The Relational Model

Why Study the Relational Model?
- Most widely used model.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- “Legacy systems” in older models
  - E.G., IBM’s IMS
- Recent competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: object-relational model
    - Informix Universal Server, Oracle, DB2

Relational Database: Definitions
- Relational database: a set of relations
- Relation: made up of 2 parts:
  - Instance: a table, with rows and columns.
    - #Rows = cardinality, #fields = degree / arity.
  - Schema: specifies name of relation, plus name and type of each column.
- Can think of a relation as a set of rows or tuples (i.e., all rows are distinct).

Example Instance of Students Relation

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Cardinality = 3, degree = 5, all rows distinct
Do all columns in a relation instance have to be distinct?

Logical DB Design: ER to Relational
- Entity sets to tables.

Example Instance

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0983763423</td>
<td>John</td>
<td>10</td>
</tr>
<tr>
<td>9384392483</td>
<td>Jane</td>
<td>10</td>
</tr>
<tr>
<td>3743923483</td>
<td>Jill</td>
<td>20</td>
</tr>
</tbody>
</table>
**Integrity Constraints (ICs)**

- IC: condition that must be true for any instance of the database
  - Domain constraints
  - Key constraints
  - Foreign key constraints (later)
- A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances
  - Avoids data entry errors too!

**Primary Key Constraints**

- A set of fields is a superkey for a relation if:
  1. No two distinct tuples can have same values in all fields
- A set of fields is a key if:
  1. The set of fields is a superkey
  2. No proper subset of the set of fields is a superkey
- If there’s >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key.
- E.g., ssn is a key for Employees. (What about name?)
  The set \{ssn, name\} is a superkey.

**Candidate Keys**

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key.

**Where do ICs Come From?**

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.
Relationship Sets to Tables

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

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)

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

Example Instance

<table>
<thead>
<tr>
<th>Employees</th>
<th>Departments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>name</td>
</tr>
<tr>
<td>0983763423</td>
<td>John</td>
</tr>
<tr>
<td>9384392483</td>
<td>Jane</td>
</tr>
<tr>
<td>3743923483</td>
<td>Jill</td>
</tr>
<tr>
<td>did</td>
<td>dname</td>
</tr>
<tr>
<td>101</td>
<td>Sales</td>
</tr>
<tr>
<td>105</td>
<td>Purchasing</td>
</tr>
<tr>
<td>108</td>
<td>Databases</td>
</tr>
</tbody>
</table>

Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation
  - Must correspond to primary key of the second relation
  - Like a ‘logical pointer’.
- **If all foreign key constraints enforced**, referential integrity is achieved, i.e., no dangling references.
  - Not like HTML links!

Enforcing Referential Integrity

- What if a new “Works_In” tuple is added that references a non-existent employee?
  - Reject it!
- What if an Employee tuple is deleted?
  - Also delete all Works_In tuples that refer to it.
  - Disallow deletion of Employee tuple that is referred to.
  - Set ssn to some default value
  - Set ssn in Works_In to null, denoting ‘unknown’
- Similar if primary key of Employee tuple is updated

Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT

CREATE TABLE Works_In(
  ssn CHAR(11),
did INTEGER,
since DATE,
PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn) REFERENCES Employees
  ON DELETE CASCADE ON UPDATE SET DEFAULT,
FOREIGN KEY (did) REFERENCES Departments
  ON DELETE SET NULL ON UPDATE CASCADE)
CREATE TABLE Employees (ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Reports_To (supervisor_ssn CHAR(11), subordinate_ssn CHAR(11),
FOREIGN KEY (supervisor_ssn) REFERENCES Employees,
FOREIGN KEY (subordinate_ssn) REFERENCES Employees)

CREATE TABLE Employees (ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments (did INTEGER,
dname CHAR(20),
budget FLOAT,
mgr_ssn CHAR(11),
PRIMARY KEY (did)
FOREIGN KEY (mgr_ssn) REFERENCES Employees)

CREATE TABLE Contracts (supplier_id INTEGER, part_id INTEGER, department_id INTEGER,
PRIMARY KEY (supplier_id, part_id, department_id),
FOREIGN KEY (supplier_id) REFERENCES Suppliers,
FOREIGN KEY (part_id) REFERENCES Parts,
FOREIGN KEY (department_id) REFERENCES Departments)

Each dept has at most one manager, according to the key constraint on Manages.

Each employee works in at least one department according to the participation constraint on Works_In.
**ER to Relational (contd.)**

A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.

**Relationship Sets to Tables**

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

**Translating Weak Entity Sets**

- Weak entity set and identifying relationship set are translated into a single table.
- When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dept_Policy (  
pname CHAR(20),  
age INTEGER,  
cost REAL,  /ssn CHAR(11) NOT NULL,  
PRIMARY KEY (pname, ssn),  
FOREIGN KEY (ssn) REFERENCES Employees,  
ON DELETE CASCADE)
```

**Translating ISA Hierarchies to Relations**

- General approach:
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (hourly_wages, hours_worked, ssn); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
  - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
- Alternative: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: ssn, name, lot, hourly_wages, hours_worked.  
  - Each employee must be in one of these two subclasses.
**Destroying and Altering Relations**

- **DROP TABLE Students**
  - Destroys the relation Students. The schema information and the tuples are deleted.

- **ALTER TABLE Students**
  - ADD COLUMN firstYear: integer
  - The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a null value in the new field.

---

**Adding and Deleting Tuples**

- Can insert a single tuple using:

  ```sql
  INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
  ```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

  ```sql
  DELETE FROM Students S WHERE S.name = 'Smith'
  ```

*Powerful variants of these commands are available; more later!*

---

**Relational Model: Summary**

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we always have domain constraints.
- Rules to translate ER to relational model