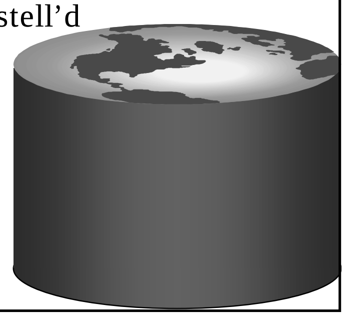
# The Relational Model

**15-415, Spring 2003, Lecture 3**

**R & G, Chap. 3**

Mine eye hath play'd the painter and hath stell'd
Thy beauty's form in table of my heart.

Shakespeare, Sonnet XXIV



# Admin

- **Next assignments: HELP!**
  – Recitations EVERY week
    (Not mandatory, but juicy! Attend!)
  – TA office hours
  – E-mail newsgroup
- **Next 2 assignments: Programming in C**
  – The longest ones ("front-loaded" semester!)
  – Read carefully the web directions
  – Ask TAs, attend recitations
  – START EARLY!!!!!

## Why Study the Relational Model?

- **Most widely used model.**
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- **"Legacy systems" in older models**
  - e.g., IBM's IMS
- **Object-oriented concepts have recently merged in**
  - *object-relational model*
    - Informix, IBM DB2, Oracle 8i
    - Early work done in POSTGRES research project at Berkeley

## Relational Database: Definitions

- *Relational database:* a set of *relations*.
- *Relation:* made up of 2 parts:
  - *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)
  - *Instance* : a *table*, with rows and columns.
    - #rows = *cardinality*
    - #fields = *degree / arity*
- Can think of a relation as a *set* of rows or *tuples*.
  - i.e., all rows are distinct

## Ex: Instance of Students Relation

| 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 |

- Cardinality = 3, arity = 5 , all rows distinct
- Do all values in each column of a relation instance have to be distinct?

## SQL - A language for Relational DBs

- **SQL* (a.k.a. "Sequel"), standard language**
- **Data Definition Language (DDL)**
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- **Data Manipulation Language (DML)**
  - Specify *queries* to find tuples that satisfy criteria
  - add, modify, remove tuples

* Structured Query Language

## SQL Overview

```
• CREATE TABLE <name> ( <field> <domain>, … )

• INSERT INTO <name> (<field names>)
       VALUES (<field values>)

• DELETE FROM <name>
        WHERE <condition>

• UPDATE <name>
     SET <field name> = <value>
   WHERE <condition>

• SELECT <fields>
    FROM <name>
   WHERE <condition>
```

## Creating Relations in SQL

- **Creates the Students relation.**
  - Note: 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 FLOAT)
```

## Table Creation (continued)

- **Another example: the Enrolled table holds information about courses students take.**

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

## Adding and Deleting Tuples

- **Can insert a single tuple using:**

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES  (‘53688’, ‘Smith’, ‘smith@cs’, 18, 3.2)
```

- **Can delete all tuples satisfying some condition (e.g., name = Smith):**

```
DELETE
  FROM Students S
 WHERE S.name = ‘Smith’
```

**Powerful variants of these commands are available; more later!**

# Keys

- Keys are a way to associate tuples in different relations
- Keys are one form of integrity constraint (IC)

Enrolled

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

FORIEGN 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

# Primary Keys

- **A set of fields is a *<u>superkey</u>* if:**
  - No two distinct tuples can have same values in all key fields
- **A set of fields is a *<u>key</u>* for a relation if :**
  - It is a superkey
  - No subset of the fields is a superkey
- **what if >1 key for a relation?**
  - one of the keys is chosen (by DBA) to be the ***primary key*.** Other keys are called ***candidate*** keys.
- **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*.**

- 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))
```

"Students can take only one course, and no two students in a course receive the same 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 the primary key of the other relation.
  - Like a `logical pointer'.

- **If all foreign key constraints are enforced, *referential integrity* is achieved (i.e., no dangling references.)**

# Foreign Keys in SQL

Example: Only students listed in the Students relation should be allowed to enroll for courses.

- *sid* is a foreign key referring to Students:

```
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 |

# Enforcing Referential Integrity

- **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 issues arise if primary key of Students tuple is updated.**

## Integrity Constraints (ICs)

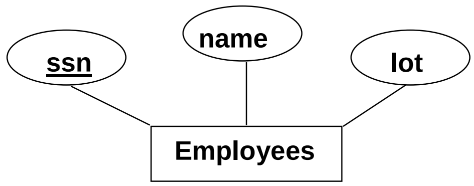
- **IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.***
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- **A *legal* instance of a relation is one that satisfies all specified ICs.**
  - DBMS should not allow illegal instances.
- **If the DBMS checks ICs, stored data is more faithful to real-world meaning.**
  - Avoids data entry errors, too!

## Where do ICs Come From?

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




| ssn | name | lot |
|---|---|---|
| 123-22-3666 | Attishoo | 48 |
| 231-31-5368 | Smiley | 22 |
| 131-24-3650 | Smethurst | 35 |

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

## Relationship Sets to Tables

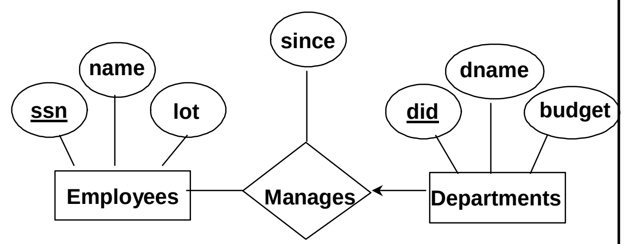
- In translating a many-to-many relationship set to a relation, attributes of the relation must include:
  1) Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
  2) All descriptive attributes.

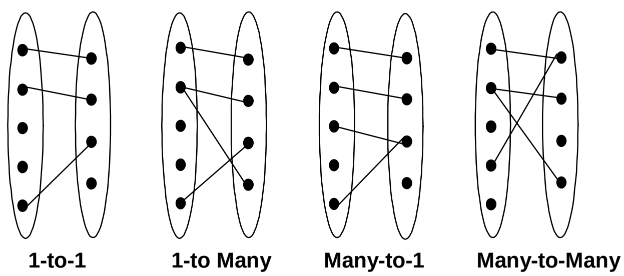
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn)
    REFERENCES Employees,
 FOREIGN KEY (did)
  REFERENCES Departments)
```

| ssn | did | since |
|---|---|---|
| 123-22-3666 | 51 | 1/1/91 |
| 123-22-3666 | 56 | 3/3/93 |
| 231-31-5368 | 51 | 2/2/92 |

# Review: Key Constraints

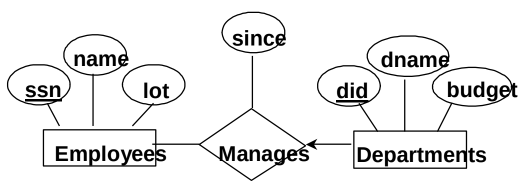
- **Each dept has at most one manager, according to the *key constraint* on Manages.**




*Translation to relational model?*

# Translating ER with Key Constraints




- Since each department has a unique manager, we could instead combine Manages and Departments.

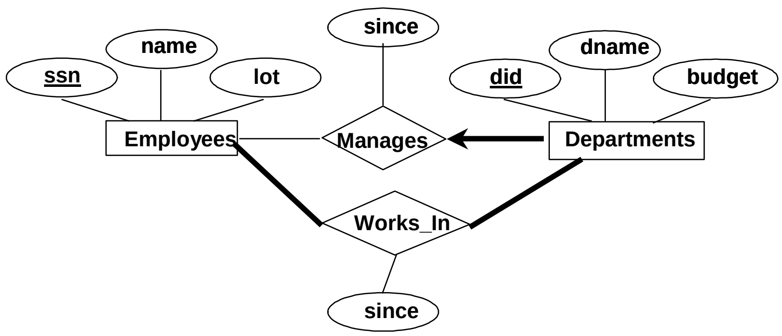
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
   FOREIGN KEY (did)
REFERENCES Departments)
```

Vs.

```
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: Participation Constraints

- **Does every department have a manager?**
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



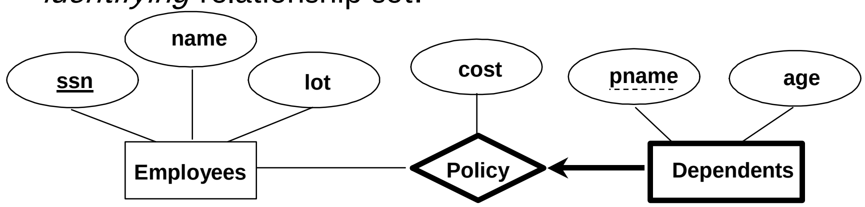

## Participation Constraints in SQL

- **We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to 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)
```

## 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.

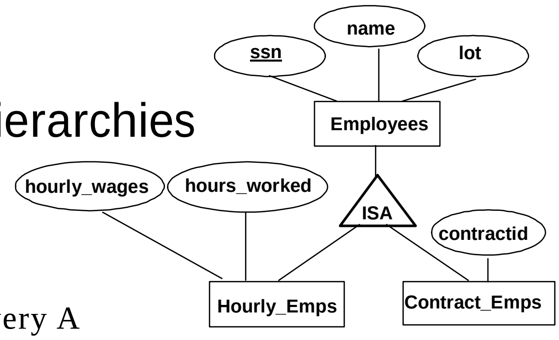



## Translating Weak Entity Sets

- **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)
```

## Review: ISA Hierarchies




- As in C++, or other PLs, attributes are inherited.
- If we declare A **ISA** B, every A entity is also considered to be a B entity.

- ***Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)**
- ***Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)***

## Translating ISA Hierarchies to Relations

- *General approach:*
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *<u>ssn</u>)*; 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: Just Hourly_Emps and Contract_Emps.**
  - *Hourly_Emps*: *<u>ssn</u>*, *name, lot, hourly_wages, hours_worked.*
  - Each employee must be in one of these two subclasses*.*

# Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used
  - Object-relational variant 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
  - In addition, we *always* have domain constraints.
- Mapping from ER to Relational is (fairly) straightforward.

- NEXT: FILES< STORAGE, BUFFERS, DISKS…
- READ CHAPTER 9!