

LAB 3 Notes

The Relational Model – Chapter 3

In the previous lab we discussed the Conceptual Database Design Phase and the ER Diagram. Today we will mainly discuss how to convert an ER model into the Relational model of a specific database.

Outline

- 1) Introduction – discuss briefly the relational model
- 2) Logical Database Design (Steps involved ER model -> Relational model).
- 3) A glance at Views
- 4) Putting it all together – An example using Postgres

1) Introduction

- Codd proposed the **relational model** in '70
- **Main advantage of Relational Model** : Simple representation (relations-tables(row, cols))
- **DDL (Data Definition Language)** – language for creating, manipulating and querying data in a relational DBMS.
- **Domain, Primary key, Foreign Key constraints are fundamental part of the relational model.**

2) Logical Database Design [ER schema -> relational database schema]

- ER schema is convenient for initial high-level description of the database design
- There is a standardized procedure to translate the ER diagram -> Relational schema (in fact software does it automatically)

A **relation schema** can be thought of as the **basic information describing a table** or relation. This includes a **set of column names**, the **data types** associated with each column, and the name associated with the entire table.

Formally: { $\langle f_1 : d_1, \dots, f_n : d_n \rangle \mid d_1 \in \text{DOM}_1, \dots, d_n \in \text{DOM}_n \}$

For **example**, a relation schema for the relation called Students could be expressed using the following representation:

Students(sid: string, name: string, login: string, age: integer, gpa: real)

There are five fields or columns, with names and types as shown above.

Domain is synonymous with data type. Attributes can be thought of as **columns in a table**. Therefore, an attribute domain refers to the data type associated with a column.

A relation instance is a set of tuples (also known as rows or records) that each conform to the schema of the relation.

The relation **cardinality** is the number of tuples in the relation.
 The relation **degree** is the number of fields (or columns) in the relation.

Relational schema: Product(Name, Price, Category, Manufacturer)

Instance:

Name	Price	Category	Manufacturer
gizmo	\$19.99	gadgets	GizmoWorks
Power gizmo	\$29.99	gadgets	GizmoWorks
SingleTouch	\$149.99	photography	Canon
MultiTouch	\$203.99	household	Hitachi

Entity Example:

Employee(ssn, name, age);

Employee

ssn	name	age
-----	------	-----

We use the **Data Definition Language** (syntax) of a DBMS to describe the generated Tables

// notation used in book, (ingres, ansi)

```
- CREATE TABLE Employee (ssn CHAR(11) NOT NULL,
                           name CHAR(30) NOT NULL,
                           age Integer,
                           Primary Key(ssn));
```

```
- ALTER TABLE Employee ADD Column surname:CHAR(40) NOT NULL;
```

// Adding to a relation

```
- INSERT INTO Employee(ssn, name, age) VALUES ("34342", "Smith", 18);
```

//Update tuple

```
- UPDATE Employee SET name="John" where name="smith";
```

//delete tuples

```
- DELETE Employee WHERE name="John"
```

//Drop Table

```
- DROP TABLE Employees;
```

Step 2: Relationships

Constraints found in a relationship

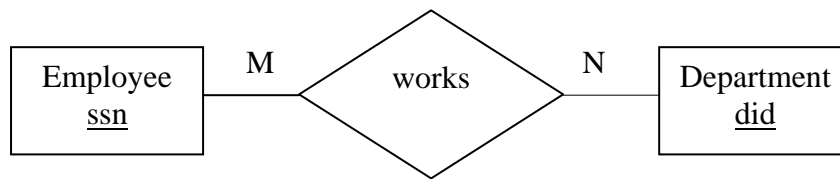
Key Constraint → determined by the “At-Most” question

Participation Constraint → determined by the “At-Least” question

2a) WITHOUT PARTICIPATION CONSTRAINT

M:N Relationships

- The relationship becomes a relation itself EMP_DEPT(ssn,did)
- Also called an Entity-Relationship

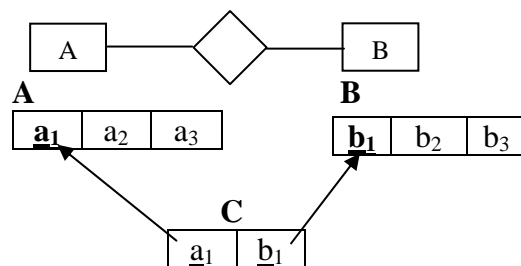


```
CREATE TABLE Employee (ssn CHAR(11) NOT NULL,
                        Primary Key(ssn));
```

```
CREATE TABLE Department (did INTEGER NOT NULL,
                          Primary Key(did));
```

```
CREATE TABLE Emp_Dept (ssn CHAR(11) NOT NULL ,
                       did INTEGER NOT NULL ,
                       Primary Key(ssn, did),
                       FOREIGN KEY (ssn) REFERENCES Employee(ssn),
                       FOREIGN KEY (did) REFERENCES Department(did));
```

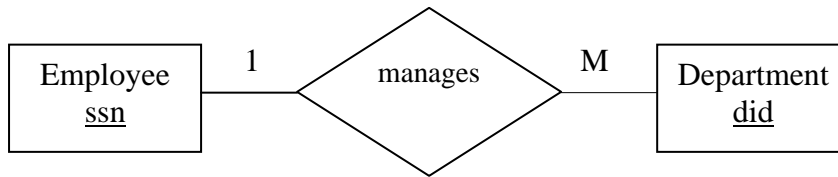
Generally:



1:M Relationships

- Suppose that an Employee manages N Departments but any department is managed only by at-most 1 person.
- The key of the one Entity moves to the direction of M.
- → Employee(ssn), Department(did, ssn)

primary key foreign key



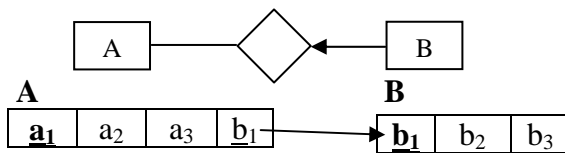
```

CREATE TABLE Employee (ssn CHAR(11) NOT NULL,
                        Primary Key(ssn));
  
```

```

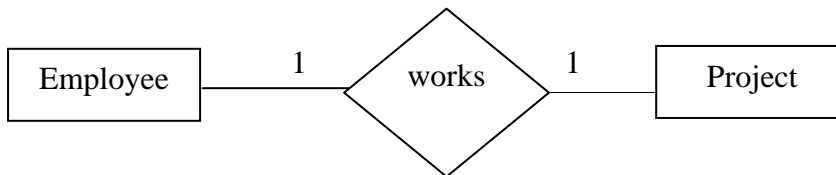
CREATE TABLE Department (did INTEGER NOT NULL,
                          ssn CHAR(11) NOT NULL
                          Primary Key(ssn),
                          FOREIGN KEY (ssn) REFERENCES Employee(ssn));
  
```

Generally:



1:1 Relationships

- Suppose an employee works on at-most 1 single-person project



- Move the key to either direction BUT remember that you want to avoid nulls
- E.g.
Employee(ssn, pid) Project(pid)

May create too many **nulls** because each employee is not said to be working on **AT-least 1 project**

The **participation constraint** that will be described in a while will make it clear to which direction we should move the key

- **Employee(ssn) Project(pid, ssn) on the other hand has no nulls since every project belongs to some employee.**

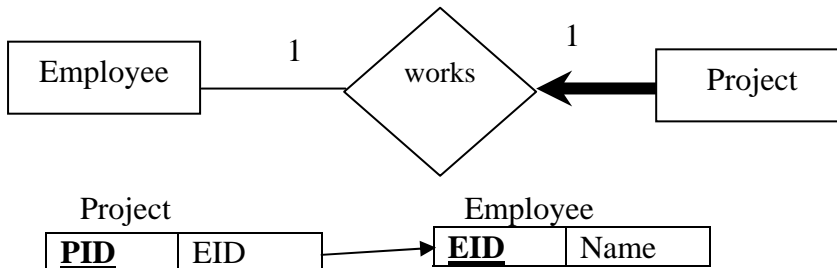
2b) WITH PARTICIPATION Constraints

- Participation Constraint → The At-Least question (bold line)
- This will give you the complete picture since any ER should capture key and participation constraints

- i) FIRST of all the participation constraint makes it clear in which direction to move the key in the case of a 1:1 relationship

Again 1:1 Relationships

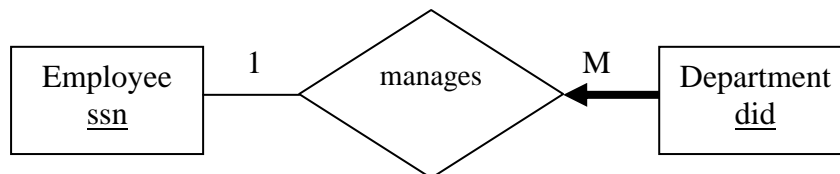
- Suppose an employee works on 0:1 projects and each project is worked on by 1:1 employees



- Now we automatically know that the key moves to the direction of the bold line. (so we avoid nulls).
- What would happen if the participation constraint was from both sides
 → the key goes to either direction

ii) **Enforcing referential integrity**

- Suppose you are given the following example
- We already know what is happening with the keys. (Dept gets the key)



ON [DELETE|UPDATE] “of the referred tuple the DBMS should”{ CASCADE, SET NULL, RESTRICT=NO ACTION=default, SET DEFAULT} “with the current relation”

a) RESTRICT=No ACTION=default

```
CREATE TABLE Department (did INTEGER,
                        ssn CHAR(11) NOT NULL
                        Primary Key(ssn)
                        FOREIGN KEY (ssn) REFERENCES Employee(ssn)
                        ON DELETE NO ACTION;
```

Means:

ON DELETE of the referred tuple in relation EMPLOYEE the DBMS should prohibit the delete of the given employee

Employee

ssn
1
2
3

Department

did	ssn
1	1
2	2
3	3

If we want to delete employee 1 i.e.

Delete FROM Employee where ssn='1';

➔ System gives “Illegal Action – Violates Referential Integrity Constraint”

We can change the manager did=1 to did=2 and then try again

e.g. UPDATE Department D

SET D.ssn=2

WHERE D.ssn=1;

b) CASCADE

```
CREATE TABLE Department (did INTEGER,  
                           ssn CHAR(11) NOT NULL  
                           Primary Key(ssn)  
                           FOREIGN KEY (ssn) REFERENCES Employee(ssn)  
                           ON DELETE CASCADE;
```

Means:

ON DELETE of the refereed tuple in relation EMPLOYEE the DBMS should cascade the delete. That means that the Department record should be deleted as well!!!!

c) SET NULL

```
CREATE TABLE Department (did INTEGER,  
                           ssn CHAR(11) NOT NULL  
                           Primary Key(ssn)  
                           FOREIGN KEY (ssn) REFERENCES Employee(ssn)  
                           ON DELETE SET NULL;
```

ILLEGAL since ssn can't be NULL

MIGHT violate PARTICIPATION constraint

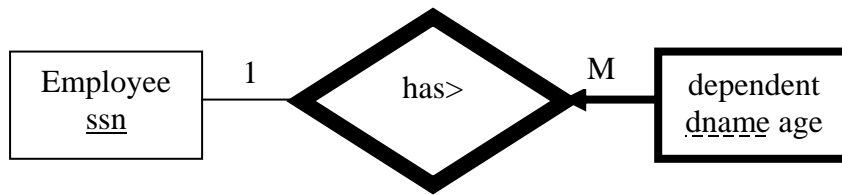
d) SET DEFAULT

```
CREATE TABLE Department (did INTEGER,  
                           ssn CHAR(11) NOT NULL DEFAULT "0";  
                           Primary Key(ssn)  
                           FOREIGN KEY (ssn) REFERENCES Employee(ssn)  
                           ON DELETE SET DEFAULT;
```

Means:

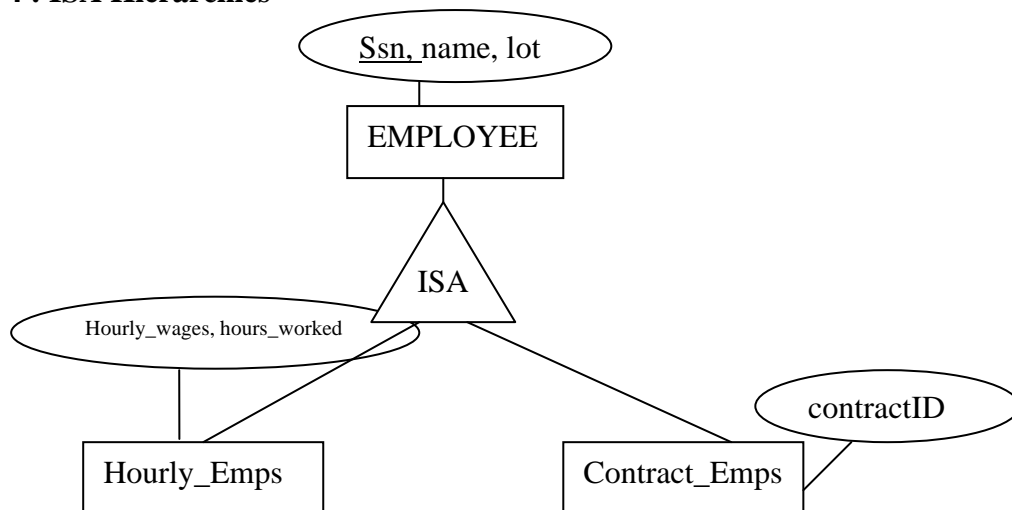
ON DELETE of the refereed tuple in relation EMPLOYEE the DBMS should set ssn in Department to the default value (i.e. 0) !!!!

Step 3: Weak Entities

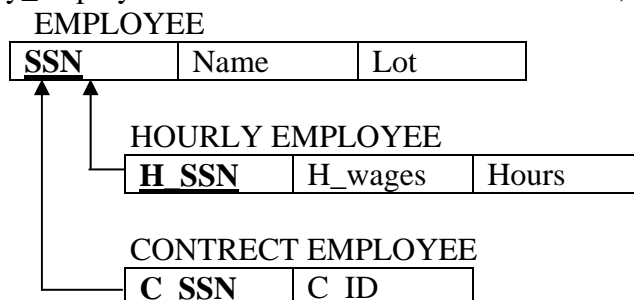


```
CREATE TABLE Dependent ( dname CHAR(11) NOT NULL,
                        ssn CHAR(11) NOT NULL,
                        age Integer,
                        Primary Key(ssn, dname)
                        FOREIGN KEY (ssn) REFERENCES Employee(ssn)
                        ON DELETE CASCADE; // deletes dependent
```

Step 4 : ISA Hierarchies



Inherit the key & create 3 relations. → In order to find e.g. name of hourly_employees we need to combine EMPLOYEE, HOURLY_EMPMS



3) VIEWS

1) Useful for hiding information – reducing degree

```
CREATE TABLE Employee (ssn CHAR(11) NOT NULL,
                       Name CHAR(30) NOT NULL,
                       PIN INTEGER NOT NULL,
                       Primary Key(ssn));
```

```
CREATE VIEW Secure_Employee(ssn, name)
AS SELECT E.ssn, E.name
FROM Employee E;
```

2) Useful for shrinking information – reducing cardinality

```
CREATE TABLE Employee (ssn CHAR(11) NOT NULL,
Name CHAR(30) NOT NULL,
BIRTHCITY char(30) NOT NULL,
Primary Key(ssn));
```

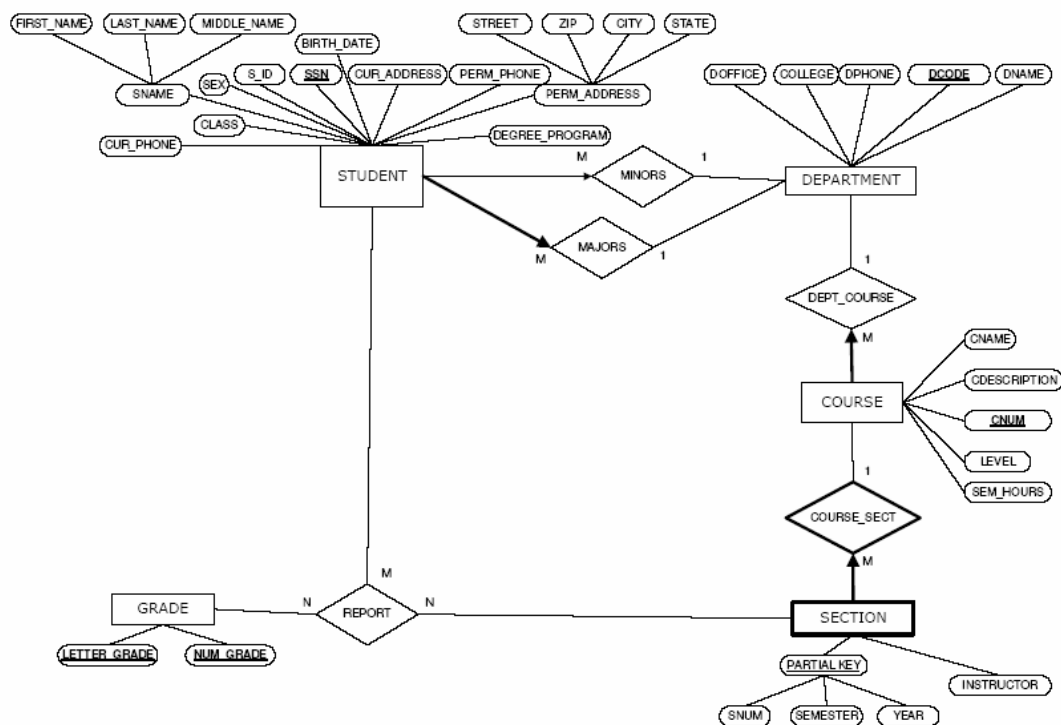
```
CREATE VIEW Chicago_Employees(ssn, name)
AS SELECT E.ssn, E.name
FROM Employee E
WHERE E. BIRTHCITY= "Chicago";
```

Attention needs to be given on updating VIEWS, It is generally a bad idea but you can still do it (updatable views)

MORE EXAMPLES GIVEN IN LAB AND OFFICE HOURS

Putting it all together

Consider the following ER diagram from last week:



Assumptions:

- 1) A student minors at most in one department.
- 2) A student has exactly one major.
- 3) A course may have many sections but each section belongs to one course.
- 4) A course is offered by only one department, while a department may offer many courses.
- 5) A student receives many grade reports and a section issues many grade reports but for a specific section a student may receive only one grade report.

Construct a corresponding relational schema and database in Postgres. In additional, create a view of reports with letter grades equal to A.