The Relational Model 2

Week 3
We have seen how to create a database schema, how do we create an actual database on our computers?

professor(PID : string, name : string)
course(PID : string, number : string, name : string)
teaches(PID : string, days : string)
...how do we create an actual database on our computers?

We use SQL, a language that allows us to build, modify and query databases.

```
professor(PID : string, name : string)
```

```
SQL> CREATE TABLE Professor
2       (pid CHAR(9),
3           name CHAR(50),
4           PRIMARY KEY(pid));
Table created.
```
SQL (Structured Query Language)

• SQL is a language that allows us to build, modify and query databases.
• SQL is an ANSI standard language. American National Standards Institute
• SQL is the “engine” behind Oracle, Microsoft SQL Server, etc.
• Most of these systems have built GUIs on top of the command line interface, so you don’t normally write statements directly in SQL (although you can).
Creating Relations in SQL

• Creates a Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

CREATE TABLE Students
(sid CHAR(20),
name CHAR(20),
login CHAR(10),
age INTEGER,
gpa REAL);

• As another example, the Enrolled table holds information about courses that students take.

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2));
Creating Domains

• Say you want to restrict the values of GPA (0 < GPA <= 4.0)

• Approach 1: Specify constraint when defining the table

```sql
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL check(gpa <= 4.0 AND gpa > 0) );
```
Creating Domains

• Approach 2: After CREATING TABLE, use ALTER TABLE

```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa REAL);

ALTER TABLE Students
ADD CONSTRAINT check_gpa CHECK(gpa > 0 AND gpa <= 4.0);
```

To specify a set of allowed values, do something like this (using either approach):
```sql
    ... CHECK(gender='M' OR gender='F')
```
Getting Info About Existing Tables

To get the list of existing tables in your database:

```
SELECT table_name
FROM user_tables;
```

or

```
SELECT table_name
FROM all_tables
WHERE owner='YOUR_ACCOUNT_NAME_IN_UPPER_CASE';
```

To get more about an existing table, say, Students:

```
Describe Students;
```
Adding and Deleting Tuples

• Can insert a single tuple using:

    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):

    DELETE
    FROM Students
    WHERE name = ‘Smith’;
The SQL Query Language

- To find all 18 year old students, we can write:

```sql
SELECT * 
FROM Students S 
WHERE S.age=18;
```

- To show just names and logins columns, replace the first line with:

```sql
SELECT S.name, S.login
```
Querying Multiple Relations

- What does this query compute?

```sql
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A';
```

We get:

<table>
<thead>
<tr>
<th>Students</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>(sid, name, login, age, gpa)</td>
<td>(sid, cid, grade)</td>
</tr>
<tr>
<td>53666 Jones jones@cs 18 3.4</td>
<td>53831 Carnatic101 C</td>
</tr>
<tr>
<td>53688 Smith smith@eecs 18 3.2</td>
<td>53831 Reggae203 B</td>
</tr>
<tr>
<td>53650 Smith smith@math 19 3.8</td>
<td>53650 Topology112 A</td>
</tr>
<tr>
<td>53666</td>
<td>53666 History105 B</td>
</tr>
</tbody>
</table>

We get:

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Topology112</td>
</tr>
</tbody>
</table>
Destroying and Altering Relations

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

  ```sql
  DROP TABLE Students;
  ```

- 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.

  ```sql
  ALTER TABLE Students
  ADD COLUMN firstYear integer;
  ```
Integrity Constraints (ICs)

- **IC**: condition that must be true for *any* instance of the database; e.g., *domain constraints*.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!
Primary Key Constraints Revisited

- A set of fields is a *key* for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. no subset of the key is also a key.
    - Otherwise, it’s a *superkey*.
    - If there is >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.

- e.g., *sid* is a key for Students. (What about *name*?) The set \{*sid, gpa*\} is a *superkey*. 
Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using `UNIQUE`), one of which is chosen as the primary key.
- What’s the difference between the two statements below?

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid) );

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid),
UNIQUE (cid, grade) );
```

Used carelessly, an IC can prevent the storage of database instances that arise in practice!
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’.
- e.g. *sid* is a foreign key referring to **Students**:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, **referential integrity** is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    - Links in HTML!
<table>
<thead>
<tr>
<th>artist_id</th>
<th>artist_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bono</td>
</tr>
<tr>
<td>2</td>
<td>Cher</td>
</tr>
<tr>
<td>3</td>
<td>Nuno Bettencourt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>artist_id</th>
<th>album_id</th>
<th>album_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>Schizophrenic</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Eat the rich</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Crave (single)</td>
</tr>
</tbody>
</table>
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```sql
CREATE TABLE Enrolled
    (sid CHAR(20), cid CHAR(20), grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid) REFERENCES Students);
```

<table>
<thead>
<tr>
<th>Enrolled</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
<td>grade</td>
</tr>
<tr>
<td>53666</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>53666</td>
</tr>
<tr>
<td>53688</td>
</tr>
<tr>
<td>53650</td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

• Consider Students and Enrolled; \( sid \) in Enrolled is a foreign key that references Students.

• What should be done if an Enrolled tuple with a non-existent student id is inserted? \((\text{Reject it!})\)

• What should be done if a Students tuple is deleted?
  – Also delete all Enrolled tuples that refer to it.
  – Disallow deletion of a Students tuple that is referred to.
  – Set \( sid \) in Enrolled tuples that refer to it to a default \( sid \).
  – (In SQL, also: Set \( sid \) in Enrolled tuples that refer to it to a special value \texttt{null}, denoting `unknown’ or `inapplicable’.)

• Similar if primary key of Students tuple is updated.
Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is **NO ACTION** *(delete/update is rejected)*
  - **CASCADE** *(also delete all tuples that refer to deleted tuple)*
  - **SET NULL / SET DEFAULT** *(sets foreign key value of referencing tuple)*

```sql
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2),
     PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid)
     REFERENCES Students
    ON DELETE CASCADE
    ON UPDATE SET DEFAULT
)
```
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.
Logical DB Design: ER to Relational

- Entity sets to tables:

```sql
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:
  - Keys for the participating entity set(s) as foreign keys
    - This set of attributes forms a superkey for the relation.
  - All descriptive attributes.

```sql
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)
```
• Each dept has at most one manager, according to the key constraint on Manages.
Translating ER Diagrams with Key Constraints

- **Option 1**: Map relationship to a table:
  - Note that *did* is the key now!
  - Separate tables for Employees and Departments.
- **Option 2**: Since each department has a unique manager, we could instead combine Manages and Departments.

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

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11),
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees);
```
CREATE TABLE Reports_To(
  supervisor_ssn  CHAR(11),
  subordinate_ssn  CHAR(11),
  PRIMARY KEY  (supervisor_ssn, subordinate_ssn),
  FOREIGN KEY (supervisor_ssn) REFERENCES Employees(ssn),
  FOREIGN KEY (subordinate_ssn) REFERENCES Employees(ssn));
Review: Participation Constraints

• Does every department have a manager?
  – If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial).
  • Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)

[Diagram of database relationships: Employees, Manages, Departments, Works_In]
Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
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);
```
Review: Weak Entities

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this identifying relationship set.

![Diagram of entities and relationships]

- Employees
  - name
  - ssn
  - lot

- Policy
  - cost
  - pname

- Dependents
  - age

Lot

Policy

Dependents
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.

```sql
CREATE TABLE Dep_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)
```
Review: Binary vs. Ternary Relationships

- What are the additional constraints in the 2nd diagram?

Bad design

Better design
Binary vs. Ternary Relationships (Cont.)

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.

CREATE TABLE Policies (  
  policyid INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (policyid),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE);  

CREATE TABLE Dependents (  
  pname CHAR(20),  
  age INTEGER,  
  policyid INTEGER,  
  PRIMARY KEY (pname, policyid),  
  FOREIGN KEY (policyid) REFERENCES Policies,  
  ON DELETE CASCADE);
ISA (‘is a’) Hierarchies

• As in C++, or other PLs, attributes are inherited.
  • If we declare A ISA B, every A entity is also considered to be a B entity.

• **Overlap constraints**: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? *(Allowed/disallowed)*

• **Covering constraints**: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

• Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.
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.
    • Contract_Emps: Every employee is recorded in Employees. Extra info recorded in Contract_Emps (contract_id); must delete Contract_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.*
  - Similar for Contract_Emps
  - Each employee must be in one of these two subclasses.
Views

- A view is just a relation, but we store a definition, rather than a set of tuples.

  CREATE VIEW YoungStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21;

- Views can be dropped using the DROP VIEW command.
Views and Security

• Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  – Given YoungStudents, but not Students or Enrolled, we can find young students who are enrolled, but not the $cids$ of the courses they are enrolled in.
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.
• Powerful and natural query languages exist.
• Rules to translate ER to relational model