Chapters 2 & 6

The Relational Model
The Relational Model

- A tabular data structure
- Tables (relations) with unique names
  - rows (tuples/entities/records)
  - columns (attributes/fields)
- Domain (or \(\text{dom}\)) of an attribute: a set of permitted atomic (non-decomposable) values
- Tuples \(t \in r(R)\), where \(R = A_1, A_2, \ldots, A_n\), is an ordered set of values, i.e.,
  \[
  t = < v_1, v_2, \ldots, v_n > \text{ where } v_1 \in \text{dom}(A_1), v_2 \in \text{dom}(A_2), \ldots, v_n \in \text{dom}(A_n).
  \]
- Relation \(r(R)\): a set of tuples
  - \(r(R) \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \ldots \times \text{dom}(A_n)\)
  - \(r\) is a relation instance
The Relational Model

- Database Schema
  - a relational DB schema consists of a set of relation schemas
  - a relation schema has a (i) name and (ii) a finite set of attributes, where each one is associated with a domain
  - \( r(R) \) denotes a relation (instance) on the relation schema \( R \)

- Definition. A candidate key \( K \) is a non-empty set of attributes which has the following properties:
  - **uniqueness**: \( K \) uniquely identifies a tuple in a relation, and
  - **minimality**: no proper subset of \( K \) satisfies the uniqueness constraint
    - a primary key is a chosen candidate key
    - a superkey satisfies (a) but may not satisfy (b)
    - a relation may have more than one candidate key
    - a foreign key in a relation schema is the primary key of another relation schema

- Keys in the relational model
  - \( K \) is a superkey for \( R \) if \( \forall t_1, t_2 \in r(R), t_1 \neq t_2, \text{ then } t_1[K] \neq t_2[K] \)
The Relational Model

- **Relational Algebra**
  - Operators used for manipulating relations, i.e., specifying retrieval/update requests (queries)
  - Relations in a relational DBMS are *closed* under the relational algebraic operations, since a *query result* is itself is a *relation*
  - A *procedural* query language
The Relational Model

- **Unary relational operators:**
  - select ($\sigma$)
  - project ($\pi$)
  - rename ($\rho$)
  - assignment ($\leftarrow$)

- **Binary relational operators:**
  - cartesian produce ($\times$)
  - union ($\cup$)
  - difference ($-$)
  - intersection ($\cap$)
  - natural join ($\bowtie$)
  - division ($\div$)

**Set Operations**
The Relational Model

- **Fundamental** relational operators:
  - select ($\sigma$)
  - project ($\pi$)
  - cartesian produce ($\times$)
  - union ($\cup$)
  - difference ($-$)
  - rename ($\rho$)

- **Additional** relational operators:
  - intersection ($\cap$)
  - natural join ($\bowtie$)
  - division ($\div$)
  - assignment ($\leftarrow$)
The Relational Model

- **Select** ($\sigma$)
  - Selects the tuples (rows) from a relation $r$ satisfying a certain selection condition $C$, i.e., $\sigma_c(r)$
  - $C$ is a Boolean expression on attributes of $r$
  - Resulting relation has the same attributes as $r$
    
    e.g., $\sigma_{(\text{std\_major} = \text{"CS"} \lor \text{std\_major} = \text{"Math"}) \land \text{std\_gpa} \geq 3.5}$ (Student)

- **Project** ($\pi$)
  - Keeps only certain attributes specified in an attribute list $L$ from a relation $r$
  - Resultant relation has only those attributes of $r$ specified in $L$
    
    e.g., $\pi_{\text{std\_name}, \text{std\_phone\#}}$ (Student)
Relational Algebra

Example.

\[ \sigma_A = "a_1" \ (r_1) \]

\[
\begin{array}{ccc}
A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_1 & b_3 & c_1 \\
\end{array}
\]

\[ \pi_{A,C} (r_1) \]

\[
\begin{array}{cc}
A & C \\
\hline
a_1 & c_1 \\
a_2 & c_2 \\
\end{array}
\]
The Relational Model

- **Cartesian Product (×)**
  - combines information from two relations
  - \( r_1(A_1, A_2, ..., A_n) \times r_2(B_1, B_2, ..., B_m) = r_3(A_1, ..., A_n, B_1, ..., B_m) \), and
    \[ \forall t \in r_3, t[A_1, ..., A_n] \in r_1, t[B_1, ..., B_m] \in r_2 \]
  - if \(|r_1| = i \) and \(|r_2| = j\), then \(|r_1 \times r_2| = i \times j\)

- **Union (∪)**
  - \( r_1 ∪ r_2 \): combines tuples in \( r_1 \) with those tuples in \( r_2 \)
  - *duplicate* tuples are eliminated
  - the two involved relations must be *union-compatible*
  - union-compatible:
    1) same *arity* (degree), i.e., same number of attributes
    2) attributes in the corresponding columns must have the same *domain*
Relational Algebra

Example (continued).

<table>
<thead>
<tr>
<th>( r_1 )</th>
<th>( r_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( A )</td>
</tr>
<tr>
<td>( B )</td>
<td>( B )</td>
</tr>
<tr>
<td>( C )</td>
<td>( C )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( a_1 )</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( b_3 )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( a_2 )</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( a_1 )</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>( b_3 )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( c_1 )</td>
</tr>
</tbody>
</table>

\[ r_1 \times r_2 \]

<table>
<thead>
<tr>
<th>( r_1.A )</th>
<th>( r_1.B )</th>
<th>( r_1.C )</th>
<th>( r_2.A )</th>
<th>( r_2.B )</th>
<th>( r_2.C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
</tbody>
</table>

\[ r_1 \cup r_2 \]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( b_3 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
</tbody>
</table>
The Relational Model

- **Set Difference (−)**
  - \( r_1 - r_2 \): contains tuples which occur in \( r_1 \) but not in \( r_2 \)
  - the two involved relations must be *union-compatible*

- **Rename (\( \rho \))**
  - \( \rho_s(r) \) returns relation \( r \) under the name \( s \), or
  - \( \rho_{B \leftarrow A}(r) \) returns attribute \( A \) under the attribute name \( B \)
Relational Algebra

Example (continued).

<table>
<thead>
<tr>
<th></th>
<th>$r_1$</th>
<th></th>
<th>$r_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$\neq$</td>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$\neq$</td>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$=$</td>
<td>$a_1$</td>
<td>$b_3$</td>
<td>$c_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$r_1 - r_2$</th>
<th></th>
<th>$\rho_s(r_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td></td>
<td>$a_1$</td>
<td>$b_3$</td>
<td>$c_1$</td>
</tr>
<tr>
<td></td>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td></td>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td></td>
<td>$a_1$</td>
<td>$b_3$</td>
<td>$c_1$</td>
</tr>
</tbody>
</table>
The Relational Model

- Additional Relational Operations
  - simplify queries constructed by using the fundamental operators
  - more convenient since less operators involved
  - do not add expressive power to the relational model

- Additional Operations:
  - Set Intersection (\(\cap\))
    - \(r_1 \cap r_2\) contains common tuples in \(r_1\) and \(r_2\)
    - the two involved relations must be union-compatible
    - can be rewritten using set difference, i.e.,

\[
r \cap s = r - (r - s)
\]
Relational Algebra

Example (continued).

<table>
<thead>
<tr>
<th>$r_1$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$r_2$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>

$A \cap r_2$

<table>
<thead>
<tr>
<th>$r_1 \cap r_2$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_3$</td>
</tr>
</tbody>
</table>
The Relational Model – Additional Operators

- **Natural Join (⋈)**: a combination of ×, σ, π in that order
  - ×: *merge* two relations into one
  - σ: *select* equal values on common attributes
  - π: *remove* duplicated columns

- A “⋈” becomes “×” if the two operand relations have no common attributes, e.g.,
  
  \[ r(A, B) \bowtie r(C, D) = r(A, B) \times s(C, D) \]

- **Rewrite ** using ×, σ and π :

  \[ r(R) \bowtie s(S) = \pi_{R\cup S} (\sigma_{r.A_1 = s.A_1} \land \ldots \land r.A_n = s.A_n (r \times s)) \]

  where \( R \cap S = \{A_1, \ldots, A_n\} \)
Relational Algebra

Example (continued).

<table>
<thead>
<tr>
<th></th>
<th>r₁</th>
<th></th>
<th>r₃</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>b₁</td>
<td>d₃</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
<td>b₃</td>
<td>d₄</td>
</tr>
<tr>
<td>a₁</td>
<td>b₃</td>
<td>c₁</td>
<td>b₁</td>
<td>d₅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>r₁ ⊗ r₃</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>d₃</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>d₅</td>
</tr>
<tr>
<td>a₁</td>
<td>b₃</td>
<td>c₁</td>
<td>d₄</td>
</tr>
</tbody>
</table>
The Relational Model – Additional Operators

- **Assignment (←)**
  - assigns a relation with a relation variable, a convenient way to express complex queries, e.g.,

  \[ V \leftarrow \pi_{R - S}(r) \]

- **Division (÷)**
  - retrieve each (resultant) tuple in the *dividend* relation that “contain” *all* the tuple values of the *divisor* relation
  - *attribute*(divisor) \(\subseteq\) *attribute*(dividend)
  - resultant schema has *attribute*(dividend) – *attribute*(divisor),
  - e.g., Has_taken(Name, Course#, Status) \(\div\) Core_course(Course#)
Example.

<table>
<thead>
<tr>
<th>Name</th>
<th>Course#</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steve</td>
<td>Math 112</td>
<td>Freshman</td>
</tr>
<tr>
<td>Steve</td>
<td>CS 103</td>
<td>Freshman</td>
</tr>
<tr>
<td>Dan</td>
<td>CS 142</td>
<td>Junior</td>
</tr>
<tr>
<td>Dan</td>
<td>Math 112</td>
<td>Junior</td>
</tr>
<tr>
<td>Dan</td>
<td>CS 143</td>
<td>Junior</td>
</tr>
<tr>
<td>Linda</td>
<td>CS 142</td>
<td>Freshman</td>
</tr>
</tbody>
</table>

\[\text{Has}_\text{taken} \div \text{Core}_\text{course}\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Junior</td>
</tr>
</tbody>
</table>
Division Operation

**Example.** Let $r$ and $s$ be the following relations:

$\begin{array}{c|c}
A & B \\
\hline
\alpha & \{1\} \\
\alpha & \{2\} \\
\alpha & \{3\} \\
\beta & \{1\} \\
\gamma & \{1\} \\
\delta & \{1\} \\
\delta & \{3\} \\
\delta & \{4\} \\
\epsilon & \{6\} \\
\epsilon & \{1\} \\
\beta & \{2\}
\end{array}$ $\div$ $\begin{array}{c|c}
B & \\
\hline
\{1\} & \\
\{2\} & \\
\end{array}$

$\begin{array}{c}
A \\
\hline
\alpha \\
\beta
\end{array}$

$r \div s$
**Division Operation**

**Example.** Let $r$ and $s$ be the following relations:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\div$</th>
<th>$s$</th>
<th>$r \div s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\alpha$</td>
<td>${a, 1}$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${a, 1}$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${b, 1}$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${a, 1}$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${b, 3}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${a, 1}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</td>
<td>${b, 1}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\beta$</td>
<td>${b, 1}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$D$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, 1}$</td>
<td>${b, 1}$</td>
</tr>
</tbody>
</table>

<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>
<tr>
<td>$\gamma$</td>
<td>$a$</td>
<td>$\gamma$</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

  \[ R - S = (A_1, \ldots, A_m) \]

  \( r \div s = \{ t \mid t \in \prod_{R-S(r)} \land \forall u \in s (tu \in r) \} \)
Division Operation

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

To see why

- \( \Pi_{R \setminus S, S}(r) \) simply reorders attributes of \( r \)

- \( \Pi_{R \setminus S}((\Pi_{R \setminus S}(r) \times s) - \Pi_{R \setminus S, S}(r)) \), i.e., (iii), gives those tuples \( t \) in \( \Pi_{R \setminus S}(r) \) such that for some tuple \( u \in s \), \( tu \notin r \).
Division Operation

- **Sample Query**: “Find all customers who have an account at all branches located in Brooklyn city.”

branch (branch-name, branch-city, assets)
depositor (customer-name, account-number)
account (account-number, branch-name, balance)

\[ \Pi_{customer-name, branch-name} (depositor \bowtie account) \div \Pi_{branch-name} (\sigma_{branch-city = “Brooklyn”} (branch)) \]
Outer Join

- An extension of the join operation that avoids loss of information.

- Computes the (natural) join and then adds tuples from one relation that does 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 (roughly speaking) false by definition.

  - Will study precise meaning of comparisons with nulls later
### Outer Join

**Example**

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

- **Natural Join**: `loan ⨯ Borrower`

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

- **Left Outer Join**: `loan ⨀ Borrower`

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

## Example

### Right Outer Join: $loan \bowtie borrower$

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

### Full Outer Join: $loan \bowtie borrower$

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

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

- **Left outer join** $(\exists s)$ defined by other algebraic operators
  
  $$ r \exists s = (r \bowtie s) \cup ( (r - \Pi_R(r \bowtie s)) \times \{ (null, \ldots, null) \} ) $$

  where $(null, \ldots, null)$ is on the schema $S - R$

- **Right outer join** $(\exists s)$ defined by other algebraic operators
  
  $$ r \exists s = (r \bowtie s) \cup ( (s - \Pi_S(s \bowtie r)) \times \{ (null, \ldots, null) \} ) $$

  where $(null, \ldots, null)$ is on the schema $R - S$

- **Full outer join** $(\exists s)$ defined by *left* and *right* outer joins
  
  $$ r \exists s = (r \exists s) \cup (r \bowtie s) $$
Null Values

- It is possible for tuples to have a null value, denoted by `null`, for some of their attributes.
- Null signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null.
- For duplicate elimination and grouping, null is treated like a value, and two nulls are assumed to be the same.
  - Alternative: assume each null is different from each other (adopted)
  - Both are arbitrary decisions
- Relational (comparison) operations (such as >, <, =, etc.) involving null values return the special truth value unknown, which is null.
Null Values

- Three-valued logic using the truth value *unknown*:

**AND:**
- \((true \ and \ unknown)\) = *unknown*
- \((false \ and \ unknown)\) = *false*
- \((unknown \ and \ unknown)\) = *unknown*

**OR:**
- \((unknown \ or \ true)\) = *true*
- \((unknown \ or \ false)\) = *unknown*
- \((unknown \ or \ unknown)\) = *unknown*

**NOT:**
- \((not \ unknown)\) = *unknown*
Null Values

- Selection: Comparisons with null values return the special truth value *unknown*, i.e., not selected

- Natural Join: joining tuples on null values cause *mismatch*

- Projection: treating nulls like any other values when eliminating duplicates; our assumption is: *difference*

- Union, Intersect, Difference: treating nulls just as the *projection* does, i.e., null values in corresponding columns in two tuples are treated as *different*

- Outer join: behaving just like *join* operations, except on tuples that do not occur in the join results, which may be added to the result, depending on the operation.
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:

  \[ r \leftarrow r - E \]

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

Example. branch (branch-name, branch-city, assets)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)

- Delete all account records in the Perryridge branch.
  \[\text{account} \leftarrow \text{account} - \sigma_{\text{branch-name} = \text{“Perryridge”}} (\text{account})\]

- Delete all loan records with amount in the range of 0 to 50.
  \[\text{loan} \leftarrow \text{loan} - \sigma_{\text{amount} \geq 0 \text{ and amount} \leq 50} (\text{loan})\]

- Delete all accounts at branches located in Needham.
  \[r_1 \leftarrow \sigma_{\text{branch-city} = \text{“Needham”}} (\text{account} \bowtie \text{branch})\]
  \[r_2 \leftarrow \Pi_{\text{account-number, branch-name, balance}} (r_1)\]
  \[\text{account} \leftarrow \text{account} - r_2\]
  \[r_3 \leftarrow \Pi_{\text{customer-name, account-number}} (r_2 \bowtie \text{depositor})\]
  \[\text{depositor} \leftarrow \text{depositor} - r_3\]
To insert data into a relation, we either:

- specify a tuple to be inserted, or
- 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

Example. account (account-number, branch-name, balance) depositor (customer-name, account-number) borrower (customer-name, loan-number) loan (loan-number, branch-name, amount)

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

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

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

- 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.

  \[
  r_1 \leftarrow (\sigma_{\text{branch-name} = \text{"Perryridge"}}(\text{borrower} \bowtie \text{loan}))
  \]

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

  \[
  \text{depositor} \leftarrow \text{depositor} \cup \Pi_{\text{customer-name, loan-number}}(r_1)
  \]
Modification

- A mechanism to change a value in a tuple without changing all the 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 \( F_i \) (1 \( \leq \) i \( \leq \) n) is either the \( i^{th} \) attribute of \( r \), if the \( i^{th} \) attribute is not updated, or a predicate involving the \( i^{th} \) attribute, if it is the attribute to be updated.

- \( F_i \) is an expression, involving only constants and the attributes of \( r \), which gives the new value for the attribute.
Modification

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

- Make interest payments by increasing all balances by 5%
  \[ \text{account} \leftarrow \Pi_{\text{account-number, branch-name, balance}} \ast 1.05 (\text{account}) \]

- Pay all accounts with balances over $10,000 6% interest and pay all others 5%
  \[ \text{account} \leftarrow \Pi_{\text{account-number, branch-name, balance}} \ast 1.06 (\sigma_{\text{balance} > 10000} (\text{account})) \cup \Pi_{\text{account-number, branch-name, balance}} \ast 1.05 (\sigma_{\text{balance} \leq 10000} (\text{account})) \]
The Relational Model

- Database Updates
  - A DB application provides a mechanism for updating data
  - Method for updating data: use an interactive query/update language
  - Updating means *inserting*, *deleting*, and *modifying* db data

  - **Delete**
    - Removes selected tuples from db tables
    - Relational algebra expression: $r \leftarrow r - E$, where $r$ is a relation & $E$ is a relational algebra query

  - **Insertion**
    - Add tuples into db tables
    - $\forall t = <a_1, a_2, \ldots, a_n>$, a tuple to be inserted into relation $r(A_1, \ldots, A_n)$,
      $a_i \in \text{dom}(A_i), 1 \leq i \leq n$
    - Relational algebra expression: $r \leftarrow r \cup E$

  - **Modification ($\delta$)**
    - Changes a value in a tuple
    - Relational algebra expression: $\delta_A \leftarrow E(r)$, where $A$ is an attr of relation $r$, & $E$ is an arithmetic expression which includes constants & attrs of $r$. 
The Relational Model

- Tuple Relational Calculus
  - A nonprocedural (declarative) query language
  - Formal DB language based on 1st order predicate calculus
  - Tuple variables range over tuples of a relation
  - Attribute values: \( t[A] \), value of attribute \( A \) in tuple variable \( t \)
  - Tuple calculus expressions:
    \[
    \{ t \mid P(t) \}
    \]
    i.e., \( \forall t \): predicate (formula) \( P \) is true for \( t \)
  - Quantifier: \( \forall \) (universal/for all), \( \exists \) (existential/there exists)
  - Free/bound tuple variable: a variable not quantified/quantified by a quantifier
    \[
    \text{e.g., } \forall t \in r_1 \land s \in r_2
    \]
  - Formulas are made up of atoms
The Relational Model

- Tuple Relational Calculus
  
  *Atoms* are of the following types:
  
  - $t \in r$, where $t$ is a *tuple variable* and $r$ is a *relation*
  
  - $t_1[X] \theta t_2[Y]$, where
    - $t_1$ and $t_2$ are (same) tuple variables
    - $X$, an attribute of $t_1$ and $Y$, an attribute of $t_2$
    - $\theta \in \{ <, \leq, =, \neq, >, \geq \}$
    - $dom(X)$ and $dom(Y)$ are $\theta$-compatible
  
  - $t[X] \theta C$, where
    - $t$ is a *tuple variable*
    - $X$, an attribute of $t$
    - $C$, a constant in $dom(X)$
    - $\theta$, same as above, a comparison operator
The Relational Model

- Tuple Calculus Expressions
  - Well-formed formula (wff):
    1. an \textbf{atom} is wff
    2. if $P$ is a wff, then
        - $P$ is a wff
        - $(P)$ is a wff
    3. if $P_1$ and $P_2$ are wffs, then
        - $P_1 \lor P_2$ is a wff
        - $P_1 \land P_2$ is a wff
        - $P_1 \Rightarrow P_2$ is a wff
    4. if $P(t)$ is a wff, where $t$ is free, then
        - $\exists t \in r \ (P(t))$ is a wff
        - $\forall t \in r \ (P(t))$ is a wff
## The Relational Model

### Relational Algebra Expressions vs. Tuple Calculus Expressions

<table>
<thead>
<tr>
<th>Relational Alg. Expression</th>
<th>Tuple Calculus Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select:</strong> $\sigma_C(r)$</td>
<td>${ t \mid t \in r \land C' }$, where $C'$ is the condition $C$ with each appearance of attribute $A$ of $r$ replaced by $t[A]$</td>
</tr>
<tr>
<td><strong>Project:</strong> $\Pi_{A_1, \ldots, A_n}(r)$</td>
<td>${ t \mid \exists u \in r \ (t[A_1] = u[A_1] \land \ldots \land t[A_n] = u[A_n]) }$</td>
</tr>
<tr>
<td><strong>Cartesian:</strong> $r(R) \times s(S)$</td>
<td>${ t \mid \exists x \in r \ (\exists y \in s \ (t[A_1] = x[A_1] \land \ldots \land t[A_n] = x[A_n] \land t[B_1] = y[B_1] \land \ldots \land t[B_m] = y[B_m])) }$</td>
</tr>
<tr>
<td><strong>Product</strong></td>
<td>where $R = {A_1, \ldots, A_n}$ and $S = {B_1, \ldots, B_m}$</td>
</tr>
<tr>
<td><strong>Union:</strong> $r \cup s$</td>
<td>${ t \mid t \in r \lor t \in s }$</td>
</tr>
<tr>
<td><strong>Difference:</strong> $r - s$</td>
<td>${ t \mid t \in r \land \neg(t \in s) }$</td>
</tr>
</tbody>
</table>
Sample Tuple Calculus Queries

Example. \( \text{loan (loan-number, branch-name, amount)} \)

- Find the \textit{loan-number, branch-name, and 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.

\[ \{ t \mid \exists n \in \text{loan} \ (t[\text{loan-number}] = n[\text{loan-number}] \land n[\text{amount}] > 1200) \} \]
Sample Tuple Calculus Queries

Example. borrower (customer-name, loan-number)
          depositor (customer-name, account-number)

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

  \{ t \mid \exists b \in borrower (t [customer-name] = b[customer-name]) \lor \\
  \exists d \in depositor (t [customer-name] = d[customer-name]) \}

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

  \{ t \mid \exists b \in borrower (t [customer-name] = b[customer-name]) \land \\
  \exists d \in depositor (t [customer-name] = d[customer-name]) \}
**Sample Tuple Calculus Queries**

**Example.**

- borrower (customer-name, loan-number)
- loan (loan-number, branch-name, amount)
- depositor (customer-name, account-number)

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

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

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

\[
\{ t \mid \exists b \in \text{borrower} (t[\text{customer-name}] = b[\text{customer-name}] \land \\
\exists n \in \text{loan} (b[\text{loan-number}] = n[\text{loan-number}] \land \\
\quad n[\text{branch-name}] = \text{“Perryridge”})) \land \\
\n\# d \in \text{depositor} (d[\text{customer-name}] = t[\text{customer-name}])\}
\]
Sample Tuple Calculus Queries

Example.

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

\[
\{ \, t \mid \exists n \in \text{loan} (n[\text{branch-name}] = \text{“Perryridge”}) \land \\
\exists b \in \text{borrower} (b[\text{loan-number}] = n[\text{loan-number}]) \land \\
\quad t[\text{customer-name}] = b[\text{customer-name}] \land \\
\exists c \in \text{customer} (c[\text{customer-name}] = b[\text{customer-name}]) \land \\
\quad t[\text{customer-city}] = c[\text{customer-city}]) \}\}
\]

\text{loan (loan-number, branch-name, amount)}
\text{borrower (customer-name, loan-number)}
\text{customer (customer-name, customer-street, customer-city)}
Sample Tuple Calculus Queries

Example.

Find the names and cities of all customers who have an account at all branches located in *Brooklyn*.

\[
\{ t \mid \exists c \in \text{customer} \ (t[\text{customer-name}] = c[\text{customer-name}] \land t[\text{customer-city}] = c[\text{customer-city}] \land \forall b \in \text{branch} \ (b[\text{branch-city}] = \text{"Brooklyn"} \Rightarrow \exists a \in \text{account} \ (a[\text{branch-name}] = b[\text{branch-name}] \land \exists d \in \text{depositor} \ (d[\text{account-number}] = a[\text{account-number}] \land d[\text{customer-name}] = c[\text{customer-name}])))\}
\]
The Relational Model

- **Safe Tuple Calculus Expressions**

  - Expressions generate *infinite* relations, e.g., \( \{ t \mid \neg (t \in r) \} \), is *not* safe
    - An expression \( E: \{ t \mid P(t) \} \) is *safe* if each value in \( E \in \text{dom}(P) \)
    - \( \text{dom}(P) = \{ \text{Constants in } P \} \cup \{ \text{Attribute values of relations in } P \} \)