# CS634 Lecture 2

Review of integrity constraints, relationships Jan 27, 2016

# Foreign Keys

Defined in Sec. 3.2.2 without mentioning nulls. Nulls covered in Sec. 5.6

First example: nice not-null foreign key column:

```
create table enrolled(
        studid char(20),
        cid  char(20),
        grade char(10),
        primary key(studid,cid),  -- so both these cols are non-null
        foreign key(studid) references students
        );
```

This FK ensures that there's a real student record for the studid listed in this row.

# Not-null Foreign Key Example

```
create table enrolled(
        studid char(20),
        cid  char(20),
        grade char(10),
        primary key(studid,cid),  -- so both these cols are non-null
        foreign key(studid) references students
        );
```

Notes:

- studid would more normally be an int, not char(20), although it doesn't matter for small tables.
- cid looks like a key for a class being taken by this student. If so, it should have a FK constraint too, and more likely int type.
- We need to load the students table before the enrolled table, and drop this table before dropping the students table.
- For mysql, we need to put "references students(sid);"

# Nullable Foreign Key columns

```
create table emp(
  eid int primary key,
  ename varchar(30) not null,
  mgrid int references emp(eid),
  age int,
  salary decimal(10,2) not null
);
```

- The big boss has no manager, so null mgrid, yet still is an employee, so belongs in the table. Just need nullable FK.
- This gives us a natural example of a FK referencing the same table.
- Also an example of a FK defined in the column definition.
- Note the SQL standard type **decimal(10,2)**, good for any salaries up to 8 dollar digits, max $99,999,999.99 , enough for most companies.
- The null FK value means that the database skips checking this value. There can be multiple null values in general.

# Null Foreign Keys: two table example

```
create table emp(
eid int primary key,
ename varchar(30) not null,  // company doesn't control enames, so can't make not null
did int not null references dept(did),
age int,
salary decimal(10,2)
);
create table dept(
did int primary key,
dname varchar(20) not null unique,  // company does control dnames…
budget decimal(10,2),
managerid int,
foreign key(managerid) references emp(eid)
);
```

Here a null managerid means that the department has no manager currently.

To insert a new department, first insert a dept row with null managerid, then move some employees to that dept, then make one of them the manager by replacing the null managerid with that eid.

# Null Foreign Keys: two table example

```
create table emp(
eid int primary key,
ename varchar(30) not null,  // company doesn't control enames, so can't make not null
did int not null references dept(did),
age int,
salary decimal(10,2)
);
create table dept(
did int primary key,
dname varchar(20) not null unique,  // company does control dnames…
budget decimal(10,2),
managerid int,
foreign key(managerid) references emp(eid)
);
```

Here a null managerid means that the department has no manager currently.

The dname is not null unique, and thus is a key.

To list each ename and coresponding manager id:

```
select e.ename, d.managerid from emp e join dept d
    on e.eid = d.managerid
```

Here the join column is emp.did,  not null, so no surprises

# Null Foreign Keys: two table example

```
create table emp(
eid int primary key,
ename varchar(30) not null,  // company doesn't control enames, so can't make unique
did int not null references dept(did),
… );
create table dept(
did int primary key,
dname varchar(20) not null unique,  // company does control dnames…
budget decimal(10,2),
managerid int,
foreign key(managerid) references emp(eid)
);
```

To list managers with their names and department names:

```
select dname, e.name from dept d, emps e
    where d.managerid = e. eid;
```

Join column = d.managerid,  nullable, so be careful in interpretation…

Suppose the HR department has no manager at the moment—what is listed here?

Nothing!  The department has disappeared from this report.

How to get it listed even with null join key??

# Null join key: outer join to the rescue!

```
select dname, e.name from dept d, emps e
    where d.managerid = e. eid;
```

Join column = d.managerid, nullable, so be careful!

This inner join ignores any null join keys and null doesn't match null. Of course it also ignores rows in either table that don't match up by join key.

The Outer join (pg. 104) is more forgiving, listing all rows somehow in one or both tables.
Need to use JOIN form: Here to preserve rows in dept, the left table:

```
select dname, e.name from dept d LEFT OUTER JOIN emps e
    on d.managerid = e.eid;
```

Now HR gets listed with dname and null for e.name.
This extra complication of nullable FKs makes a lot of people nervous about them, along with worries about proper implementation in databases, but note that mysql/Innodb works fine on this. (Of course Oracle does too.)

# Constraint Names: example from Wikipedia

```
CREATE TABLE Supplier (
SupplierNumber INTEGER NOT NULL,
Name VARCHAR(20) NOT NULL,
Address VARCHAR(50) NOT NULL,
Type VARCHAR(10),
CONSTRAINT supplier_pk PRIMARY KEY(SupplierNumber),
CONSTRAINT number_value CHECK (SupplierNumber > 0) )
```

Useful for later deletion of the constraint:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

Check constraint: not supported by mysql

# Referential Actions

```
CREATE TABLE Invoices (
    InvoiceNumber   INTEGER NOT NULL,
    SupplierNumber  INTEGER NOT NULL,
    Text            VARCHAR(4096),
    CONSTRAINT invoice_pk PRIMARY KEY(InvoiceNumber),
    CONSTRAINT inumber_value CHECK (InvoiceNumber > 0),
    CONSTRAINT supplier_fk FOREIGN KEY(SupplierNumber)
      REFERENCES Supplier(SupplierNumber)
      ON UPDATE CASCADE ON DELETE RESTRICT )
```

- On update cascade: if the supplierNumber in supplier is changed, say from 100 to 200, then all the rows in invoices with supplierNumber = 100 will also be changed to 200.
- On delete restrict: If supplierNumber 100 is deleted in supplier, that delete will fail.
- Book, pg. 71: Add RESTRICT, nearly same as NO ACTION.

# Book Example, pg. 71

```
CREATE TABLE Enrolled ( studid CHAR(20) ,
cid CHAR(20) ,
grade CHAR(10),
PRIMARY KEY (studid, dd),
FOREIGN KEY (studid) REFERENCES Students
ON DELETE CASCADE
ON UPDATE NO ACTION)
```

On delete cascade: student deleted means all enrolled rows for student deleted. Makes sense: enrollment records are details on the student.
On update no action: student studid updated (who has enrolled courses): update fails

# Foreign Keys added to table

ALTER TABLE <table identifier>
  ADD [ CONSTRAINT <constraint identifier> ]
    FOREIGN KEY ( <column expression> )
    REFERENCES <table identifier> [ ( <column expression> {, <column expression>}... ) ]
    [ ON UPDATE <referential action> ]
    [ ON DELETE <referential action> ]

Useful for loading tables when there is a cycle in FK relationships.

# Book Example, pg. 72: cycle in non-null FKs

CREATE TABLE Students ( sid CHAR(20) ,
name CHAR(30), login CHAR (20) ,
age INTEGER, honors CHAR(10) NOT NULL, gpa REAL)
PRIMARY KEY (sid),
FOREIGN KEY (honors) REFERENCES Courses (cid))

CREATE TABLE Courses (cid CHAR(10),
cname CHAR ( 10) , credits INTEGER, grader CHAR(20) NOT NULL,
PRIMARY KEY (dd)
FOREIGN KEY (grader) REFERENCES Students (sid))

Loading: create one table without FK, load it first, then other, then alter table to add constraint.

Insert a new course and grader in a transaction: can get constraint checking deferred until commit—see pg. 72

Another solution: make one FK column nullable.

# E-R Translation to Tables (Sec. 3.5)

Simplest case: PK of one column, just an entity table, example, pg. 75

Relationship tables: PK has multiple columns.

From pg. 76:
To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship.

Thus, the attributes of the relation (the relationship table) include:

- The primary key attributes of each participating entity set, as foreign key fields.
- The descriptive attributes of the relationship set.

# E-R Translation to Tables (Sec. 3.5)

Most important Relationship: Binary, with two columns in PK of relationship table, so two related entities.

Binary Relationships: Entity  Entity

N-1 relationships can be translated to a FK in one of the tables
Example: each employee has a department, so did is in the emps table, and as a FK to dept. (i.e., don't *need* a relationship table for this case, though can be used)

N-N relationships are translated to a relationship table, with (key1, key2) rows, PK (key1, key2), key1 and key2 with FKs to their entity tables. There can be additional columns.

# N-N Relationships and their tables

Consider student and class entities, and their N-N relationship "enrolled". A student is enrolled in multiple classes, and a class has multiple students.

```
create table enrolled(
snum int,                  -- key value in student
cname varchar(40),  --key value in class
primary key(snum,cname), -- two keys, and both are not-null
foreign key(snum) references student(snum),
foreign key(cname) references class(name)
);
```

We can call this a relationship table, or join table. When you see a table that fits this pattern, it's implementing an N-N relationship between the pointed-to tables. If it's close to this, but missing something, consider if it's right or not.

# Example of an entity that looks like a relationship

Concept: a sailor reserves a boat for a certain day (pg. 102)
Use of a verb as its name makes it sound like a pure relationship, but is it?

**create table reserves(**
sid **int not null,** --key value of sailor table
bid **int not null,** -- key value of boat table
day **varchar(10),** -- day of reservation
**foreign key (sid) references sailors(sid),**
**foreign key (bid) references boats(bid)**
);
What is missing from this compared to the N-N relationship table?

# Example of an entity that looks like a relationship, continued

Concept: a sailor reserves a boat for a certain day

```
create table reserves(
sid int not null,  --key value of sailor table
bid int not null, -- key value of boat table
day varchar(10),  -- day of reservation
foreign key (sid) references sailors(sid),
foreign key (bid) references boats(bid)
);
```

What is missing from this compared to the N-N relationship table????
Answer: Primary key(sid, bid).
What is the PK here?
Answer: the set of all columns, the PK of last resort.
The above table schema means one sailor can reserve the same boat for different days. OK, that makes sense. I'd call this table "reservation" or "reservations", using a noun to go with its being an entity, not a pure relationship.

# Pure relationships can have attributes

**Concept:** a certain part supplied by a certain supplier has a certain cost.

```
create table catalog(
sid int, -- key value in suppliers
pid int, -- key value in parts
cost decimal(10,2),
primary key(sid,pid),
foreign key(sid) references suppliers(sid),
foreign key(pid) references parts(pid)
);
```

Here we see the pure relationship-table pattern, so we can classify this as a pure binary relationship table, in spite of its noun name. So what's cost doing in here?

It's an attribute of the relationship: suppliers supply parts, each with its particular cost. Different suppliers can charge different amounts for the same generic part. The PK (sid,pid) means that for a certain pid and sid, we have just one cost.

# Catalog: Entity or Relationship?

Do we have to call catalog a relationship?

No, it's just that we can call it that. Any N-N relationship with attributes can alternatively be considered an entity with two N-1 relationships outbound from it.

This flexibility is often used in data modeling environments that don't support attributes of relationships.

The down-side of this alternative is that we can't express the constraint of the PK (sid, pid) in the diagram.

We can draw two E-R diagrams for this…on the board anyway.

# Weak entities

Weak entities: entities that are details of another entity, and depend on them for their identity.

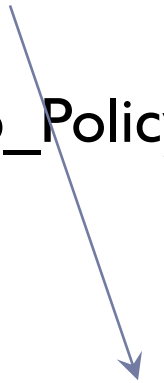
That means the PK of the weak entity contains the PK of the related entity.

Example, pg. 82, dependents and employees
Concept: an employee may have several dependents, not of interest in themselves but only as details of the employee. We may not know their SSNs or other unique ids, but that's OK.

Employee has PK ssn
Dependents has PK (pname, ssn)
FK (ssn) references employees on delete cascade

# Weak Entity Example from pg. 82

FK in PK along with relative id pname



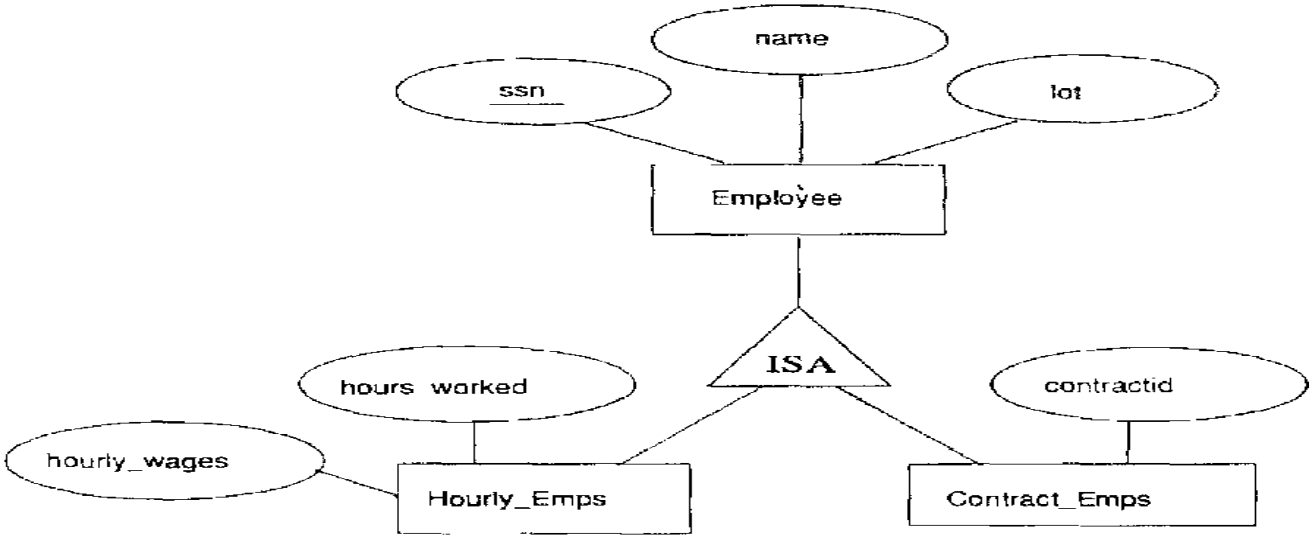
```
CREATE TABLE Dep_Policy (pname CHAR(20) ,
age INTEGER,
cost REAL,
ssn CHAR (11) ,
PRIMARY KEY (pname, ssn),
FOREIGN KEY (ssn) REFERENCES Employees
ON DELETE CASCADE )
```

An employee can have dependent policies, identified by their own ssn along with the dependent's name.

# Translating class hierarchies

Example on pg. 83
Employees may be Hourly_Emps or Contract_Emps, with different fields needed to describe them beyond the Employee fields.

This is a hard case with several solutions, none perfect.



# Translating class hierarchies

Example on pg. 83
Employees may be Hourly_Emps or Contract_Emps, with different fields needed to describe them beyond the Employee fields.

This is a hard case with several solutions, none perfect.

- In practice, we often just throw all the needed fields into the employees table and use nulls for fields that aren't relevant to a particular employee.

- It's important to have a classifier column so it's clear when nulls are appropriate for a column and row.

- Obviously we are giving up the power of "not null" here!

# Translating class hierarchies

A more normalized approach models the IS-A relationship shown on pg. 83:
An Hourly_Emp IS-A Employee
A Contract_Emp IS-A Employee

These are N-1 relationships, so the Hourly_emp table will have a FK to Employees (ssn) as well as the "extra" fields for hourly emp's.

Here ssn is unique in hourly_emp as well as in employee, so we have a case where a single column (ssn) is both a PK and a FK.

Though this is more normalized, it is not that easy to query without the help of object-relational tools.