The Relational Model

Chapter 3
Why Study the Relational Model?

❖ Most widely used model by industry.
  ▪ IBM, Informix, Microsoft, Oracle, Sybase, etc.

❖ It is simple, elegant, and efficient
  ▪ Entities and relations are represented as tables
  ▪ Tables allow for arbitrary referencing
    (Tables can refer to other tables)
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* or *arity*.
  - **Schema**: specifies name of relation, plus a name and type for each column.
    - e.g. Students(*sid*: string, *name*: string, *login*: string, 
      *age*: integer, *gpa*: real).
- Can think of a relation as a *set* of rows or *tuples*. 
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@cs</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 in a relation instance distinct—each relation is defined to be a set of unique tuples
Relational Query Languages

- The relational model supports simple and powerful *querying* of data.
- Often *declarative* instead of *imperative*
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
The SQL Query Language

❖ Developed by IBM (system R) in the 1970s
❖ Need for a standard since it is used by many vendors
❖ Standards:
  ▪ SQL-86
  ▪ SQL-89 (minor revision)
  ▪ SQL-92 (major revision)
  ▪ SQL-99 (major extensions, current standard)
To find all 18 year old students, we can write:

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

To find just names and logins, replace the first line

```
SELECT S.name, S.login
```

When a relation is referenced only once, the use of variables is optional.
Querying Multiple Relations

- What does the following query compute?

Given the following instances of Enrolled and Students:

Students:

<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@cs</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>

Enrolled:

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</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>

**Query:**

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A'
```
What does the following query compute?

Given the following instances of Enrolled and Students:

**Students:**

<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@cs</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>

**Enrolled:**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</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>

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>
Querying Multiple Relations

❖ What does the following query compute?

Given the following instances of Enrolled and Students:

Students:

<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@cs</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>

Enrolled:

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</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>

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

“Joins” two tables

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>
Creating Relations in SQL

- SQL for Students relation.
- 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)
```
Creating Relations in SQL

❖ Another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
    (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2))
```
Destroying and Altering Relations

DROP TABLE Students

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

❖ Insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53675, 'Smith', 'smith@phys', 18, 3.5)
```

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

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

Powerful variants of these commands are available.
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.
Try it

Online SQL interpreter
Primary Key Constraints

❖ A set of fields is a key for a relation if:
   1. No two tuples can have same values in all key fields
   2. This is not true for any subset of the key
❖ If the key is overspecified (Rule 2 violated), it is called a superkey.
❖ If there’s more than one key for a relation, one is chosen (by DBA) as the primary key.
❖ E.g., sid is a key for Students. (What about name?) The set \{sid, gpa\} is a superkey.
Keys in SQL

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

For a given student and course, there is a single grade.

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

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

Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid),
UNIQUE (cid, grade) )
Foreign Keys

- **Foreign key**: Set of fields in one relation that is used to “reference” 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?
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>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</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>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@cs</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>
Enforcing Referential Integrity

❖ What should be done if an Enrolled tuple with a non-existent student id is inserted? *(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 *null*, denoting *unknown*.)

❖ 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
  - 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.
- We can check a database instance to see if an IC is violated, but we can NEVER infer an IC by looking at an instance.
  - An IC is a statement about all possible instances!
- Key and foreign key ICs are the most common; more general ICs supported too.
Logical DB Design: ER to Relational

- Entity sets to tables:

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
Review: ISA Hierarchies

It is often useful to subdivide entities into classes, like in an OOL. 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)*
Translating ISA 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.
Translating ISA to Relations

Alternative approach:
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.
Views

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

❖ Views can be dropped using the **DROP VIEW** command.
  ❖ How to handle **DROP TABLE** if there’s a view on the table?
  ❖ **DROP TABLE** command has options to let the user specify this.
CREATE VIEW YoungActiveStudents (name, grade) 
    AS SELECT S.name, E.grade 
    FROM Students S, Enrolled E 
    WHERE S.sid = E.sid and S.age<21
Views to support ISA relations

❖ The common elements of an ISA hierarchy can be supported using views.

```sql
CREATE VIEW Employee(ssn, name, lot)
AS SELECT H.ssn, H.name, H.lot
FROM Hourly_Emps H
UNION
SELECT C.ssn, C.name, C.lot
FROM Contract_Emps C
```
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 students who have are enrolled, but not their sid’s, cid’s, or even their ages.

<table>
<thead>
<tr>
<th>login</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith@cs</td>
<td>C</td>
</tr>
<tr>
<td>smith@cs</td>
<td>B</td>
</tr>
<tr>
<td>smith@math</td>
<td>A</td>
</tr>
<tr>
<td>jones@cs</td>
<td>B</td>
</tr>
</tbody>
</table>
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.