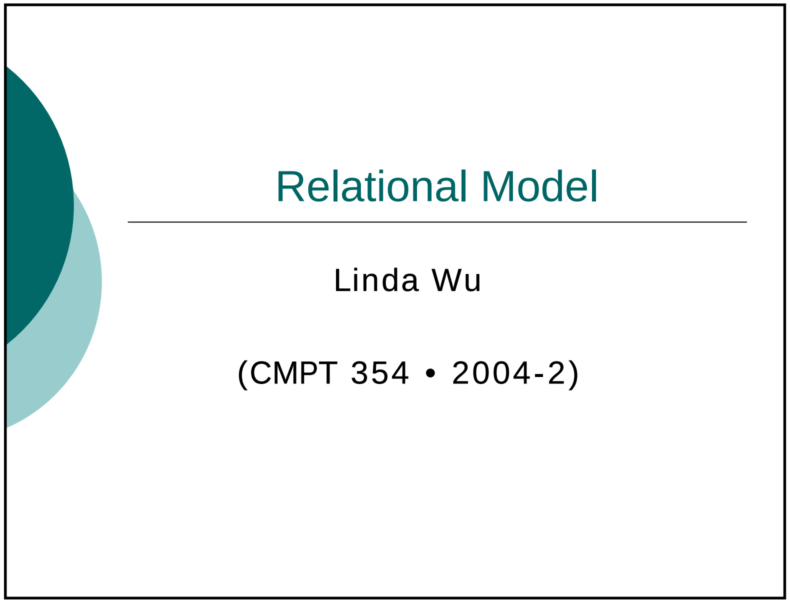
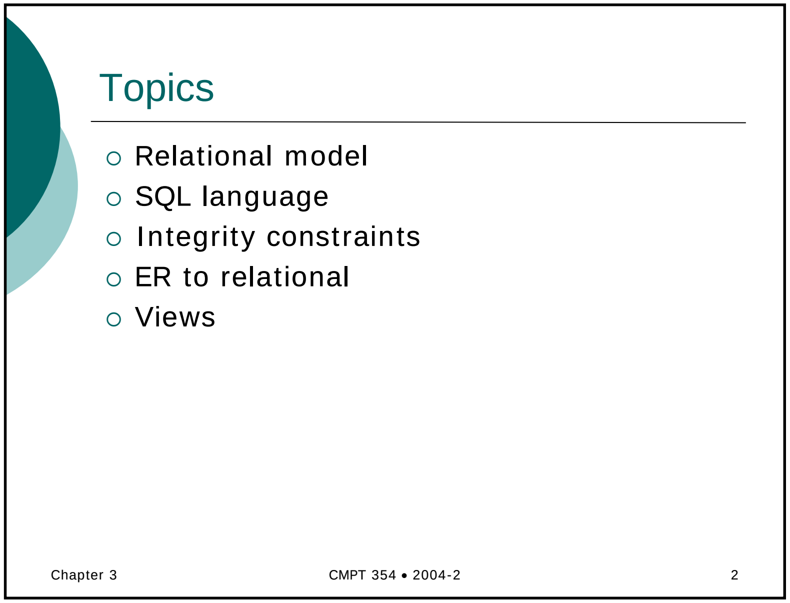


Relational Model

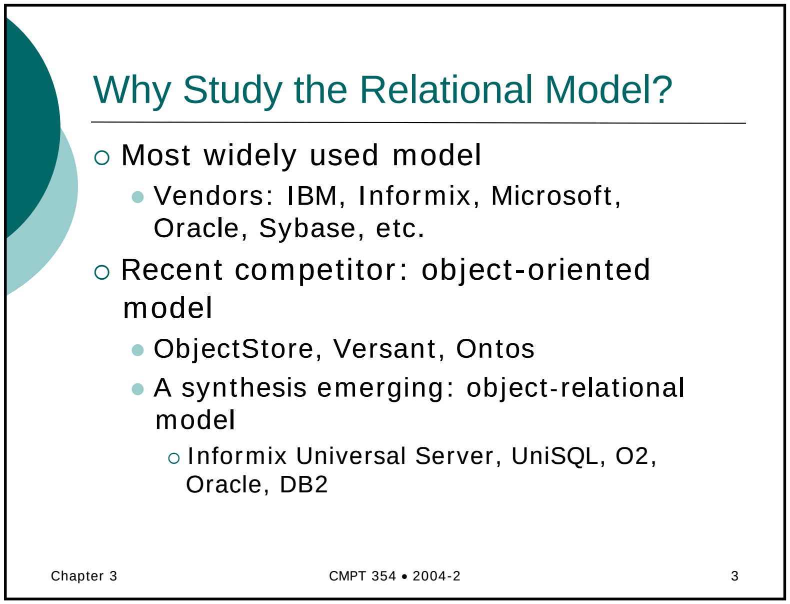
Linda Wu

(CMPT 354 • 2004-2)



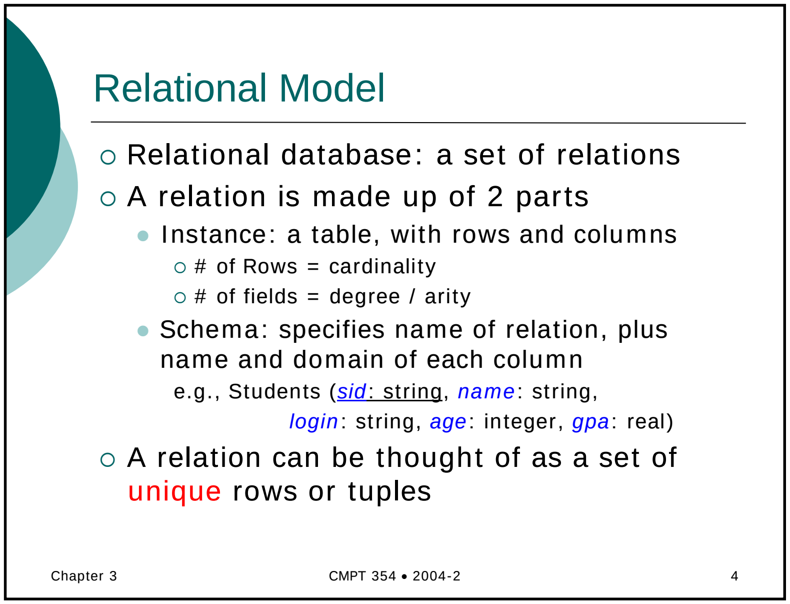
Topics

- Relational model
- SQL language
- Integrity constraints
- ER to relational
- Views



Why Study the Relational Model?

- Most widely used model
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Recent competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: object-relational model
    - Informix Universal Server, UniSQL, O2, Oracle, DB2



Relational Model

- Relational database: a set of relations
- A relation is made up of 2 parts
  - Instance: a table, with rows and columns
    - # of Rows = cardinality
    - # of fields = degree / arity
  - Schema: specifies name of relation, plus name and domain of each column

    e.g., Students (*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
- A relation can be thought of as a set of unique rows or tuples

Relational Model (Cont.)

sid	*name*	*login*	*age*	*gpa*
S0001	Jones	jones@cs	18	3.4
S0002	Smith	smith@ee	19	3.2
S0003	Smith	smith@math	18	3.8
S0004	Mary	mary@music	14	1.8

Cardinality = 4, degree = 5, all rows distinct

Relational Model (Cont.)

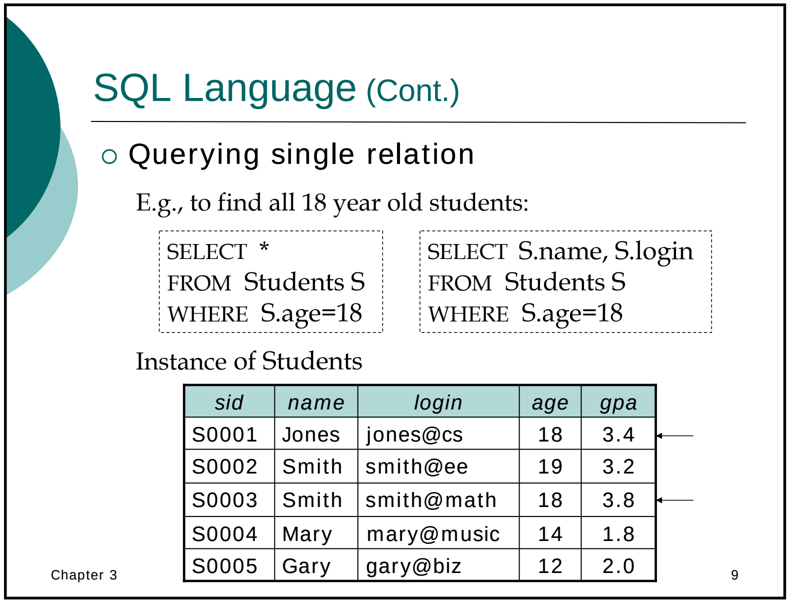
- A major strength of relational model
  - Supports simple, powerful querying of data
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation
  - The key: precise semantics for relational queries
  - Allows the optimizer to extensively re-order operations, and still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

SQL Language

- Query
  - A question about the data
  - Its answer consists of a relation containing the result
- Query language
  - A specialized language for writing query
- SQL (Structured Query Language)
  - The standard language for creating, manipulating, and querying data in a relational DBMS
  - Originally developed by IBM (system-R) in the 1970s

SQL Language (Cont.)

- Need for a standard since SQL products are offered by many vendors
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extension, current standard)



SQL Language (Cont.)

- Querying single relation

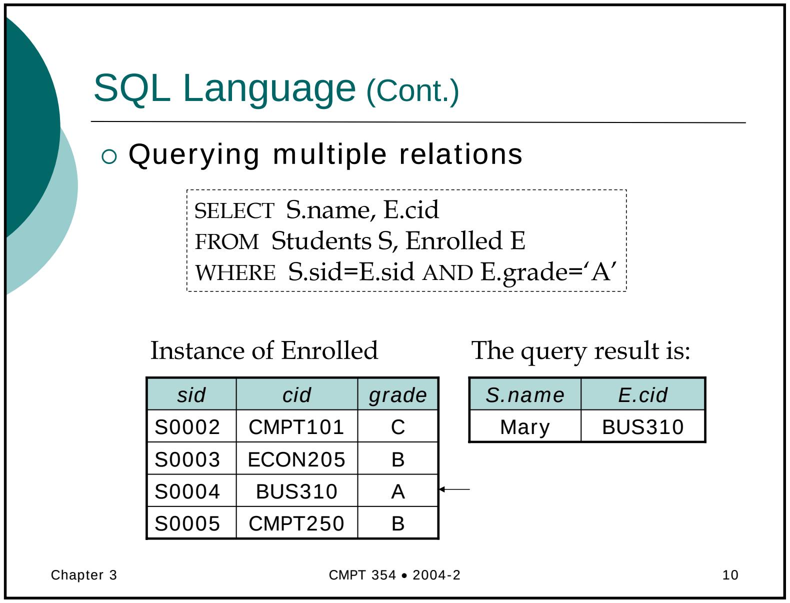
E.g., to find all 18 year old students:

```
SELECT  *
FROM  Students S
WHERE  S.age=18
```

```
SELECT  S.name, S.login
FROM  Students S
WHERE  S.age=18
```

Instance of Students

sid	*name*	*login*	*age*	*gpa*
S0001	Jones	jones@cs	18	3.4
S0002	Smith	smith@ee	19	3.2
S0003	Smith	smith@math	18	3.8
S0004	Mary	mary@music	14	1.8
S0005	Gary	gary@biz	12	2.0



SQL Language (Cont.)

- Querying multiple relations

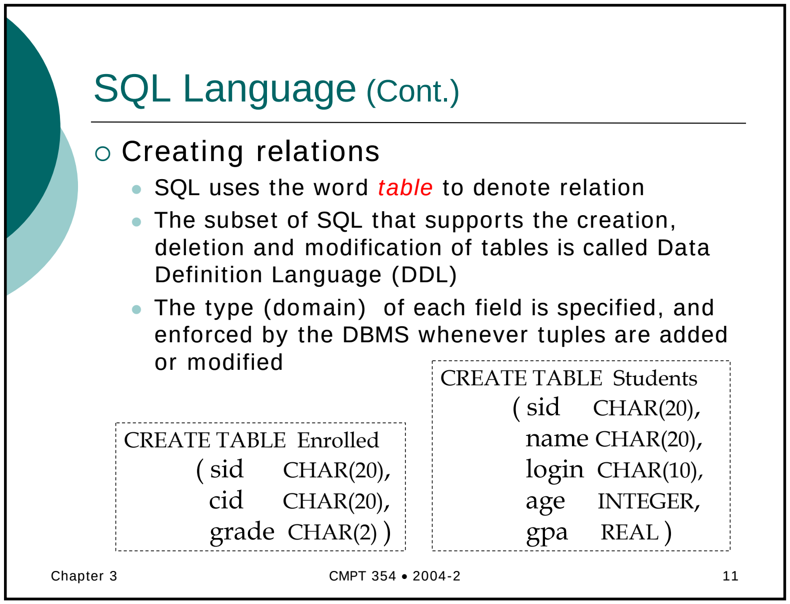
```
SELECT  S.name, E.cid
FROM  Students S, Enrolled E
WHERE  S.sid=E.sid AND E.grade='A'
```

Instance of Enrolled

sid	*cid*	*grade*
S0002	CMPT101	C
S0003	ECON205	B
S0004	BUS310	A
S0005	CMPT250	B

The query result is:

S.name	*E.cid*
Mary	BUS310

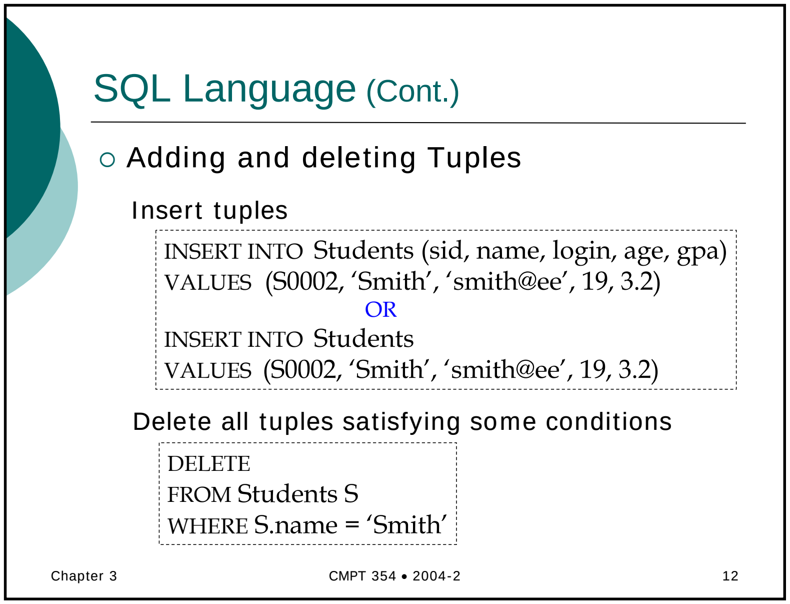


SQL Language (Cont.)

- Creating relations
  - SQL uses the word *table* to denote relation
  - The subset of SQL that supports the creation, deletion and modification of tables is called Data Definition Language (DDL)
  - 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     REAL )
```

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



SQL Language (Cont.)

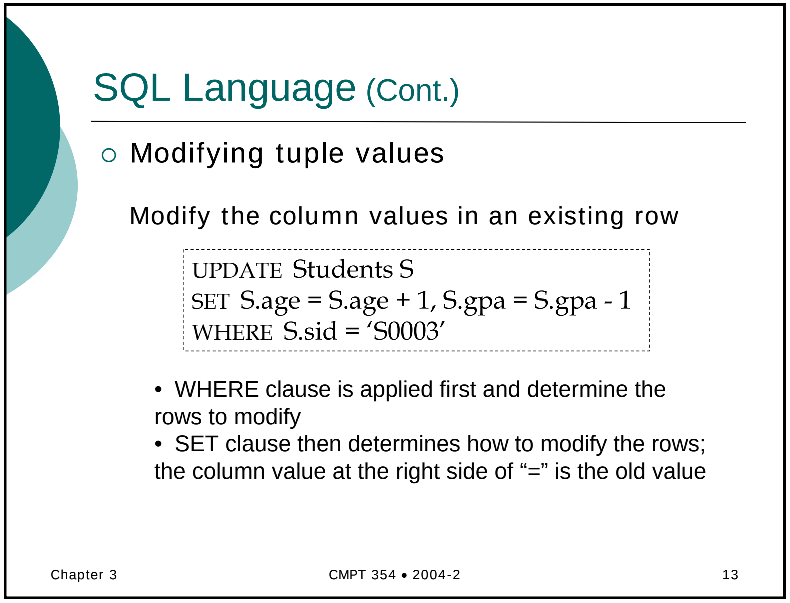
- Adding and deleting Tuples

Insert tuples

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES  (S0002, 'Smith', 'smith@ee', 19, 3.2)
                    OR
INSERT INTO  Students
VALUES  (S0002, 'Smith', 'smith@ee', 19, 3.2)
```

Delete all tuples satisfying some conditions

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



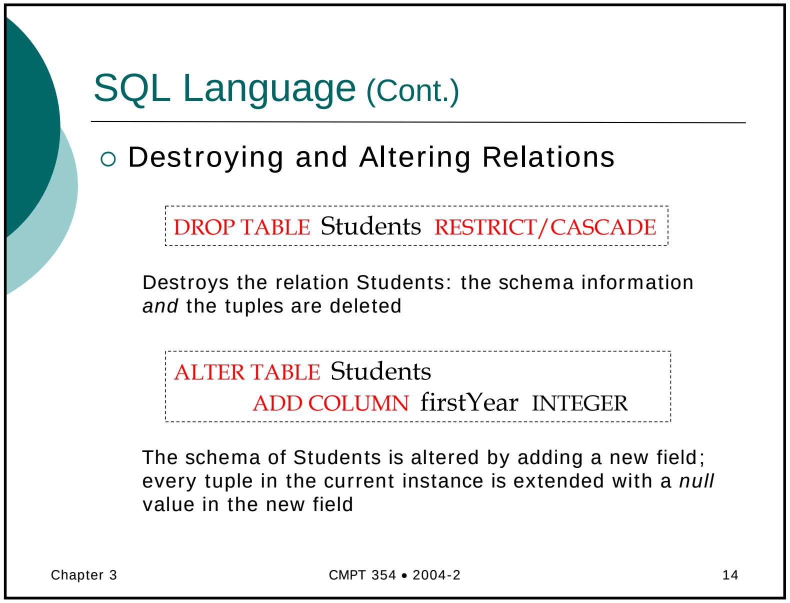
SQL Language (Cont.)

○ Modifying tuple values

Modify the column values in an existing row

```
UPDATE Students S
SET S.age = S.age + 1, S.gpa = S.gpa - 1
WHERE S.sid = 'S0003'
```

- WHERE clause is applied first and determine the rows to modify
- SET clause then determines how to modify the rows; the column value at the right side of "=" is the old value



SQL Language (Cont.)

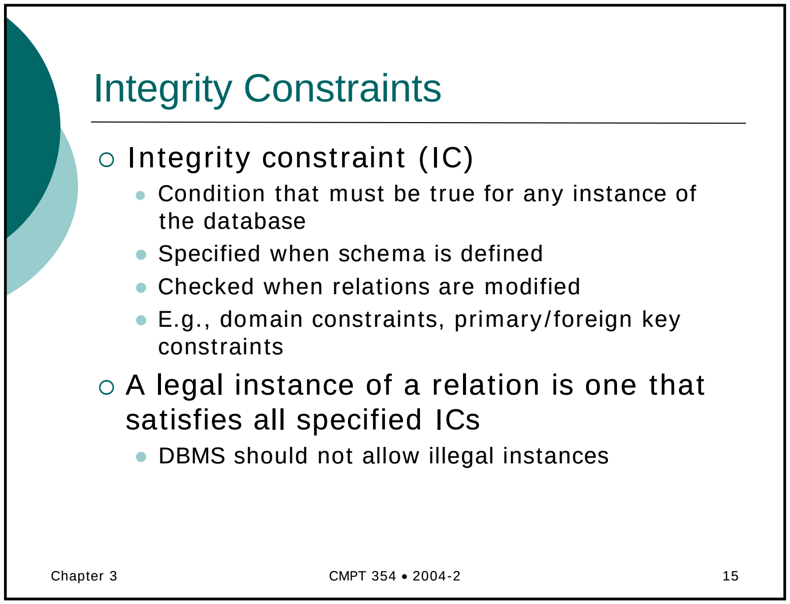
○ Destroying and Altering Relations

```
DROP TABLE Students RESTRICT/CASCADE
```

Destroys the relation Students: the schema information *and* the tuples are deleted

```
ALTER TABLE Students
          ADD COLUMN firstYear INTEGER
```

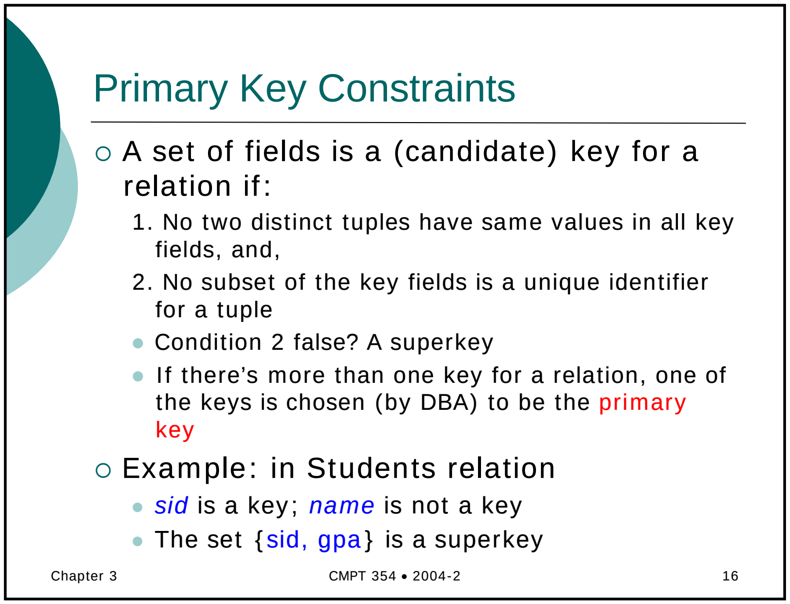
The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field



Integrity Constraints

○ Integrity constraint (IC)
- Condition that must be true for any instance of the database
- Specified when schema is defined
- Checked when relations are modified
- E.g., domain constraints, primary/foreign key constraints

○ A legal instance of a relation is one that satisfies all specified ICs
- DBMS should not allow illegal instances



Primary Key Constraints

○ A set of fields is a (candidate) key for a relation if:

1. No two distinct tuples have same values in all key fields, and,
2. No subset of the key fields is a unique identifier for a tuple

- Condition 2 false? A superkey
- If there's more than one key for a relation, one of the keys is chosen (by DBA) to be the primary key

○ Example: in Students relation
- *sid* is a key; *name* is not a key
- The set {sid, gpa} is a superkey

Primary Key Constraints (Cont.)

- Specifying key constraints in SQL
  - candidate keys specified using UNIQUE
  - primary key specified using PRIMARY KEY

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

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),
    UNIQUE (cid, grade) )
```

Students can take only one course, and receive a single grade for that course; no two students in a course receive the same grade.

Primary Key Constraints (Cont.)

- Enforcing primary key constraints
  - Insertion and update can cause violations
  - Deletion does not cause a violation
  - Examples
    - Insert (S0002, Mike, mike@cs, 20, 2.7) to Students
    - Insert (null, Mike, mike@cs, 20, 2.7) to Students
    - Change sid “S0001” to “S0002” in the first tuple of Students

Foreign Key Constraints

- Foreign key
  - A set of fields in one relation that is used to “refer” to a tuple in another relation like a “logical pointer”
  - The foreign key in the referencing relation must match the primary key of the referenced relation

E.g., Enrolled (*sid*: string, *cid*: string, *grade*: string)

- *sid* is a foreign key referring to Students
- If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references

Foreign Key Constraints (Cont.)

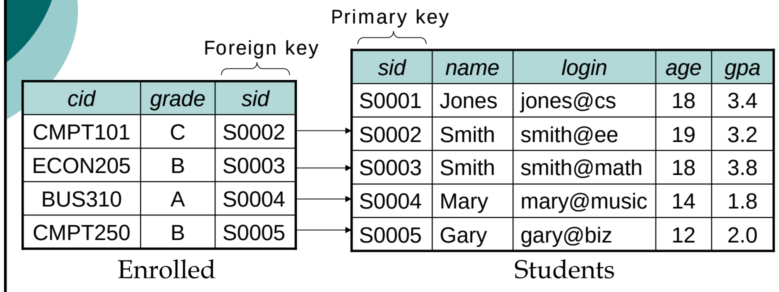
- Foreign Keys in SQL

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

- Only students listed in the Students relation are allowed to enroll in courses
- A student has exactly one grade for each course s/he is enrolled in

Foreign Key Constraints (Cont.)

- Enforcing Referential Integrity



Primary key

Foreign key

cid	grade	sid
CMPT101	C	S0002
ECON205	B	S0003
BUS310	A	S0004
CMPT250	B	S0005

Enrolled

sid	name	login	age	gpa
S0001	Jones	jones@cs	18	3.4
S0002	Smith	smith@ee	19	3.2
S0003	Smith	smith@math	18	3.8
S0004	Mary	mary@music	14	1.8
S0005	Gary	gary@biz	12	2.0

Students

- Insert (PHYS110, A, S1234) to Enrolled (rejected!)
- Delete (S0002, Smith, smith@ee, 19, 3.2) from Students (?)
- Change sid "S005" to "S9999" in the last tuple (?)

Foreign Key Constraints (Cont.)

- A foreign key could refer to the same relation

sid	name	login	age	gpa	partner
S0001	Jones	jones@cs	18	3.4	S0002
S0002	Smith	smith@ee	19	3.2	S0001
S0003	Smith	smith@math	18	3.8	null
S0004	Mary	mary@music	14	1.8	null
S0005	Gary	gary@biz	12	2.0	null

Students

a foreign key referring to Students

Foreign Key Constraints (Cont.)

- Referential Integrity in SQL: SQL-92/99 support 4 options on DELETE and UPDATE
  - NO ACTION (default)
    - delete/update is rejected
  - CASCADE
    - Also delete/update all tuples that refer to deleted/updated tuple
  - SET NULL
    - Set foreign key value of the referencing tuple to null
  - SET DEFAULT
    - Set foreign key value of the referencing tuple to a default value

Foreign Key Constraints (Cont.)

```
CREATE  TABLE  Enrolled
     ( sid       CHAR(20)  DEFAULT  'S9999',
       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!
- Key and foreign key ICs are the most common
- More general ICs are supported too

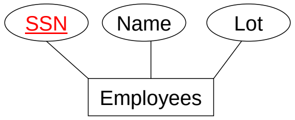
Logical DB Design: ER to Relational

- Entity set
- Relationship set without constraints
- Relationship set with key constraints
- Relationship set with participation constraints
- Weak entity set
- Class hierarchy
- Aggregation

Entity sets to Tables

- Attributes of entity set → attributes of table
- Domain of each attribute
- Primary key of entity set

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

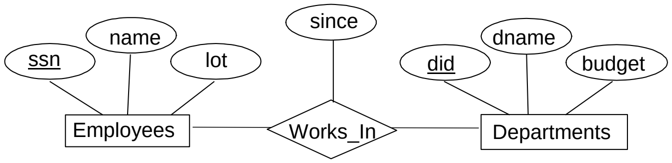


ssn	*name*	*lot*
123-22-3666	Andrew	44
231-31-5368	Emily	22
131-24-3650	Smith	15

Relationship Sets to Tables

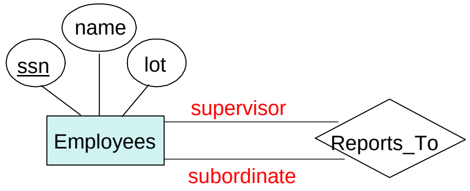
- In translating a relationship set without key or participation constraints to a relation, attributes of the relation must include:
  - The primary key attributes for each participating entity set, as foreign keys
    - This set of attributes forms a superkey for the relation
    - If no key constraints, this set of attributes is a candidate key
  - All descriptive attributes

Relationship Sets to Tables (Cont.)



```
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 )
```

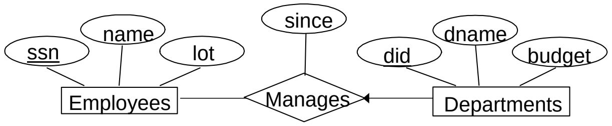
Relationship Sets to Tables (Cont.)



```
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 (subordiante_ssn) REFERENCES Employees (ssn) )
```

Translating Relationship Set with Key Constraints

- Map Manages relationship to a table
  - *did* is the 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 )
```

Translating Relationship Set with Key Constraints (Cont.)

- Since each department has a unique manager, we could instead combine Manages and Departments
  - Avoid a distinct table for the relationship set, therefore, fast query
  - Space would be wasted if some departments have no managers (null)

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

Translating Relationship Set with Participation Constraints

- Every department has a manager
  - Use a combined table for Manages and Departments

```
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 )
```

- If wish to delete an Employees tuple: first change the Dept_Mgr tuple to have a new employee as manager

Translating Relationship Set with Participation Constraints (Cont.)

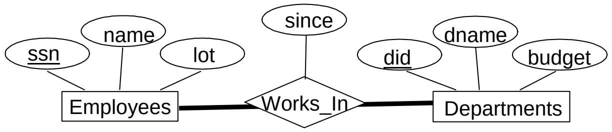
- If use a separate table for Manages:
  - Does not ensure that a manager is initially appointed for each department

```
CREATE TABLE  Manages (
         ssn      CHAR(11)  NOT  NULL,
         did      INTEGER,
         since   DATE,
         PRIMARY KEY  (did),
         FOREIGN KEY  (ssn)  REFERENCES Employees,
         FOREIGN KEY  (did)  REFERENCES Departments )
```

Combined table (Dept_Mgr) is better than separate table (Manages) for one-to-many relationship, especially when the entity set with key constraint also has a total participation constraint

Translating Relationship Set with Participation Constraints (Cont.)

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

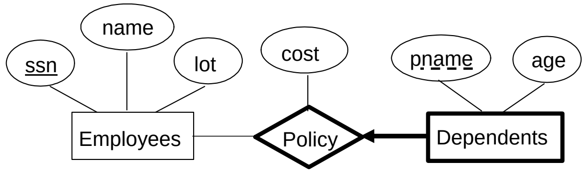


Ensuring total participation of Departments in Works_In
- Possible way: declares that *did* in Departments is a foreign key referring to Works_In
- Problem: this is not a valid FK constraint, since *did* is not a key in Works_In
- Solution: assertion

Translating Weak Entity Set

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity
  - Owner entity set and weak entity set participate in a one-to-many relationship set: 1 owner, many weak entities
  - Weak entity set have total participation in its identifying relationship set
- 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

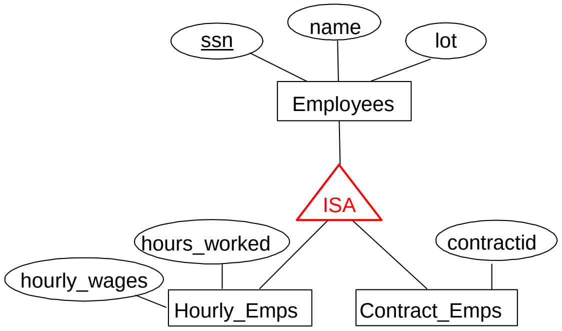
Translating Weak Entity Set (Cont.)



```
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 )
```

Translating Class Hierarchies

- Example: Hourly_Emps and Contract_Emps



Translating Class Hierarchies (Cont.)

- General approach: 3 distinct relations
  - Employees (ssn, *name*, *lot*)
    - All employees are recorded
  - Hourly_Emps (ssn, *hourly_wages*, *hours_worked*)
    - *ssn* is FK referencing Employees
    - Must delete Hourly_Emps tuple if the referenced Employees tuple is deleted
  - Contract_Emps (ssn, *contractid*)

* Queries involving all employees easy

* Queries involving just Hourly_Emps require a join with Employees to get some attributes

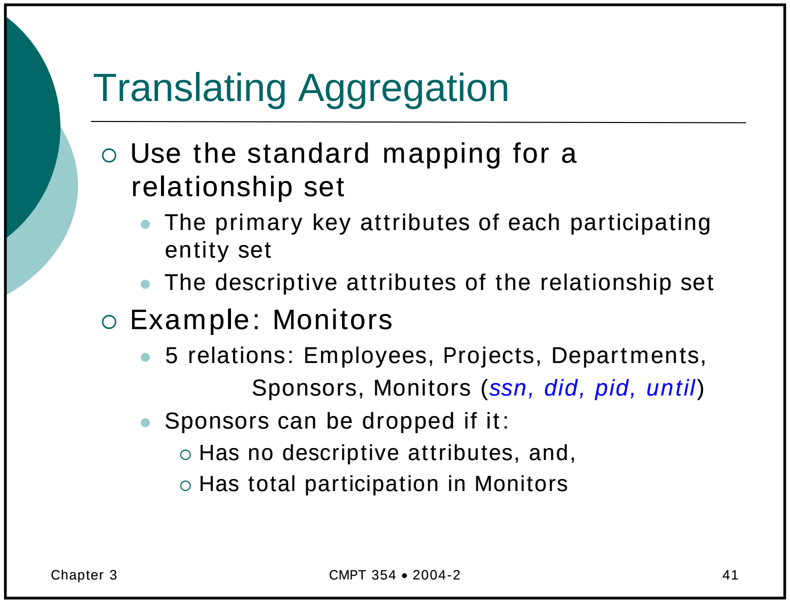
Translating Class Hierarchies (Cont.)

- Alternative: 2 relations
  - Hourly_Emps: (ssn, *name*, *lot*, *hourly_wages*, *hours_worked*)
  - Contract_Emps: (ssn, *name*, *lot*, *contractid*)

* Each employee must be in one of these two subclasses

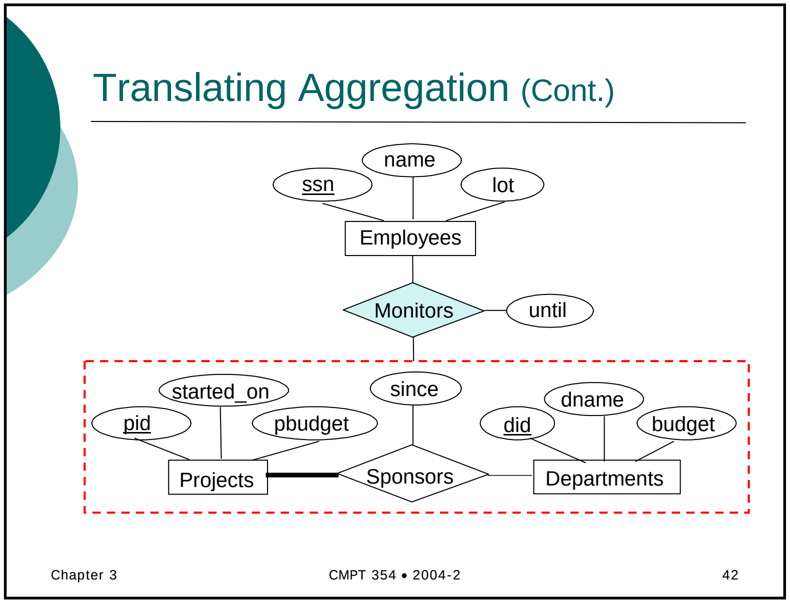
* Duplication: *name* and *lot* are stored twice if an employee is both an Hourly_Emps and a Contract_Emps

* Queries involving all employees have to examine two relations

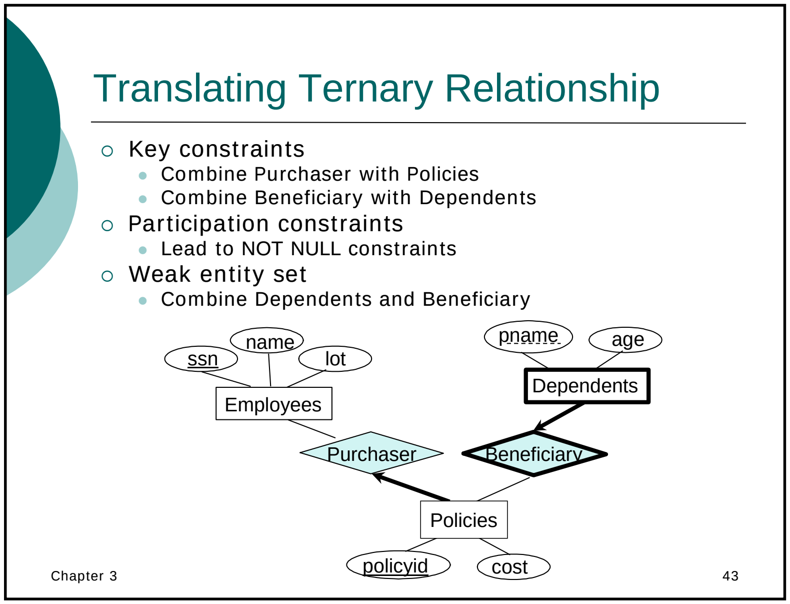


Translating Aggregation

- Use the standard mapping for a relationship set
  - The primary key attributes of each participating entity set
  - The descriptive attributes of the relationship set
- Example: Monitors
  - 5 relations: Employees, Projects, Departments, Sponsors, Monitors (*ssn, did, pid, until*)
  - Sponsors can be dropped if it:
    - Has no descriptive attributes, and,
    - Has total participation in Monitors

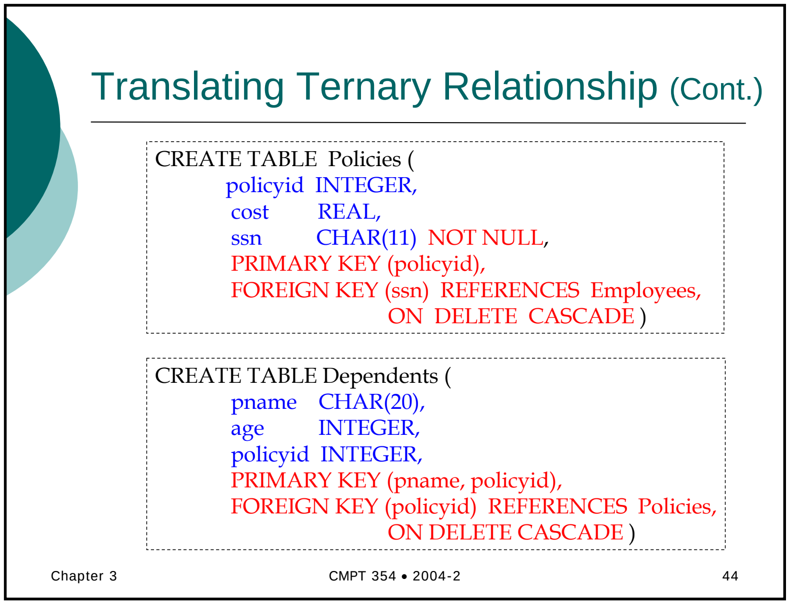


Translating Aggregation (Cont.)



Translating Ternary Relationship

- Key constraints
  - Combine Purchaser with Policies
  - Combine Beneficiary with Dependents
- Participation constraints
  - Lead to NOT NULL constraints
- Weak entity set
  - Combine Dependents and Beneficiary



Translating Ternary Relationship (Cont.)

```
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 just a relation, but we store a definition of it, 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
```

* To drop a view:
DROP VIEW command

* To drop a table when there is a view on the table:
DROP TABLE command with CASCADE option

Views (Cont.)

- View provides support for logical data independence
- Views provides support for security
  - View can be used to present necessary information (or a summary), while hiding details in underlying relations
- We can insert a row into a view by inserting a row into its base table
- An INSERT or UPDATE to the base table may lead to a row not in the view

Relational Model: Summary

- A tabular representation of data
- Simple and intuitive, 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