**Review Session**

- ER and Relational
  - ER \(\cong\) Relational
  - Constraints, Weak Entities, Aggregation, ISA
- Relational Algebra \(\cong\) Relational Calculus
  - Selections/Projections/Joins/Division
- SQL (Division, Outer-Joins, Constraints)
- Your questions

**ER & Relational Review**

**Employees** (ssn CHAR(11), name CHAR(20), did INTEGER)

**Departments** (did INTEGER, dname CHAR(20), budget INTEGER)

**Manages** (did INTEGER, ssn CHAR(11), since DATE)
  - FOREIGN KEY (ssn) REFERENCES Employees
  - FOREIGN KEY (did) REFERENCES Departments

Each employee can manage zero, one or more departments. Each department has zero, one or more managers.

**Key Constraint**

**Employees** (ssn CHAR(11), name CHAR(20), lot INTEGER)

**Dept Mgr** (did INTEGER, ssn CHAR(11), dname CHAR(20), budget INTEGER, since DATE)
  - FOREIGN KEY (ssn) REFERENCES Employees

Each employee can manage zero, one or more departments. Each department has at most one manager.

**Key + Participation Constraint**

**Employees** (ssn CHAR(11), name CHAR(20), lot INTEGER)

**Dept Mgr** (did INTEGER, ssn CHAR(11), dname CHAR(20), budget INTEGER, since DATE)
  - FOREIGN KEY (ssn) REFERENCES Employees
  - FOREIGN KEY (did) REFERENCES Employees
  - ssn NOT NULL
  - ON DELETE CASCADE

Each employee can manage zero, one or more departments. Each department has exactly one manager.

**Participation Constraint**

**Employees** (ssn CHAR(11), name CHAR(20), lot INTEGER)

**Departments** (did INTEGER, dname CHAR(20), budget INTEGER)

**Manages** (did INTEGER, ssn CHAR(11), since DATE)
  - FOREIGN KEY (ssn) REFERENCES Employees
  - FOREIGN KEY (did) REFERENCES Employees

Insufficient! Additional checks required.

Each employee manages at least one department. Each department has at least one manager.

**Weak Entities**

**Dep_Policy** (pname CHAR(20), ssn CHAR(11), age INTEGER, cost REAL)
  - FOREIGN KEY (ssn) REFERENCES Employees
  - NOT NULL
  - ON DELETE CASCADE

Weak entities have only a “partial key” (dashed underline)
Aggregation

Used to model a relationship involving a relationship set.
Allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

<table>
<thead>
<tr>
<th>Employees</th>
<th>Monitors</th>
<th>Departments</th>
<th>Projects</th>
<th>Sponsors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employees(ssn, name, lot)</td>
<td>Monitors(pid FK, ssn FK, did_FK, until)</td>
<td>Departments(did, dname, budget)</td>
<td>Projects(pid, pbudget, started_on)</td>
<td>Sponsors(pid FK, did FK, since)</td>
</tr>
</tbody>
</table>

ISA (`is a`) Hierarchies

<table>
<thead>
<tr>
<th>ISA (<code>is a</code>) Hierarchies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA (<code>is a</code>)</td>
</tr>
<tr>
<td>ISA (<code>is a</code>)</td>
</tr>
<tr>
<td>ISA (<code>is a</code>)</td>
</tr>
<tr>
<td>ISA (<code>is a</code>)</td>
</tr>
</tbody>
</table>

Relational Algebra: 5 Basic Operations

- **Selection** (\( \sigma \)) Selects a subset of rows from relation (horizontal).
- **Projection** (\( \pi \)) Retains only wanted columns from relation (vertical).
- **Cross-product** (\( \times \)) Allows us to combine two relations.
- **Set-difference** (\( - \)) Tuples in r1, but not in r2.
- **Union** (\( \cup \)) Tuples in r1 and/or in r2.
- **Renaming** (\( \rho \)) E.g. \( \rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 x R1) \)

Division A/B

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td>p2</td>
<td>B1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
<td>p2</td>
<td>B2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
<td>p4</td>
<td>B2</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td>s1</td>
<td>A/B1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
<td>s2</td>
<td>A/B2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
<td>s3</td>
<td>A/B2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
<td>s4</td>
<td>A/B2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
<td>s1</td>
<td>A/B3</td>
</tr>
</tbody>
</table>

Compound Operations

- **Intersect** (\( \cap \))
- **Join**
  - **Condition Join** \( S1.Sid = R1.Sid \)
  - **Equijoin**: Special case where \( c \) contains only equalities
  - **Natural join**
    - Compute \( R \times S \)
    - Select rows where attributes that appear in both relations have equal values
    - Project all unique attributes and one copy of each of the common ones.

Tuple Relational Calculus

- **Query has the form: \( \{ T \mid p(T) \} \)**
  - \( p(T) \) denotes a formula in which tuple variable \( T \) appears.
  - \( S \mid S \subseteq Sailors \mid S.rating > 7 \mid R(R \subseteq Reserves \mid R.sid = S.sid \mid R.bid = 103) \)
  - Only one free variable
  - Bounded variable
Algebra ⇔ Calculus

Find the names of sailors who’ve reserved a red boat

\[ \text{Projection} \]
\[ \text{Join} \]
\[ \text{Join} \]
\[ \text{Selection} \]

Division example

Find the names of sailors who’ve reserved ALL red boats

\[ \text{Projection} \]
\[ \text{Join} \]
\[ \text{Join} \]
\[ \text{Selection} \]

SQL Query

The target-list contains (i) list of column names & (ii) terms with aggregate operations (e.g., \(\text{MIN}(S\text{.age})\)).

• column name list (i) can contain only attributes from the grouping-list.

Conceptual Evaluation

• Compute Cartesian product of all tables in FROM clause
• Discard rows not satisfying WHERE clause (Selection)
• Group the remaining rows according to Grouping-List
• Apply HAVING clause
• Apply SELECT list (Projection)
• If there is DISTINCT, eliminate duplicates
• Order remaining tuples according to ORDER BY

Division in SQL

Find sailors who’ve reserved ALL boats

\[ \text{Simpler} \]

```
SELECT s.sid, s.name, r.bid
FROM Sailors s
LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```
Constraints Over Multiple Relations

CREATE TABLE Sailors
(
  sid INTEGER,
  sname CHAR(10),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
  CHECK
  (
    (SELECT COUNT (S.sid) FROM Sailors S)
    + (SELECT COUNT (B.bid) FROM Boats B) < 100
  )
)

- Awkward and wrong!
- Only checks sailors!
- Only required to hold if the associated table is non-empty.
- ASSERTION is the right solution; not associated with either table.

CREATE ASSERTION smallClub
CHECK
  (
    (SELECT COUNT (S.sid) FROM Sailors S)
    + (SELECT COUNT (B.bid) FROM Boats B) < 100
  )

Number of boats plus number of sailors is < 100