The Relational Model

Chapter 3

ECS 165A – Winter 2020

Mohammad Sadoghi
Exploratory Systems Lab
Department of Computer Science

UC DAVIS
UNIVERSITY OF CALIFORNIA
Why Study the Relational Model?

❖ Most widely used model.
  ▪ Vendors: IBM, Microsoft, Oracle, etc.
❖ “Legacy systems” in older models
  ▪ E.G., IBM’s IMS
❖ Old competitors:
  ▪ Hierarchical Model
  ▪ Network Model
❖ Competitors: object-oriented model
❖ Object-relational model
Relational Database: Definitions

❖ Relational database: a set of relations

❖ Relation: made up of 2 parts:
  ▪ Schema: specifies name of relation, plus name and type of each column.
    • E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).

  ▪ Instance: a table, with rows and columns.
    #Rows = cardinality, #fields = degree / arity.

❖ 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?
Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: 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)
Example: University Database

- **Schema:**
  - `Students(sid: string, name: string, login: string, age: integer, gpa: real)`
  - `Courses(cid: string, cname: string, credits: integer)`
  - `Enrolled(sid: string, cid: string, grade: string)`
The SQL Query Language

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

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

- To find just names and logins, replace the first line:

  ```sql
  SELECT  S.name, S.login
  ```
Creating Relations in SQL

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

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

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

CREATE TABLE Enrolled
(sid: CHAR(20),
cid: CHAR(20),
grade: CHAR(2))
Removing and Altering Relations

DROP TABLE Students

- Removes 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:

\[
\text{INSERT INTO Students (sid, name, login, age, gpa)} \\
\text{VALUES (53688, ‘Smith’, ‘smith@ee’, 18, 3.2)}
\]

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

\[
\text{DELETE} \\
\text{FROM Students S} \\
\text{WHERE S.name = ‘Smith’}
\]
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

- A set of fields is a *key* for a relation if no two distinct tuples (records) can have same values in all key fields, and
Primary and Candidate 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.”

❖ “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,cid) )

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

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

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

Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<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>

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@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>
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? *(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’ 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)

```
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)
```
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