JACOBS UNIVERSITY

The Relational Model

Ramakrishnan & Gehrke, Chapter 3

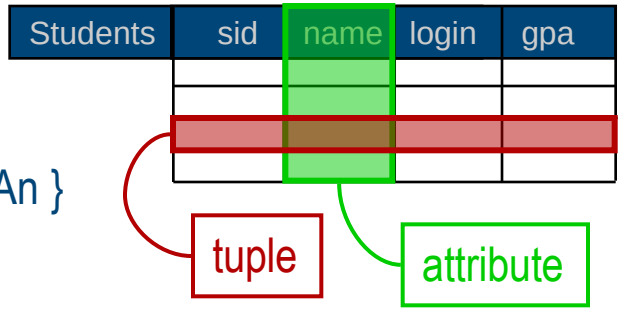
Relational Database: Definitions

JACOBS UNIVERSITY

- Technically: Relation made up of 2 parts:
  - Schema: specifies name of relation, plus name and type of each column
    - Ex: Students(sid: string, name: string, login: string, gpa: real)
  - Instance: a table, with rows and columns
    - *# rows = cardinality, # fields = degree / arity*

does not change often

changes all the time



- Mathematically:
  - Let A1, …, An (n>0) be value sets, called attribute domains
  - relation $R \subseteq A1 \times \ldots \times An = \{ (a1,\ldots,an) \mid a1 \in A1, \ldots, an \in An \}$
- Can think of a relation as a set of rows or tuples
  - i.e., all rows are distinct = no duplicates (hmm…)
  - atomic attribute types only – no fancies like sets, trees, …
- Relational database: a set of relations

Example Instance of Students Relation

JACOBS UNIVERSITY

Sid	Name	Login	Gpa
53666	Jones	jones@cs	3.4
53688	Smith	smith@eecs	3.2
53650	Smith	smith@math	3.8

- Cardinality = 3, degree = 4, all rows distinct
- Do all *columns* in a relation instance have to be distinct?

Querying Relational Databases

JACOBS UNIVERSITY

- A major strength of the relational model: simple, powerful *querying* of data
  - Data organised in tables, query results are tables as well
  - Small set of generic operations, work on any table structure
- Query describes structure of result ("what"),
  not algorithm how this result is achieved ("how")
  - data independence, optimizability
- Queries can be written intuitively,
  and the DBMS is responsible for efficient evaluation
  - The key: precise (mathematical) semantics for relational queries
  - Allows the optimizer to extensively re-order operations,
    and still ensure that the answer does not change

SQL, Structured English Query Language

JACOBS UNIVERSITY

- "find all students with GPA less than 3.6"
  - SELECT *
    FROM Students S
    WHERE S.gpa < 3.6

sid	name	login	gpa
53666	Jones	jones@cs	3.4
53688	Smith	smith@eecs	3.2
~~53650~~	~~Smith~~	~~smith@math~~	~~3.8~~

- To find just names and logins, replace the first line:
  - SELECT S.name, S.login

~~sid~~	name	login	~~gpa~~
53666	Jones	jones@cs	3.4
53688	Smith	smith@eecs	3.2

name	login
Jones	jones@cs
Smith	smith@eecs

SQL Joins: Querying Multiple Relations

JACOBS UNIVERSITY

- 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 Students and Enrolled:

```
sid   name  login       gpa
---------------------------
53666 Jones jones@cs    3.4
53688 Smith smith@eecs  3.2
53650 Smith smith@math  3.8
```

```
sid   cid          grade
----------------------
53831 Carnatic101  C
53831 Reggae203    B
53666 Topology112  A
53688 History105   B
```

- we get:

```
S.name   E.cid
--------------
Jones Topology112
```

DML: Adding and Deleting Tuples

JACOBS UNIVERSITY

- insert a single tuple:
  - INSERT INTO Students(sid, name, login, gpa)
    VALUES (53688, ‘Smith’, ‘smith@ee’, 3.2)
- delete all tuples satisfying some condition:
  - DELETE FROM Students S
    WHERE S.name = ‘Smith’
- change all tuples satisfying some condition:
  - UPDATE Students S
    SET gpa = 3.0
    WHERE S.name = ‘Smith’

Integrity Constraints

JACOBS UNIVERSITY

- Integrity constraint = 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

JACOBS UNIVERSITY

- 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 → superkey
  - If >1 key for relation,
    one of the keys is chosen (by DBA) to be primary key
- Example:
  - sid key for Students (what about name?)
  - The set {sid, gpa} is a superkey

Primary and Candidate Keys in SQL

JACOBS UNIVERSITY

- 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

JACOBS UNIVERSITY

- *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'
- Example: 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?

Enforcing Referential Integrity

JACOBS UNIVERSITY

- 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

JACOBS UNIVERSITY

- 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 )
```

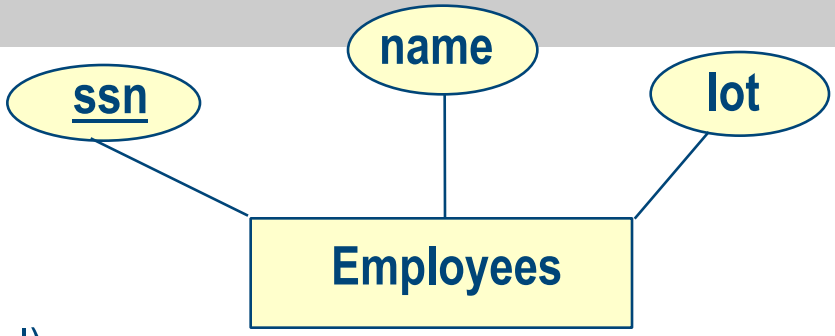
treat corresponding Enrolled tuple when Students (!) tuple is deleted

Where do ICs Come From?

JACOBS UNIVERSITY

- based upon the semantics of the real-world enterprise that is being described in the database relations
- can check a database instance to see if an IC is violated, but 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

Logical DB Design: ER to Relational



- Entity sets to tables:
  - ER attribute → table attribute
    (can do that because ER constrained
    to simple types, same as in relational model)
  - Declare key attribute “Primary key”

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

- Best practice (not followed by book):
  Add “abstract” identifying key attribute
  - No further semantics
  - System generated
  - use only this as primary key & for referencing

```
CREATE TABLE Employees
 ( sid INTEGER,
   ssn CHAR(11) UNIQUE,
   …,
   PRIMARY KEY (sid) )
```

Relationship Sets to Tables

JACOBS UNIVERSITY

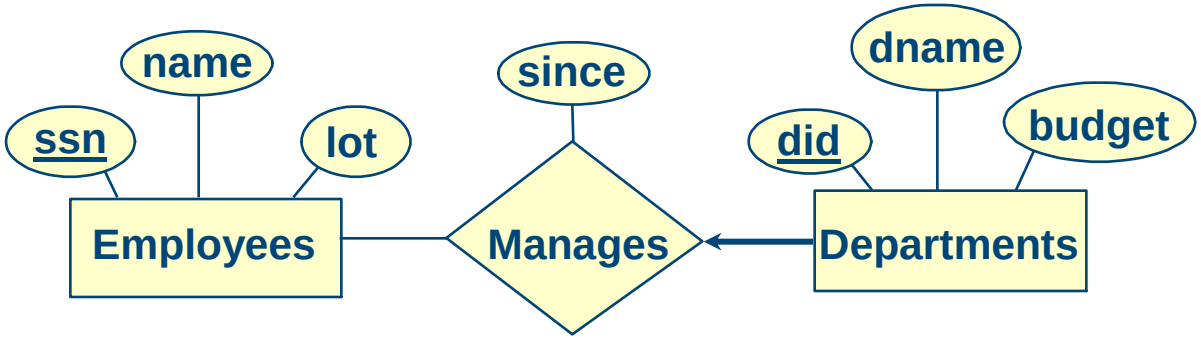
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys)
    - *a superkey for the relation*
  - All descriptive attributes

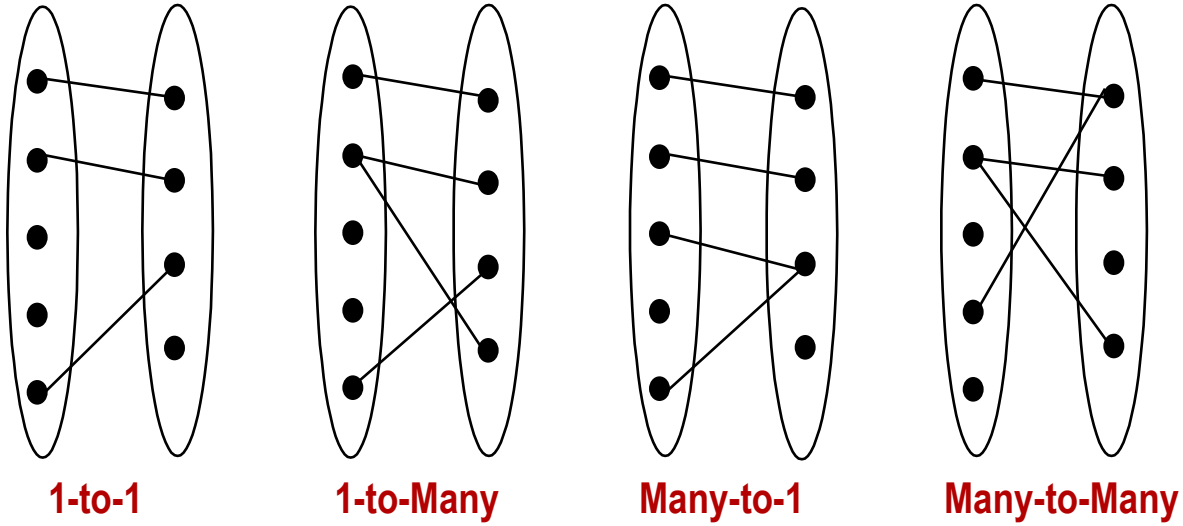
```
CREATE TABLE Works_In
( ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
       REFERENCES Employees,
  FOREIGN KEY (did)
       REFERENCES Departments )
```

Review: Key Constraints

JACOBS UNIVERSITY

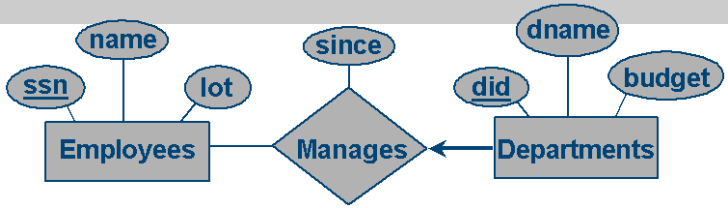
- Each dept has at most one manager, according to the key constraint on Manages





Translation to relational model? …see next!

ER Diagrams with Key Constraints



- Map relationship to table:
  - did key now!
  - Separate tables for Employees 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 )
```

- each department has unique manager

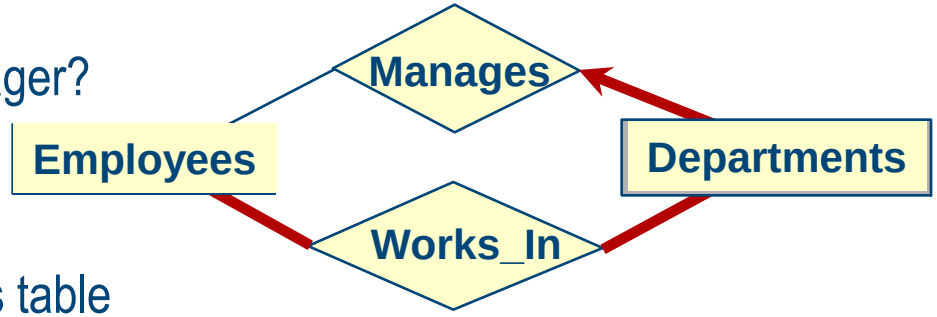
  → can combine Manages and Departments

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

Participation Constraints in SQL

JACOBS UNIVERSITY

- Review: Participation Constraints
  - Does every department have a manager?
    → participation constraint
  - Every did value in Departments table must appear in a row of the Manages table (with non-null ssn value!)



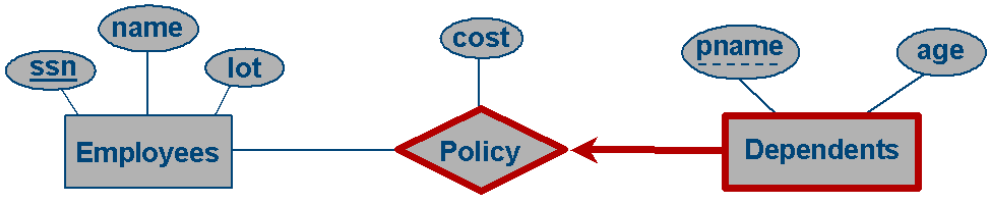
- can capture participation constraints involving one entity set in a binary relationship
  - but little else (w/o CHECK constraints)

```
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 )
```

Translating Weak Entity Sets

JACOBS UNIVERSITY

- Review: weak entity:
  identifiable uniquely only by *owner* entity
  - one-to-many relationship set
    (1 owner, many weak entities)
  - Weak entity:
    total participation in identifying relationship set



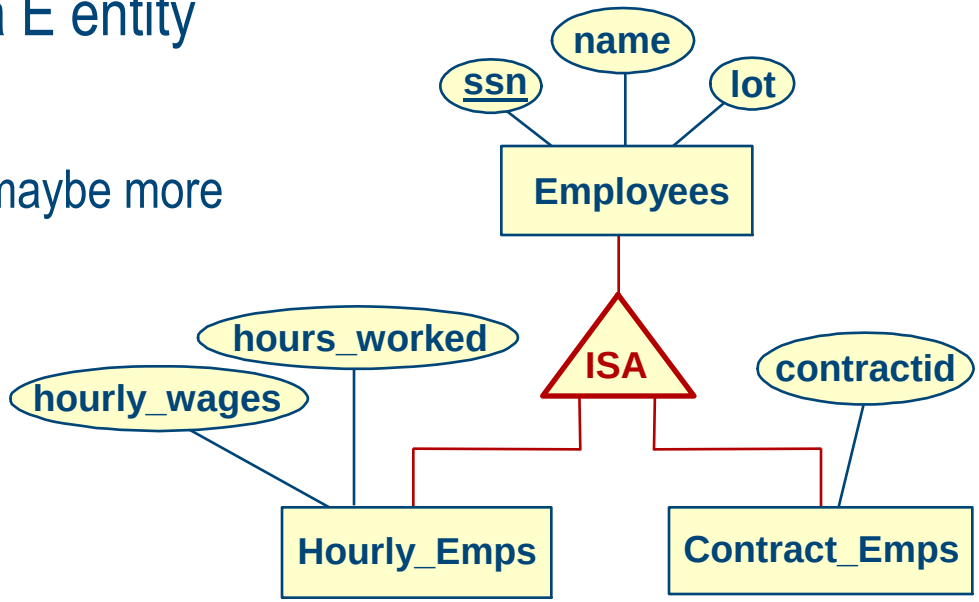
- Weak entity set & identifying relationship set
  → single table
- When owner entity is deleted:
  delete all owned weak entities

```
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 )
```

Review: ISA Hierarchies

JACOBS UNIVERSITY

- H ISA E: every H entity is also a E entity ("H inherits from E")
  - H attributes = E attributes + plus maybe more
  - H subclass, E superclass
- Mapping to Relations
  - Several choices
  - Constraints determine



Translating ISA Hierarchies to Relations

JACOBS UNIVERSITY

- General approach: separate relation per entity set
  → 3 relations: Employees, Hourly_Emps, Contract_Emps
  - Every employee recorded in Employees
  - For hourly emps, extra info recorded in Hourly_Emps (hourly_wages, hours_worked, ssn)
  - must delete Hourly_Emps tuple if referenced Employees tuple is deleted
  - Queries involving all employees easy,
    those involving just Hourly_Emps require a join to get some attributes
- Alternative: relations only entity sets with instances
  → 2 relations: Hourly_Emps, Contract_Emps
  - Hourly_Emps: ssn, name, lot, hourly_wages, hours_worked
  - Each employee must be in one of these two subclasses

Views

JACOBS UNIVERSITY

- view is just a relation, but we store definition rather than a set of tuples
  - 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 dropped using DROP VIEW
  - DROP TABLE if there's a view on the table? → options

Views and Security

JACOBS UNIVERSITY

- Views useful for personalized information (or a summary), while hiding details in underlying relation(s)
  - Given YoungStudents, but not Students or Enrolled, we can find students who are enrolled
  - …but not the cid's of the courses they are enrolled in