The Relational Model

Problem Set #1:
Will be out later tonight
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)

❖ Recent competitor: object-oriented model
  ▪ ObjectStore, Versant, Ontos
  ▪ A synthesis emerging: object-relational model
    • Informix Universal Server, UniSQL, O2, Oracle, DB2
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 a name and type for each column.
- Can think of a relation as a *set* of rows or *tuples*. 
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@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 *have to be distinct*—each relation is defined to be a *set* of unique tuples
Relational Query Languages

❖ A major strength of the relational model is that it 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
❖ A portable and long-lasting standard
❖ Standards:
  ▪ SQL-86
  ▪ SQL-89 (minor revision)
  ▪ SQL-92 (major revision)
  ▪ SQL-1999 (major extensions, Current baseline)
  ▪ SQL-2003, SQL-2006 (added XML support)
  ▪ SQL-2008, (minor additions)
  ▪ SQL-2011, (temporal support)
Creating Relations in SQL

- SQL for creating the Students relation.
- Observe that 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 that students take.

CREATE TABLE Students (sid INTEGER, name TEXT, login TEXT, age INTEGER, gpa REAL)

CREATE TABLE Enrolled (sid INTEGER, cid TEXT, grade TEXT)
Destroying and Altering Relations

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

```
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.

```
ALTER TABLE Students
ADD COLUMN admitYear: integer
```
SQL Queries

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

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

Situation:

```
<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>
</tbody>
</table>
```

• To find just names and logins, replace the first line

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

• When a relation is referenced only once and attributes are unique, the use of variables is optional
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"
```

Effectively “Joins” or connects two tables

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>
Adding and Deleting Tuples

❖ Can 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; more later!
**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:
   1. No two tuples can have same values for all their corresponding 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.
Primary and Candidate Keys in SQL

- Possibly many *candidate keys*, one of which is chosen as the *primary key*. Alternative, non primary keys can be specified using `UNIQUE`.

- “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!

```sql
CREATE TABLE Enrolled (
    sid INTEGER,
    cid  TEXT,
    grade TEXT,
    PRIMARY KEY (sid),
    UNIQUE (cid, grade) )
```

```sql
CREATE TABLE Enrolled (
    sid INTEGER,
    cid  TEXT,
    grade TEXT,
    PRIMARY KEY (sid,cid) )
```
Foreign Keys, Referential Integrity

- **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?

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 INTEGER,  
cid TEXT,  
grade TEXT,  
PRIMARY KEY (sid,cid),  
FOREIGN KEY (sid) REFERENCES Students )
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Carnatic101</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>53688</td>
<td>Reggae203</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
<td></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

- 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?
  1. Also delete all Enrolled tuples that refer to it.
  2. Disallow deletion of a Students tuple that is referred to.
  3. Set \( sid \) in Enrolled tuples that refer to it to a default \( sid \).
  4. (In SQL, also: Set \( sid \) in Enrolled tuples that refer to it to a special value \texttt{null}, denoting \textit{unknown} or does not apply.)
- 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** (no consequences on delete or update)
- **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 INTEGER,  
cid TEXT,  
grade TEXT,  
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.
Views

❖ A **view** is just a relation, but it is derived from other relations. Thus, we store a **definition**, rather than a set of tuples.

```
CREATE VIEW YoungActiveStudents(name, grade)
AS SELECT S.login, 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.
  - How to handle **DROP TABLE** if there’s a view on the table?
  - **DROP TABLE** command has options to let the user specify this.
Views to support ISA relations

❖ The common elements of an ISA hierarchy can be supported using views.
❖ For example, consider this implementation of Alternate 2 from slide 29

```
CREATE VIEW Employee(ssn, name, jobtitle)
AS SELECT H.ssn, H.name, H.jobtitle
    FROM Hourly_Emps H
UNION
    SELECT C.ssn, C.name, C.jobtitle
    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).

❖ Ex. A list of grades made by YoungStudents, sorted by email addresses.

<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>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.