Relational Model

Chapter 3

Relational Database: Definitions

- Relational database: a set of *relations*

- Relation: made up of 2 parts:
  - **Schema** : specifies name of relation, plus name and type of each column.
    - e.g. **Student**(**sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**: real).
  - **Instance** : a *table*, with rows and columns. #Rows = *cardinality*, #fields = *degree (arity)*.

Relational Database: Definitions

- We can think of a relation as a **set** of *rows* or *tuples* (i.e., all rows are distinct).

- In relational model, a **field** can only contain atomic values, i.e., lists or sets are **disallowed**!

Example Instance of Student 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?
- While depicting a relation instance, order of rows doesn’t matter!

Alternative Terminology for Relational Model

Formal Terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

A Relational Database Schema for a University

- **Student**(**sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**:real)
- **Course**(**cid**: string, **cname**:string, **credits**:integer)
- **Enrolled**(**sid**:string, **cid**:string, grade:string)

- This is a conceptual schema.

Query Languages

- A major strength of DBMSs:
  - **Data Definition Language** (DDL) is used to set up schema
  - **Data Manipulation Language** (DML) is used to modify and query the data
- SQL:
  - Standard language for creating, manipulating, and querying data in a **relational DBMS**

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.
  - 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
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)
  - SQL-2003, SQL-2006 (extensions on XML)
  - SQL- 2008
  - SQL-2011

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Elements of Table Declarations

- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.
  - DATE (e.g.:`'2007-09-30'`)
  - TIME (e.g.:`'15:30:02.5'`)
  - Any value can be NULL

Creating Relations in SQL

- Creates the Student relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Student
    (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:

INSERT INTO *Student* (*sid*, *name*, *login*, *age*, *gpa*)
VALUES (300, 'Smith', 'smith@ee', 18, 3.4)

sid	name	login	age	gpa
300	Smith	smith@ee	18	3.4

* *Powerful variants of these commands are available.*

Adding and Deleting Tuples

- Can **delete** all tuples

```
DELETE
FROM Student S
```

sid	name	login	age	gpa
300	Smith	smith@ee	18	3.4
100	John	john@ceng	19	1.7
200	Fatma	fatma@ee	18	3.6

* *Powerful variants of these commands are available.*

Adding and Deleting Tuples

- Can **delete** all tuples satisfying some condition (e.g., name = John):

```
DELETE
FROM Student S
WHERE S.name = 'John'
```

sid	name	login	age	gpa
300	Smith	smith@ee	18	3.4
200 100	Fatma John	fatma@eng john@ee	18 19	3.6 3.7
200	Fatma	fatma@ee	18	3.6

* *Powerful variants of these commands are available.*

Modifying Tuples

- Can **modify** the column values in an existing row using:

```
UPDATE  Student S
SET     S.age = S.age + 1
```

sid	name	login	age	gpa
300	Smith	smith@ee	19	3.2
100	John	john@ceng	20	1.7
200	Fatma	fatma@ee	19	3.6

Modifying Tuples

- Can **modify** the column values in an existing row using:

```
UPDATE  Student S
SET     S.age = S.age + 1, S.gpa = S.gpa –1
WHERE   S.sid = 200
```

sid	name	login	age	gpa
300	Smith	smith@ee	18	3.4
100	John	john@ceng	19	1.7
200	Fatma	fatma@ee	19	2.6

Modifying Tuples

- Can **modify** the column values in an existing row using:

```
UPDATE Student S
SET    S.gpa = S.gpa – 0.1
WHERE  S.gpa >= 3.3
```

sid	name	login	age	gpa
300	Smith	smith@ee	18	3.3
100	John	john@ceng	19	1.7
200	Fatma	fatma@ee	18	3.5

Remark: In the examples of DELETE and UPDATE, the WHERE clause is applied *first* to determine the rows to be deleted or updated.

Integrity Constraints (ICs)

- IC: a condition specified on a DB schema & restricts the data that can be stored in an instance
- 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!

Key Constraints

- A set of fields is a (**candidate**) **key** for a relation if :
  1. **uniqueness:** Two distinct tuples cannot have identical values in all the fields of a key, and
  2. **minimality:** No subset of the set of fields in a key is a unique identifier for a tuple
  - If part 2 false, then it is a ***superkey***.
  - If there is more than one (candidate) key for a relation, one of the keys is chosen (by DBA) to be the ***primary key***.

The DBMS optimizes several tasks for the use of PK (eg., while referring to a tuple from another relation or while accessing the table via an index, etc.)

Key Constraints: Example

- **Student**(**sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**:real)
- Based on the application requriements of a university DB:
  - What can be the candidate keys for Student? **sid** **or:** **login**
  - What can be the superkeys?
  - Which one would you choose as PK?

sid,name
sid,gpa
sid,name,gpa
sid,name,gpa,login,age
login,name
login,name,gpa
sid,login
…

sid

Remark: ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.

Primary and Candidate Keys in SQL

- Possibly many *candidate keys*, **one** of which is chosen as the *primary key*.

```
CREATE TABLE Student
        (sid      CHAR(20),
         name     CHAR(20),
         login    CHAR(10),
         age      INTEGER,
         gpa      REAL,
        PRIMARY KEY (sid),
        UNIQUE  (login))
```

- For **domain, PK and unique constraints**: If a DB modification operation (add/del/update) violates them , operation is **rejected**

Primary and Candidate Keys in SQL

- "For a given student and course, there is a single grade."

sid	cid	grade
53666	Ceng351	A
53666	Ceng351	B-
53666	Ceng331	A

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

✓ **Good design if Enrolled is intended to keep the current semester's data!**

An IC can prevent storage of instances that arise in practice! So, DBA must set ICs carefully depending on the nature of the application domain!

Primary and Candidate Keys in SQL

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

- Implication: "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."

"sid	cid	grade
53666	Ceng351	A
53666	Ceng331	B-
53444	Ceng351	A

- An IC can prevent storage of instances that arise in practice!

X **Bad design (or your university is really weird!!!)**

PRIMARY KEY vs. UNIQUE

1. There can be <u>only one</u> PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

For unknown/missing/inapplicable values

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 *Student*:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign keys…

Enrolled(*sid*: string, *cid*: string, ...)

Student(**sid**:string, …)

Primary key

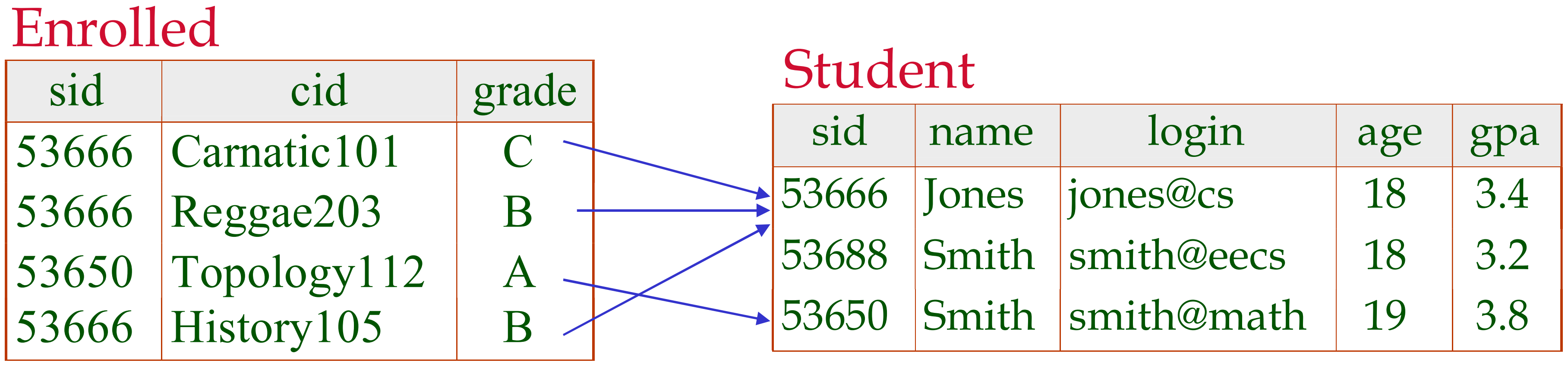
Course(**cid**:string, ..

Primary ke

Foreign Keys in SQL

- Only students listed in the Student 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 Student )
```



Enrolled

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

Student

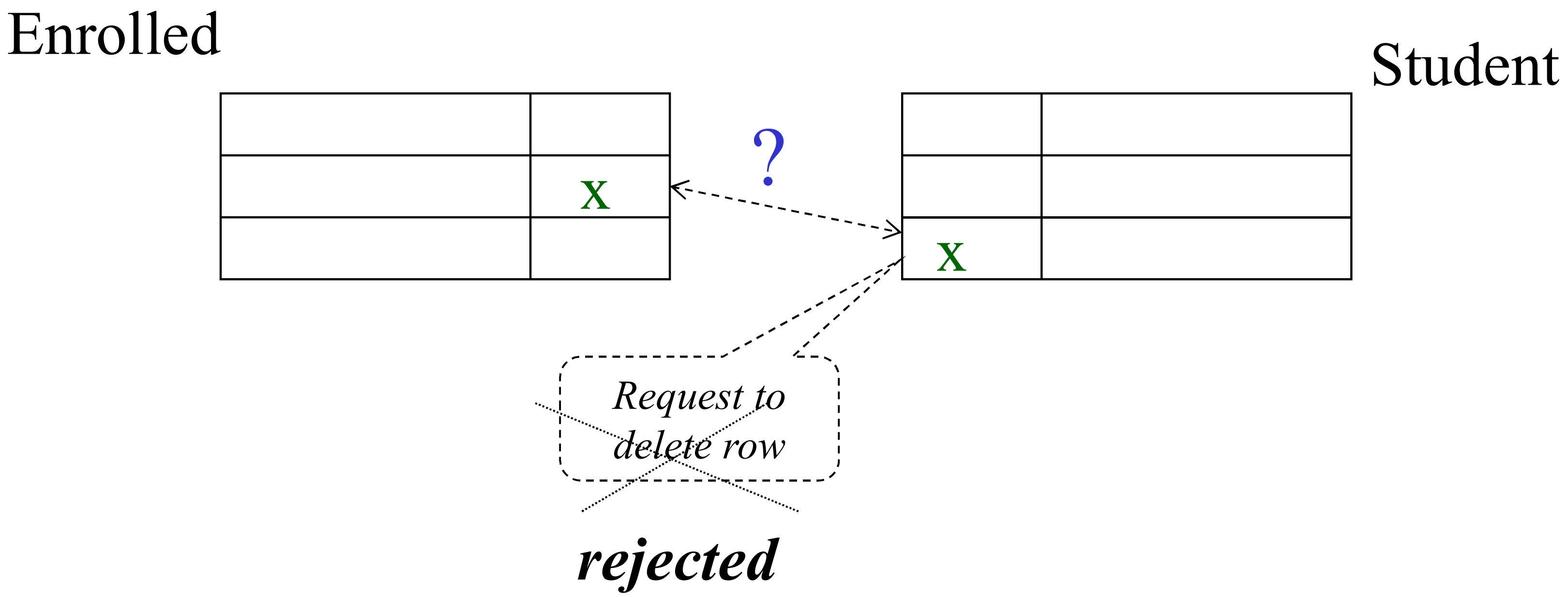
sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider *Student* and *Enrolled*; *sid* in *Enrolled* is a foreign key that references *Student*.
- What should be done if an *Enrolled* tuple with a non-existent *student id* is **inserted**? (*Reject it!*)
- What should be done if a *Student* tuple is **deleted**?
  - Disallow deletion of a *Student* 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*'.)
  - Also delete all *Enrolled* tuples that refer to it.
- Similar if primary key of *Student* tuple is **updated**.

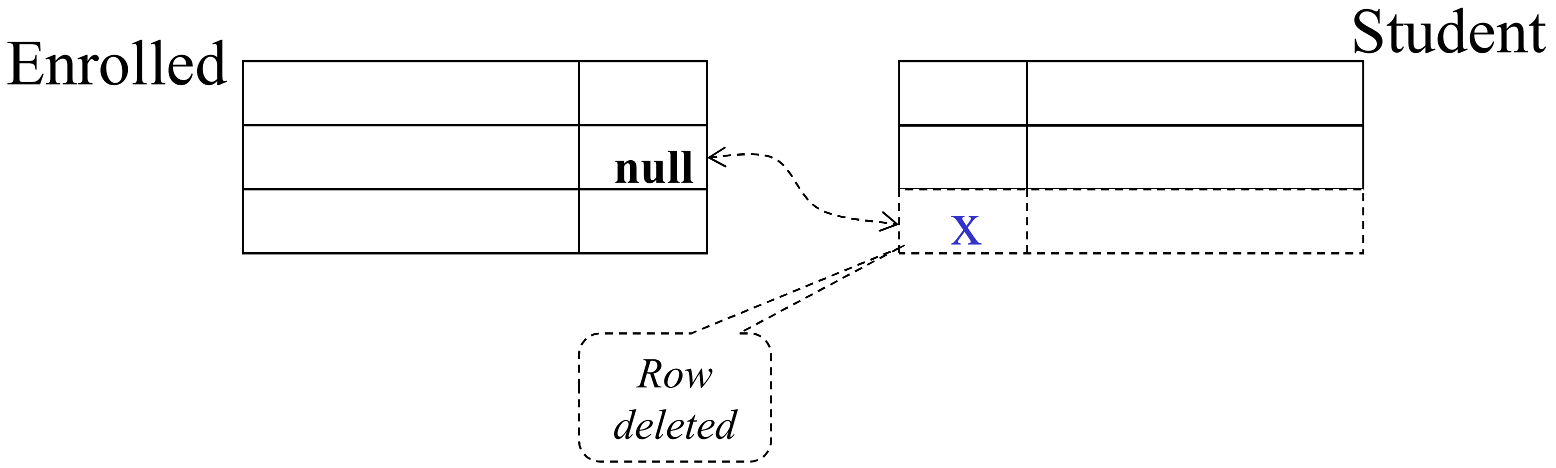
Handling Foreign Key Violations

- Deletion from *Student*:
  - NO ACTION: Reject if row(s) in *Enrolled* references row to be deleted (default response)



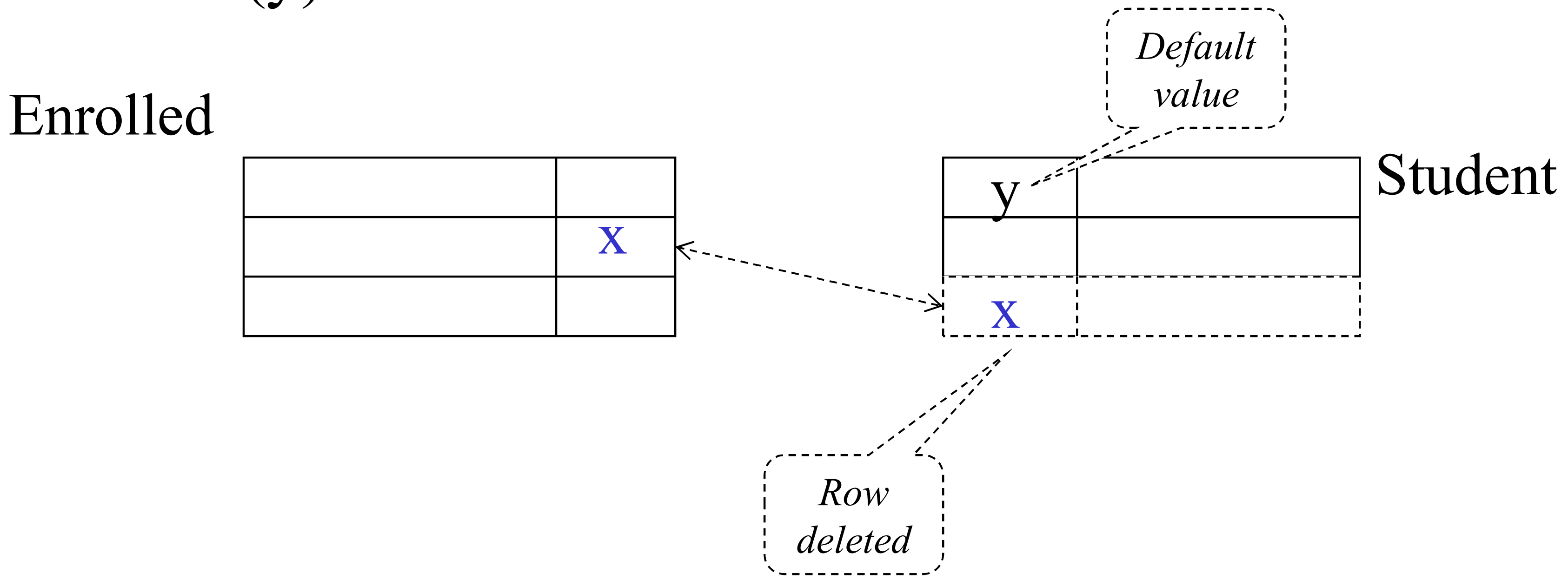
Handling Foreign Key Violations (cont'd)

- Deletion from *Student* (cont'd):
  - SET NULL: Set value of foreign key in referencing row(s) in A to **null**



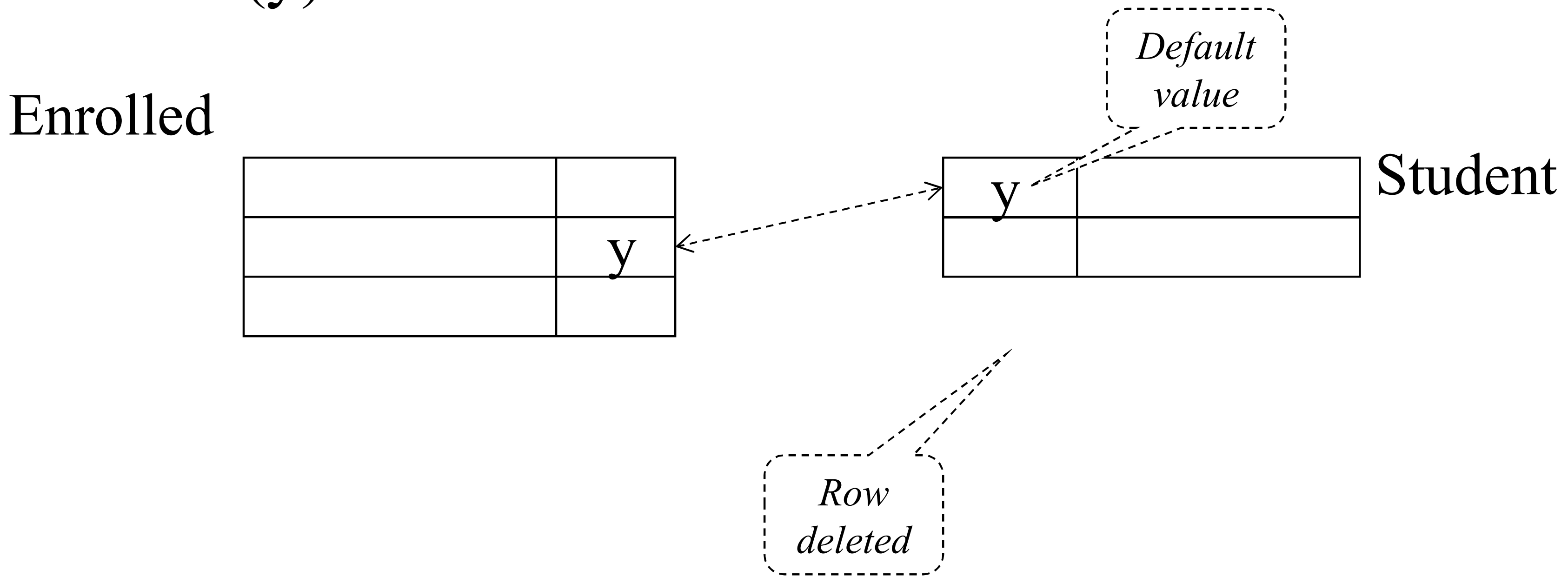
Handling Foreign Key Violations (cont'd)

- Deletion from *Student* (cont'd):
  - SET DEFAULT: Set value of foreign key in referencing row(s) in *Enrolled* to default value (y) which must exist in *Student*



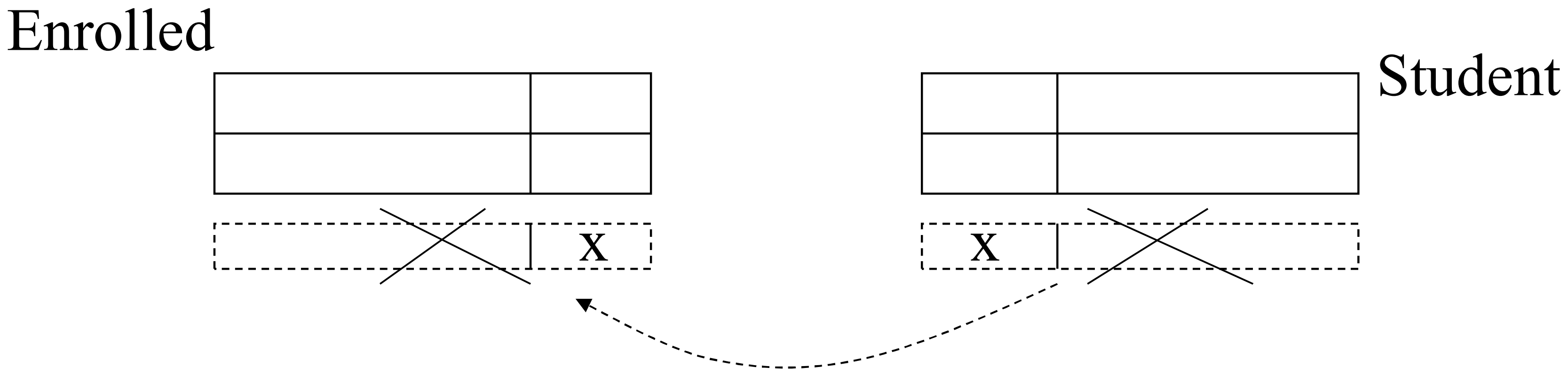
Handling Foreign Key Violations (cont'd)

- Deletion from *Student* (cont'd):
  - SET DEFAULT: Set value of foreign key in referencing row(s) in *Enrolled* to default value (y) which must exist in *Student*



Handling Foreign Key Violations (cont'd)

- Deletion from *Student* (cont'd):
  - CASCADE: Delete referencing row(s) in *Enrolled* as well



Referential Integrity in SQL/92

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

Which of these make sense for Enrolled?

- SQL/92 supports all 4 options on deletes and updates:

  NO ACTION / SET NULL / SET DEFAULT / CASCADE

```
CREATE TABLE Enrolled
   (sid CHAR(20) DEFAULT '9999',
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid) REFERENCES Student ON DELETE [redacted]N
                                         ON UPDATE ([redacted]E
   FOREIGN KEY (cid) REFERENCES Course   ON DELETE NO ACTION
                                         ON UPDATE CASCADE )
```

Foreign Keys: Remarks

- NULL can appear in a FK (unless FK is also the part of the PK in the child table, of course)
- Assume in Student relation, we also keep student's advisor id, which refers to Instructor relation (with PK field **iid**):

Student

sid	name	login	age	gpa	iid
53666	Jones	jones@cs	18	3.4	1
53688	Smith	Smith@eecs	18	3.2	Null
53650	Smith	Smith@math	19	3.8	1

Instructor

iid	name	salary
1	Altingovde	
2	Karagoz	
3	Chair	

```
CREATE TABLE Student
   (sid     CHAR(20),
    name  CHAR(20),
    login   CHAR(10),
    age      INTEGER,
    gpa      REAL,
    iid       CHAR(20) DEFAULT '3',
   PRIMARY KEY (sid),
   FOREIGN KEY (iid) REFERENCES Instructor ON DELETE [redacted]
                                                     ON UPDATE [redacted] )
```

Student

sid	name	login	age	gpa	iid
53666	Jones	jones@cs	18	3.4	1
53688	Smith	Smith@eecs	18	3.2	Null
53650	Smith	Smith@math	19	3.8	1

iid	name	salary
1	Altingovde	
2	Karagoz	
3	Chair	

Foreign Keys: Remarks

- NULL can appear in a FK (unless FK is also the part of the PK in the child table, of course)
- The field names in the FK may be different than that of the referred PK, but the number of fields should be same and field types should be compatible

```
CREATE TABLE Enrolled
  (student-id CHAR(20) DEFAULT '9999',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (student-id,cid),
   FOREIGN KEY (student-id) REFERENCES Student(sid)
                                   ON DELETE NO ACTION
                                   ON UPDATE CASCADE
```

Foreign Keys: Remarks

- A FK can refer to the same relation
  - In Student relation, there could be the partner field that is intended to store the sid of another student…
  - More examples will be discussed in the ER week…

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 not 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.