

The Relational Model

Module 1, Lecture 2



Why Study the Relational Model?

- ❖ Most widely used model.
 - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- ❖ “Legacy systems” in older models
 - e.g., IBM’s IMS
- ❖ Recent competitor: Object-Oriented model
 - ObjectStore, Versant, Ontos
 - a synthesis emerging: *object-relational model*
 - ◆ Informix Universal Server, UniSQL, O2



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*: string, *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

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- ❖ 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.
- ❖ 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))
```



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 (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 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!
- ```
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.

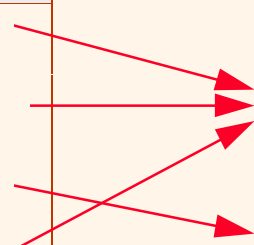
```
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students)
```

Enrolled

| sid   | cid         | grade |
|-------|-------------|-------|
| 53666 | Carnatic101 | C     |
| 53666 | Reggae203   | B     |
| 53650 | Topology112 | A     |
| 53666 | History105  | B     |

Students

| sid   | name  | login      | age | gpa |
|-------|-------|------------|-----|-----|
| 53666 | Jones | jones@cs   | 18  | 3.4 |
| 53688 | Smith | smith@eecs | 18  | 3.2 |
| 53650 | Smith | smith@math | 19  | 3.8 |





# *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.



# *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 standard is SQL-92.
- ❖ To find all 18 year old students, we can write:

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

| sid   | name  | login    | age | gpa |
|-------|-------|----------|-----|-----|
| 53666 | Jones | jones@cs | 18  | 3.4 |
| 53688 | Smith | smith@ee | 18  | 3.2 |

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

```
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"
```

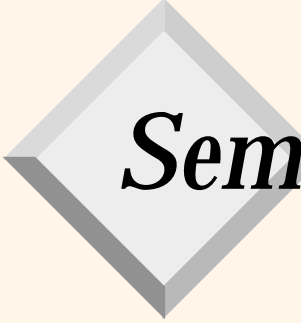
Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

| sid   | cid         | grade |
|-------|-------------|-------|
| 53831 | Carnatic101 | C     |
| 53831 | Reggae203   | B     |
| 53650 | Topology112 | A     |
| 53666 | History105  | B     |

we get:

| S.name | E.cid       |
|--------|-------------|
| Smith  | Topology112 |





# 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*

| S.sid        | S.name       | S.login           | S.age     | S.gpa      | E.sid        | E.cid              | E.grade  |
|--------------|--------------|-------------------|-----------|------------|--------------|--------------------|----------|
| 53666        | Jones        | jones@cs          | 18        | 3.4        | 53831        | Carnatic101        | C        |
| 53666        | Jones        | jones@cs          | 18        | 3.4        | 53832        | Reggae203          | B        |
| 53666        | Jones        | jones@cs          | 18        | 3.4        | 53650        | Topology112        | A        |
| 53666        | Jones        | jones@cs          | 18        | 3.4        | 53666        | History105         | B        |
| 53688        | Smith        | smith@ee          | 18        | 3.2        | 53831        | Carnatic101        | C        |
| 53688        | Smith        | smith@ee          | 18        | 3.2        | 53831        | Reggae203          | B        |
| 53688        | Smith        | smith@ee          | 18        | 3.2        | 53650        | Topology112        | A        |
| 53688        | Smith        | smith@ee          | 18        | 3.2        | 53666        | History105         | B        |
| 53650        | Smith        | smith@math        | 19        | 3.8        | 53831        | Carnatic101        | C        |
| 53650        | Smith        | smith@math        | 19        | 3.8        | 53831        | Reggae203          | B        |
| <b>53650</b> | <b>Smith</b> | <b>smith@math</b> | <b>19</b> | <b>3.8</b> | <b>53650</b> | <b>Topology112</b> | <b>A</b> |
| 53650        | Smith        | smith@math        | 19        | 3.8        | 53666        | History105         | B        |



# *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.