The Relational Model
Ramakrishnan & Gehrke, Chap. 3

Review
• Why use a DBMS? OS provides RAM and disk
  – Concurrency
  – Recovery
  – Abstraction, Data Independence
  – Query Languages
  – Efficiency (for most tasks)
  – Security
  – Data Integrity

Glossary
• Byte
• Kilobyte
• Megabyte
• Gigabyte
• Terabyte
  – A handful of these for files in EECS
  – Biggest single online DB is Wal-Mart, >100TB
  – Internet Archive WayBack Machine is > 100 TB
• Petabyte
  – 11 of these in email in 1999
• Exabyte
  – 8 of these projected to be sold in new disks in 2003
• Zettabyte
• Yottabyte
Data Models

- DBMS models real world
- Data Model is link between user’s view of the world and bits stored in computer
- Many models exist
- We will concentrate on the Relational Model

Why Study the Relational Model?

- Most widely used model.
  - Vendors: IBM, Microsoft, Oracle, Sybase, etc.
- ”Legacy systems” in older models
  - e.g., IBM’s IMS
- Object-oriented concepts have recently merged in
  - object-relational model
  - IBM DB2, Oracle 9i, IBM Informix
  - Will touch on this toward the end of the semester
  - Based on POSTGRES research project at Berkeley
    - Postgres still represents the cutting edge on some of these features!

Relational Database: Definitions

- Relational database: a set of relations.
- Relation: made up of 2 parts:
  - Instance: a table, with rows and columns.
    - |rows| = cardinality
  - 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)
    - |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, arity = 5, all rows distinct
- Do all values in each column of a relation instance have to be distinct?
SQL - A language for Relational DBs

- **SQL:** standard language
- **Data Definition Language (DDL)**
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- **Data Manipulation Language (DML)**
  - Specify queries to find tuples that satisfy criteria
  - add, modify, remove tuples

---

Creating Relations in SQL

- Creates the Students relation.
- Note: the type (domain) of each field is specified, and enforced by the DBMS
  - whenever tuples are added or modified.
- Another example: the Enrolled table holds information about courses students take.

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

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

---

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!
Keys

- Keys are a way to associate tuples in different relations.
- Keys are one form of integrity constraint (IC).

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>
</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>

Primary Keys

- A set of fields is a superkey if:
  - No two distinct tuples can have same values in all key fields.
  - A set of fields is a key for a relation if:
    - It is a superkey.
    - No subset of the fields is a superkey.
- >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.

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

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

vs.

"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."

- Used carelessly, an IC can prevent the storage of database instances that should arise in practice!

Foreign Keys

- A Foreign Key is a field whose values are keys in another relation.

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, cid))
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.)

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 )
```

### 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!

Where do ICs Come From?

- ICs are based upon the semantics of the real-world 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.**
Enforcing Referential Integrity

- Remember 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.
  - 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.

Administrivia

- Homework 0 is posted!
  - Check the instructions again.
- Other textbooks
  - Korth/Silberschatz/Sudarshan
  - O'Neil and O'Neil
  - Garcia-Molina/Ullman/Widom

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

- The most widely used relational query language.
  - Current std is SQL99; SQL92 is a basic subset
- 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
  ```
Querying Multiple Relations

- What does the following query compute?

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

Given the following instance of Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</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>

Semantics of a Query

- A conceptual evaluation method for the previous query:
  1. do FROM clause: compute cross-product of Students and Enrolled
  2. do WHERE clause: Check conditions, discard tuples that fail
  3. do SELECT clause: Delete unwanted fields

- Remember, this is conceptual. Actual evaluation will be much more efficient, but must produce the same answers.

Cross-product of Students and Enrolled Instances

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@cs</td>
<td>18</td>
<td>3.2</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@math</td>
<td>18</td>
<td>3.8</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@math</td>
<td>18</td>
<td>3.8</td>
<td>53650</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@math</td>
<td>18</td>
<td>3.8</td>
<td>53666</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@math</td>
<td>18</td>
<td>3.8</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used
  - Object-relational variant gaining ground
  - XML support being added
- 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.