Chapter 2: Intro to Relational Model
&
Chapter 6.1: Relational Algebra
Example of a Relation

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>58583</td>
<td>Califéri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
</tbody>
</table>

Relation
(or table)

attributes
(or columns)

(tuples
(or rows)
Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute.
- Attribute values are (normally) required to be **atomic**; that is, indivisible.
- The special value **null** is a member of every domain.
- The null value causes complications in the definition of many operations.
Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are attributes
- $R = (A_1, A_2, \ldots, A_n)$ is a relation schema
  
  Example:
  
  $$instructor = (ID, \; name, \; dept\_name, \; salary)$$

- Formally, given sets $D_1, D_2, \ldots, D_n$ a relation $r$ is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  
  Thus, a relation is a set of $n$-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

- 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
Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: instructor relation with unordered tuples

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
</tbody>
</table>
Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts
  
  \[
  \text{instructor} \\
  \text{student} \\
  \text{advisor}
  \]

- Bad design:
  
  \[
  \text{univ (instructor -ID, name, dept\_name, salary, student\_Id, ..)}
  \]

  results in
  
  - repetition of information (e.g., two students have the same instructor)
  - the need for null values (e.g., represent an student with no advisor)

- Normalization theory (Chapter 7) deals with how to design “good” relational schemas
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 $r(R)$
  - Example: $\{ID\}$ and $\{ID, name\}$ are both superkeys of `instructor`.
- Superkey $K$ is a **candidate key** if $K$ is minimal
  - Example: $\{ID\}$ is a candidate key for `Instructor`
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
    - Example: `teaches(ID, course_id, sec_id, semester, year)`
  - **Referenced** relation: referenced attributes must be **primary key attributes**
    - Example: `instructor(ID, name, dept_name, salary)`
Relational Query Languages

- Procedural vs. non-procedural (declarative)

- “Pure” languages: fundamental, lacking the “syntactic sugar”
  - Relational algebra (procedural)
  - Tuple relational calculus (non-procedural)
  - Domain relational calculus (non-procedural)
Relational Algebra

- Algebra: operators and operands
  - Relational algebra
    - Operands: relations
    - Operators: basic operators (+ additional operations)
- Six basic operators
  - select: $\sigma$
  - project: $\Pi$
  - union: $\cup$
  - set difference: $-$
  - Cartesian product: $\times$
  - rename: $\rho$
- The operators take one or two relations as inputs and produce a new relation as a result.
Select Operation – Example

- Relation $r$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>
Select Operation

- Notation: $\sigma_p(r)$
- $p$ is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{ t | t \in r \text{ and } p(t) \}$$

Where $p$ is a formula in propositional calculus consisting of **terms** connected by: $\land$ (and), $\lor$ (or), $\neg$ (not)

Each **term** is one of:

$$<\text{attribute}> \ op \ <\text{attribute}> \text{ or } <\text{constant}>$$

where $op$ is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\text{instructor (ID, name, dept\_name, salary)}$$

$$\sigma_{\text{dept\_name=“Physics”}}(\text{instructor})$$
## 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>1</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>40</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

- \( \Pi_{A,C} (r) \)

\[
\begin{array}{cc}
A & C \\
\alpha & 1 \\
\alpha & 1 \\
\beta & 1 \\
\beta & 2 \\
\end{array}
\quad = 
\begin{array}{cc}
A & C \\
\alpha & 1 \\
\beta & 1 \\
\beta & 2 \\
\end{array}
\]
Project Operation

- Notation: \[ \Pi_{A_1, A_2, \ldots, A_k}(r) \]
  
  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.

- Example: To eliminate the \textit{dept\_name} attribute of \textit{instructor}
  
  \textit{instructor} (ID, name, dept\_name, salary)

  \[ \Pi_{ID, name, salary}(\textit{instructor}) \]
Composition of Operations

- Can build expressions using multiple operations
- Example: $\Pi_{B,C} (\sigma_{A=\alpha} (r))$

Relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

$\sigma_{A=\alpha} (r)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

$\Pi_{B,C} (\sigma_{A=\alpha} (r))$

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>5</td>
</tr>
</tbody>
</table>
Exercise

employee (person_name, street, city, salary)

- Find the names of all employees who live in city “Seoul”

- Find the names of all employees whose salary is greater than 100,000

- Find the names of all employees who live in “Seoul” and whose salary is greater than 100,000
Union 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}
  
  \]

- \( r \cup s \):
  \[
  \begin{array}{|c|c|}
  \hline
  A & B \\
  \hline
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \beta & 3 \\
  \hline
  \end{array}
  \]

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 (example: 2$^{nd}$ column of $r$ deals with the same type of values as does the 2$^{nd}$ column of $s$)

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

  $\text{section (course_id, sec_id, semester, year, building, room_number, time_slot_id)}$

  $$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{“Fall”} \land \text{year}=2009} (\text{section})) \cup$$

  $$\Pi_{\text{course_id}} (\sigma_{\text{semester}=\text{“Spring”} \land \text{year}=2010} (\text{section}))$$
Set difference of two relations

- 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>

  $r$

<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>

  $s$

- $r - s$:

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

- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

\[
\Pi_{\text{course_id}} \left( \sigma_{\text{semester} = \text{"Fall"}} \land \text{year=2009} \left( \text{section} \right) \right) - \\
\Pi_{\text{course_id}} \left( \sigma_{\text{semester} = \text{"Spring"}} \land \text{year=2010} \left( \text{section} \right) \right)
\]
**Cartesian-Product Operation – Example**

- Relations $r$, $s$:

  $$
  \begin{array}{cc}
  A & B \\
  \alpha & 1 \\
  \beta & 2 \\
  \end{array}
  \quad
  \begin{array}{ccc}
  C & D & E \\
  \alpha & 10 & a \\
  \beta & 10 & a \\
  \beta & 20 & b \\
  \gamma & 10 & b \\
  \end{array}
  $$

- $r \times s$:

  $$
  \begin{array}{cccccc}
  A & B & C & D & E \\
  \alpha & 1 & \alpha & 10 & a \\
  \alpha & 1 & \beta & 10 & a \\
  \alpha & 1 & \beta & 20 & b \\
  \alpha & 1 & \gamma & 10 & b \\
  \beta & 2 & \alpha & 10 & a \\
  \beta & 2 & \beta & 10 & a \\
  \beta & 2 & \beta & 20 & b \\
  \beta & 2 & \gamma & 10 & b \\
  \end{array}
  $$
Cartesian-Product Operation

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

- Same attribute name may appear in both $r$ and $s$
  - Attach to an attribute the name of the relation from which the attribute originally came
    e.g.) $(instructor.ID, instructor.name, instructor.dept_name, instructor.salary$
    $teaches.ID, teaches.course_id, teaches.sec_id, teacher.semester, teaches.year)$
  - Can drop relation-name prefix for the attributes that appear in only one schema

- Assume that attributes of $r(R)$ and $s(S)$ are disjoint. (That is, $R \cap S = \emptyset$).
- Even then, if attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming must be used.
  e.g.) Cartesian-product of a relation with itself
Exercise

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

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

- Find the names of all customers who have a loan at the “Gwanak” branch.

- Find the names of all customers who have a loan at the “Gwanak” branch but do not have an account at any branch of the bank.
Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

\[ \rho_x(E) \]

returns the expression \( E \) under the name \( X \)

- If a relational-algebra expression \( E \) has arity \( n \), then

\[ \rho_{x}(A_1, A_2, \ldots, A_n)(E) \]

returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, \ldots, A_n \)
Example Query

■ Find the largest salary in the university

\[ \text{instructor (ID, name, dept\_name, salary)} \]

- Step 1: find instructor salaries that are less than some other instructor salary (i.e. not maximum)
  - using a copy of instructor under a new name \( d \)

\[ \prod_{\text{instructor} . \text{salary}} (\sigma_{\text{instructor} . \text{salary} < d . \text{salary}} (\text{instructor} \times \rho_d (\text{instructor}))) \]

- Step 2: Find the largest salary

\[ \prod_{\text{salary}} (\text{instructor}) - \]

\[ \prod_{\text{instructor} . \text{salary}} (\sigma_{\text{instructor} . \text{salary} < d . \text{salary}} (\text{instructor} \times \rho_d (\text{instructor}))) \]
Example Queries

- Find the names of all instructors in the Physics department, along with the course_id of all courses they have taught

- Query 1
  \[ \Pi_{\text{instructor.ID, course_id}} (\sigma_{\text{dept_name}="\text{Physics"}} (\sigma_{\text{instructor.ID=teaches.ID}} (\text{instructor x teaches})))) \]

- Query 2
  \[ \Pi_{\text{instructor.ID, course_id}} (\sigma_{\text{instructor.ID=teaches.ID}} (\sigma_{\text{dept_name}="\text{Physics"}} (\text{instructor} \times \text{teaches})))) \]
A basic expression in the relational algebra consists of either one of the following:

- 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:

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_p (E_1)$, $P$ is a predicate on attributes in $E_1$
- $\Pi_s(E_1)$, $S$ is a list consisting of some of the attributes in $E_1$
- $\rho_x (E_1)$, $x$ is the new name for the result of $E_1$
Additional Operations

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

- Set intersection
- Natural join
- Assignment
- Outer join
Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
  $$r \cap s = \{ t | t \in r \text{ and } t \in s \}$$
- Assume:
  - $r, s$ have the same *arity*
  - attributes of $r$ and $s$ are compatible
- Note: $r \cap s = r - (r - s)$
### Set-Intersection Operation – Example

- **Relation** \( r, s \):

<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>( \alpha )</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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

- **\( r \cap s \)**:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Natural-Join Operation

- Notation: \( r \bowtie s \)

Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively. Then, \( r \bowtie s \) is a relation on schema \( R \cup S \) obtained as follows:
  
  - Consider 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 \( R \cap S \), add a tuple \( t \) to the result, where
    - \( t \) has the same value as \( t_r \) on \( r \)
    - \( t \) has the same value as \( t_s \) on \( s \)

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 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>$r$</td>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>$\beta$</td>
<td>2</td>
<td>$\gamma$</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>$\gamma$</td>
<td>4</td>
<td>$\beta$</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>1</td>
<td>$\gamma$</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>$\delta$</td>
<td>2</td>
<td>$\beta$</td>
<td>b</td>
</tr>
</tbody>
</table>

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

$r \bowtie s$

<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>$r \bowtie s$</td>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>a</td>
<td>$\alpha$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>a</td>
<td>$\gamma$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>1</td>
<td>$\gamma$</td>
<td>a</td>
<td>$\alpha$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>1</td>
<td>$\gamma$</td>
<td>a</td>
<td>$\gamma$</td>
</tr>
<tr>
<td></td>
<td>$\delta$</td>
<td>2</td>
<td>$\beta$</td>
<td>b</td>
<td>$\delta$</td>
</tr>
</tbody>
</table>
Natural Join and Theta Join

Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach

\[ \Pi_{name, \text{title}} (\sigma_{\text{dept_name} = \text{Comp. Sci.}} (\text{instructor} \bowtie \text{teaches} \bowtie \text{course})) \]

Natural join is associative

\[ (\text{instructor} \bowtie \text{teaches}) \bowtie \text{course} \] is equivalent to \[ \text{instructor} \bowtie (\text{teaches} \bowtie \text{course}) \]

Natural join is commutative

\[ \text{instruct} \bowtie \text{teaches} \] is equivalent to \[ \text{teaches} \bowtie \text{instructor} \]

The **theta join** operation \( r \bowtie_{\theta} s \) is defined as

\[ r \bowtie_{\theta} s = \sigma_{\theta} (r \times s) \]
Exercise

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

- Find all customers who have an account from at least the “Gwanak” and “Gangnam” branches.
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 a result of the query.
  - Assignment must always be made to a temporary relation variable.

- Modification of the database can be expressed using the assignment operator
Assignment Example

- Rewrite $r \times s$ with assignment operations

\[
\begin{align*}
temp1 & \leftarrow r \times s \\
temp2 & \leftarrow \sigma_{r.A_1 = s.A_1 \land r.A_2 = s.A_2 \land \ldots \land r.A_n = s.A_n} (temp1) \\
result & \leftarrow \Pi_{R \cap S}(temp2)
\end{align*}
\]
Outer Join

- An extension of the join operation that avoids loss of information
- Computes the 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.
    - We shall study precise meaning of comparisons with nulls later
Natural Join – Example

Relation `course`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation `prereq`

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- Natural Join

`course` \( \bowtie \) `prereq`
# Left Outer Join – Example

## Relation `course`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

## Relation `prereq`

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- **Left Outer Join**

  \[
  \text{course} \LeftJoin \text{prereq}
  \]

## Result Table

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
<td>null</td>
</tr>
</tbody>
</table>
Right Outer Join – Example

Relation course

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- Right Outer Join
  
  course \( \bowtie \) prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>CS-101</td>
</tr>
</tbody>
</table>
Full Outer Join – Example

Relation *course*

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation *prereq*

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- Full Outer Join

  course ⨆ prereq
Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. \( r \bowtie s \) can be written as
    \[
    (r \bowtie s) \cup (r - \Pi_{R}(r \bowtie s)) \times \{(null, \ldots, null)\}
    \]
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*.

- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)
Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* is used instead of *unknown*?
    
    $$(1 < \text{null}) = \text{false} \implies \neg (1 < \text{null}) = \text{true} \quad (!)$$

- Three-valued logic using the truth value *unknown*:
  - **OR:**
    
    $$(\text{unknown or true}) = \text{true},$$
    $$(\text{unknown or false}) = \text{unknown}$$
    $$(\text{unknown or unknown}) = \text{unknown}$$

  - **AND:**
    
    $$(\text{true and unknown}) = \text{unknown},$$
    $$(\text{false and unknown}) = \text{false},$$
    $$(\text{unknown and unknown}) = \text{unknown}$$

  - **NOT:**
    
    $$(\neg \text{unknown}) = \text{unknown}$$

- In SQL “*P is unknown*” evaluates to true if predicate *P* evaluates to *unknown*

- Result of select predicate is treated as *false* if it evaluates to *unknown*
Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are $m$ copies of $t1$ in $r$, and $n$ copies of $t2$ in $s$, there are $m \times n$ copies of $t1.t2$ in $r \times s$
  - Other operators similarly defined
    - E.g. union: $m + n$ copies, intersection: $\min(m, n)$ copies
    - difference: $\min(0, m - n)$ copies
End of Chapter 2 & 6.1