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

Chapter 3: Relational Model
customer-name \times customer-street \times customer-city

over

\{ \text{Curtis, North, Rye}, \text{Lindsey, Park}, \text{Pitkethly} \} \quad \text{Then } r \quad \{ \text{jones, Main, Harrison}, \text{Smith, North, Rye} \}

\{ \text{Harrison, Rye, Pitkethly} \} \quad \text{customer-city}

\{ \text{Main, North, Park} \} \quad \text{customer-street}

\{ \text{Jones, Smith, Curtis, Lindsey} \} \quad \text{customer-name}

\text{Example: If } \quad a_i \in A_i

\text{Thus a relation is a set of n-tuples } (a_1, a_2, \ldots, a_n) \quad \text{where}

\prod_{i=1}^{n} A_i \times \cdots \times A_2 \times A_1 \quad \text{Given sets } A_1, A_2, \ldots, A_n, \text{ a relation } r \text{ is a subset of }

\text{Basic Structure}
\[\text{customer} (\text{customer-schema})\]

\[\pi_R (R) \text{ is a relation on the relation schema } R\]

\[(\text{customer-city})\]

\[\text{customer-schema} = (\text{customer-name, customer-street})\]

\[(\forall A_1, A_2, \ldots, A_n \text{ is a relation schema} R)\]

\[R = \pi_{A_1, A_2, \ldots, A_n} R\]

\[A_1, A_2, \ldots, A_n \text{ are attributes}\]
### Customer

<table>
<thead>
<tr>
<th>Pittsfield</th>
<th>Park</th>
<th>Lindsay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ry e</td>
<td>North</td>
<td>Curry</td>
</tr>
<tr>
<td>Ry e</td>
<td>North</td>
<td>Smith</td>
</tr>
<tr>
<td>Harrison</td>
<td>Main</td>
<td>Jones</td>
</tr>
</tbody>
</table>

- An element of \( r \) is a tuple, represented by a row in a table.
- By a table.
- The current values (relation instance) of a relation are specified.
have the same name), and no subset of it is a superkey. Since it is a superkey (assuming no two customers can possibly have the same name), \( Y \) is a candidate key for \( \text{Customer} \).

Example: \{ \text{customer-name} \} is a candidate key for \( \text{Customer} \).

\( Y \) is a candidate key if \( Y \) is minimal.

Customers can possibly have the same name.

Example: \{ \text{customer-name} \} are both superkeys of \( \text{Customer} \).

Example: \{ \text{customer-name, customer-street} \} and

modeled.


\[ Y \] is a superkey of \( R \) if values for \( Y \) are sufficient to identity a unique tuple of each possible relation \( (r)(R) \). By "possible tuples", we mean a relation \( R \) that could exist in the enterprise we are modeling.

\[ \text{Let } K \supseteq R \]

---

**Keys**
be that of either entity set. For one-to-one relationship sets, the relation’s primary key can
the "many" entity set becomes the relation’s primary key.
For binary many-to-many relationship sets, the primary key of
also the primary key.
For binary many-to-many relationship sets, above super key is
related entity sets becomes a super key of the relation.

- Relationship set. The union of the primary keys of the
  discriminator of the weak entity set.

- Weak entity set. The primary key of the relation consists of
  the primary key of the relation.

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

Determining Keys from E-R Sets
people use: •
• Pure Languages form underlying basis of query Languages that
  Domain Relational Calculus
  Tuple Relational Calculus
  Relational Algebra
  "Pure" Languages:
  • Non-Procedural
  • Procedural

Categories of Languages: database

Language in which user requests information from the

Query Languages
new relation as a result.

The operators take two or more relations as inputs and give a

- Rename
- Cartesian product
- Set difference
- Union
- Project
- Select

Six basic operators

Procedural Language

Relational Algebra
\[(\text{not}) \land (\text{or}) \land (\text{and}) \lor \land \lor : \text{connected by} \]

\[\neq \]

\[\text{terms of the form:} \]

\[\langle \text{attribute} \rangle < \langle \text{attribute} \rangle \]

\[\langle \text{attribute} \rangle \text{ or constant} \]

\[\{ (\exists \tau \land \mathcal{A} \in \tau) \land \tau \} = (\tau)^{\mathcal{D} \mathcal{A}} \]

\text{Defined as:}\n
\[\text{Notation:} \]

\[\tau \]
\[ (\forall x \in \mathbb{Z} \quad x < d \vee b = a \quad \bullet) \]

**Example**

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Relation** 

Select **Operation** – **Example**
Duplicate rows removed from result, since relations are sets

erasing the columns that are not listed

The result is defined as the relation of f columns obtained by

where A₁, A₂ are attribute names and ρ is a relation name.

\[ \Pi_{\text{A₁, A₂, \ldots, Aₙ}} \rho \]

Notation

Project Operation
\[
\begin{array}{c|c}
\mathcal{C} & \mathcal{A} \\
\hline
2 & \emptyset \\
1 & \emptyset \\
\mathcal{C} & \mathcal{A}
\end{array}
\]

\[
\begin{array}{c|c}
\emptyset & \emptyset \\
\mathcal{C} & \mathcal{A}
\end{array}
\]

(II.1.2) \quad \text{Relation } r

\[
\begin{array}{c|c|c}
2 & 40 & \emptyset \\
1 & 30 & \emptyset \\
1 & 20 & \emptyset \\
1 & 10 & \emptyset \\
\mathcal{C} & \mathcal{B} & \mathcal{A}
\end{array}
\]


Project Operation Example
column of $s$)

or $r$ deals with the same type of values as does the 2nd

2. The attribute domains must be compatible (e.g., 2nd column

$(r,s)$ must have the same arity (same number of attributes)

For $r \cup s$ to be valid,

$$\{ s \ni \exists t \ni r \cup t \} = s \cup r$$

\textbf{Defined as:}

\textbf{Notation:} $r \cup s$
The diagram illustrates the union operation on two relations, 's' and 't'.
Set Difference Operation

- Attribute domains of \( r \) and \( s \) must be compatible
- \( r \) and \( s \) must have the same arity

Set differences must be taken between compatible relations.

\[
\{ s \neq t \text{ and } u \in t \mid t \} = s - r
\]

Defined as:

Notation: \( r \setminus s \)
Set Difference Operation – Example

\[ r - s \]

\[
\begin{array}{c|c}
I & \emptyset \\
I & a \\
B & A \\
\end{array}
\]

\[
\begin{array}{c|c}
3 & \emptyset \\
2 & a \\
B & A \\
\end{array}
\]

\[
\begin{array}{c|c}
1 & \emptyset \\
2 & a \\
1 & a \\
B & A \\
\end{array}
\]

Relations \( r \):
must be used.

If attributes of \( \mathcal{S}(s) \) and \( \mathcal{H}(\tau) \) are not disjoint, then renaming \( \cdot \)

\[
\emptyset = S \cup \mathcal{H}
\]

Assume that attributes of \( \mathcal{S}(s) \) and \( \mathcal{H}(\tau) \) are disjoint. (That is,

\[
\{s \in b \text{ and } \mu \in \tau | b \tau \} = s \times \tau
\]

Defined as: \( \cdot \)

Notation: \( \cdot \times s \)
### Cartesian Product Operation - Example

#### Relations $r_s$, $s$:  

<table>
<thead>
<tr>
<th>$s$</th>
<th>10</th>
<th>20</th>
<th>10</th>
<th>10</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$+$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$+$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$+$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$r$</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$B$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
</tr>
</tbody>
</table>

- $s \times r$
### Example of operations

**Composition of Operations**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

\( \sigma \times \iota \) **Can build expressions using multiple operations**

\( \sigma \times \iota \) **Example:**

\( A \boxtimes B \) **Composition**
attributes renamed to $A_1$, $A_2$, \ldots, $A_n$. \[ \text{returns the result of expression } F \text{ under the name } x, \text{ and with the} \]
\[ (F) \left( \forall^{n-1} x \right) \]

If a relational-algebra expression $F$ has arity $n$, then it returns the expression $F$ under the name $x$.

\[ (F) x \]

Example:

- Allows us to refer to a relation by more than one name.
- Relational-algebra expressions allow us to name, and therefore to refer to, the results of renaming operations.
borrower (customer-name, loan-number)

deposit (customer-name, account-number)

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

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

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

branch (branch-name, branch-city, assets)
II loan number \( \text{amount} < 1200 \) (loan)

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

\( \text{amount} < 1200 \) (loan)

Find all loans of over $1200

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

Find the names of all customers who have a loan and an account, or both, from the bank.

Example Queries
Example Queries:

- Find the names of all customers who have a loan at the "Perryridge" branch.

- Find the names of all customers who have a loan at any branch of the bank.

- Find the names of all customers who have a loan at the "Perryridge" branch but do not have an account at any branch of "Perryridge".
\[
\text{\textcolor{red}{\textbf{Example Queries}}}
\]

1. **Query 1**

   PerTyridge branch.

   Find the names of all customers who have a loan at the

2. **Query 2**

   PerTyridge branch.

   \[
   \left( \left( \text{customer-name} \times \text{store-name} \right) \text{loan-number = loan-number} \right)
   \]

   \[
   \left( \text{branch-name = "PerTyridge"} \text{loan-number = loan-number} \right)
   \]

   \[
   \left( \text{customer-name = "PerTyridge"} \text{branch-name = "PerTyridge"} \right)
   \]
\[(\text{balance} \times \text{account}) \times \text{account} \times \text{account} \times \text{balance} < \text{balance} \times \text{balance} \times \text{balance} \]

The query is:

- Rename account relation as \(d\)
- Find the largest account balance

Example Queries
A basic expression in the relational algebra consists of either one of the following:

- \( \rho_{x} (E_{1}) \), \( x \) is the new name for the result of \( E_{1} \)
- \( \Pi_{S} (E_{1}) \), \( S \) is a list consisting of some of the attributes in \( E_{1} \)
- \( E_{1} \cup E_{2} \)
- \( E_{1} \setminus E_{2} \)
- A relation in the database
- A constant relation

Let \( E_{1} \) and \( E_{2} \) be relational-algebra expressions; the following are all relational-algebra expressions:

- \( \sigma_{P} (E_{1}) \), \( P \) is a predicate on attributes in \( E_{1} \)
- \( E_{1} \times E_{2} \)

Formal Definition
Assignment •

Division •

Natural Join •

Set Intersection •

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.
Set-Intersection Operation

\[(s \setminus t) \setminus t = s \cup t\]

Note: \(s \cup t\) and \(s\) are compatible attributes of \(t\) and \(s\) have the same arity.

Assume:

\[\{s \in t \land t \in t | t \}\} = s \cup t\]

Defined as:

\[\text{Notation: } s \cup t\]
Set-Intersection Operation – Example

\[ R \cap S \]

\[ s \cup \]

\[ \begin{array}{c|c}
3 & \emptyset \\
2 & a \\
B & A \\
\end{array} \]

\[ \begin{array}{c|c}
1 & \emptyset \\
2 & a \\
1 & a \\
B & A \\
\end{array} \]
(s × t) \Delta R = s \Delta R \vee t \Delta R \equiv (s \Delta R \vee t \Delta R) \land (s \Delta R \land t \Delta R)

\text{If } s \text{ and } t \text{ have the same value on } t^s \text{ from } s.

\text{Example: If } t \text{ and } t^s \text{ have the same value on each of the attributes in } s, \text{ consider each pair of tuples } t^t \text{ from } t \text{ and } t^s \text{ from } s.

\text{Let } R \text{ and } s \text{ be relations on schemas } R \text{ and } S \text{ respectively. The result is a relation on schema } R \cap S \text{ which is obtained by }

\text{Notation: } R \bowtie S

\text{Natural Join Operation}
### Natural Join Operation - Example

Let's consider an example of a natural join operation on two relations `s` and `t`. These relations are as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>b</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>2</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>3</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>f</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>2</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>3</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

The natural join of these two relations, denoted as `s ∩ t`, is obtained by selecting those tuples where `s` and `t` have matching values for their common attributes. In this case, the common attributes are `a` and `b`, and the join operation will result in the following relation:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>3</td>
<td>a</td>
</tr>
</tbody>
</table>

This is because the join operation requires that the matching values for the common attributes must be equal in both relations. Therefore, the join result considers only the tuples where `s` and `t` have the same values for `a` and `b`.
\{ (r \in \mathbb{R}) \exists n A \land (r')^S \exists (t \in t \mid t) = s \div r \}

\left( \forall^w A, \ldots, \forall^w A \right) = S - R

The result of \( r \) \( \div \) \( s \) is a relation on schema

\left( \forall^w B, \ldots, \forall^w B \right) = S - R

\left( \forall^w B, \ldots, \forall^w A, \ldots, \forall^w A \right) = R - R

\text{where}

\text{Let } r \text{ and } s \text{ be relations on schemas } R \text{ and } S \text{ respectively,}

\text{suitable to queries that include the phrase "for all".

\( s \div r \)
\[
\neg \in \{ t \mid \exists \, s \in S \, t \in (s \times (t)_{S-R \Pi \Pi})_{S-R \Pi \Pi} \} \]

gives those tuples \( t \) such that for some tuple \( s \in S \), \( t \) simply reorders attributes of \( t \).

To see why:

\[
((t)_{S-R \Pi \Pi} \times (s \times (t)_{S-R \Pi \Pi}))_{S-R \Pi \Pi} = (s \div t)
\]

Let \( H \) and \( s \) be relations, and let \( \rho(H) \) and \( \mu(H) \) be relations in terms of the basic algebra operation.

Definition: in terms of the basic algebra operation:

\[
\rho \subseteq s \times \mu \text{ then is the largest relation satisfying:} \]

\[
s \div \rho \subseteq \mu\]

Property:

\textbf{Division Operation (Cont.)}
A

B C

E

\[ s \div r \cdot \]

Relations \( r \), \( s \):

Another Division Example
- May use variable in subsequent expressions.

- Variable on the left of the →

- The result to the right of the → is assigned to the relation

\[
\text{result} = \text{temp}_2 - (s \times (s^{R} \cap R^{S}(I)))
\]

Example: Write \( s \div r \) as \( s \rightarrow \text{temp}_1 \rightarrow \text{temp}_2 \rightarrow \text{temp}_1 \)

- Assignment must always be made to a temporary relation

- Assignment whose value is displayed as the result of the query

- 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

**Assignment Operation**
Example Queries

Query 1

Find all customers who have an account in at least the "Downtown" and "Uptown" branches.

Query 2

Where \( \mathit{CN} \) denotes customer-name and \( \mathit{BN} \) denotes

\[
\left\{ \left( \mathit{account} \right) \left( \mathit{depositor} \right) \left( \mathit{branch-name} \right) \mathit{Downtown} \right\} \div \left( \left( \mathit{account} \right) \left( \mathit{depositor} \right) \left( \mathit{branch-name} \right) \mathit{Uptown} \right)
\]

\( \mathit{query} \)
Example Queries

\[
\Pi_{\text{branch-name \ (branch)}} \div \Pi_{\text{customer-name \ (branch-name \ (depositor \ \text{account})}}
\]

in Brooklyn.

Find all customers who have an account at all branches located in Brooklyn.
Tuple Relational Calculus

- $\Phi$ is a formula similar to that of first-order predicate calculus
- $t \in r$ denotes that tuple $t$ is in relation $r$
- $A$ denotes the value of tuple $t$ on attribute $A$
- $t$ is a tuple variable; $[A]$ denotes the value of tuple $t$
- $\{ (t) | \Phi \}$ is the set of all tuples $t$ such that predicate $\Phi$ is true for $t$

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

\[ \{ (t) | \Phi \} \]
In relation \( \mathcal{R} \)

\[
\forall t \in \mathcal{R} \quad \forall x \in \mathcal{O} \quad (t \in \mathcal{O}) \iff (t \in \mathcal{R})
\]

such that predicate \((t)\mathcal{O}\) is true

\[
\forall t \in \mathcal{O} \quad \exists! t \in \mathcal{R} \quad (t \in \mathcal{R}) \iff (t \in \mathcal{O})
\]

\[
\forall t \in \mathcal{R} \quad (t \in \mathcal{O}) \iff (t \in \mathcal{R})
\]

5. Set of quantifiers:

\[
\forall \wedge x \iff \exists \iff x
\]

4. Implication:

\[
(\iff) \quad (\iff) \quad (\iff) \quad (\iff)
\]

3. Set of connectives: and, or, not

\[
(\iff) \quad (\iff) \quad (\iff) \quad (\iff) \quad (\iff) \quad (\iff) \quad (\iff) \quad (\iff)
\]

2. Set of comparison operators: \(=\), \(<\), \(\leq\), \(\geq\), \(\neq\), \(\in\), \(\notin\)

1. Set of attributes and constants

Predicate Calculus Formula
borrower (customer-name, loan-number)
depositor (customer-name, account-number)
loan (branch-name, loan-number, amount)
account (branch-name, account-number, balance)
customer (customer-name, customer-street, customer-city)
branch (branch-name, branch-city, assets)
Notice that a relation on schema [customer-name] is implicitly defined by the query:

\[
\{ \text{loan-number} \mid \text{amount} < 1200 \vee (\text{amount} = 1200) \} \\
\]

$1200$

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

$1200$

Find the branch-name, loan-number, and amount for loans of

---

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

\[ \{ [\text{customer-name}] \in [\text{depositor}] \wedge [\text{customer-name}] \in [\text{borrower}] \mid t \} \]

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

\[ \{ [\text{customer-name}] \in [\text{depositor}] \wedge [\text{customer-name}] \in [\text{borrower}] \mid t \} \]
\{ [\text{ff}] \text{customer-name} | \exists \text{ deposit} (t) \} \lor
(\exists s [\text{loan-name} = \text{loan-number} \land
\text{customer-name} \land \text{deposit-ltv}])
\lor
\exists s [\text{customer-name} = \text{loan-name} \land \text{depositor-branch}] \land
\exists s [\text{customer-name} = \text{loan-number} \land \text{depositor-branch}]

Example Queries

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

\{ ([\text{loan-number} = \text{loan-name} \land
\text{customer-name} = \text{loan-number} \land
\text{depositor-branch}] \land
\exists s [\text{customer-name} = \text{loan-number} \land \text{depositor-branch}]) \land
\exists s [\text{customer-name} = \text{loan-number} \land \text{depositor-branch}] \land
\exists s [\text{customer-name} = \text{loan-number} \land \text{depositor-branch}]

• Find the names of all customers having a loan at the
  Perpendicular branch
\{((\text{[customer-city]} \land \text{[customer-name]} \land \text{[loan-number]} \land \text{[branch-name]} \land \text{"Perryridge"}) \lor \text{[branch-name]} ) \mid \text{[loan-name]} \in \text{[branch-name]} \}\}
{(((account-number)n = [account-number]s ∨ [name]s = [customer-name]n) ∨ (depositor (s) name) ∨ [name]n = [branch-name]s) branch ∈ [branch-name]s} ⊆ Brooklyn

branches located in Brooklyn:

Find the names of all customers who have an account at all
or constants that appear in $D$ or every component of $t$ appears in one of the relations, tuples.

An expression to safe expressions.

To guard against the problem, we restrict the set of allowable domains of any attribute of relation $t$ is infinite.

For example, results in an infinite relation if the

It is possible to write tuple calculus expressions that generate

Safety of Expressions
calculus
represents a formula similar to that of the predicate
represent domain variables

\{ \langle u, x \rangle \mid \langle u, x \rangle \}

Each query is an expression of the form:

tuple relational calculus

A nonprocedural query language equivalent in power to the

Domain Relational Calculus
\{((\text{" Perryfriede"} = q \lor \text{ loan } \in < q, l', q >) \land q \in \forall b) \lor \in < q, l', c > \} \\

Find the branch and the loan amount.

Find the names of all customers who have a loan from the

\{((0 < a \lor \text{ loan } \in < a, l', q >) \land q \in \forall b) \lor \in < q, l', c > \} \\

Find the names of all customers who have a loan of over $1200.

Find the names of all customers who have a loan for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

\{0 < a \lor \text{ loan } \in < a, l', q > \land q > 1200 \} \\

Find the branch-name, loan-number, and amount for loans of

Example Queries
\{ ( \exists \text{ account} \in \text{deposit} \ ( q, a, c, > \land \forall \text{ branch} \in \text{Brooklyn} \ ( z, y, x, > ) \land A \mid < c > ) \}

\text{branches located in Brooklyn:}

Find the names of all customers who have an account at all

\{ ( ( \exists \text{ deposit} \ ( q, a, c, > ) \land \forall \text{ account} \in \text{ Brooklyn} \ ( q, a, n, > ) \land \forall \text{ loan} \in \text{ borrower} \ ( l, q, a, > ) \land A \mid < c > ) \}

\text{both at the Brooklyn branch:}

Find the names of all customers having a loan, an account, or

Example Queries
from $\text{dom}(P)$.

3. For every "for all" subformula of the form $\forall x (\, (P) \, (x))$ such that $(P) (x)$ is true, the subformula is true if and only if there is a value $x$ in $(\forall x (\, (P) \, (x)) (x)$.

2. For every "there exists" subformula of the form $\exists x \in P \quad (P) (x)$ that is, the values appear either in or in a tuple of a relation mentioned in $P$.

1. All values that appear in tuples of the expression are values is safe if all of the following hold:

$$\{ (\, u_1 x_1, \ldots, u_k x_k \, ) \mid \not< \, u_1 x_1, \ldots, u_k x_k \, \}$$
Aggregate Functions

Outer Join

Generalized Projection

Extended Relational-Algebra-Operations
II. Customer-name, limit – credit-balance (credit-info)

Find how much more each person can spend:

- Given relation credit-info(customer-name, limit, credit-balance),

- Constants and attributes in the schema of R.

Each of \( F_1, F_2, \ldots, F_u \) are arithmetic expressions involving \( R \) is any relational-algebra expression \( \pi_{F_1, F_2, \ldots, F_u}(R) \)

- Functions to be used in the projection list.

- Extends the projection operation by allowing arithmetic

**Generalized Projection**
Outer Join

- All comparisons involving null are false by definition.
- null signifies that the value is unknown or does not exist.

Uses null values:
- Join:
  do not match tuples in the other relation to the result of the
  Computes the join and then adds tuples from one relation that
  An extension of the join operation that avoids loss of
<table>
<thead>
<tr>
<th>Customer-name</th>
<th>Loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hayes</td>
<td>L-155</td>
</tr>
<tr>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td></td>
<td>Loan-number</td>
</tr>
<tr>
<td></td>
<td>1700</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>Loan-number</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0000</td>
</tr>
<tr>
<td>Branch-Name</td>
<td>Loan-Number</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>null</td>
<td>T-230</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>null</td>
<td>T-170</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Redwood</td>
<td>T-230</td>
</tr>
<tr>
<td></td>
<td>T-170</td>
</tr>
</tbody>
</table>

**Example - Outer Join**
<table>
<thead>
<tr>
<th>Name</th>
<th>Amount</th>
<th>Loan-Number</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hayes</td>
<td>null</td>
<td>1-155</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>4000</td>
<td>1-260</td>
<td>Perryridge</td>
</tr>
<tr>
<td>Smith</td>
<td>3000</td>
<td>1-230</td>
<td>Redwood</td>
</tr>
<tr>
<td>Jones</td>
<td></td>
<td>1-170</td>
<td>Downtown</td>
</tr>
</tbody>
</table>

Outer Join Example
\( \forall \) is an attribute name

\( ? \) is an aggregate function

\( \{ \} \) is a list of attributes on which to group

\( c_1, c_2, \ldots, c_n \) is any relational algebra expression

\( (\mathcal{E})^{\mu}_{\forall \, c_1, c_2, \ldots, c_n} \) 

\text{count} \quad \text{number of values}

\text{sum} \quad \text{sum of values}

\text{max} \quad \text{maximum value}

\text{min} \quad \text{minimum value}

\text{ave} \quad \text{average value}

A single value as a result.

\text{Aggregate Functions}
\[
\begin{array}{c|c|c}
10 & \emptyset & \emptyset \\
3 & \emptyset & \emptyset \\
2 & \emptyset & \varnothing \\
2 & \varnothing & \varnothing \\
\hline
\emptyset & B & A
\end{array}
\]

Relation \( R \): {}
<table>
<thead>
<tr>
<th>Branch-name</th>
<th>Account-name</th>
<th>Account-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>Redwood</td>
<td>A-222</td>
</tr>
<tr>
<td>750</td>
<td>Bigphon</td>
<td>A-215</td>
</tr>
<tr>
<td>1300</td>
<td>Periyide</td>
<td>A-217</td>
</tr>
<tr>
<td>900</td>
<td>Periyide</td>
<td>A-201</td>
</tr>
<tr>
<td>400</td>
<td>Periyide</td>
<td>A-102</td>
</tr>
</tbody>
</table>

Relation account grouped by branch-name: 

Example - Aggregate Function
All these operations are expressed using the assignment operator.

- Updating
- Insertion
- Deletion

The content of the database may be modified using the following operations.
where $R$ is a relation and $A$ is a relational algebra query.

$$A - R \rightarrow R$$

A deletion is expressed in relational algebra by:

- particular attributes
- can delete only whole tuples; cannot delete values on only
- removed from the database
- instead of displaying tuples to the user, the selected tuples are
- a delete request is expressed similarly to a query,
depositor \rightarrow \text{account}

\text{account} \rightarrow \text{account}

\text{customer-name, account-number} \rightarrow \text{balance}

\text{branch-name, account-number, balance} \rightarrow \text{branch}

\text{branch} \rightarrow \text{branch-name} = \text{Needham}

\text{Delete all accounts at branches located in Needham.}

\text{loan} \rightarrow \text{loan}

\text{amount} \geq 0 \text{ and } \text{amount} \leq 50

\text{Delete all loan records with amount in the range to 50.}

\text{account} \rightarrow \text{account}

\text{branch-name} = \text{Perryridge}

\text{Delete all account records in the Perryridge branch.}

\text{Examples}
In relational algebra, an insertion is expressed by:

\[ \mathcal{E} \cap I \rightarrow I \]

To insert data into a relation, we either:

```
Insert
```
$n_1 \leftarrow (\text{branch-name} = \text{"Perryridge"}, \text{borrower} \times \text{loan})$

account \leftarrow account \cup \Pi (\text{branch-name, loan-number, 200} (r_1))

depositor \leftarrow \text{depositor} \cup \Pi (\text{customer-name, loan-number} (r_1))

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.

Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.
attirbutes of \( r \), which gives the new value for the attribute

\[ \frac{1}{r} \text{ if } \frac{1}{r} \text{ is an expression involving only constants and the} \]

not updated, or, if the attribute is to be updated

\[ \text{Each } x_i \text{ is either the } i \text{th attribute of } r, \text{ if the } i \text{th attribute is} \]

\[ \left( \frac{1}{r} \right)^{u_{F_1}^{F_n}} \rightarrow r \]

Use the generalized projection operator to do this task

\[ \bigcup \text{ values in the tuple} \]

\[ \text{A mechanism to change a value in a tuple without changing all} \]

Updating
$((account \ 0000 \geq \ \text{BAL} \ \text{BAL} \ * \ 1.05) \ \cap \\
((account \ 0000 \ < \ \text{BAL} \ \text{BAL} \ * \ 1.06) \rightarrow \ \text{BAL} \ \text{BAL} \ \text{BN} \ \text{AN} \\
\cap \ \text{account} \rightarrow \ \text{account})$ \\

6 percent interest and pay all others 5 percent. 

Pay all accounts with balances over $10,000 

Branch-name and account-number, respectively. 

Where \text{BAL, BN, and AN} stand for balance. 

$((account \ * \ 1.05) \rightarrow \ \text{BAL} \ \text{BAL} \ \text{BN} \ \text{AN} \\
\cap \ \text{account} \rightarrow \ \text{account})$ \\

Make interest payments by increasing all balances by 5 percent. 

Update Examples
made visible to a user as a "virtual relation" is called a view.

Any relation that is not part of the conceptual model but is defined in terms of views.

\[
\text{II} \text{customer-name, loan-number (portion, loan)}
\]

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

Consider a person who needs to know a customer's loan

(database).

logical model (i.e., all the actual relations stored in the

In some cases, it is not desirable for all users to see the entire

VIEWS

views
the view.

the saving of an expression to be substituted into queries using
evaluating the query expression. Rather, a view definition causes

View definition is not the same as creating a new relation by

the virtual relation that the view generates.

Once a view is defined, the view name can be used to refer to

expression. The view name is represented by a

where query expression is any legal relational algebra query

create view a as query expression

the form

A view is defined using the create view statement which has
We can find all customers of the Bermuda branch by writing:

\[
\text{create view all-customer as}
\]

and their customers.

Consider the view (named all-customer) consisting of branches consisting of branches.
\{(\text{Person}) \cap \text{Branch-loan}\} \in \text{Branch-loan} \rightarrow \text{Branch-loan}

Since we allow a view name to appear wherever a relation

\text{II}
\text{Branch-name, loan-number (loan)}
\text{CREATE VIEW Branch-loan AS}
\text{Branch-loan, is defined as:}
\text{Branch-loan, except loan-amount. The view given to the person,}
\text{Consider the person who needs to see all loan data in the loan}
\text{updates through views views must be translated to modifications of the actual relations in the database.}
Updates Through Views (cont.)
A view relation vi is said to be recursive if it depends on itself.

Only if there is a path in the dependency graph from v2 to vi.

A view relation v1 is said to depend on view relation v2 if and only if v2 is used in the expression defining v1.

A view relation v1 is said to depend directly on a view relation v2 if v2 may be used in the expression defining another view.
terminate.

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

until no more view relations are present in $v_1$

Replace the view relation $v_i$ by the expression defining $v_i$

Find any view relation $v_i$ in $v_1$

repeat

replacement step:

View expansion of an expression repeats the following

contain uses of view relations.

Let view $v_i$ be defined by an expression $e_i$ that may itself

views.

A way to define the meaning of views defined in terms of other