Database Applications (15-415)

ER to Relational & Relational Algebra
Lecture 4, January 20, 2015

Mohammad Hammoud
Today...

- **Last Session:**
  - The relational model

- **Today’s Session:**
  - ER to relational
  - Relational algebra
    - Relational query languages (in general)
    - Relational operators

- **Announcements:**
  - PS1 is due on Thursday, Jan 22 by midnight
  - In the next recitation we will practice on translating ER designs into relational databases
  - The recitation time and location will remain the same for the whole semester (i.e., every Thursday at 4:30PM in Room # 1190)
Outline

Translating ER Diagrams to Tables and Summary

Query Languages

Relational Operators
Strong Entity Sets to Tables

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must include:
  1. Keys for each participating entity set (as foreign keys)
     - This set of attributes forms a superkey for the relation
  2. All descriptive attributes

- Relationship sets
  - 1-to-1, 1-to-many, and many-to-many
  - Key/Total/Partial participation
M-to-N Relationship Sets to Tables

CREATE TABLE Works_In(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (ssn, did),
    FOREIGN KEY (ssn)
        REFERENCES Employees,
    FOREIGN KEY (did)
        REFERENCES Departments)
1-to-M Relationship Sets to Tables

CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn)
    REFERENCES Employees,
    FOREIGN KEY (did)
    REFERENCES Departments)

CREATE TABLE Departments(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    PRIMARY KEY (did),
)

Approach 1:
Create separate tables for Manages and Departments
# 1-to-M Relationship Sets to Tables

**Approach 2:**
Create a table for only the Departments entity set (i.e., take advantage of the key constraint)

```
CREATE TABLE Dept_Mgr(
    ssn    CHAR(11),
    did    INTEGER,
    since  DATE,
    dname  CHAR(20),
    budget REAL,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees
)
```

Can `ssn` take a `null` value?
One-Table vs. Two-Table Approaches

- The one-table approach:
  (+) Eliminates the need for a separate table for the involved relationship set (e.g., Manages)
  (+) Queries can be answered without combining information from two relations
  (-) Space could be wasted!
    - What if several departments have no managers?

- The two-table approach:
  - The opposite of the one-table approach!
Translating Relationship Sets with Participation Constraints

- What does the following ER diagram entail (with respect to Departments and Managers)?

Every *did* value in Departments table must appear in a row of the Manages table-*if defined-* (with a non-null ssn value!)

```
<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>since</th>
<th>did</th>
<th>dname</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employees</th>
<th>Manages</th>
<th>Departments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Works_In</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Translating Relationship Sets with Participation Constraints

- Here is how to create the “Dept_Mgr” table using the one-table approach:

```
CREATE TABLE Dept_Mgr(
   did INTEGER,
   dname CHAR(20),
   budget REAL,
   ssn CHAR(11) NOT NULL,
   since DATE,
   PRIMARY KEY (did),
   FOREIGN KEY (ssn) REFERENCES Employees,
   ON DELETE NO ACTION)
```

Can this be captured using the two-table approach?
Translating Relationship Sets with Participation Constraints

- Here is how to create the “Dept_Mgr” table using the one-table approach:

```
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE SET NULL
)
```

Would this work?
Translating Weak Entity Sets

- A weak entity set always:
  - Participates in a one-to-many binary relationship
  - Has a key constraint and total participation

- Which approach is ideal for that?
  - The one-table approach
Translating Weak Entity Sets

- Here is how to create “Dep_Policy” using the one-table approach

```
CREATE TABLE Dep_Policy (
    dname CHAR(20),
    age INTEGER,
    cost REAL,
    ssn CHAR(11) NOT NULL,
    PRIMARY KEY (dname, ssn),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```
Translating ISA Hierarchies to Relations

- Consider the following example:
Translating ISA Hierarchies to Relations

- General approach:
  - *Create* 3 relations: “Employees”, “Hourly_Emps” and “Contract_Emps”

- How many times do we record an employee?
- What to do on deletions?
- How to retrieve *all* info about an employee?
Translating ISA Hierarchies to Relations

- Alternatively:
  - Just create 2 relations “Hourly_Emps” and “Contract_Emps”
  - Each employee **must be** in one of these two subclasses

- ‘black’ is gone!

Duplicate Values!
Translating Aggregations

- Consider the following example:
Translating Aggregations

- Standard approach:
  - The Employees, Projects and Departments entity sets and the Sponsors relationship sets are translated as described previously.

- For the Monitors relationship, we create a relation with the following attributes:
  - The key attribute of Employees (i.e., ssn)
  - The key attributes of Sponsors (i.e., did, pid)
  - The descriptive attributes of Monitors (i.e., until)
The Relational Model: A Summary

- A tabular representation of data

- Simple and intuitive, currently one of the most widely used
  - Object-relational variant is gaining ground

- Integrity constraints can be specified (by the DBA) based on application semantics (DBMS checks for violations)
  - Two important ICs: primary and foreign keys
  - Also: not null, unique
  - In addition, we always have domain constraints

- Mapping from ER to Relational is (fairly) straightforward!
ER to Tables - Summary of Basics

- **Strong entities:**
  - Key -> primary key

- **(Binary) relationships:**
  - Get keys from all participating entities:
    - 1:1 -> either key can be the primary key
    - 1:N -> the key of the ‘N’ part will be the primary key
    - M:N -> both keys will be the primary key

- **Weak entities:**
  - Strong key + partial key -> primary key
  - ..... ON DELETE CASCADE
ER to Tables - Summary of Advanced

- **Total/Partial participation:**
  - NOT NULL

- **Ternary relationships:**
  - Get keys from all; decide which one(s) -> primary Key

- **Aggregation:** like relationships

- **ISA:**
  - 3 tables (most general)
  - 2 tables (‘total coverage’)
Relational Query Languages

- Query languages (QLs) allow manipulating and retrieving data from databases

- The relational model supports simple and powerful QLs:
  - Strong formal foundation based on logic
  - High amenability for effective optimizations

- Query Languages != programming languages!
  - QLs are not expected to be “Turing complete”
  - QLs are not intended to be used for complex calculations
  - QLs support easy and efficient access to large datasets
Formal Relational Query Languages

- There are two mathematical Query Languages which form the basis for commercial languages (e.g., SQL)
  - Relational Algebra
    - Queries are composed of operators
    - Each query describes a step-by-step procedure for computing the desired answer
    - Very useful for representing execution plans
  
- Relational Calculus
  - Queries are subsets of first-order logic
  - Queries describe desired answers without specifying how they will be computed
  - A type of non-procedural (or declarative) formal query language
Formal Relational Query Languages

- There are two mathematical Query Languages which form the basis for commercial languages (e.g., SQL)
  - Relational Algebra
    - Queries are composed of operators
    - Each query describes a step-by-step procedure for computing the desired answer
    - Very useful for representing execution plans
  - Relational Calculus
    - Queries are subsets of first-order logic
    - Queries do not describe the specific procedures for computing how they will be computed
    - A type of non-procedural (or declarative) formal query language

This session’s topic

Next session’s topic (very briefly)
Relational Algebra

- Operators (with notations):
  1. Selection (\( \bigcirc \))
  2. Projection (\( \bar{\pi} \))
  3. Cross-product (\( \times \))
  4. Set-difference (\( - \))
  5. Union (\( \cup \))
  6. Intersection (\( \cap \))
  7. Join (\( \Join \))
  8. Division (\( \div \))
  9. Renaming (\( \rho \))

- Each operation returns a relation, hence, operations can be composed! (i.e., Algebra is “closed”)
Relational Algebra

- Operators (with notations):
  1. Selection (\(\sigma\))
  2. Projection (\(\pi\))
  3. Cross-product (\(\times\))
  4. Set-difference (\(-\))
  5. Union (\(\cup\))
  6. Intersection (\(\cap\))
  7. Join (\(\Join\))
  8. Division (\(\div\))
  9. Renaming (\(\rho\))

- Each operation returns a relation, hence, operations can be composed! (i.e., Algebra is “closed”)
The Projection Operation

- **Projection:** $\pi_{\text{att-list}}(R)$
  - "Project out" attributes that are NOT in $\text{att-list}$
  - The schema of the output relation contains ONLY the fields in $\text{att-list}$, with the same names that they had in the input relation

- **Example 1:** $\pi_{\text{sname, rating}}(S_2)$

<table>
<thead>
<tr>
<th>Input Relation:</th>
<th>Output Relation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>
The Projection Operation

- Example 2: \( \pi_{\text{age}}(S2) \)

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

- The projection operator eliminates duplicates!
  - Note: real DBMSs typically do not eliminate duplicates unless explicitly asked for
The Selection Operation

- **Selection**: \( \sigma_{condition} \ (R) \)
  - Selects rows that satisfy the selection condition
  - The schema of the output relation is identical to the schema of the input relation

- **Example**: \( \sigma_{rating > 8} \ (S2) \)

**Input Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Output Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Operator Composition

- The output relation can be the input for another relational algebra operation! (Operator composition)

- Example:

  \[
  \pi_{\text{sname}, \text{rating}} (\sigma_{\text{rating} > 8} (S2))
  \]

**Input Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Intermediate Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Final Output Relation:**

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

\[S2\]
The Union Operation

- **Union:** \( R \cup S \)
  - The two input relations must be union-compatible
    - Same number of fields
    - ‘Corresponding’ fields have the same type
  - The output relation includes all tuples that occur “in either” \( R \) or \( S \) “or both”
  - The schema of the output relation is identical to the schema of \( R \)

- **Example:** \( S_1 \cup S_2 \)

**Input Relations:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Output Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>
The Intersection Operation

- **Intersection**: $R \cap S$
  - The two input relations must be *union-compatible*
  - The output relation includes all tuples that occur “in both” $R$ and $S$
  - The schema of the output relation is identical to the schema of $R$

- **Example**: $S_1 \cap S_2$

### Input Relations:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Output Relation:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
The Set-Difference Operation

- **Set-Difference:** $R - S$
  - The two input relations must be *union-compatible*
  - The output relation includes all tuples that occur in $R$ “but not” in $S$
  - The schema of the output relation is identical to the schema of $R$

- **Example:** $S_1 - S_2$

**Input Relations:**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yummy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Output Relation:**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
</tbody>
</table>
The Cross-Product and Renaming Operations

- Cross Product: $R \times S$
  - Each row of R is paired with each row of S
  - The schema of the output relation concatenates S1’s and R1’s schemas
  - Conflict: R and S might have similar field names
  - Solution: Rename fields using the “Renaming Operator”
  - Renaming: $\rho(R(F), E)$

- Example: $S_1 \times R_1$

<table>
<thead>
<tr>
<th>Input Relations:</th>
<th>Output Relation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

Conflicts: Both S1 and R1 have a field called sid
The Cross-Product and Renaming Operations

- Cross Product: $\mathbf{RXS}$
  - Each row of R is paired with each row of S
  - The schema of the output relation concatenates S1’s and R1’s schemas
  - Conflict: R and S might have the same field name
  - Solution: Rename fields using the “Renaming Operator”
  - Renaming: $\rho(R(F), E)$

- Example: $S1 \times R1$

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>31</td>
<td>103</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Input Relations:

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>31</td>
<td>103</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Output Relation:

<table>
<thead>
<tr>
<th>(sid)</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$
Next Class

Relational Algebra (Cont’d)