The Relational Model
Why Study the Relational Model?

- Most widely used model.
  IBM, Microsoft, Oracle, MySQL, PostgreSQL, ...
- "Legacy systems" in older models
  e.g., IBM’s IMS
- Competitor: Object-oriented model
  - a synthesis: object-relational model

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.
    E.g., Students\((sid: \text{string}, \text{name: string, login: string, age: integer, gpa: real})\)
- 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?

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.

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))
Adding and Deleting Tuples

❖ Can insert a single tuple using:

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

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

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

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

---

Integrity Constraints (ICs)

❖ IC: condition that must be true for any instance of the database; e.g., \textit{domain constraints}.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.

❖ A \textit{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:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? 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., 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.
- “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 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 without referential integrity?

Links in HTML!

---

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>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</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.

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.
Constraints Beyond the ER Model

- **Functional dependencies:**
  - *e.g.,* A dept can’t order two distinct parts from the same supplier. Can’t express this wrt ternary Contracts relationship.
  - Normalization refines ER design by considering FDs.

- **Inclusion dependencies:**
  - Special case: Foreign keys (ER model can express these).
  - *e.g.,* At least 1 person must report to each manager. (Set of ssn values in Manages must be subset of supervisor_ssn values in Reports_To.) Foreign key? Expressible in ER model?

- **General constraints:**
  
  *e.g.,* Manager’s discretionary budget less than 10% of the combined budget of all departments he or she manages.

Several kinds of integrity constraints can be expressed in the ER model: key constraints, participation constraints, and overlap/covering constraints for ISA hierarchies. Some foreign key constraints are also implicit in the definition of a relationship set.

- Some of these constraints can be expressed in SQL only if we use general CHECK constraints or assertions.
- Some constraints (notably, functional dependencies) cannot be expressed in the ER model.
- Constraints play an important role in determining the best database design for an enterprise.
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.
- 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?

\[
\text{SELECT S.name, E.cid}
\text{FROM Students S, Enrolled E}
\text{WHERE S.sid=E.sid AND E.grade="A"}
\]

Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

<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>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>E.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>53832</td>
<td>Reggae203</td>
<td>B</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>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

ER to Relational

Entity sets get mapped to tables.

CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))
**ER to Relational**

Relationship sets without constraints also get mapped to a relation.

In translating a relationship set to a relation, attributes of the relation must include:

- Keys for each participating entity set (as foreign keys).
  This set of attributes forms **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
)
```

**Translating ER Diagrams with Key Constraints**

- Map relationship to a table:
  - Note that **did** is the key now!
  - Separate tables for Employees and Departments.
- 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
)

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees
)
```
**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).

```sql
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)
```

**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),
  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.

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.