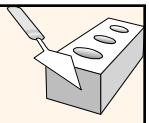
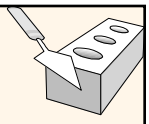


Integrity Constraints (ICs)

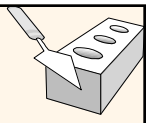
- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal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



Primary Key Constraints

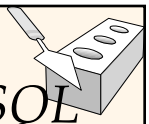
- A set of fields is a (candidate) *key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Primary Key Constraints (Cont.)



```
CREATE TABLE Students
  (sid        CHAR(20),
   name       CHAR(30),
   login      CHAR(20),
   age        INTEGER,
   gpa        REAL,
  UNIQUE (login, age),
  CONSTRAINT StudentsKey PRIMARY KEY  (sid))
```


Primary and Candidate Keys in SQL

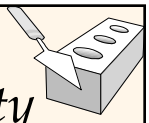


- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.
- "For a given student and course, there is a single grade." vs. "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same 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) )
```

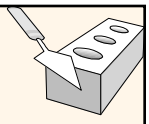
Foreign Keys, Referential Integrity



- *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 – 404 Error

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 )
```

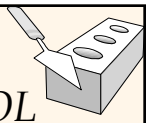
Enrolled

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

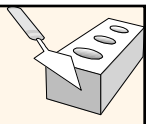
Specifying Foreign Key Constraints in SQL



```
CREATE TABLE Enrolled
  (studid     CHAR(20),
   cid        CHAR(20),
   grade      CHAR(10),
   PRIMARY KEY  (studid, cid),
   FOREIGN KEY (studid) REFERENCES Students)
```

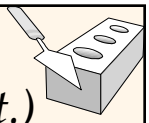
- The same attribute name is not required.

Enforcing Referential Integrity



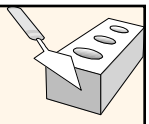
- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Disallow deletion of a Students tuple that is referred to.
  - Also delete all Enrolled tuples that refer to it.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: set an attribute to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)
- Similar if primary key of Students tuple is updated.

Enforcing Referential Integrity (Cont.)



- Deletion does not cause a violation of domain, PRIMARY KEY, or UNIQUE constraints. However, an update can cause violations, similar to an insertion (for a single table without pointers on it).
- For foreign key violations:
  - Deletions of Enrolled tuples do not violate referential integrity, but insertions could.
  - Insertions of Students tuples do not violate referential integrity, but deletions could.

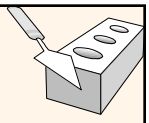
Referential Integrity in SQL



- 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 CASCADE )
```

Transactions and Constraints

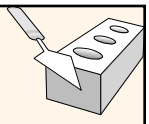


```
CREATE TABLE Students
  (sid        CHAR(20),
   name       CHAR(20),
   honors     CHAR(20) NOT NULL,
   PRIMARY KEY  (sid),
   FOREIGN KEY (honors) REFERENCES Courses (cid))

CREATE TABLE Courses
  (cid        CHAR(20),
   cname      CHAR(20),
   grader     CHAR(20) NOT NULL,
   PRIMARY KEY  (cid),
   FOREIGN KEY (grader) REFERENCES Students (sid))
```

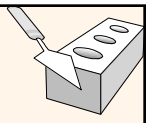
- SET CONSTRAINT Constraint_X DEFERRED

Transaction example



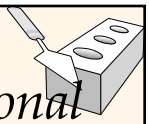
```
BEGIN TRAN T1;
CREATE TABLE Students ...;
CREATE TABLE Courses ...;
SET CONSTRAINT Constraint_X DEFERRED;
SET CONSTRAINT Constraint_Y DEFERRED;
INSERT  INTO Students ...;
...
...
UPDATE table3 ...;
COMMIT TRAN T1;
```

- https://learn.microsoft.com/en-us/sql/t-sql/language-elements/begin-transaction-transact-sql?redirectedfrom=MSDN&view=sql-server-ver16



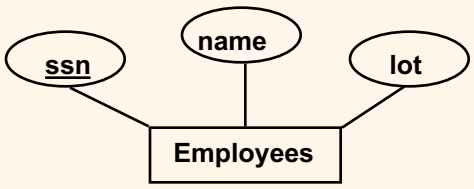
Where do ICs Come From?

- 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!



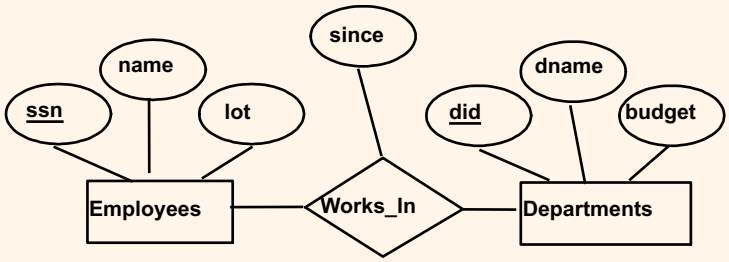
Logical DB Design: ER to Relational

- Entity sets to tables:




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

Relationship Sets to Tables

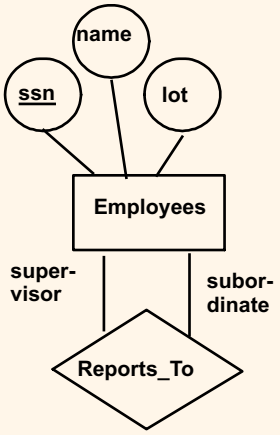


Relationship Sets to Tables (Cont.)

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

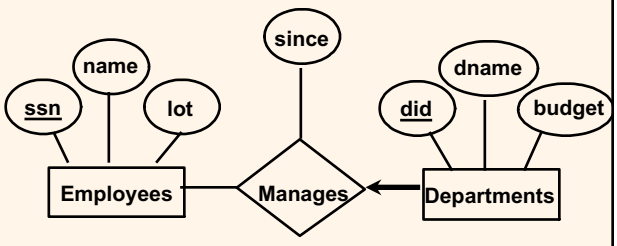
Relationship Sets to Tables (Cont.)

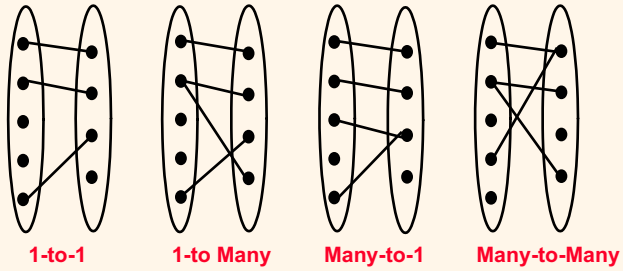



```
CREATE TABLE Reports_To (
  supervisor_ssn  CHAR(11),
  subordinate_ssn  CHAR(11),
  PRIMARY KEY (supervisor_ssn,
subordinate_ssn),
  FOREIGN KEY (supervisor_ssn)
  REFERENCES Employees (ssn),
  FOREIGN KEY (subordinate_ssn)
      REFERENCES Employees (ssn))
```


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

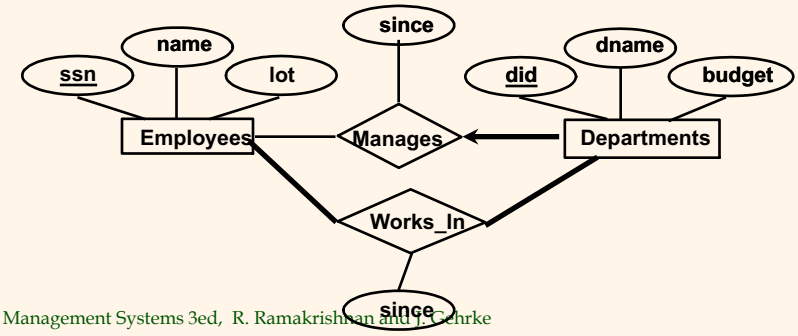
- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

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

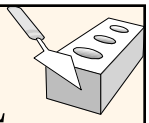
```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```


Review: 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

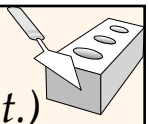


- We can capture participation constraints involving one entity set in a binary relationship.

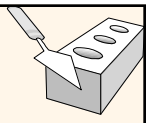
```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11) NOT NULL,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE NO ACTION)
```

Not CASCADE?

Participation Constraints in SQL (Cont.)

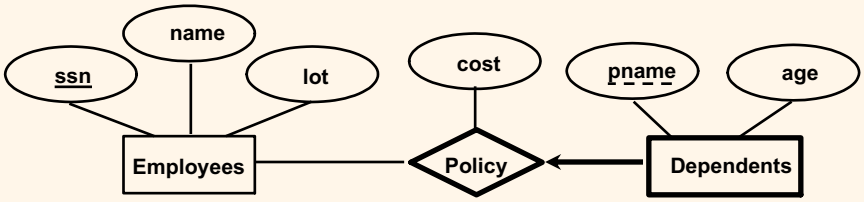


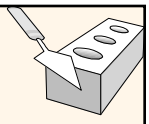
- Can we capture participation constraints in the Works_In relationship?
  - No. We need to use *table constraints* or *assertions*.
  - *Table constraints* and *assertions* can be specified using the full power of the SQL query language and are very expressive but also very expensive to check and enforce (more details in Section 5.7).



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.



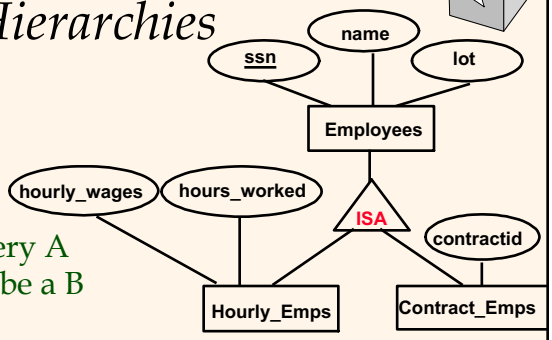


Translating Weak Entity Sets

- 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)
  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.

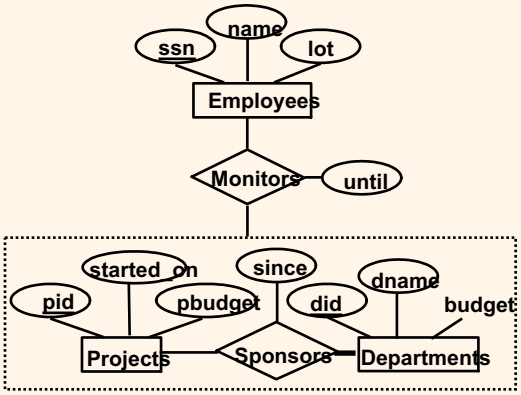
- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

They are usually expressed in SQL by using assertions.

Translating ISA Hierarchies to Relations

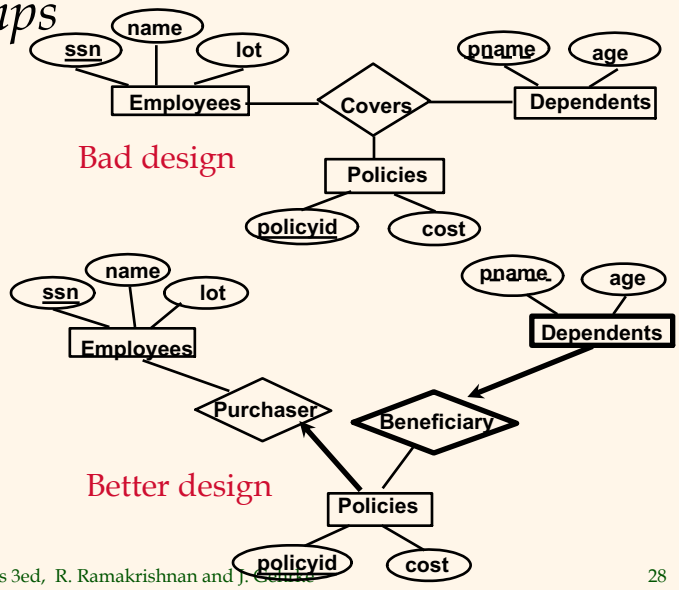
- ***General approach:***
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages, hours_worked, ssn*); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).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

Translating ER Diagrams with Aggregation

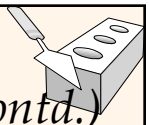


For the Monitors relationship set, we create a relation with the following attributes: ssn (Employees), did, pid (Sponsors), and the descriptive attribute of Monitors (until).

Review: Binary vs. Ternary Relationships



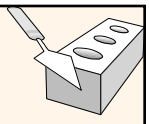
Binary vs. Ternary Relationships (Contd.)



- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.
- What if Policies is a weak entity set?

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


Views

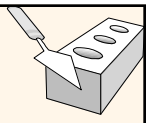


- A *view* is like a relation, but we store a *definition*, rather than a set of tuples.

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

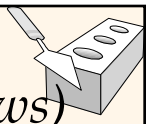
- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Views and Security



- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungActiveStudents, but not Students or Enrolled, we can find students who have enrolled, but not the *cid's* of the courses they are enrolled in.

Updates on Views (updatable views)

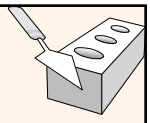


```
CREATE VIEW GoodStudents1 (sid, gpa)
        AS SELECT S.sid, S.gpa
           FROM Students S
           WHERE S.gpa > 3.0

CREATE VIEW GoodStudents2 (sname, gpa)
        AS SELECT S.name, S.gpa
           FROM Students S
           WHERE S.gpa > 3.0
```

http://msdn.microsoft.com/en-us/library/ms187956.aspx

Relational Model: Summary



- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- 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.
- Powerful and natural query languages exist.
- Rules to translate ER to relational model