CAS CS 460/660
Relational Model
E/R Model:

- Entities, relationships, attributes
- Cardinalities: 1:1, 1:n, m:1, m:n
- Keys: superkeys, candidate keys, primary keys
■ Weak Entity sets, identifying relationship
■ Discriminator, total participation, one-to-many
Review

- **Generalization-specialization**

  ![Diagram of generalization-specialization]

- **Aggregation**

  ![Diagram of aggregation]
Review

- Data models: framework for organizing and interpreting data
- E/R Model
- OO, Object relational, XML
- Relational Model
  - Intro
  - E/R to relational
  - SQL preview
Relational Data Model

- Introduced by Ted Codd (early 70’s) (Turing Award, ‘81)

- Relational data model contributes:
  1. Separation of logical and physical data models (data independence)
  2. Declarative query languages
  3. Formal semantics
  4. Query optimization (key to commercial success)

- First prototypes:
  - Ingres -> postgres, informix (Stonebraker, UC Berkeley) 
    Turing Award 2015!!!!!!
  - System R -> Oracle, DB2 (IBM) (Jim Gray, Turing Award 1998)
Relations

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-202</td>
<td>450</td>
</tr>
<tr>
<td>Brookline</td>
<td>A312</td>
<td>600</td>
</tr>
</tbody>
</table>

- Rows (tuples, records)
- Columns (attributes)
- Tables (relations)

- Why relations?
Relations

■ Mathematical relations (from set theory):

Given 2 sets \( R = \{1, 2, 3, 5\}, S = \{3, 4\} \)

\[ R \times S = \{(1,3), (1, 4), (2, 3), (2,4), (3,3), (3,4), (5,3), (5,4)\} \]

\[ \text{A relation between } R \text{ and } S \text{ is any subset of } R \times S \]

\[ \text{e.g., } \{(1,3), (2,4), 5,3)\} \]

■ Database relations:

Given attribute domains:

\[ \text{bname} = \{\text{Downtown, Brighton, ….}\} \]
\[ \text{acct\_no} = \{\text{A-101, A-102, A-203, ….}\} \]
\[ \text{balance} = \{\text{…. 400, 500, ….}\} \]

\[ \text{account} \ subset of \ bname \times acct\_no \times balance \]

\[ \{\text{(Downtown, A-101, 500),} \]
\[ \text{(Brighton, A-202, 450),} \]
\[ \text{(Brookline, A-312, 600)}\} \]
### Storing Data in a 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@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>

- Data about individual students
- One row per student
- How to represent course enrollment?
Students may enroll in more than one course. The most efficient way to keep track of enrollment is to keep the enrollment data in a separate table.

<table>
<thead>
<tr>
<th>cid</th>
<th>grade</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carnatic101</td>
<td>C</td>
<td>53666</td>
</tr>
<tr>
<td>Reggae203</td>
<td>B</td>
<td>53666</td>
</tr>
<tr>
<td>Topology112</td>
<td>A</td>
<td>53650</td>
</tr>
<tr>
<td>History105</td>
<td>B</td>
<td>53666</td>
</tr>
</tbody>
</table>
How to connect student data to enrollment?
Need a Key

<table>
<thead>
<tr>
<th>cid</th>
<th>grade</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carnatic101</td>
<td>C</td>
<td>53666</td>
</tr>
<tr>
<td>Reggae203</td>
<td>B</td>
<td>53666</td>
</tr>
<tr>
<td>Topology112</td>
<td>A</td>
<td>53650</td>
</tr>
<tr>
<td>History105</td>
<td>B</td>
<td>53666</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@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>
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
In other words...

- **Data Model** – a way to organize information
- **Schema** – one particular organization,
  - i.e., a set of fields/columns, each of a given type
- **Relation**
  - a name
  - a schema
  - a set of tuples/rows, each following organization specified in schema
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 (degree) = 5, all rows distinct
SQL - A language for Relational DBs

- SQL: standard language (based on SEQUEL in System R (IBM now DB2))

- 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
SQL Overview

- CREATE TABLE <name> ( <field> <domain>, ... )

- INSERT INTO <name> (<field names>)
  VALUES (<field values>)

- DELETE FROM <name>
  WHERE <condition>

- UPDATE <name>
  SET <field name> = <value>
  WHERE <condition>

- SELECT <fields>
  FROM <name>
  WHERE <condition>
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.

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))
Adding and Deleting Tuples

■ Can insert a single tuple using:

```
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):

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

✧ Powerful variants of these commands are available; more later!
Integrity Constraints (IC): conditions that restrict the data that can be stored in the database

Keys are a way to associate tuples in different relations

Keys are one form of integrity constraint (IC)

### 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>

### Enrolled

<table>
<thead>
<tr>
<th>cid</th>
<th>grade</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carnatic101</td>
<td>C</td>
<td>53666</td>
</tr>
<tr>
<td>Reggae203</td>
<td>B</td>
<td>53666</td>
</tr>
<tr>
<td>Topology112</td>
<td>A</td>
<td>53650</td>
</tr>
<tr>
<td>History105</td>
<td>B</td>
<td>53666</td>
</tr>
</tbody>
</table>
Primary Keys - Definitions

- **Key**: A minimal set of attributes that uniquely identify a tuple

- 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 **candidate key** for a relation if:
  - It is a superkey
  - No subset of the fields is a superkey

- >1 **candidate keys** 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 storage of database instances that should be permitted!

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

```sql
CREATE TABLE Enrolled
    (sid CHAR(20)
     cid  CHAR(20),
     grade CHAR(2),
    PRIMARY KEY (sid,cid))
```
A Foreign Key is a field whose values are keys in another relation.

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>sid</td>
</tr>
<tr>
<td>Carnatic101</td>
<td>53666</td>
</tr>
<tr>
<td>Reggae203</td>
<td>53666</td>
</tr>
<tr>
<td>Topology112</td>
<td>53650</td>
</tr>
<tr>
<td>History105</td>
<td>53666</td>
</tr>
</tbody>
</table>
**Foreign Keys, Referential Integrity**

- **Foreign key**: Set of fields in one relation used to `refer’ to tuples in another relation.
  - Must correspond to primary key of the second relation.
  - Like a `logical pointer’.

- **E.g. sid in Enrolled is a foreign key referring to Students:**
  - Enrolled(\textit{sid}: string, \textit{cid}: string, \textit{grade}: string)
  - If all foreign key constraints are enforced, referential integrity is achieved (i.e., no dangling references.)
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>
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!
E/R to Relations

E/R diagram

Relational schema, e.g.
account=(bname, acct_no, bal)

E = (a₁, ..., aₙ)

R₁ = (a₁, b₁, c₁, ..., cᵦ)
More on relationships

■ What about:

■ Could have:

\[ R_1 = (a_1, b_1, c_1, \ldots, c_k) \]

since \( a_1 \) is the key for \( R_1 \) (also for \( E_1 = (a_1, \ldots, a_n) \))

■ Another option is to merge \( E_1 \) and \( R_1 \)

\[ \text{ignore } R_1 \]

\[ \text{Add } b_1, c_1, \ldots, c_k \text{ to } E_1 \text{ instead, i.e.} \]

\[ E_1 = (a_1, \ldots, a_n, b_1, c_1, \ldots, c_k) \]
E1 = (a₁, ..., aₙ) E2 = (b₁, ..., bₘ)

R1 = (a₁, b₁, c₁, ..., cₖ)

E1 = (a₁, ..., aₙ, b₁, c₁, ..., cₖ)
E2 = (b₁, ..., bₘ)

E1 = (a₁, ..., aₙ)
E2 = (b₁, ..., bₘ, a₁, c₁, ..., cₖ)

Treat as n:1 or 1:m
### Weak entity sets

- E1 = \( (a_1, \ldots, a_n) \)
- E2 = \( (a_1, b_1, \ldots, b_m) \)

### Multivalued Attributes

- Emp = \( (ssn, name) \)
- Emp-Dept = \( (ssn, dept) \)
E/R to Relational

Method 1:
\[ E = (a_1, \ldots, a_n) \]
\[ S_1 = (a_1, b_1, \ldots, b_m) \]
\[ S_2 = (a_1, c_1, \ldots, c_k) \]

Method 2:
\[ S_1 = (a_1, \ldots, a_n, b_1, \ldots, b_m) \]
\[ S_2 = (a_1, \ldots, a_n, c_1, \ldots, c_k) \]

Q: When is method 2 not possible?
E/R to Relational

- Aggregation

E1, R1, E2, E3 as before

\[ R2 = (c_1, a_1, b_1, d_1, \ldots, d_j) \]