

ICS 321 Spring 2011

The Relational Model of Data (ii)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Defining Relational Schema in SQL

- Two aspects:
 - **Data definition language** – declaring database schemas
 - **Data manipulation language** – querying & modifying the database
- Three kinds of relations
 - Stored relations
 - Views
 - Temporary tables
- **CREATE TABLE** statement

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))
```

SQL Data Types

- Character Strings
 - CHAR(n), VARCHAR(n)
- Bit Strings
 - BIT(n), BIT VARYING(n)
- Boolean - BOOLEAN
- Integer
 - INT, INTEGER, SHORTINT, BIGINT
- Floating point numbers
 - FLOAT, REAL, DOUBLE PRECISION, DECIMAL(n,d)
- Dates and Times
 - DATE (eg. '1948-05-14'), TIME (eg. '15:00:02.5')

Destroying and Altering Relations

DROP TABLE Students

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

ALTER TABLE Students **ADD** firstYear

- 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 **DROP** age

- Deletes the age column

Default Values

- Specify default values for fields in table declaration

```
CREATE TABLE MovieStar (...  
    gender CHAR(1) DEFAULT '?',  
    birthdate DATE DEFAULT DATE '0000-00-00')
```

- Or in an alter table statement

```
ALTER TABLE MovieStar ADD phone CHAR(16)  
DEFAULT 'unlisted';
```

Adding and Deleting Tuples

- Insert a single tuple:

```
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?

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?
- 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.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
 - Default is **NO ACTION** (*delete/update is rejected*)
 - **CASCADE** (also delete all tuples that refer to deleted tuple)
 - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

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