ICS 421 Spring 2010

Relational Model & Normal Forms

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa

Review

- ER model models the application data at the conceptual level
  - it does not assume any data model at the logical level
- A rigorous way to reason about ER is using set theory / Venn diagrams
  - Entity sets are collections of entities
  - Relationship sets are collections of edges connecting entities of entity sets
- Relational model – logical database design

Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation:* made up of 2 parts:
  - *Instance* : a *table,* with rows and columns. #Rows = *cardinality*, #fields = *degree / arity.*
  - *Schema* : specifies name of relation, plus name and type of each column.
    - E.G. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Example Instance of Students 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?

Relational Query Languages

- A major strength of the relational model: simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: 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 was developed by IBM (system R) in the 1970s
  - Standards: SQL-86, SQL-89 (minor revision), SQL-92 (major revision), SQL-99 (major extensions, current standard)

The SQL Query Language Syntax

- A simple SQL query takes the following form:

  SELECT <list of column names>
  FROM <list table names>
  WHERE <conditions>

- Conditions can be a boolean combination using AND, OR, NOT
- SQL queries can be nested into the FROM and WHERE clauses
- Conceptually, results of a SQL query is also a relation

Example: SQL Query on Single Table

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

- To find just names and logins:

```
SELECT S.name, S.login
FROM Students S
WHERE S.age=18
```

Querying Multiple Relations

- What does the following query compute?

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

Given the following instances of Enrolled and Students:

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

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

we get:

S.name	E.cid
Smith	Topology112

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.

```
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))
```

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?
    - Links in HTML!

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 )
```

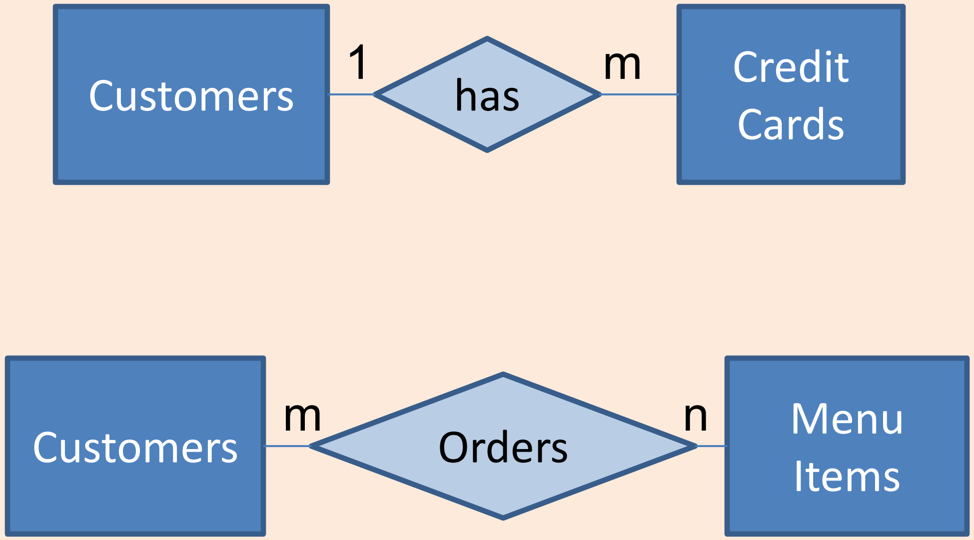
Enrolled

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

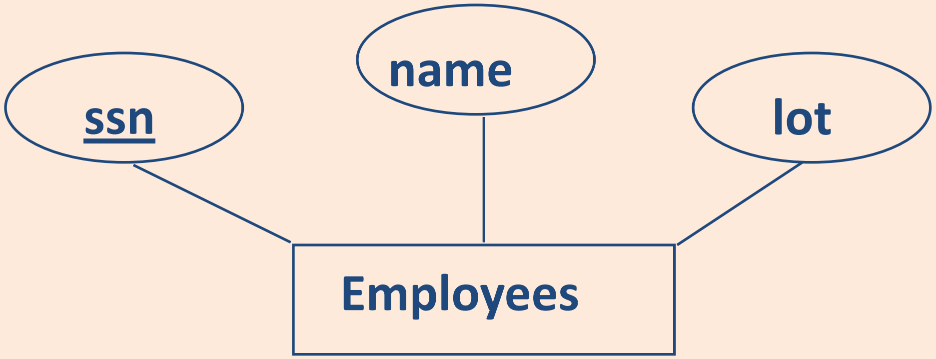
Students

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

Logical DB Design: ER to Relational



Entity Sets to Tables




```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

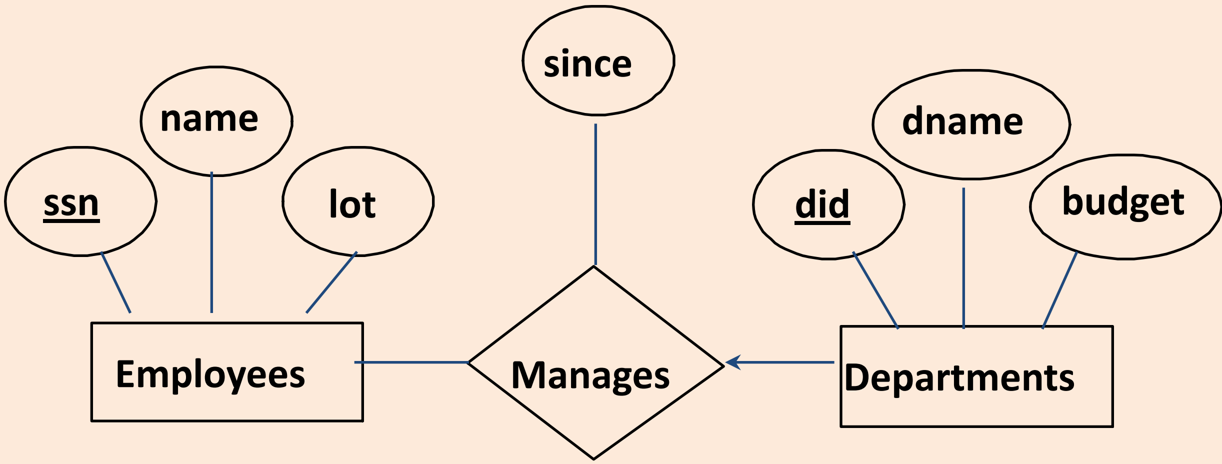
Relationship Sets to Tables

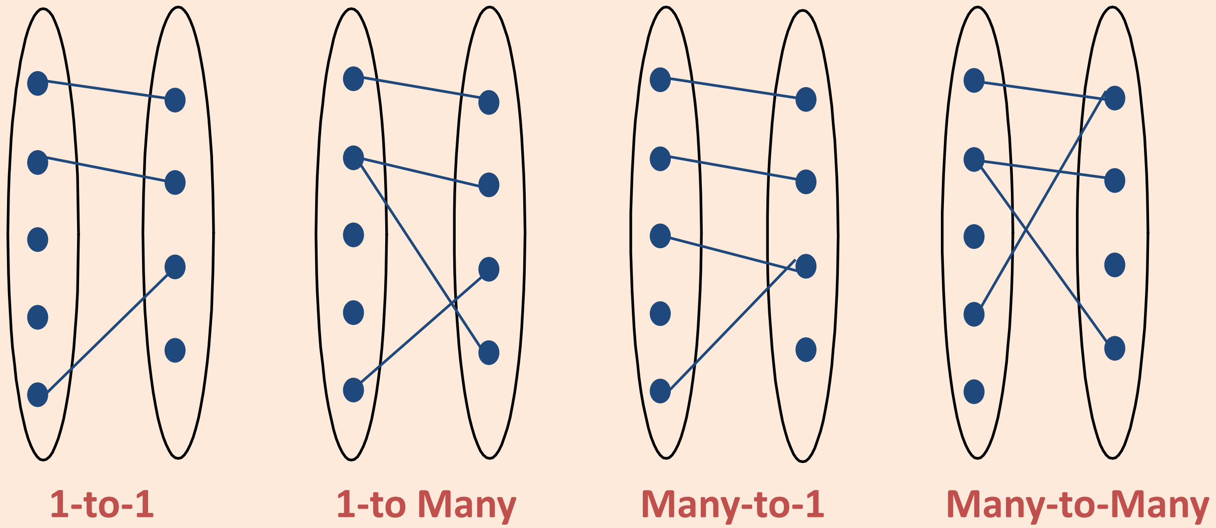
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

```
CREATE TABLE Works_In(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
      REFERENCES Employees,
  FOREIGN KEY (did)
      REFERENCES Departments)
```

Review: Key Constraints

- Each dept has at most one manager, according to the *key constraint* on Manages.





Translation to relational model?

Translating ER Diagrams with Key Constraints

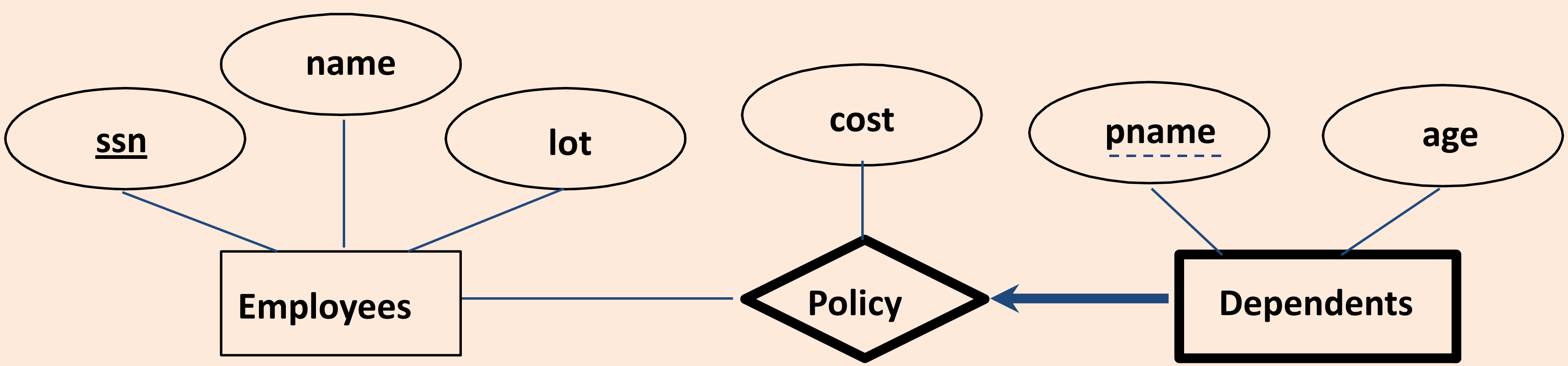
- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```

Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

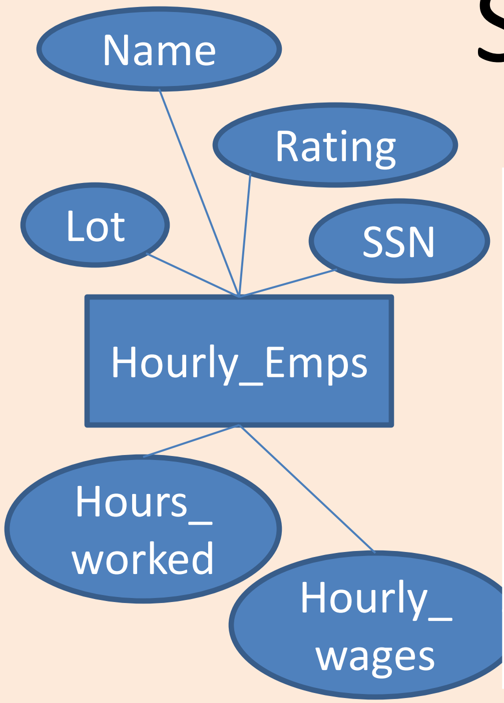


Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Dep_Policy (
  pname  CHAR(20),
  age  INTEGER,
  cost  REAL,
  ssn  CHAR(11) NOT NULL,
  PRIMARY KEY  (pname, ssn),
  FOREIGN KEY  (ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```

Schema Refinement



Hourly_Emps

<u>SSN</u>	Name	Lot	Rating	Hourly_wages	Hours_worked
123-22-2366	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Suppose hourly wages are determined by rating
- **Redundant storage** : (8,10) stored multiple times
- **Update anomaly** : change hourly wages in row 1
- **Insertion anomaly** : requires knowing hourly wages for the rating
- **Deletion anomaly** : deleting all (8,10) loses info

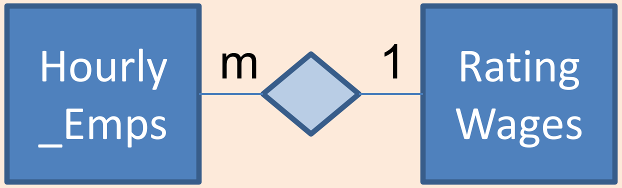
Using Two Smaller Tables

Hourly_Emps

SSN	Name	Lot	Rating	Hours_ worked
123-22-2366	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

RatingWages

Rating	Hourly_ wages
5	7
8	10



- *Notation*: denote relation schema by listing the attributes SNLRWH
- **Update anomaly** : Can we change W for Attishoo?
- **Insertion anomaly** : What if we want to insert an employee and don't know the hourly wage for his rating?
- **Deletion anomaly** : If we delete all employees with rating 5, do we lose the information about the wage for rating 5?

Decomposition

Hourly_Emps

SSN	**Name**	**Lot**	**Rating**	**Hours_ worked**
123-22-2366	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

RatingWages

Rating	**Hourly_ wages**
5	7
8	10

- Remove redundancy by decomposition
  - Since hourly wage is completely determined by rating, factor out hourly wage.
- Pros: less redundancy less anomalies
- Cons: retrieving the hourly wage of an employee requires a join

Functional Dependency

- A functional dependency X -> Y holds over relation R if, for every allowable instance *r* of R:
  - for all tuples *t1,t2* in *r*,

    $$\pi_X(t1) = \pi_X(t2) \text{ implies } \pi_Y(t1) = \pi_Y(t2)$$

  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that K -> R
  - However, K -> R does not require K to be *minimal*!

FD Example

Hourly_Emps

SSN	Name	Lot	Rating	Hourly_wages	Hours_worked
123-22-2366	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Two FDs on Hourly_Emps:
  - *ssn* is the key: S -> SNLRWH
  - *rating* determines *hourly_wages*: R -> W

Reasoning about FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn -> did, did -> lot* implies *ssn -> lot*
- Armstrong's Axioms
  - Let X, Y, Z are sets of attributes:
  - *Reflexivity*: If X is a subset of Y, then Y -> X
  - *Augmentation*: If X -> Y, then XZ -> YZ for any Z
  - *Transitivity*: If X -> Y and Y -> Z, then X -> Z
- These are *sound* and *complete* inference rules for FDs!

Example: Armstrong's Axioms

Hourly_Emps

<u>SSN</u>	Name	Lot	Rating	Hourly_**W**ages	**H**ours_worked
123-22-2366	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- *<u>Reflexivity</u>*: If X is a subset of Y, then Y -> X
  - SNLR is a subset of SNLRWH, SNLRWH -> SNLR
- *<u>Augmentation</u>*: If X -> Y, then XZ -> YZ for any Z
  - S -> N, then SLR -> NLR
- *<u>Transitivity</u>*: If X -> Y and Y -> Z, then X -> Z
  - S -> R, R -> W, then S -> W

Preparations for next class

- Install DB2 Express-C edition on your laptops by Thursday’s class
- Bring your laptops to class on Thursday.