( \gamma )</td>
<td></td>
</tr>
<tr>
<td>( \delta )</td>
<td>2</td>
<td>( \beta )</td>
<td>b</td>
<td>( \delta )</td>
<td></td>
</tr>
</tbody>
</table>
Division Operation

\[ r \div s \]

- Suited to queries that include the phrase “for all.”

- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively, where
  \[ R = (A_1, \ldots, A_m, B_1, \ldots, B_n) \]
  \[ S = (B_1, \ldots, B_n) \]

The result of \( r \div s \) is a relation on schema \( R - S = (A_1, \ldots, A_m) \)

\[ r \div s = \{ t \mid t \in \Pi_{R-S}(r) \land \forall u \in s (tu \in r) \} \]
Division Operation (Cont.)

- Property
  - Let \( q = r \div s \)
  - Then \( q \) is the largest relation satisfying: \( q \times s \subseteq r \)

- Definition in terms of the basic algebra operation
  Let \( r(R) \) and \( s(S) \) be relations, and let \( S \subseteq R \)

\[
  r \div s = \Pi_{R-S}(r) - \Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)
\]

To see why:
- \( \Pi_{R-S,S}(r) \) simply reorders attributes of \( r \)
- \( \Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r) \) gives those tuples \( t \) in \( \Pi_{R-S}(r) \) such that for some tuple \( u \in s \), \( tu \notin r \).
Division Operation – Example

- Relations $r$, $s$:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1</td>
</tr>
<tr>
<td>$\delta$</td>
<td>1</td>
</tr>
<tr>
<td>$\delta$</td>
<td>3</td>
</tr>
<tr>
<td>$\delta$</td>
<td>4</td>
</tr>
<tr>
<td>$\delta$</td>
<td>6</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>1</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

- $r \div s$

<table>
<thead>
<tr>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>
Another Division Example

- Relations $r$, $s$:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\alpha$</td>
<td>$a$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$a$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$b$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$a$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$b$</td>
<td>$3$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$a$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>$b$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\beta$</td>
<td>$b$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

- $r \div s$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</td>
</tr>
</tbody>
</table>
Assignment Operation

- The assignment operation (←) provides a convenient way to express complex queries; write query as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

- Assignment must always be made to a temporary relation variable.

- Example: Write \( r \div s \) as

\[
\begin{align*}
temp_1 & \leftarrow \Pi_{R-S} (r) \\
temp_2 & \leftarrow \Pi_{R-S} ((temp_1 \times s) - \Pi_{R-S,S}(r)) \\
result &= temp_1 - temp_2
\end{align*}
\]

- The result to the right of the ← is assigned to the relation variable on the left of the ←.

- May use variable in subsequent expressions.
Example Queries

- Find all customers who have an account from at least the “Downtown” and “Uptown” branches.
  - Query 1
    \[ \Pi_{CN}(\sigma_{BN = "Downtown"}(\text{depositor} \Join \text{account})) \cap \Pi_{CN}(\sigma_{BN = "Uptown"}(\text{depositor} \Join \text{account})) \]
    where \( CN \) denotes \textit{customer-name} and \( BN \) denotes \textit{branch-name}.
  - Query 2
    \[ \Pi_{\text{customer-name}, \text{branch-name}}(\text{depositor} \Join \text{account}) \]
    \[ \div \rho_{\text{temp}(\text{branch-name})}(\{("Downtown"), ("Uptown")\}) \]
Example Queries

- Find all customers who have an account at all branches located in Brooklyn.

\[
\Pi_{\text{customer-name}, \text{branch-name}} (\text{depositor} \Join \text{account})
\div \Pi_{\text{branch-name}} (\sigma_{\text{branch-city} = \text{"Brooklyn"}} (\text{branch}))
\]
Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

\( \{ t \mid P(t) \} \)

- It is the set of all tuples \( t \) such that predicate \( P \) is true for \( t \)
- \( t \) is a tuple variable; \( t[A] \) denotes the value of tuple \( t \) on attribute \( A \)
- \( t \in r \) denotes that tuple \( t \) is in relation \( r \)
- \( P \) is a formula similar to that of the predicate calculus
 Predicate Calculus Formula

1. Set of attributes and constants

2. Set of comparison operators: (e.g., <, ≤, =, ≠, >, ≥)

3. Set of connectives: and (∧), or (∨), not (¬)

4. Implication (⇒): \( x \Rightarrow y \), if \( x \) if true, then \( y \) is true
   \[
   x \Rightarrow y \equiv \neg x \lor y
   \]

5. Set of quantifiers:
   - \( \exists t \in r (Q(t)) \equiv \) “there exists” a tuple \( t \) in relation \( r \) such that predicate \( Q(t) \) is true
   - \( \forall t \in r (Q(t)) \equiv Q \) is true “for all” tuples \( t \) in relation \( r \)
Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-city)

account (branch-name, account-number, balance)

loan (branch-name, loan-number, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)
Example Queries

- Find the \textit{branch-name}, \textit{loan-number}, and \textit{amount} for loans of over $1200

\[
\{ t \mid t \in \text{loan} \land t[\text{amount}] > 1200 \}
\]

- Find the loan number for each loan of an amount greater than $1200

\[
\{ t \mid \exists s \in \text{loan} \ (t[\text{loan-number}] = s[\text{loan-number}] \\
\land s[\text{amount}] > 1200) \}
\]

Notice that a relation on schema \textit{[customer-name]} is implicitly defined by the query.
Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

\[
\{ t \mid \exists s \in borrower(t[\text{customer-name}] = s[\text{customer-name}]) \\
\quad \land \exists u \in depositor(t[\text{customer-name}] = u[\text{customer-name}]) \}\]

- Find the names of all customers who have a loan and an account at the bank.

\[
\{ t \mid \exists s \in borrower(t[\text{customer-name}] = s[\text{customer-name}]) \\
\quad \land \exists u \in depositor(t[\text{customer-name}] = u[\text{customer-name}]) \}\]
Example Queries

- Find the names of all customers having a loan at the Perryridge branch

\[
\{ t \mid \exists s \in borrower(t[customer-name] = s[customer-name] \\
\quad \land \exists u \in loan(u[branch-name] = \text{“Perryridge”} \\
\quad \quad \land u[loan-number] = s[loan-number]) \}\}
\]

- Find the names of all customers who have a loan at the Perryridge branch, but no account at any branch of the bank

\[
\{ t \mid \exists s \in borrower(t[customer-name] = s[customer-name] \\
\quad \land \exists u \in loan(u[branch-name] = \text{“Perryridge”} \\
\quad \quad \land u[loan-number] = s[loan-number]) \\
\quad \land \neg \exists v \in depositor (v[customer-name] = t[customer-name]) \}\}
\]
Example Queries

- Find the names of all customers having a loan from the Perryridge branch and the cities they live in

\[
\{ t \mid \exists s \in \text{loan} \ (s[\text{branch-name}] = \text{“Perryridge”}) \\
    \land \exists u \in \text{borrower} \ (u[\text{loan-number}] = s[\text{loan-number}]) \\
    \land t[\text{customer-name}] = u[\text{customer-name}] \\
    \land \exists v \in \text{customer} \ (u[\text{customer-name}] = v[\text{customer-name}]) \\
    \land t[\text{customer-city}] = v[\text{customer-city}]) )))
\]
Example Queries

- Find the names of all customers who have an account at all branches located in Brooklyn:

\[
\{ t \mid \forall s \in \text{branch} \ (s[\text{branch-city}] = \text{"Brooklyn"} \ \Rightarrow \\
\exists u \in \text{account} \ (s[\text{branch-name}] = u[\text{branch-name}] \\
\land \exists s \in \text{depositor} \ (t[\text{customer-name}] = s[\text{customer-name}] \\
\land s[\text{account-number}] = u[\text{account-number}])})}\]

Database Systems Concepts 3.36 Silberschatz, Korth and Sudarshan ©1997
It is possible to write tuple calculus expressions that generate infinite relations.

For example, \( \{ t \mid \neg t \in r \} \) results in an infinite relation if the domain of any attribute of relation \( r \) is infinite.

To guard against the problem, we restrict the set of allowable expressions to *safe* expressions.

An expression \( \{ t \mid P(t) \} \) in the tuple relational calculus is *safe* if every component of \( t \) appears in one of the relations, tuples, or constants that appear in \( P \).
Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus.
- Each query is an expression of the form:

\[ \{ <x_1, x_2, \ldots, x_n> \mid P(x_1, x_2, \ldots, x_n) \} \]

- \( x_1, x_2, \ldots, x_n \) represent domain variables
- \( P \) represents a formula similar to that of the predicate calculus
Example Queries

- Find the branch-name, loan-number, and amount for loans of over $1200:
  \[ \{ \langle b, l, a \rangle \mid \langle b, l, a \rangle \in \text{loan} \land a > 1200 \} \]

- Find the names of all customers who have a loan of over $1200:
  \[ \{ \langle c \rangle \mid \exists b, l, a (\langle c, l \rangle \in \text{borrower} \land \langle b, l, a \rangle \in \text{loan} \land a > 1200) \} \]

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:
  \[ \{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \land \exists b (\langle b, l, a \rangle \in \text{loan} \land b = \text{“Perryridge”})) \} \]
Example Queries

- Find the names of all customers having a loan, an account, or both at the Perryridge branch:

\[
\{ <c> \mid \exists l (\langle c, l \rangle \in \text{borrower} \\
\quad \land \exists b, a (\langle b, l, a \rangle \in \text{loan} \land b = \text{"Perryridge"}) \\
\quad \lor \exists a (\langle c, a \rangle \in \text{depositor} \\
\quad \quad \land \exists b, n (\langle b, a, n \rangle \in \text{account} \land b = \text{"Perryridge"})) \}
\]

- Find the names of all customers who have an account at all branches located in Brooklyn:

\[
\{ <c> \mid \forall x, y, z (\langle x, y, z \rangle \in \text{branch} \land y = \text{"Brooklyn"}) \Rightarrow \exists a, b (\langle x, a, b \rangle \in \text{account} \land \langle c, a \rangle \in \text{depositor}) \}
\]
Safety of Expressions

\{<x_1, x_2, ..., x_n> \mid P(x_1, x_2, ..., x_n)\}

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from $dom(P)$ (that is, the values appear either in $P$ or in a tuple of a relation mentioned in $P$).

2. For every “there exists” subformula of the form $\exists x \; (P_1(x))$, the subformula is true if and only if there is a value $x$ in $dom(P_1)$ such that $P_1(x)$ is true.

3. For every “for all” subformula of the form $\forall x \; (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for all values $x$ from $dom(P_1)$.
Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions
Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

\[ \Pi_{F_1,F_2,\ldots,F_n}(E) \]

- \( E \) is any relational-algebra expression

- Each of \( F_1, F_2, \ldots, F_n \) are arithmetic expressions involving constants and attributes in the schema of \( E \).

- Given relation \textit{credit-info}(customer-name, limit, credit-balance), find how much more each person can spend:

\[ \Pi_{customer-name, limit - credit-balance} (credit-info) \]
Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist.
  - All comparisons involving *null* are **false** by definition.
### Outer Join – Example

- **Relation loan**

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
</tr>
</tbody>
</table>

- **Relation borrower**

<table>
<thead>
<tr>
<th>customer-name</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-155</td>
</tr>
</tbody>
</table>
Outer Join – Example

- \( \text{loan} \bowtie \text{Borrower} \)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
</tbody>
</table>

- \( \text{loan} \leftarrow\bowtie \text{borrower} \)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Outer Join – Example

- $loan \bowtie Borrower$

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>null</td>
<td>L-155</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

- $loan \bowtie borrower$

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
<td>$null$</td>
</tr>
<tr>
<td>null</td>
<td>L-155</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>
Aggregate Functions

- Aggregation operator \( \mathcal{G} \) takes a collection of values and returns a single value as a result.

  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values

\[
G_1, G_2, ..., G_n \mathcal{G} F_1 A_1, F_2 A_2, ..., F_m A_m(E)
\]

- \( E \) is any relational-algebra expression
- \( G_1, G_2, \ldots, G_n \) is a list of attributes on which to group
- \( F_i \) is an aggregate function
- \( A_i \) is an attribute name
Aggregate Function – Example

- Relation \( r \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( \alpha )</td>
<td>7</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta )</td>
<td>7</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( \beta )</td>
<td>3</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( \beta )</td>
<td>10</td>
</tr>
</tbody>
</table>

- \( \sum_C(r) \)

\[
\begin{align*}
\text{sum-C} \\
27
\end{align*}
\]
Aggregate Function – Example

- Relation *account* grouped by *branch-name*:

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>A-102</td>
<td>400</td>
</tr>
<tr>
<td>Perryridge</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>750</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-215</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>A-222</td>
<td>700</td>
</tr>
</tbody>
</table>

- \( g_{\text{sum balance}}(\text{account}) \)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>sum-balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>1300</td>
</tr>
<tr>
<td>Brighton</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>700</td>
</tr>
</tbody>
</table>
Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating

- All these operations are expressed using the assignment operator.
Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes.
- A deletion is expressed in relational algebra by:

\[ r \leftarrow r \ominus E \]

where \( r \) is a relation and \( E \) is a relational algebra query.
**Deletion Examples**

- Delete all account records in the Perryridge branch.

  \[
  \text{account} \leftarrow \\
  \text{account} \leftarrow \sigma_{\text{branch-name} = \text{"Perryridge"}}(\text{account})
  \]

- Delete all loan records with amount in the range 0 to 50.

  \[
  \text{loan} \leftarrow \text{loan} \leftarrow \sigma_{\text{amount} \geq 0 \text{ and } \text{amount} \leq 50}(\text{loan})
  \]

- Delete all accounts at branches located in Needham.

  \[
  r_1 \leftarrow \sigma_{\text{branch-city} = \text{"Needham"}}(\text{account} \bowtie \text{branch}) \\
  r_2 \leftarrow \Pi_{\text{branch-name}, \text{account-number}, \text{balance}}(r_1) \\
  r_3 \leftarrow \Pi_{\text{customer-name}, \text{account-number}}(r_2 \bowtie \text{depositor}) \\
  \text{account} \leftarrow \text{account} \setminus r_2 \\
  \text{depositor} \leftarrow \text{depositor} \setminus r_3
  \]
To insert data into a relation, we either:
- specify a tuple to be inserted
- write a query whose result is a set of tuples to be inserted

In relational algebra, an insertion is expressed by:

\[ r \leftarrow r \cup E \]

where \( r \) is a relation and \( E \) is a relational algebra expression.

The insertion of a single tuple is expressed by letting \( E \) be a constant relation containing one tuple.
Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

\[
\text{account} \leftarrow \text{account} \cup \{(\text{"Perryridge"}, A-973, 1200)\} \\
\text{depositor} \leftarrow \text{depositor} \cup \{(\text{"Smith"}, A-973)\}
\]

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

\[
\begin{align*}
\text{r}_1 & \leftarrow (\sigma_{\text{branch-name} = \text{"Perryridge"}} (\text{borrower} \bowtie \text{loan})) \\
\text{account} & \leftarrow \text{account} \cup \Pi_{\text{branch-name, loan-number,200}} (\text{r}_1) \\
\text{depositor} & \leftarrow \text{depositor} \cup \Pi_{\text{customer-name, loan-number}} (\text{r}_1)
\end{align*}
\]
• A mechanism to change a value in a tuple without changing *all* values in the tuple

• Use the generalized projection operator to do this task

\[ r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r) \]

  - Each \( F_i \) is either the \( i \)th attribute of \( r \), if the \( i \)th attribute is not updated, or, if the attribute is to be updated
  
  - \( F_i \) is an expression, involving only constants and the attributes of \( r \), which gives the new value for the attribute
Update Examples

- Make interest payments by increasing all balances by 5 percent.

\[ \text{account} \leftarrow \Pi_{BN,AN,BAL} \leftarrow BAL \times 1.05 \ (\text{account}) \]

where \( BAL, BN \) and \( AN \) stand for balance, branch-name and account-number, respectively.

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent.

\[ \text{account} \leftarrow \Pi_{BN,AN,BAL} \leftarrow BAL \times 1.06 \ (\sigma_{BAL > 10000} \ (\text{account})) \]
\[ \cup \ \Pi_{BN,AN,BAL} \leftarrow BAL \times 1.05 \ (\sigma_{BAL \leq 10000} \ (\text{account})) \]
Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)

- Consider a person who needs to know a customer’s loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

\[ \Pi_{\text{customer-name, loan-number}} (\text{borrower} \bowtie \text{loan}) \]

- Any relation that is not part of the conceptual model but is made visible to a user as a “virtual relation” is called a view.
A view is defined using the `create view` statement which has the form

```
create view v as <query expression>
```

where `<query expression>` is any legal relational algebra query expression. The view name is represented by `v`.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression. Rather, a view definition causes the saving of an expression to be substituted into queries using the view.
View Examples

- Consider the view (named *all-customer*) consisting of branches and their customers.

```sql
create view all-customer as
  \( \Pi_{\text{branch-name}, \text{customer-name}} (\text{depositor} \bowtie \text{account}) \)
  \cup \( \Pi_{\text{branch-name}, \text{customer-name}} (\text{borrower} \bowtie \text{loan}) \)
```

- We can find all customers of the Perryridge branch by writing:

```sql
\( \Pi_{\text{customer-name}} (\sigma_{\text{branch-name} = \text{“Perryridge”}} (all-customer)) \)
Updates Through Views

- Database modifications expressed as views must be translated to modifications of the actual relations in the database.

- Consider the person who needs to see all loan data in the loan relation except amount. The view given to the person, branch-loan, is defined as:

  \[
  \text{create view branch-loan as}
  \]
  \[
  \Pi_{\text{branch-name, loan-number}} \ (\text{loan})
  \]

Since we allow a view name to appear wherever a relation name is allowed, the person may write:

\[
\text{branch-loan} \leftarrow \text{branch-loan} \cup \{ (\text{“Perryridge”, L-37}) \}
\]
Updates Through Views (Cont.)

- The previous insertion must be represented by an insertion into the actual relation `loan` from which the view `branch-loan` is constructed.

- An insertion into `loan` requires a value for `amount`. The insertion can be dealt with by either
  - rejecting the insertion and returning an error message to the user
  - inserting a tuple ("Perryridge", L-37, null) into the `loan` relation
Views Defined Using Other Views

- One view may be used in the expression defining another view.
- A view relation $v_1$ is said to depend directly on a view relation $v_2$ if $v_2$ is used in the expression defining $v_1$.
- A view relation $v_1$ is said to depend on view relation $v_2$ if and only if there is a path in the dependency graph from $v_2$ to $v_1$.
- A view relation $v$ is said to be recursive if it depends on itself.
View Expansion

- A way to define the meaning of views defined in terms of other views.
- Let view $v_1$ be defined by an expression $e_1$ that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

  repeat
  
  Find any view relation $v_i$ in $e_1$
  
  Replace the view relation $v_i$ by the expression defining $v_i$

  until no more view relations are present in $e_1$

- As long as the view definitions are not recursive, this loop will terminate.