



The Relational Model

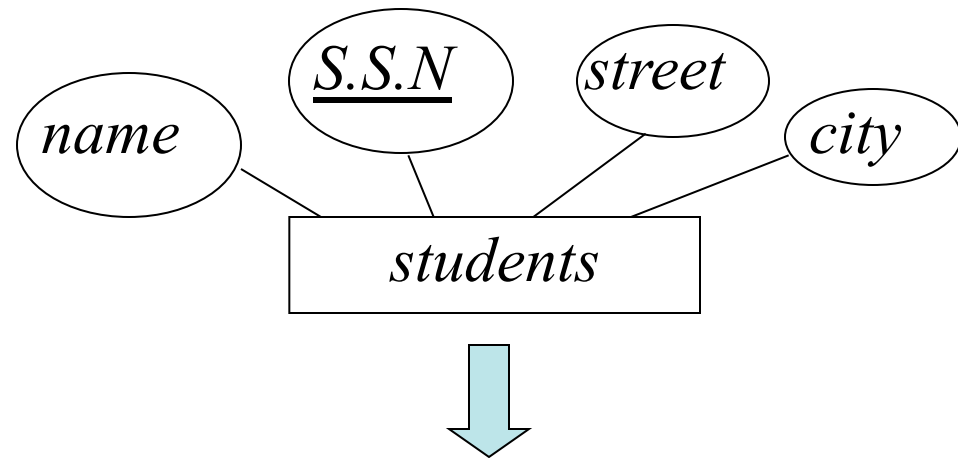
Week 2

Relations

A **relation** is a more concrete construction, of something we have seen before, the ER diagram.

A relation is (just!) a table!

We will use **table** and **relation** interchangeably, except where there is a possibility of confusion.



<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax
Sue	5555	Lee	Vienna

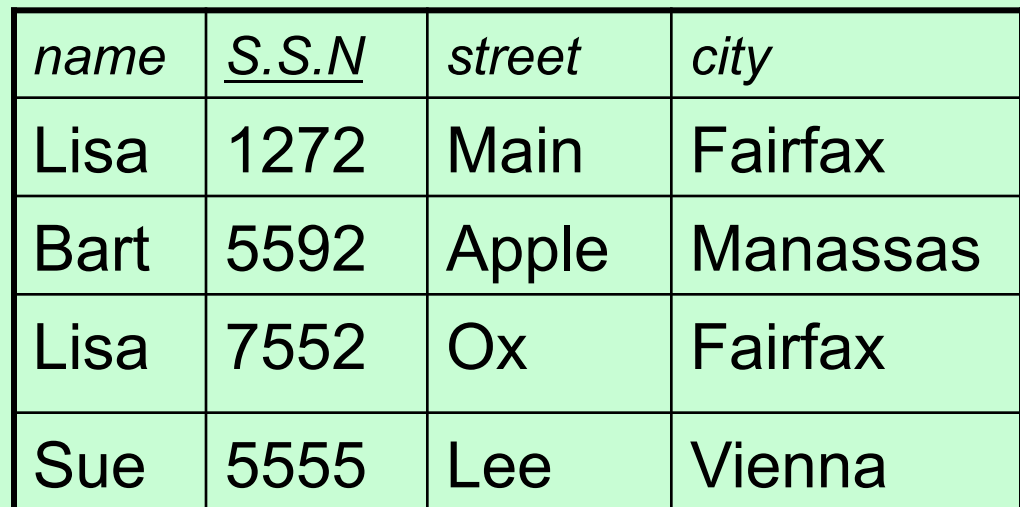
The students relation

A **relation** consists of a **relational schema** and a **relational instance**.

A **relation schema** is essentially a list of column names with their data types. In this case...

`students(name : string, S.S.N : string, street : string, city : string)`

- A **relation instance** is made up of zero or more **tuples** (rows, records)



<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax
Sue	5555	Lee	Vienna

A schema specifies a relation's name.

students(*name* : string, *S.S.N* : string, *street* : string, *city* : string)

A schema also specifies the name of each **field**, and its domain.

Fields are often referred to as columns, attributes, dimensions

A minor, but important point about relations, they are unordered.

<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax
Sue	5555	Lee	Vienna

<i>name</i>	<u><i>S.S.N</i></u>	<i>city</i>	<i>street</i>
Lisa	1272	Fairfax	Main
Bart	5592	Manassas	Apple
Lisa	7552	Fairfax	Ox
Sue	5555	Vienna	Lee

This is not a problem, since we refer to fields by name.

However sometimes, we refer to the fields by their column number, in which case the ordering becomes important. I will point this out when we get there.

Also, the tuples are unordered too!

Note that every tuple in our instance is unique. This is not a coincidence. The definition of relation demands it.

Later we will see how we can represent weak entities in relations.

<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax
Sue	5592	Lee	Vienna

The number of fields is called the **degree** (or **arity**, or dimensionality of the relation).

Below we have a table of degree 4.

The number of tuples = **cardinality** of the relation

Of course, we don't count the row that has the labels!

To the right we have a table of cardinality 3.

<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax

students(*name* : string, *S.S.N* : string, *street* : string, *city* : string)

Note that relations have primary keys, just like ER diagrams. Remember that the primary key might not be one field, it may be a combination of two or more fields.

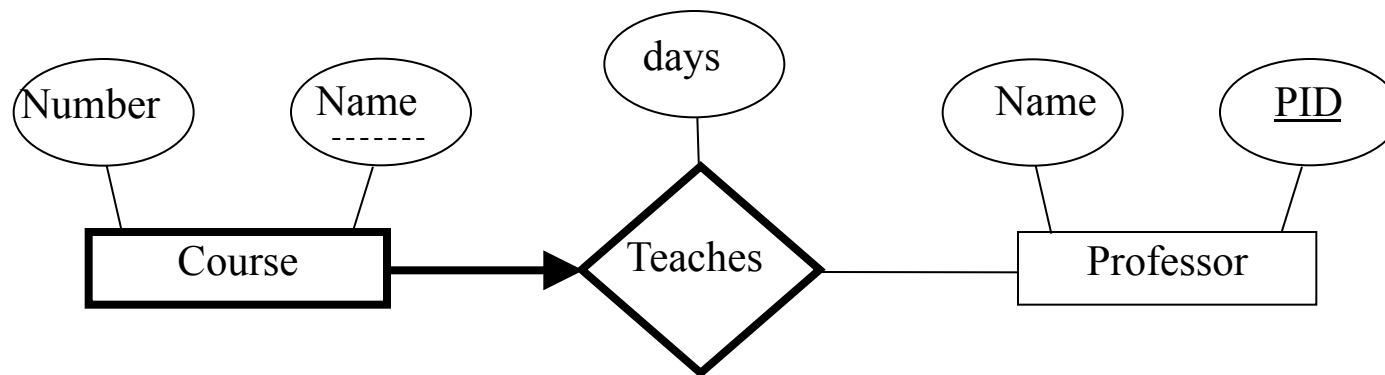
<i>name</i>	<u><i>S.S.N</i></u>	<i>street</i>	<i>city</i>
Lisa	1272	Main	Fairfax
Bart	5592	Apple	Manassas
Lisa	7552	Ox	Fairfax
Sue	5555	Lee	Vienna

Translating ER diagrams into Relations

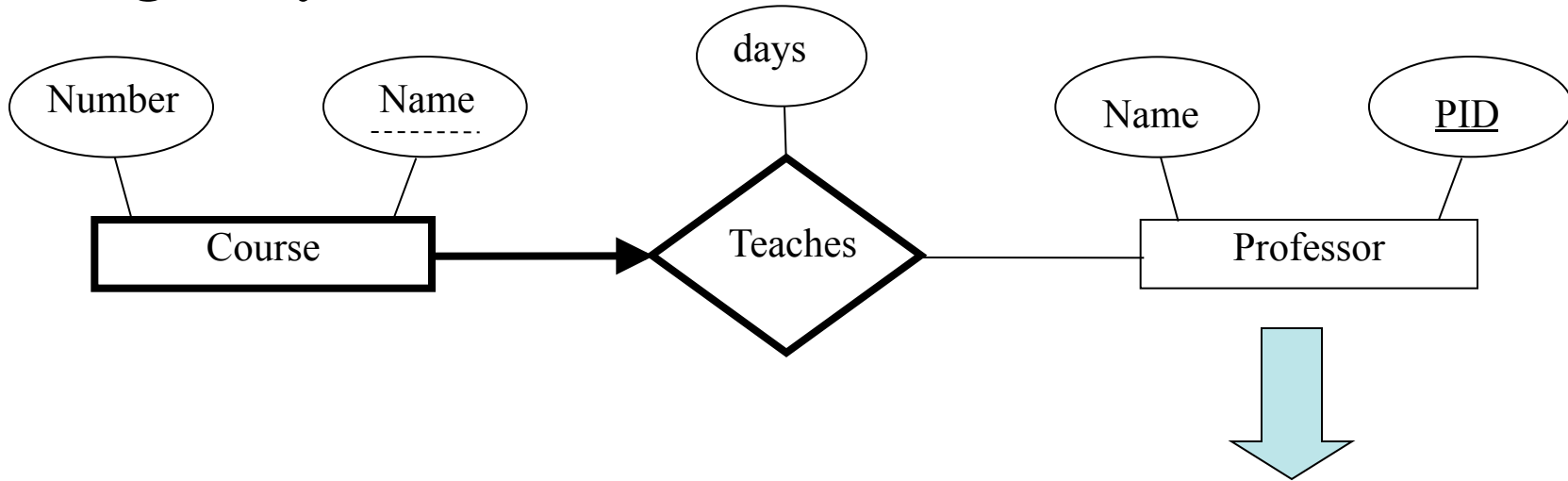
We need to figure out how to translate ER diagrams into relations.

There are only three cases to worry about.

- Strong entity sets
- Weak entity sets
- Relationship sets



- Strong entity sets

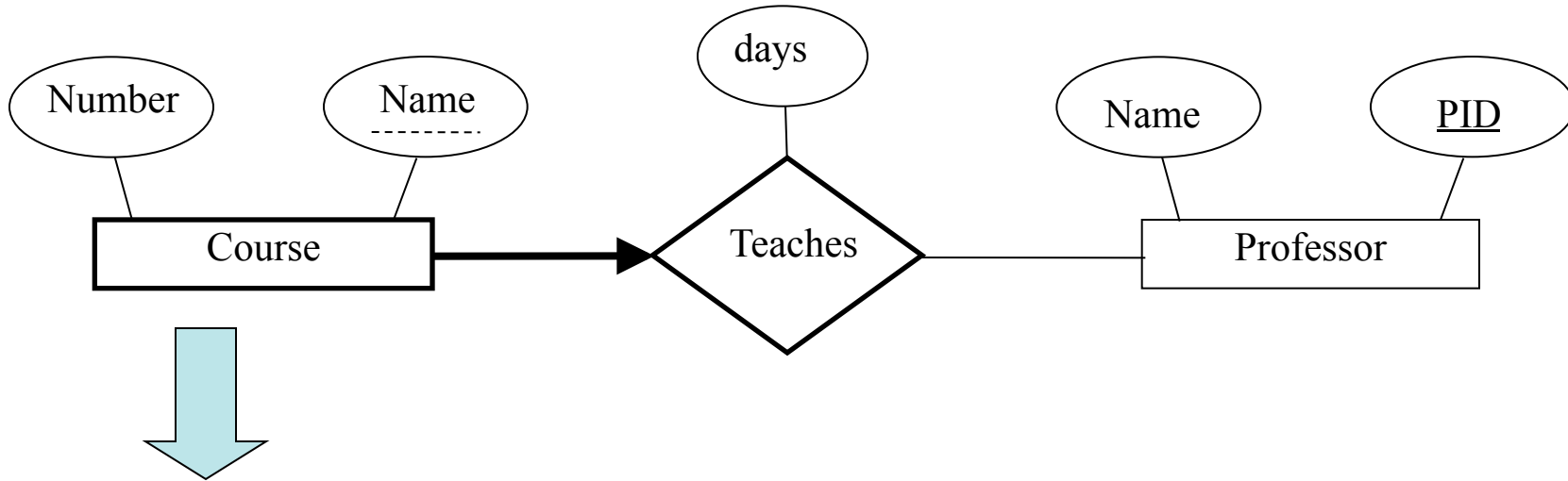


professor(PID : string, name : string)

This is trivial, the primary key of the ER diagram becomes the primary key of the relation. All other fields are copied in (in any order)

<u>PID</u>	name
1234	Lin
3421	Lee
2342	Smyth
4531	Lee

- **Weak entity sets**



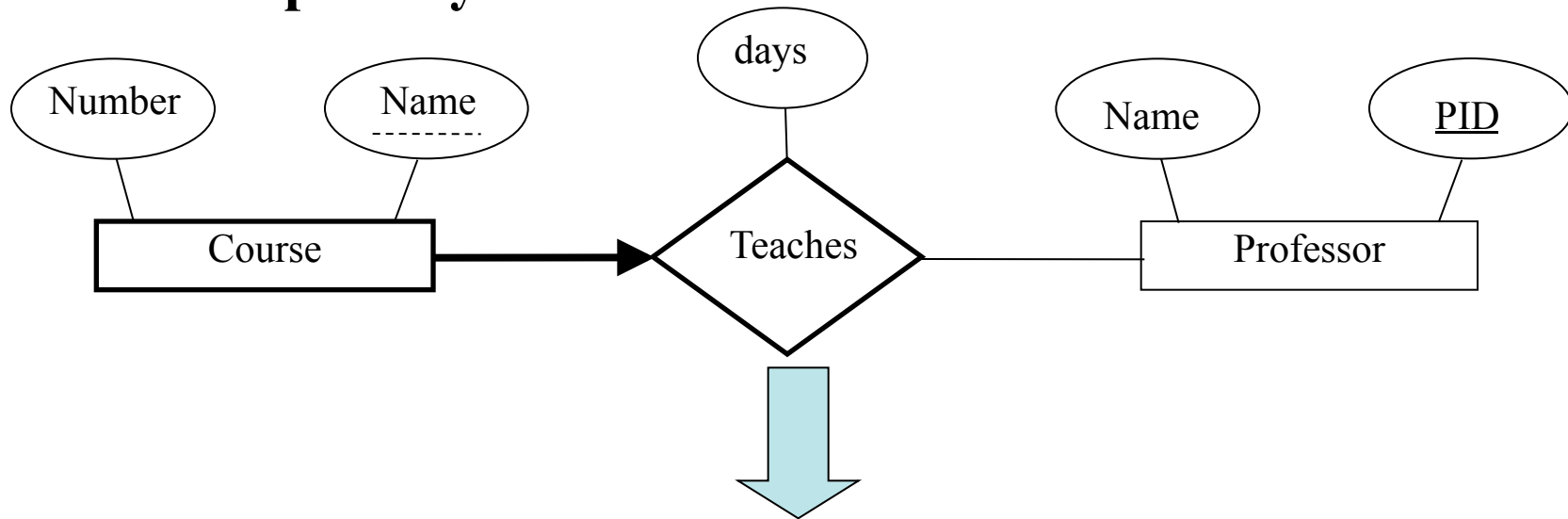
course(PID : string, *number* : string, name : string)

<u>PID</u>	<i>number</i>	<u>name</u>
1234	CS12	C++
3421	CS11	Java
2342	CS12	C++
4531	CS15	LISP

The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set. The “imported key from the strong entity set is called the **foreign key**.

All other fields are copied in (in any order)

• Relationship entity sets



teaches(PID : string, days : string)

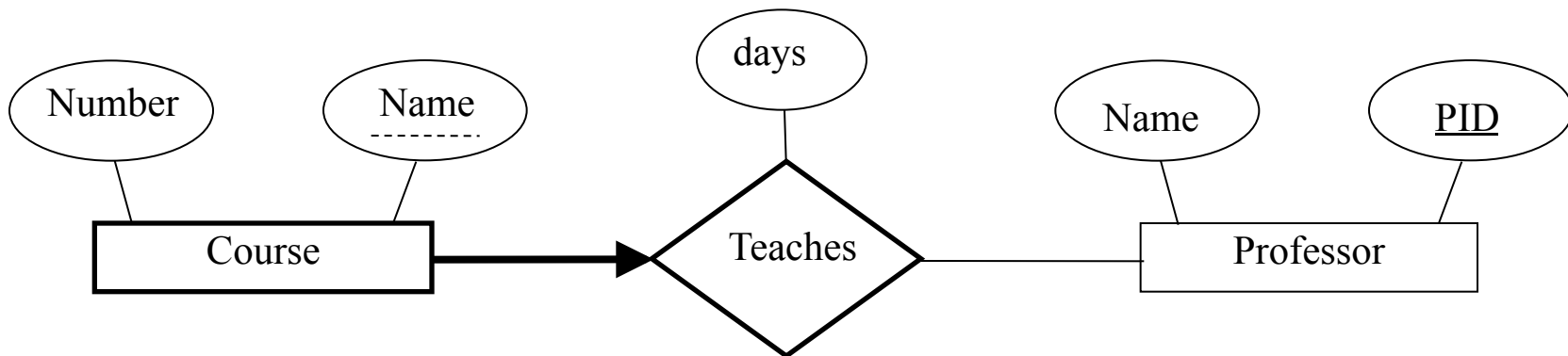
For one-to-one relationship sets, the relation's primary key can be that of either entity set.

- For many-to-many relationship sets, the union of the primary keys becomes the relation's primary key

- For other cases, the relation's primary key is taken from the strong entity set.

<u>PID</u>	days
1234	mwf
3421	wed
2342	tue
4531	sat

So, this ER Model...




... maps to this **database schema**

professor(PID : string, name : string)

course(PID : string, number : string, name : string)

teaches(PID : string, days : string)

Later we'll see how we can take advantage of the key constraint and come up with a better design.



We have seen how to create a database schema, how do we create an actual database on our computers?

professor(*PID* : string, *name* : string)

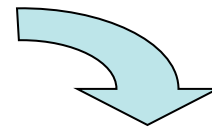
course(*PID* : string, *number* : string, *name* : string)

teaches(*PID* : string, *days* : string)

...how do we create an actual database on our computers?

We use SQL, a language that allows us to build, modify and query databases.

professor(PID : string, name : string)



```
SQL> CREATE TABLE Professor
2   (pid CHAR(9),
3   name CHAR(50),
4   PRIMARY KEY(pid));

Table created.
```

SQL (Structured Query Language)

- SQL is a language that allows us to build, modify and query databases.
- SQL is an ANSI standard language. American National Standards Institute
- SQL is the “engine” behind Oracle, Microsoft SQL Server, etc.
- Most of these systems have built GUIs on top of the command line interface, so you don’t normally write statements directly in SQL (although you can).

Creating Relations in SQL

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

```
CREATE TABLE Students  
(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));
```

Creating Domains

- Say you want to restrict the values of GPA
($0 < \text{GPA} \leq 4.0$)
- Approach 1: Specify constraint when
defining the table

```
CREATE TABLE Students
  (sid CHAR(20),
   name CHAR(20),
   login CHAR(10),
   age INTEGER,
   gpa REAL check(gpa <= 4.0 AND gpa > 0) );
```

Creating Domains

- Approach 2: After CREATING TABLE, use ALTER TABLE

```
CREATE TABLE Students
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL);
```

```
ALTER TABLE Students
ADD CONSTRAINT check_gpa CHECK(gpa > 0 AND gpa <= 4.0);
```

To specify a set of allowed values, do something like this (using either approach):
... CHECK(gender='M' OR gender='F')

Getting Info About Existing Tables

To get the list of existing tables in your database:

```
SELECT table_name  
FROM user_tables;
```

or

```
SELECT table_name  
FROM all_tables  
WHERE owner='YOUR_ACCOUNT_NAME_IN_UPPER_CASE';
```

To get more about an existing table, say, Students:

```
Describe Students;
```

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  
WHERE name = 'Smith';
```

The SQL Query Language

- 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 show just names and logins columns, replace the first line with:

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

Querying Multiple Relations

- What does this query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A';
```

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

Enrolled

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

we get:

S.name	E.cid
Smith	Topology112

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` firstYear integer;

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 Revisited

- 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. no subset of the key is also a key.
 - 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*.
- What's the difference between the two statements below?

```
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) );
```

Used carelessly, an IC can prevent the storage of database instances that arise in practice!

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!

artist_id	artist_name
1	Bono
2	Cher
3	Nuno Bettencourt

Link Broken

artist_id	album_id	album_name
3	1	Schizophonic
4	2	Eat the rich
3	3	Crave (single)

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.

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.