Database Management Systems INFO 210

The Relational Model
Lecture 5

Franz Wotawa
TU Graz, Institut for Software Technologie
Inffeldgasse 16b/2
wotawa@ist.tugraz.at

1

Today…

- Last Session:
  - The ER model
- Today's Session:
  - The relational model
    - Basic SQL
    - ER to relational databases

2

Outline

- The Relational Model: Basic SQL
- Translating ER Diagrams to Tables and Summary
- Query Languages

3

What is the Relational Model?

- The relational model adopts a "tabular" representation
  - A database is a *collection* of one or more relations
  - Each relation is a *table* with rows and columns

- What is unique about the relational model as opposed to older data models?
  - Its simple data representation
  - Ease with which complex queries can be expressed

4

Basic Constructs

- The main construct in the relational model is the *relation*
- A relation consists of:
  1. A schema which includes:
     - The relation's name
     - The name of each column
     - The *domain* of each column
  2. An instance which is a set of tuples
     - Each tuple has the same number of columns as the relation schema

5

Creating Relations in SQL

- S1 can be used to create the "Students" relation
- S2 can be used to create the "Enrolled" relation

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

S1

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

S2

The DBMS enforces domain constraints whenever tuples are added or modified

6

Adding and Deleting Tuples

- We can insert a single tuple to the "Students" relation using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2);
```

- We can delete all tuples from the "Students" relation which satisfy some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith';
```

Powerful variants of these commands are available; *more on this next week*!

7

Integrity Constraints (ICs)

- An IC is a condition that must be true for *any* instance of the database (e.g., *domain constraints*)
  - ICs are specified when schemas are 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!

8

Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraints (ICs)

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

FOREIGN Key

Students

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

PRIMARY Key

9

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- "For a given student and course, there is a single grade"

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

vs.

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

Q: What does this mean?

10

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- “For a given student and course, there is a single grade”

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

vs.

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

“A student can take only one course, and no two students in a course receive the same grade”

11

Foreign Keys in SQL

- Example: Only existing students may 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	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

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

12

Enforcing Referential Integrity

- 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?
  - Disallow its deletion
  - Delete all Enrolled tuples that refer to it
  - Set sid in Enrolled tuples that refer to it to a *default sid*
  - Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*
- What if a "Students" tuple is updated?

13

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates
  - Default is NO ACTION (i.e., *delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to the 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 );
```

What does this mean?

14

Views

- A view is a table whose rows are not explicitly stored but computed as needed

```
CREATE VIEW YoungActiveStudents (name, grade)
        AS SELECT S.name, E.grade
        FROM Students S, Enrolled E
        WHERE S.sid = E.sid and S.age<21;
```

- Views can be queried
  - Querying YoungActiveStudents would necessitate computing it first then applying the query on the result as being like any other relation
- Views can be dropped using the DROP VIEW command
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this

15

Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is *changed*, a view can be defined to represent the old schema
    - This allows applications to *transparently* assume the old schema
  - Views can be defined to give a group of users access to just the information they are allowed to see
    - E.g., we can define a view that allows students to see other students' names and ages, but not GPAs (also students can be prevented from accessing the underlying "Students" relation)

16

Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is *changed*, a view can be defined to represent the old schema
    - This allows applications to *transparently* assume the old schema

  Logical Data Independence!

  - Views can be defined to give a group of users access to just the information they are allowed to see
    - E.g., we can define a view that allows students to see other students' names and ages, but not GPAs (also students can be prevented from accessing the underlying "Students" relation)

17

Views and Security

- Views can be used to present necessary information, while hiding details in underlying relation(s)
  - If the schema of an old relation is *changed*, a view can be defined to represent the old schema
    - This allows applications to *transparently* assume the old schema

  Logical Data Independence!

  - Views can be defined to give a group of users access to just the information they are allowed to see
    - E.g., we can define a view that allows students to see other students' names and ages, but not GPAs (also students can be prevented from accessing the underlying "Students" relation)

  Security!

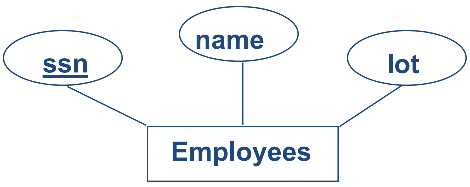
18

Outline

- The Relational Model: Basic SQL
- Translating ER Diagrams to Tables and Summary ✓
- Query Languages

19

Strong Entity Sets to Tables




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

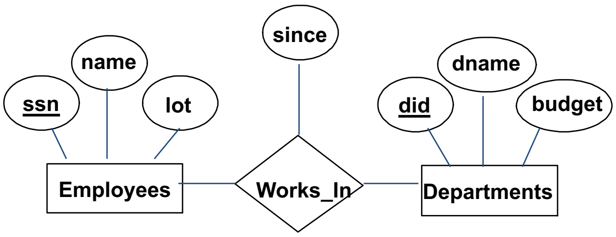
20

Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must include:
  1. Keys for each participating entity set (as foreign keys)
     - This set of attributes forms a *superkey* for the relation
  2. All descriptive attributes
- Relationship sets
  - 1-to-1, 1-to-many, and many-to-many
  - Key/Total/Partial participation

21

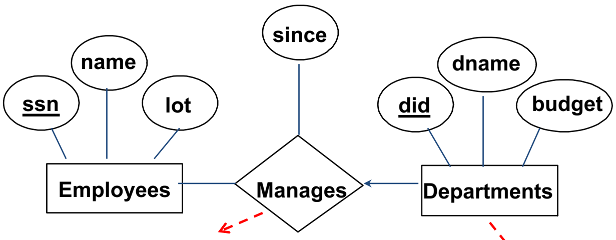
M-to-N Relationship Sets to Tables




```
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);
```

22

1-to-M Relationship Sets to Tables



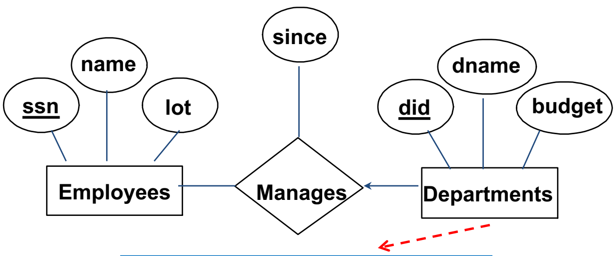

```
CREATE TABLE  Manages(
 ssn     CHAR(11),
 did     INTEGER,
 since   DATE,

 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
 FOREIGN KEY (did)
REFERENCES Departments);
```

```
CREATE TABLE  Departments(
 did          INTEGER,
 dname     CHAR(20),
 budget    REAL,
 PRIMARY KEY  (did)
);
```

Approach 1:
Create separate tables for Manages and Departments

23

1-to-M Relationship Sets to Tables




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

Can ssn take a *null* value?

Approach 2:
Create a table for only the Departments entity set (i.e., take advantage of the key constraint)

24

One-Table vs. Two-Table Approaches

- The one-table approach:

  (+) Eliminates the need for a separate table for the involved relationship set (e.g., Manages)

  (+) Queries can be answered without combining information from two relations

  (-) Space could be wasted!

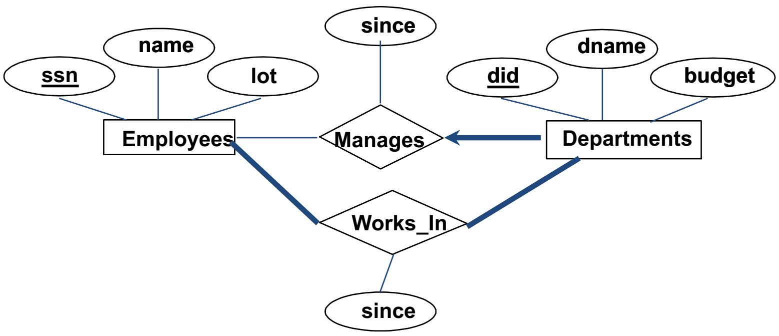
  - What if several departments have no managers?

- The two-table approach:
  - The opposite of the one-table approach!

25

Translating Relationship Sets with Participation Constraints

- What does the following ER diagram entail (with respect to Departments and Managers)?



Every *did* value in Departments table must appear in a row of the Manages table- *if defined*- (with a non-null *ssn* value!)

26

Translating Relationship Sets with Participation Constraints

- Here is how to create the “Dept_Mgr” table using the one-table approach:

```
CREATE TABLE  Dept_Mgr(
  did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  ssn  CHAR(11) NOT NULL,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (ssn) REFERENCES Employees,
    ON DELETE NO ACTION);
```

Can this be captured using the two-table approach?

27

Translating Relationship Sets with Participation Constraints

- Here is how to create the “Dept_Mgr” table using the one-table approach:

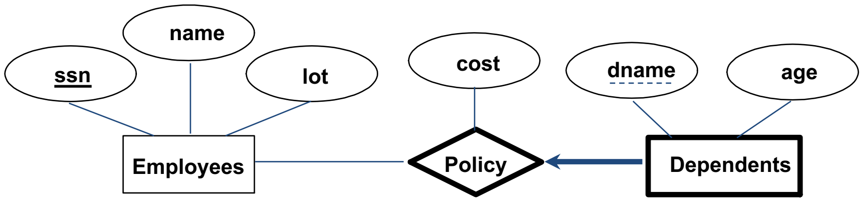
```
CREATE TABLE  Dept_Mgr(
  did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  ssn  CHAR(11) NOT NULL,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (ssn) REFERENCES Employees,
    ON DELETE SET NULL);
```

Would this work?

28

Translating Weak Entity Sets

- A weak entity set always:
  - Participates in a one-to-many binary relationship
  - Has a key constraint and total participation

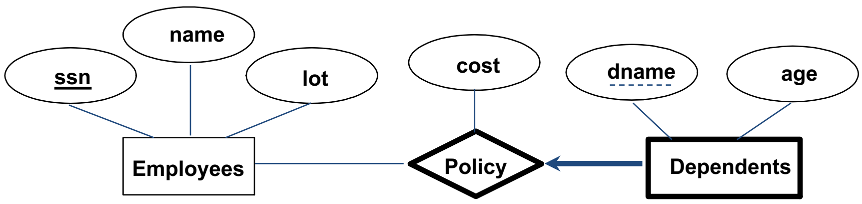


- Which approach is ideal for that?
  - The one-table approach

29

Translating Weak Entity Sets

- Here is how to create “Dep_Policy” using the one-table approach

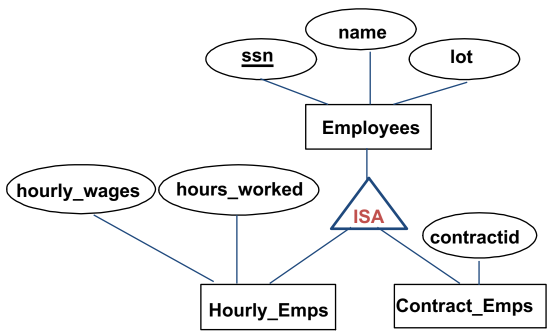



```
CREATE TABLE  Dep_Policy (
   dname   CHAR(20),
   age     INTEGER,
   cost    REAL,
   ssn     CHAR(11) NOT NULL,
   PRIMARY KEY  (dname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE);
```

30

Translating ISA Hierarchies to Relations

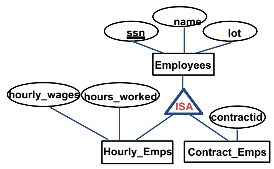
- Consider the following example:

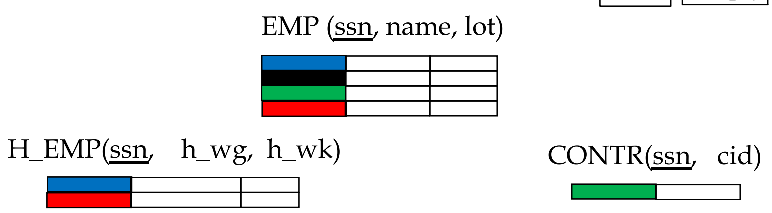


31

Translating ISA Hierarchies to Relations

- General approach:
  - *Create* 3 relations: "Employees", "Hourly_Emps" and "Contract_Emps"



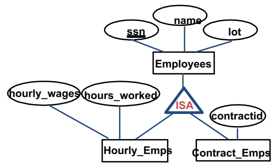


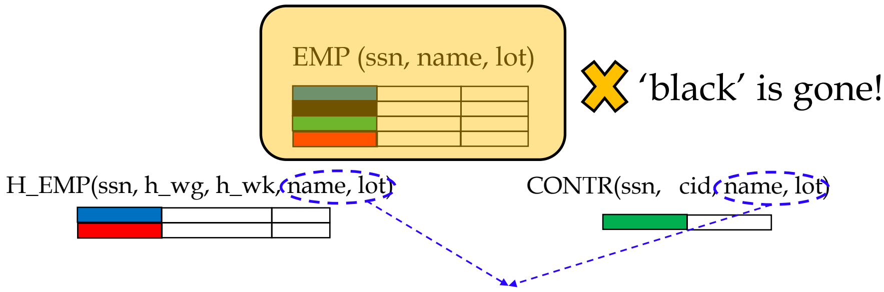
- How many times do we record an employee?
- What to do on deletions?
- How to retrieve ***all*** info about an employee?

32

Translating ISA Hierarchies to Relations

- Alternatively:
  - Just create 2 relations "Hourly_Emps" and "Contract_Emps"
    - Each employee **must be** in one of these two subclasses

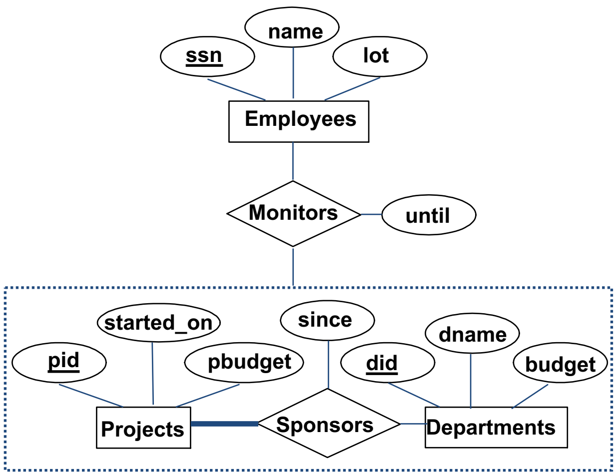




33

Translating Aggregations

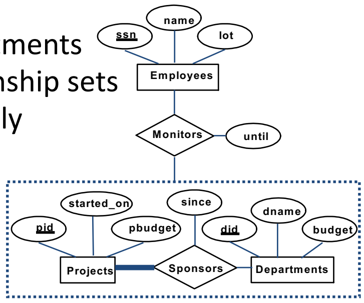
- Consider the following example:



34

Translating Aggregations

- Standard approach:
  - The Employees, Projects and Departments entity sets and the Sponsors relationship sets are translated as described previously



  - For the Monitors relationship, we create a relation with the following attributes:
    - The key attribute of Employees (i.e., ssn)
    - The key attributes of Sponsors (i.e., did, pid)
    - The descriptive attributes of Monitors (i.e., until)

35

The Relational Model: A Summary

- A tabular representation of data
- Simple and intuitive, currently one of the most widely used
  - Object-Relational Mapping (ORM) hides the relational model
  - Non-relational NoSQL model is gaining ground
- Integrity constraints can be specified (by the DBA) based on application semantics (DBMS checks for violations)
  - Two important ICs: primary and foreign keys
  - Also: not null, unique
  - In addition, we *always* have domain constraints
- Mapping from ER to Relational is (fairly) straightforward!

36

ER to Tables - Summary of Basics

- Strong entities:
  - Key -> primary key
- (Binary) relationships:
  - Get keys from all participating entities:
    - 1:1 -> either key can be the primary key
    - 1:N -> the key of the 'N' part will be the primary key
    - M:N -> both keys will be the primary key
- Weak entities:
  - Strong key + partial key -> primary key
  - ..... ON DELETE CASCADE

37

ER to Tables - Summary of Advanced

- Total/Partial participation:
  - NOT NULL
- Ternary relationships:
  - Get keys from all; decide which one(s) -> primary Key
- Aggregation: like relationships
- ISA:
  - 3 tables (most general)
  - 2 tables ('total coverage')

38

Outline

- The Relational Model: Basic SQL
- Translating ER Diagrams to Tables and Summary
- Query Languages ✓

39

Relational Query Languages

- Query languages (QLs) allow *manipulating* and *retrieving* data from databases

- The relational model supports simple and powerful QLs:
  - Strong formal foundation based on logic
  - High amenability for effective optimizations

- Query Languages != programming languages!
  - QLs are not expected to be “Turing complete”
  - QLs are not intended to be used for complex calculations
  - QLs support easy and efficient access to large datasets

40

Formal Relational Query Languages

- There are two mathematical Query Languages which form the basis for commercial languages (e.g., SQL)
  - Relational Algebra
    - Queries are composed of operators
    - Each query describes a step-by-step procedure for computing the desired answer
    - Very useful for representing *execution plans*
  - Relational Calculus
    - Queries are subsets of first-order logic
    - Queries describe desired answers without specifying how they will be computed
    - A type of *non-procedural* (or *declarative*) formal query language

41

Next Class

Relational Algebra

42