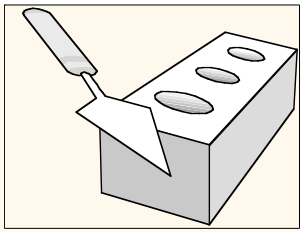
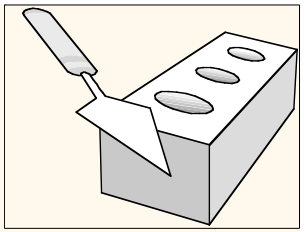


The Relational Model

Chapter 3

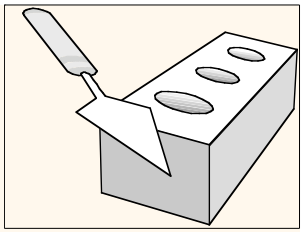
Slides modified by Rasmus Pagh for
Database Systems, Fall 2006
IT University of Copenhagen



Today

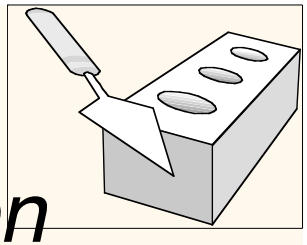
- Recap: Relations
- SQL's data definition language
- Expressing integrity constraints
- From an ER model to a relational model (case study based on RG exercise 3.16)
- Modeling tools: Demonstration of Rational Data Architect.

Relational Database: Definitions



- *Relational database:* a set of *relations*
- *Relation:* Word used in two senses:
  - *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).

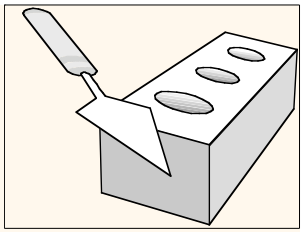
Example Instance of Students Relation



sid	name	login	age	gpa
3666	Jones	jones@cs	18	3.4
3688	Smith	smith@eecs	18	3.2
3650	Smith	smith@math	19	3.8

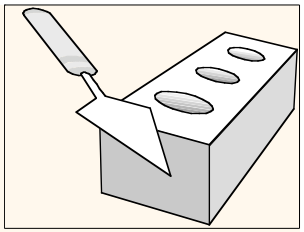
- Cardinality = 3, degree = 5, all rows distinct
- Do all columns in a relation instance have to be distinct?

Why Study the Relational Model?



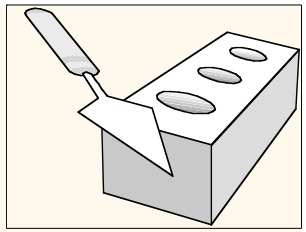
- Most widely used model.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- "Legacy systems" in older models
  - E.G., IBM's IMS
- Recent competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: *object-relational model*
    - Informix Universal Server, UniSQL, O2, Oracle, DB2

Relational Query Languages



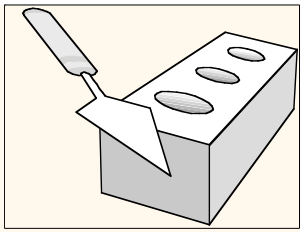
- A major strength of the relational model: supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

"The" SQL Language



- First developed by IBM (system R) in the 1970s
- Has constructs for data definition (DDL), data manipulation (DML), and queries.
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86, SQL-89 (minor revision), SQL-92 (major revision), SQL-99 (major extensions), SQL:2003 (current standard).
- Major DBMSs all have some level of standards compliance, but many differences among SQL in different DBMSs exist.
- In the course (and book) focus is on the common parts of SQL.

Creating Relations in SQL

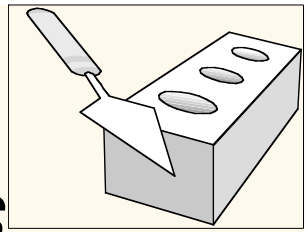


- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

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

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

Destroying and Altering Relations



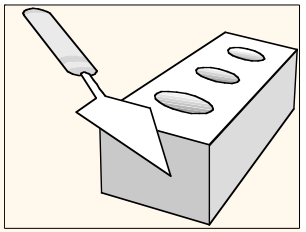
DROP TABLE Students

- Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples



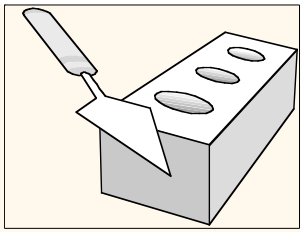
- Can insert a single tuple using:

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

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

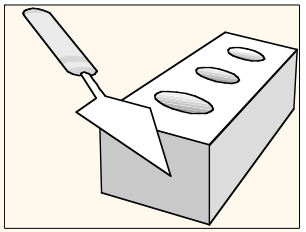
```
DELETE
FROM Students
WHERE name = 'Smith'
```

Integrity Constraints (ICs)



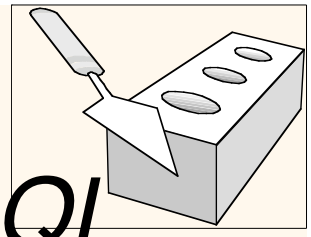
- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal 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

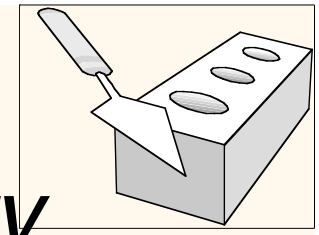


- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.
- "For a given student and course, there is a single grade." vs. "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

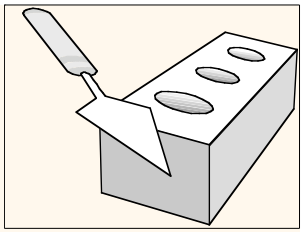
```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
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.



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 )
```

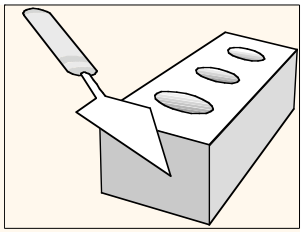
Enrolled

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

Students

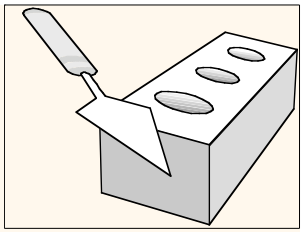
sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

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? Several possibilities:
  - 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.

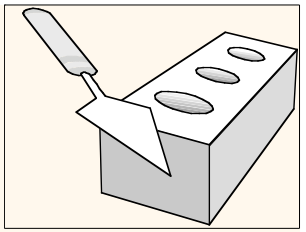
Referential Integrity in SQL



- SQL supports 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)

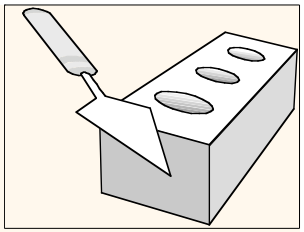
```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

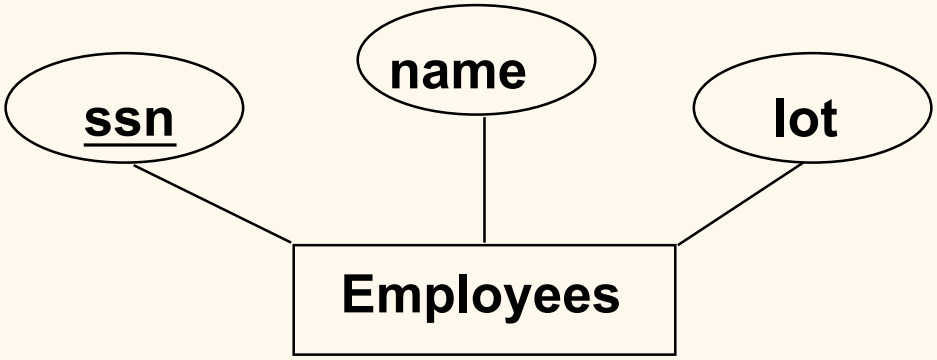


- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - 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.

Conversion: ER to Relational

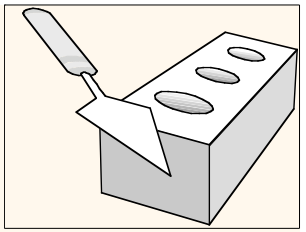


- Entity sets to tables:
  - Relation with same set of attributes.
  - Key declared as primary key.




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

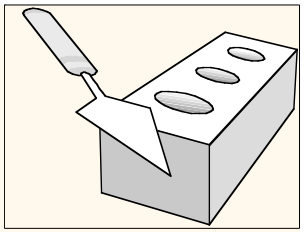
Relationship Sets to Tables



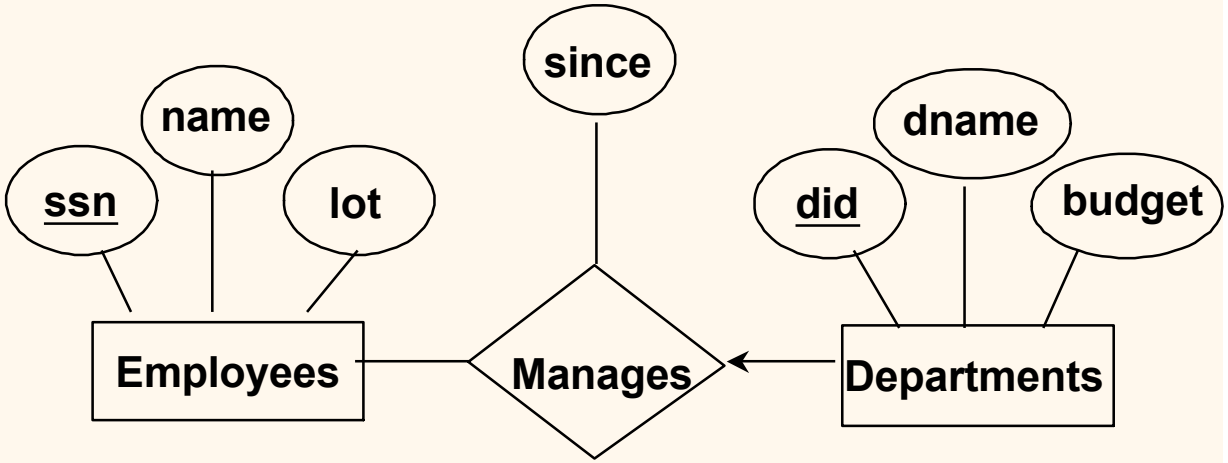
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys). It might be necessary to rename attributes if duplicates exist.
  - All descriptive attributes of the relationship set.
  - The combined set of keys of the participating entity sets is declared a key.

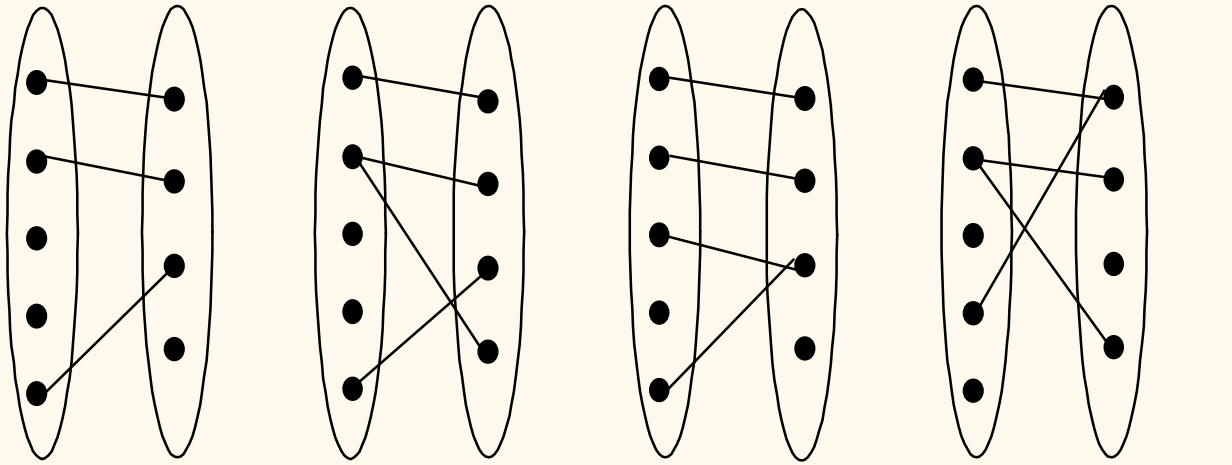
```
CREATE TABLE Works_In(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
        REFERENCES Employees,
  FOREIGN KEY (did)
        REFERENCES
Departments
)
```

Review: Key Constraints



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

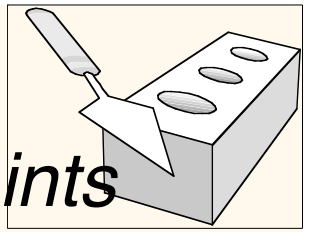




1-to-1 | 1-to Many | Many-to-1 | Many-to-Many

Translation to relational model?

Translating ER Diagrams with Key Constraints

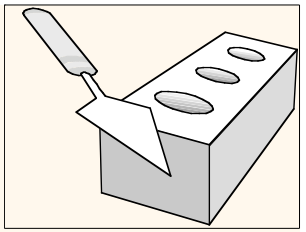


- Map relationship to a table:
  - Note that did can be used as primary key.
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.
- What about a 1-1 relationship?

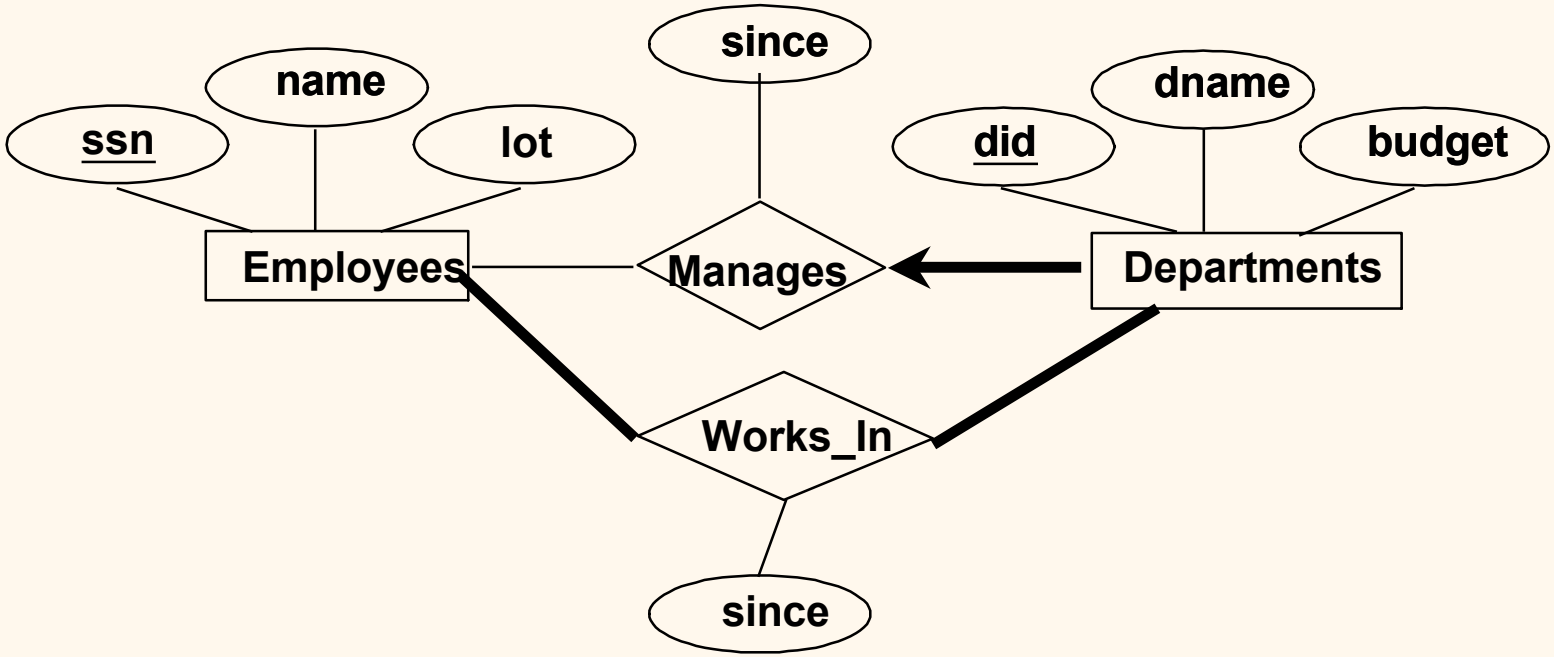
```
CREATE TABLE Manages(
   ssn CHAR(11),
   did INTEGER,
   since DATE,
   PRIMARY KEY (did),
   FOREIGN KEY (ssn) REFERENCES Employees,
   FOREIGN KEY (did) REFERENCES Departments)
```

```
CREATE TABLE Departments2(
   did INTEGER,
   dname CHAR(20),
   budget REAL,
   ssn CHAR(11),
   since DATE,
   PRIMARY KEY (did),
   FOREIGN KEY (ssn) REFERENCES Employees)
```

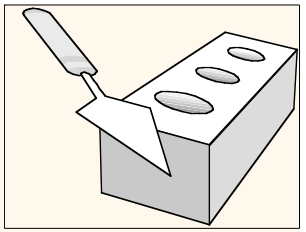
Review: Participation Constraints



- 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!)

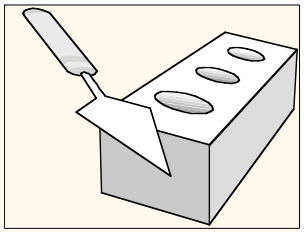


Participation Constraints in SQL



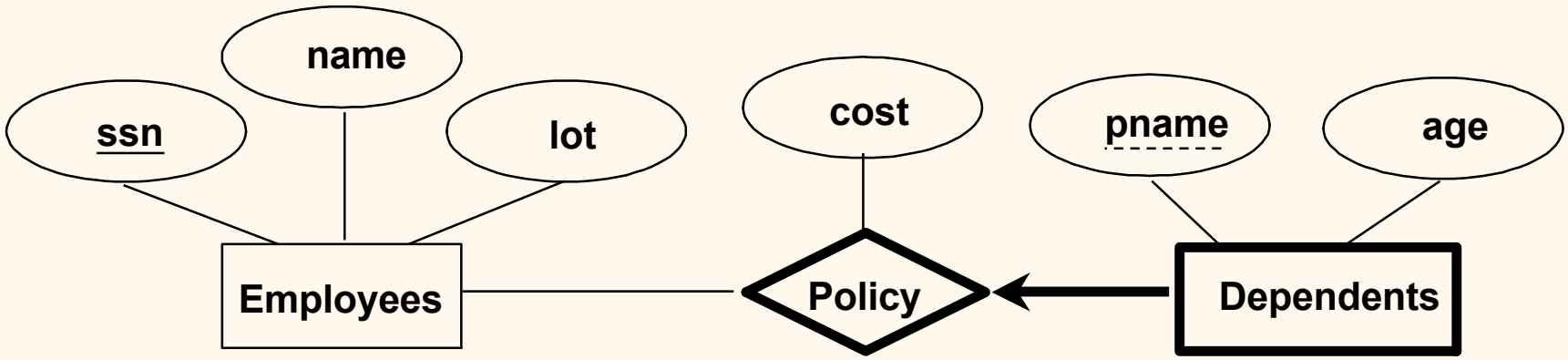
- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

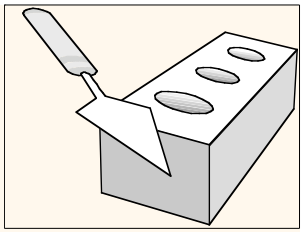
```
CREATE TABLE Deptartments2(
   did INTEGER,
   dname CHAR(20),
   budget REAL,
   ssn CHAR(11) NOT NULL,
   since DATE,
   PRIMARY KEY (did),
   FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```



Review: Weak Entities

- *Weak entities* can be identified uniquely only by considering the primary key(s) of other "*owner*" entities.
  - Owner entity sets 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 the *identifying* relationship set(s).



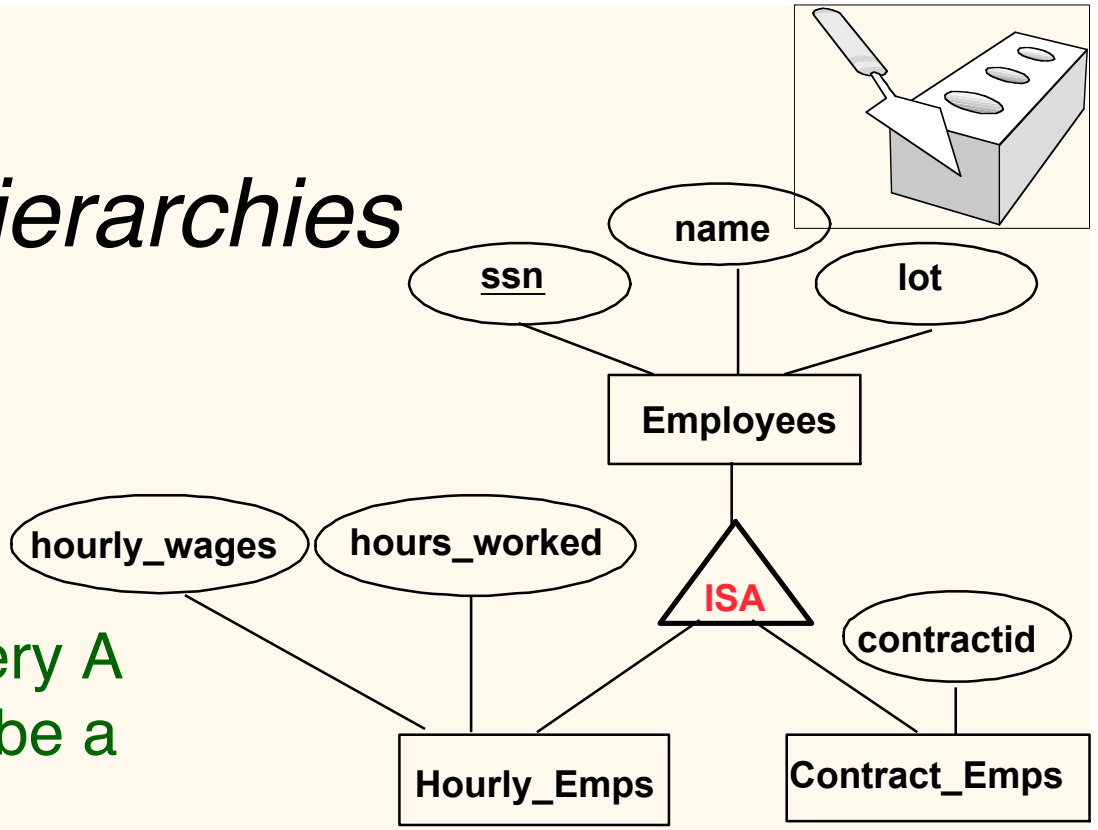


Translating Weak Entity Sets

- Weak entity set and identifying relationship set(s) are translated into a single table.
- Primary key consists of all key attributes of owner entity sets and underlined attributes of the weak entity set.

```
CREATE TABLE  Dependents2 (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

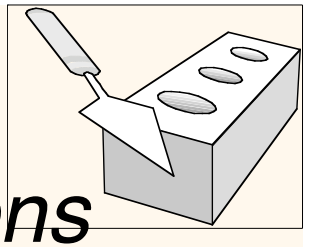
Review: ISA Hierarchies



- 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.

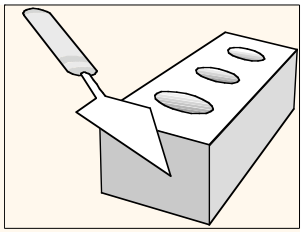
- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

Translating ISA Hierarchies to Relations



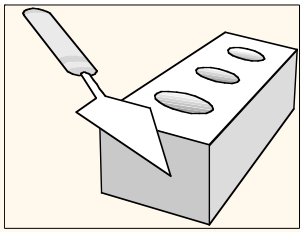
- ***General approach:***
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees.
    - For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn*)
    - must delete Hourly_Emps tuple if referenced Employees tuple is deleted.
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.

Translating ISA Hierarchies, cont.



- 1st alternative:
  Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: *ssn*, *name, lot, hourly_wages, hours_worked.*
  - Each employee must be in one of these two subclasses. (If they do not cover Employees, we need also Employees).
- 2nd alternative (not in book):
  Extend Employees' attribute set
  - Let all subclass attributes become attributes of Employees.
  - Attributes that do not apply to a given entity are given the value NULL.
- Which method to choose depends on how we are going to access the data. There are often tradeoffs between efficiency, ease of updating, and ease of querying.

Views

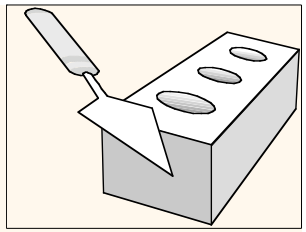


- The set of relations is not the full story on conceptual organization of relational data.
- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungStudents (name)
        AS SELECT name
        FROM Students
        WHERE age<21
```

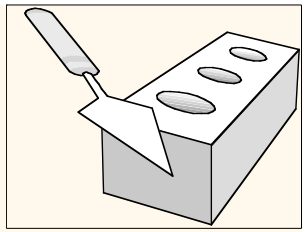
- Views can be dropped using the DROP VIEW command.

Applications of Views



- Data independence: A view may stay the same even if we alter the relation schemas. Any queries using the view can then stay unchanged.
- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students, we can find the names of young students, but no other information about them.

Most important points



- How to create and update relations in SQL.
- How to express integrity constraints in SQL:
  - Keys (primary and unique)
  - Foreign keys
- How to translate an ER model to a relational model.
  - Choices must be made in ISA hierarchies and 1-1 relationships.