ISYS2120 – Data & Information Management

Week 3: The Relational Data Model
(Kifer/Bernstein/Lewis - Chapter 3; Ramakrishnan/Gehrke - Chapter 3)

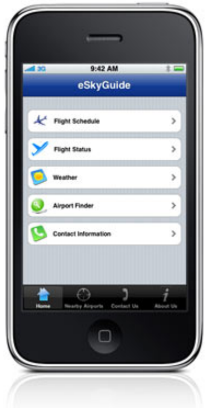
Prof Alan Fekete / Dr. Matloob Khushi
School of Information Technologies

Outline

- Introduction to the Relational Data Model
- Creating Relational Database Schemas using SQL
- Mapping E-R Diagrams to Relational Schemas

Based on slides from Kifer/Bernstein/Lewis (2006) “Database Systems”
and from Ramakrishnan/Gehrke (2003) “Database Management Systems”,
and including material from Fekete and Röhm.

How can one system support such diverse applications?



Relational Data Model

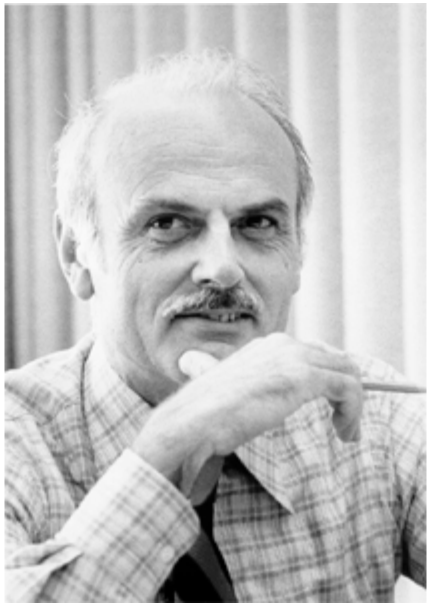


Photo of Edgar F. Codd

- The relational model was first proposed by Dr. E.F. 'Ted' Codd of IBM in 1970 in:

  "A Relational Model for Large Shared Data Banks", Communications of the ACM, June 1970.

  - *This paper caused a major revolution in the field of database management and earned Ted Codd the coveted ACM Turing Award in 1981.*

- The relational model of data is based on the mathematical concept of ***Relation***.
  - Studied in Discrete Mathematics

- The strength of the relational approach to data management comes from its simple way of structuring data, based on a formal foundation provided by the theory of relations.

“Being an IT professional and not knowing the relational data model is like practising medicine without a license.”

Chris Date

Data Model vs. Schema

- **Data model**: a collection of concepts for describing data
  - *Structure of the data*
  - *Operations on the data*
  - *Constraints on the data*

- **Schema**: a description of a particular collection of data at some abstraction level, using a given data model

- **Relational data model** is the most widely used model today
  - Main concept: **relation**, basically a table with rows and columns
  - Every relation has a **schema**, which describes the columns, or fields

Definition of Relation

- **Informal Definition:**
  A ***relation*** is a named, two-dimensional table of data
  - Table consists of rows (record) and columns (attribute or field)
- Example: Staff

Attributes (also: columns, fields)

Tuples (rows, records)

eid	fname	lname	gender	address
1234	Peter	Pan	M	Neverland
5658	Dan	Murphy	M	Alexandria
8877	Sarah	Sander	F	Glebe

Conventions: we try to follow a general convention that relation names begin with a capital letter, while attribute names begin with a lower-case letter

Some Remarks

- Not all tables qualify as a relation.
- Requirements:
  - Every relation must have a unique name.
  - Attributes (columns) in tables must have unique names.
    => The order of the columns is irrelevant.
  - All tuples in a relation have the same structure;
    constructed from the same set of attributes
  - Every attribute value is atomic (not multivalued, not composite).
  - Every row is unique
    (can't have two rows with exactly the same values for all their fields)
  - The order of the rows is immaterial

- The restriction of atomic attributes is also known as **First Normal Form (1NF).**

Example

- Is this a correct relational table Staff1?

eid	fname	lname	gender	address	phones
1234	Peter	Pan	M	Neverland	0403 567123
5658	Dan	Murphy	M	Alexandria	02 67831122 0431 567312
9876	Jin	Jiao	F	Bankstown	
8877	Sarah	Sander	F	Glebe	02 8789 8876
1234	Peter	Pan	M	Neverland	0403 567123

Formal Definition of a Relation

- **A Relation is a mathematical concept based on the ideas of sets.**
  - ***Relation R***
    Given sets $D_1, D_2, \ldots, D_n$, a relation R is a subset of $D_1 \times D_2 \times \ldots \times D_n$
    Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where $a_i \in D_i$
- Example:
  If
  studentid = {12345678, 23456789, 345354345, 44455666, etc}
  name = {Jones, Smith, Kerry, Lindsay, etc}
  date = {1985-11-09, 1984-07-15, 1984-12-01, 1986-01-01, etc}
  then
  R = { (12345678, Jones, 1984-07-15),
  (345354345, Lindsay, 1986-01-01),
  (44455666, Kerry, 1985-11-09),
  (23456789, Kerry, 1994-07-15) }
  is a relation over *studentid* x *name* x *date*

Relation Schema vs. Relation Instance

- A relation *R* has a **relation schema**:
  - specifies name of relation, and name and data type of each attribute.
    - $A_1, A_2, \ldots, A_n$ are ***attributes***
    - $R = (A_1, A_2, \ldots, A_n)$ is a ***relation schema***
    - e.g. `Student(sid: string, name: string, unikey: string, birthdate: date, gender: char)`

- A **relation instance**: a set of tuples (*table*) for a schema
  - $D_1, D_2, \ldots, D_n$ are the domains
  - each attribute corresponds to one domain:
    $dom(A_i) = D_i \ , 1 <= i <= n$
  - $R \subseteq D_1 \text{ x } D_2 \text{ x} \ldots \text{x } D_n$
  - #rows = ***cardinality***,
    #fields = ***degree*** (or ***arity***) of a relation

Relational Database

- **Data Structure:** A relational database is a set of relation instances (tables) with tuples (rows) and fields (columns) - a simple and consistent structure.
  - The collection of all the corresponding relational schemata is the **relational database schema**.
- **Data Manipulation:** Powerful operators to manipulate the data stored in relations.
- **Data Integrity:** facilities to specify a variety of rules to maintain the integrity of data when it is manipulated.

Theory vs. Technology

- A relational DBMS supports data in a form close to, but not exactly, matching the mathematical relation
  - RDBMS allows null ‘values’ for unknown information
  - RDBMS allows duplicate rows
    - The formal relational model does not allow for duplicates
  - plus some more differences which we will see later (e.g. RDBMS support an order of tuples or attributes)

The Special NULL ‘Value’

- RDBMS allows a special entry ***NULL*** in a column to represent facts that are not relevant, or not yet known
  - Eg a new employee has not yet been allocated to a department
  - Eg salary, hired may not be meaningful for adjunct lecturers
  - INSTRUCTOR table in a university

lname	fname	salary	birth	hired
Jones	Peter	35000	1970	1998
Smith	Susan	null	1983	null
Smith	Alan	35000	1975	2000

Pro and Con of NULL

- Pro:
  NULL is useful because using an ordinary value with special meaning does not always work
  - Eg if salary=-1 is used for “unknown” in the previous example, then averages won’t be sensible

- Con:
  NULL causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effects later

Creating and Deleting Relations in SQL

- Creation of tables (relations):

```
CREATE TABLE name ( list-of-columns )
```

  - Example: Create the Students relation.
    Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Instructor (lname  VARCHAR(20),
                         fname  VARCHAR(20),
                         salary INTEGER,
                         birth  DATE,
                         hired  DATE );
```

- Deletion of tables (relations):

```
DROP TABLE name
```

  - the schema information <u>and</u> the tuples are deleted.
  - Example: Destroy the Instructor relation

```
DROP TABLE Instructor
```

Base Datatypes of SQL

Base Datatypes	Description	Example Values
SMALLINT INTEGER BIGINT	Integer values	1704, 4070
DECIMAL(p,q) NUMERIC(p,q)	Fixed-point numbers with precision *p* and *q* decimal places	1003.44, 160139.9
FLOAT(p) REAL DOUBLE PRECISION	floating point numbers with precision *p*	1.5E-4, 10E20
CHAR(q) VARCHAR(q) CLOB(q)	alphanumerical character string types of fixed size *q* respectively of variable length of up to *q* chars	‚The quick brown fix jumps…', 'INFO2120'
BLOB(r)	binary string of size r	B'01101', X'9E'
DATE	date	DATE '1997-06-19', DATE '2001-08-23'
TIME	time	TIME '20:30:45', TIME '00:15:30'
TIMESTAMP	timestamp	TIMESTAMP '2002-08-23 14:15:00'
INTERVAL	time interval	INTERVAL '11:15' HOUR TO MINUTE

(cf. Türker, ORDBMS 2004/2005)

Create Table Example

Student	
sid	name

Enrolled		
sid	ucode	grade

UnitOfStudy		
ucode	title	credit_pts

```
CREATE TABLE Student (
    sid    INTEGER,
    name VARCHAR(20)
);
CREATE TABLE UnitOfStudy  (
    ucode CHAR(8),
    title VARCHAR(30),
    creditPoints INTEGER
);
CREATE TABLE Enrolled (
     sid INTEGER, ucode CHAR(8), grade INTEGER
);
```

SQL Schemas & Table Modifications

- Database Servers typically shared by multiple users
  - We want separate schemas per user (naming: `schema.tablename`)
  - **`CREATE SCHEMA`** … command
    - E.g. http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_6014.htm or http://www.postgresql.org/docs/8.3/static/sql-createschema.html
  - If not provided, automatic schema by user name (cf. Oracle)

- Several base data types available in SQL
  - E.g. `INTEGER, REAL, CHAR, VARCHAR, DATE`, …
  - but each system has its specialities such as specific BLOB types or value range restrictions
    - E.g. Oracle calls a string for historical reasons `VARCHAR2`
  - cf. online documentation

- Existing schemas can be changed

  `ALTER TABLE` *`name`* **`ADD COLUMN`** … | **`ADD CONSTRAINT`**… | …
  - Huge variety of vendor-specific options; cf. online documentation

Modifying Instances using SQL

- Insertion of new data into a table / relation
  - **Syntax:**
    INSERT INTO *table* [“**(**”*list-of-columns*“**)**”] **VALUES** “**(**“ list-of-*expression* “**)**”
  - Example:

```
INSERT INTO Student (sid, name) VALUES (12345678, 'Smith')
```

- Updating of tuples in a table / relation
  - **Syntax:**
    UPDATE *table* **SET** *column*“**=**“*expression* {“**,**”*column*“**=**“*expression*}
    [**WHERE** *search_condition*]
  - Example:

```
UPDATE Student
   SET address = '4711 Water Street'
 WHERE sid = 123456789
```

- Deleting of tuples from a table / relation
  - **Syntax:**
    DELETE FROM *table* [**WHERE** *search_condition*]
  - Example:

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

WEHERE clause
As in SELECT statement

Integrity Constraints

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

Non-Null Columns

- One domain constraint is to insist that no value in a given column can be null
  - The value can't be unknown; The concept can't be inapplicable

- In SQL, append NOT NULL to the field declaration

```
CREATE TABLE Instructor
                  ( lname VARCHAR(20) NOT NULL,
                    fname  VARCHAR(20) NOT NULL,
                    salary INTEGER,
                    birth  DATE NOT NULL,
                    hired  DATE );
```

ICs to avoid Duplicate Rows

- In a SQL-based RDBMS, it is possible to insert a row where every attribute has the same value as an existing row
  - The table will then contain two identical rows
  - This isn’t possible for a mathematical relation, which is a *set* of n-tuples

lname	fname	salary	birth	hired
Jones	Peter	35000	1970	1998
Smith	Susan	75000	1983	2006
Smith	Alan	35000	1975	2000
Jones	Peter	35000	1970	1998

Identical rows

Duplicate Rows are bad

- Duplicating information
  - waste of storage
  - Huge danger of inconsistencies if we miss duplicates during updates

- Need for a mechanism to avoid duplicated data…

=> **Key Constraints**

Relational Keys

- **Primary keys** are unique, minimal identifiers in a relation.
  - Examples include employee numbers, social security numbers, etc. This is how we can guarantee that all rows are unique.
  - There may be several **candidate keys** to choose from
  - If we just say **key**, we typically mean *candidate key*

- **Foreign keys** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)
  - Must refer to a candidate key of the parent relation
  - Like a `logical pointer'

- Keys can be **simple** (single attribute) or **composite** (multiple attributes)
- Keys usually are used as indices to speed up the response to user queries (more on this later in the semester)

Example: Relational Keys

Primary key identifies each tuple of a relation.

Composite Primary Key consisting of more than one attribute.

Student

<u>sid</u>	name
31013	John

Enroll

<u>sid</u>	<u>ucode</u>	grade
31013	I2120	CR

Units_of_study

<u>ucode</u>	title	credit_pts
I2120	DB Intro	4

Foreign key is a (set of) attribute(s) in one relation that `refers' to a tuple in another relation (like a `logical pointer').

Relational Keys in more Detail

- 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, we call them each a *candidate key*, and one of the keys is chosen (by DBA) to be the ***primary key***.

- E.g., *sid* is a key for Student.
  - What about *name*?
  - And the set {*sid, name*}? This is a superkey.

Summary of Key Constraints in SQL

- Primary keys and foreign keys can be specified as part of the SQL CREATE TABLE statement:
  - The `PRIMARY KEY` clause lists attributes that comprise the *primary key*.
  - The `FOREIGN KEY` clause lists the attributes that comprise the *foreign key* and the name of the relation referenced by the foreign key.
  - The `UNIQUE` clause lists attributes that comprise a *candidate key*.
- By default, a foreign key references the primary key attributes of the referenced table

```
FOREIGN KEY (sid) REFERENCES Student
```

- Reference columns in the referenced table can be explicitly specified
  - but *must be declared as primary or candidate keys*

```
FOREIGN KEY (lecturer) REFERENCES Lecturer(empid)
```

- Tip: Name them using `CONSTRAINT` clauses

```
CONSTRAINT Student_PK PRIMARY KEY (sid)
```

Create Table Example with PKs/FKs

Student	
sid	name

Enrolled		
sid	ucode	grade

Unit_of_Study		
ucode	title	credit_pts

```
CREATE TABLE Student ( sid INTEGER, ... ,
    CONSTRAINT Student_PK PRIMARY KEY (sid)
);
CREATE TABLE UoS  ( ucode CHAR(8), ... ,
    CONSTRAINT UoS_PK PRIMARY KEY (ucode)
);
CREATE TABLE Enrolled ( sid INTEGER, ucode CHAR(8), grade CHAR(2),
    CONSTRAINT Enrolled_FK1 FOREIGN KEY (sid) REFERENCES Student,
    CONSTRAINT Enrolled_FK2 FOREIGN KEY (ucode) REFERENCES UoS,
    CONSTRAINT Enrolled_PK  PRIMARY KEY (sid,ucode)
);
```

Choosing the Correct Key Constraints

- Careful: Used carelessly, an IC can prevent the storage of database instances that arise in practice!
- Example:
  Attempt to model that a student can get only a single grade per course.

```
CREATE TABLE Enrolled  (
  sid   INTEGER,
  cid   CHAR(8),
  grade CHAR(2),
  PRIMARY KEY (sid,cid) )
```

vs.

```
CREATE TABLE Enrolled  (
  sid   INTEGER,
  cid   CHAR(8),
  grade CHAR(2),
  PRIMARY KEY (sid, cid),
  UNIQUE (sid, grade) )
```

- "For a given student and course, there is a single grade; the same grade can be achieved by several students in a course."

- "For a given student and course, there is a single grade; but a student can achieve *a certain grade only once*."

Brainteaser

- Given the following example a table

```
CREATE TABLE Test (
        a INTEGER,
        b INTEGER UNIQUE,
        PRIMARY KEY (a,b)
    );
```

- Would the following be a legal database instance?

{ (1, 1) ,
(1, 2) ,
(1, 3) ,
(2, 1) ,
(2, 4) }

Keys and NULLs

- PRIMARY KEY
  - Must be unique and do not allow NULL values
- UNIQUE (candidate key)
  - Possibly many *candidate keys* (specified using UNIQUE)
  - According to the ANSI standards SQL:92, SQL:1999, and SQL:2003, a UNIQUE constraint should disallow duplicate non-NULL values, but allow multiple NULL values.
  - Many DBMS (e.g. Oracle or SQL Server) implement only a crippled version of this, allowing a single NULL but disallowing multiple NULL values….
- FOREIGN KEY
  - By default allows nulls
  - If there must be a parent tuple, then must combine with NOT NULL constraint

Foreign Keys & Referential Integrity

- **Referential Integrity**:
  for each tuple in the referring relation whose foreign key value is α, there must be a tuple in the referred relation with a candidate key that also has value α
  - e.g. *sid* is a foreign key referring to Student:
    Enrolled(*sid*: integer, ucode: string, grade: string)
  - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references

Q: Can you name a data model w/o referential integrity?

Enforcing Referential Integrity

- Consider Student and Enrolled;
  sid in Enrolled is a foreign key that references Student.

- What should be done if an Enrolled tuple with a non-existent student *sid* is inserted? (*Reject it!*)

- What should be done if a Student tuple is deleted? Choices:
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Student 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 Student tuple is updated.

Referential Integrity in SQL

- SQL/92, SQL:1999 and SQL:2003 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 UnitOfStudy
( uosCode    CHAR(8),
  title      VARCHAR(80),
  credit_pts   INTEGER,
  taughtBy INTEGER DEFAULT 1,
    PRIMARY KEY (uosCode),
    FOREIGN KEY (taughtBy)
    REFERENCES  Professor
      ON UPDATE CASCADE
      ON DELETE SET DEFAULT )
```

UnitOfStudy

<u>uosCode</u>	title	credit_pts	taughtBy

Professor

<u>empid</u>	name

This Week's Agenda

- Introduction
- Creating Relational Database Schemas using SQL
- **Mapping E-R Models to Relations**

Correspondence with E-R Model

- Relations (tables) correspond to entity types (entity sets) and to many-to-many relationship types/sets.
- Rows correspond with entities and with many-to-many relationships.
- Columns correspond with attributes or one-to-many relationships.
- Note:
  The word relation (in relational database) is NOT the same as the word relationship (in E-R model).

Mapping E-R Diagrams into Relations

- Mapping rules for
  - Strong Entities
  - Weak Entities
  - Relationships
    - One-to-many, Many-to-many, One-to-one
    - Unary Relationships
    - Ternary Relationships

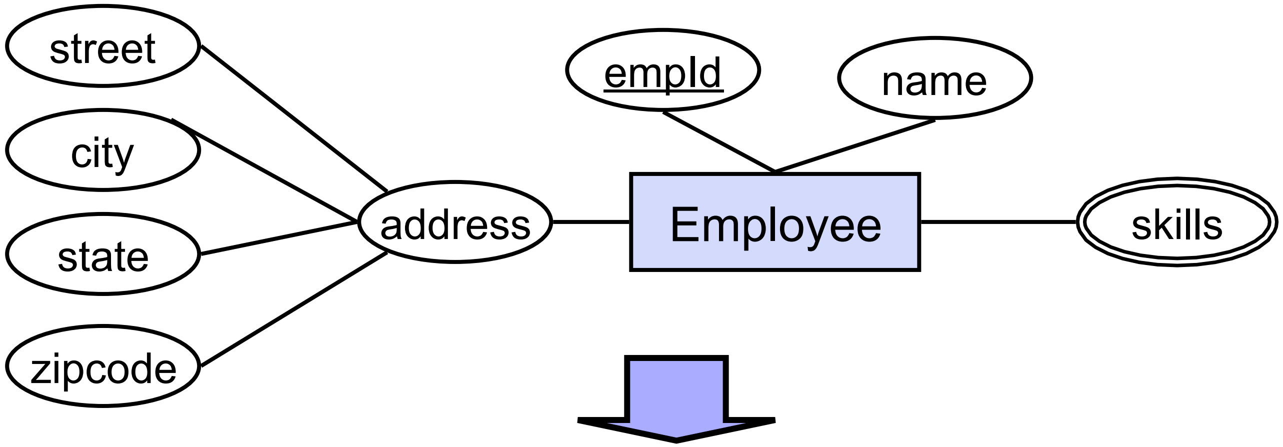
- We will concentrate in the lecture on typical examples…

1. Mapping Regular Entities to Relations

- Each **entity type** becomes a relation
  - **Simple attributes**
    E-R attributes map directly onto the relation
  - **Composite attributes**
    Composite attributes are flattened out by creating a separate field for each component attribute
    => We use only their simple, component attributes
  - **Multi-valued attribute**
    Becomes a separate relation with a foreign key taken from the superior entity

Example: Mapping Entity Types

- Employee entity type with composite/multi-valued attributes



Employee

empId	name	street	city	state	zipcode

Skills

empId	skill

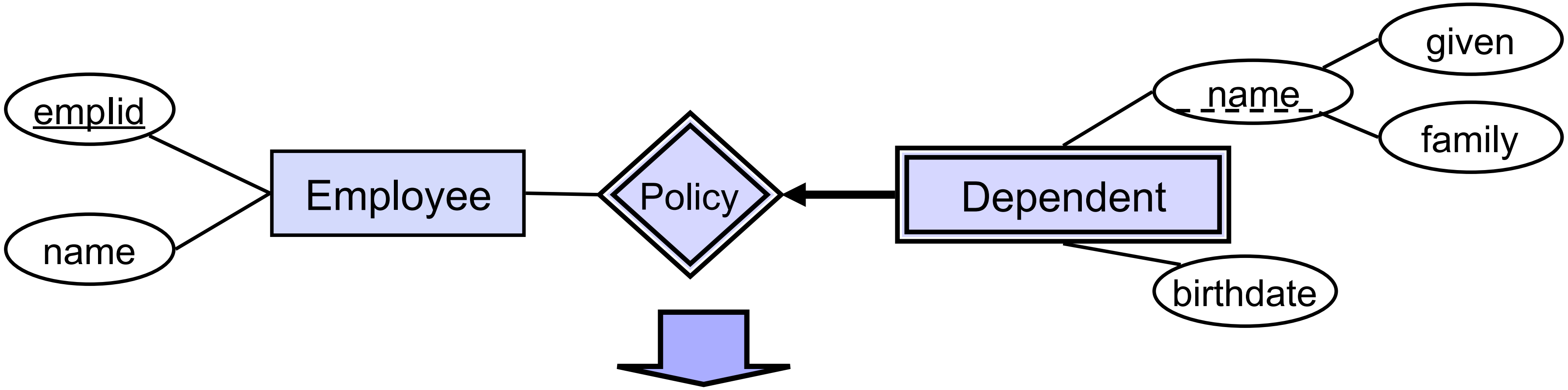
PK-/FK reference between Employee table and table for multi-valued Skills attribute.

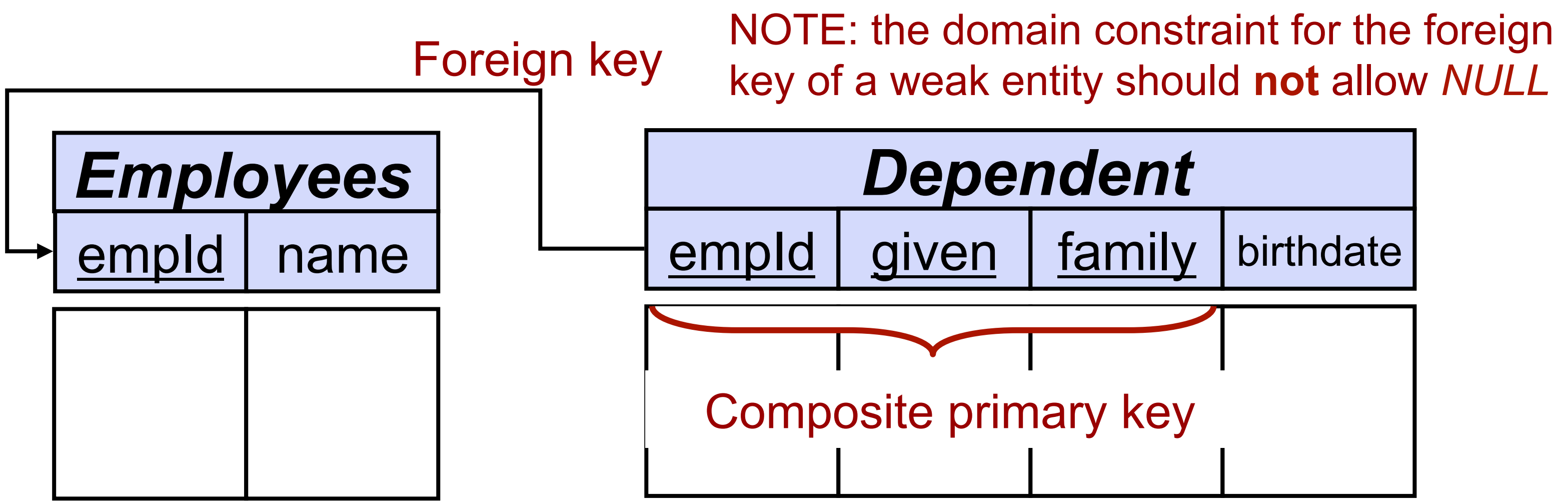
2. Mapping of Weak Entity Types

- **Weak Entity Types**
  - become a separate relation with a foreign key taken from the superior entity
  - primary key composed of:
    - Partial key (discriminator) of weak entity
    - Primary key of identifying relation (strong entity)
  - Mapping of attributes of weak entity as shown before

Example: Mapping of Weak Entity

- Weak entity set 'Dependent' with composite partial key





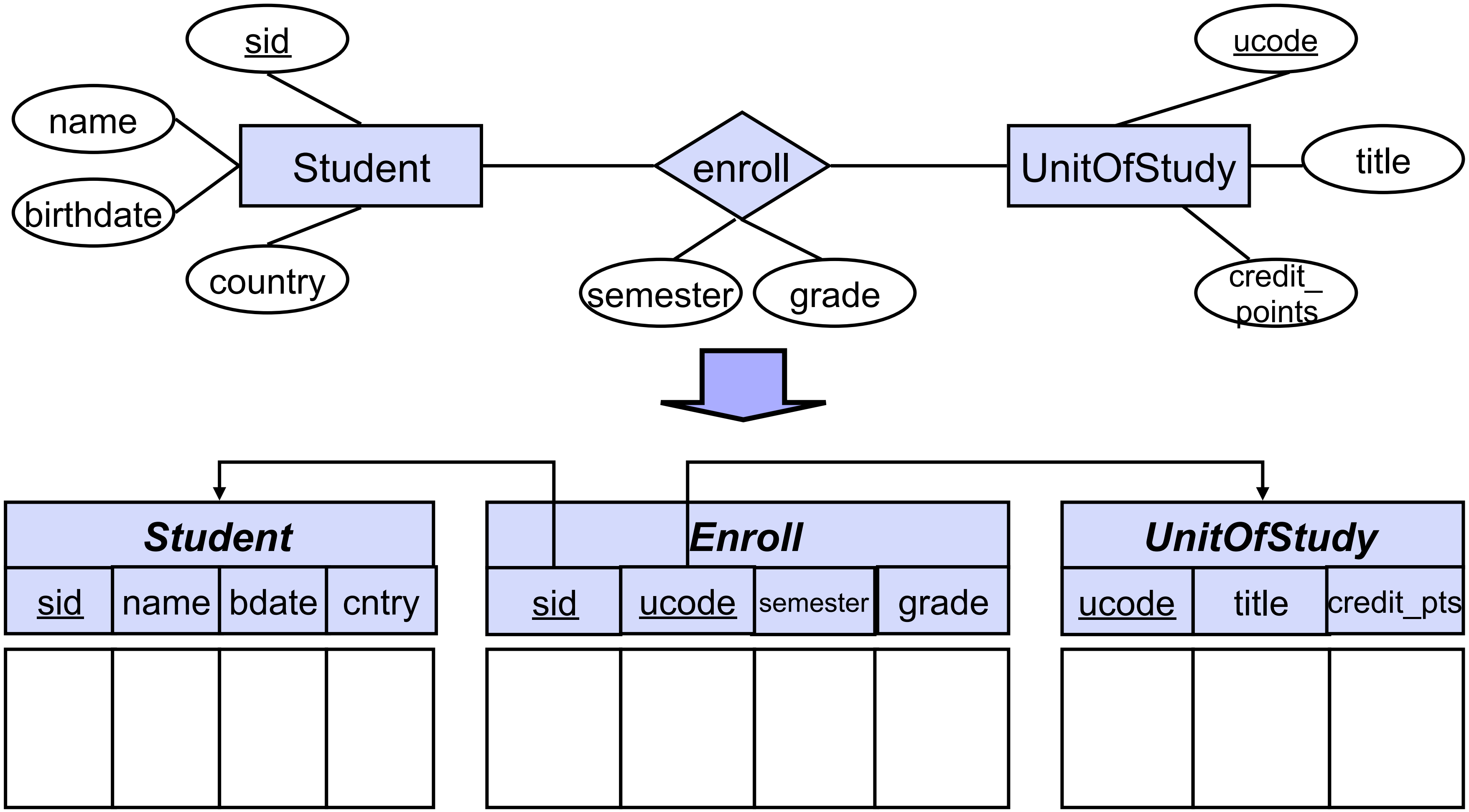
3. Mapping of Relationship Types

- **Many-to-Many** - Create a ***new relation*** with the primary keys of the two entity types as its primary key
- **One-to-Many** - Primary key on the one side becomes a foreign key on the many side
  - Participation Constraint: total side becomes NOT NULL
- **One-to-One** - Primary key on the mandatory side becomes a foreign key on the optional side

- **Relationship attributes** - become fields of either the dependent, respectively new relation

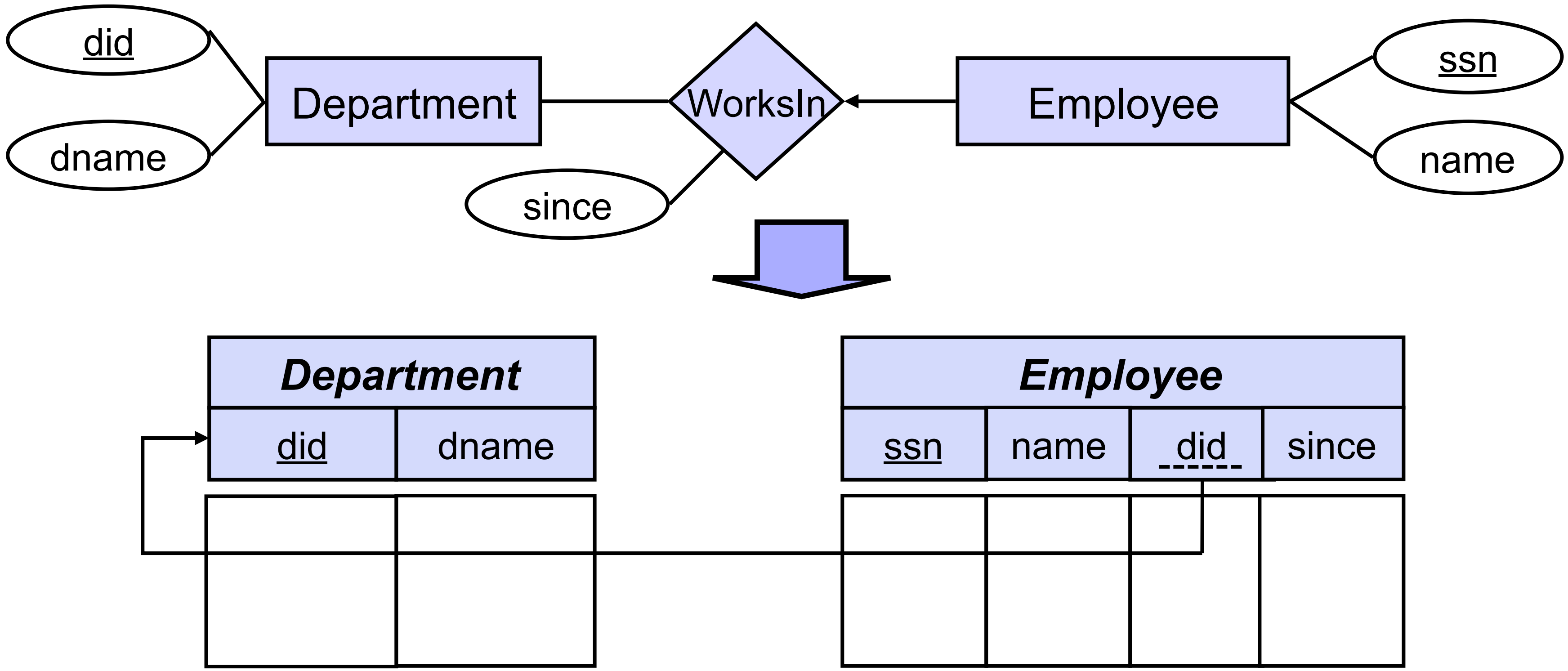
Example: Mapping of Many-to-Many Relationship Types

- Many-to-many relationship between Student & UnitOfStudy



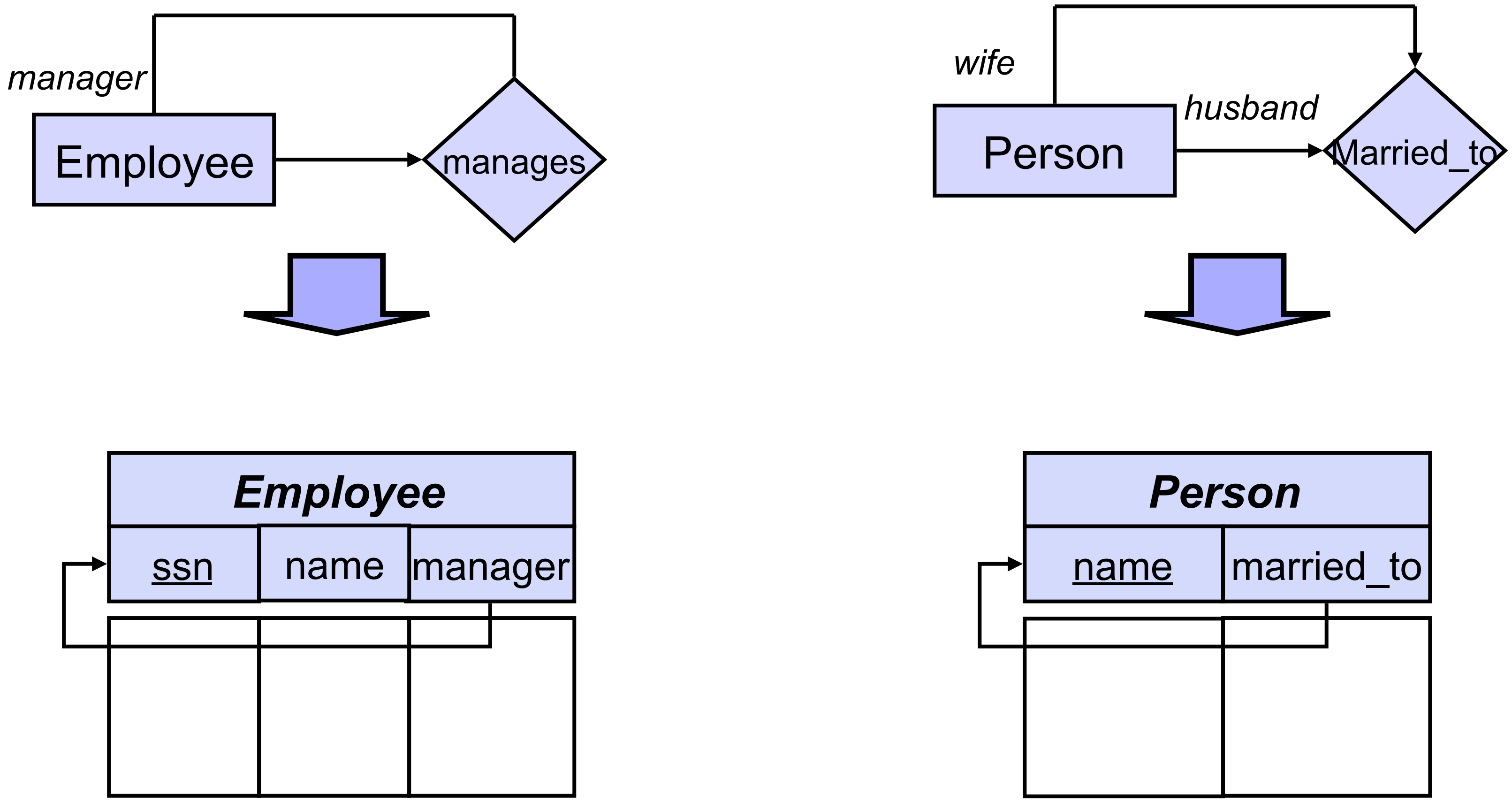
Example: Mapping of One-to-Many Relationships

- **Key Constraint**: One-to-many relationship type



- **Participation Constraint:** NOT NULL on foreign key

Further Examples



This Week's Agenda

- Introduction
- Creating Relational Database Schemas using SQL
- Mapping E-R Models to Relations

You should now be able to:

- **The Relational Model**
  - Design a relational schema for a simple use case
  - Map an Entity-Relationship diagram to a relational schema
  - Identify candidate and primary keys for a relational schema
  - Explain the basic rules and restrictions of the relational data model
  - Explain the difference between candidate, primary and foreign keys
  - Create and modify a relational database schema using SQL
    - including domain types, NULL constraints and PKs/FKs

References

- Kifer/Bernstein/Lewis (2nd edition)
  - Chapter 3
- Ramakrishnan/Gehrke (3rd edition - the 'Cow' book)
  - Chapter 3.1-3.4 and 3.6-3.7, plus Chapter 1.5
- Ullman/Widom (3rd edition)
  - Chapter 2.1 - 2.3, Section 7.1 and Chapter 8.1-8.2
  - *views and foreign keys come later, instead relational algebra is introduced very early on; also briefly compares RDM with XML,*
- Silberschatz/Korth/Sudarshan (5th edition - 'sailing boat')
  - Chapter 2.1 - 2.2; Chapter 3.1-3.2 and 3.9
  - *starts with relational algebra early on which we do later*
- Elmasri/Navathe (5th edition)
  - Chapter 2.1-2.3; Chapter 5; Section 8.8
  - *talks first more about system architectures and conceptual modeling*

Next Week

- Relational Algebra
- More Complex SQL
  - Subqueries, group by …

- Readings:
  - Kifer/Bernstein/Lewis book, Chapter 5
  - <u>Or</u> alternatively (if you prefer those books):
    - Ramakrishnan/Gehrke, Chapter 5 & Section 4.2
    - Ullman/Widom, Chapter 6