COMP9120

Week 3: Relational Data Model & Logical Database Design
Semester 2, 2016

(Ramakrishnan/Gehrke - Chapter 3;

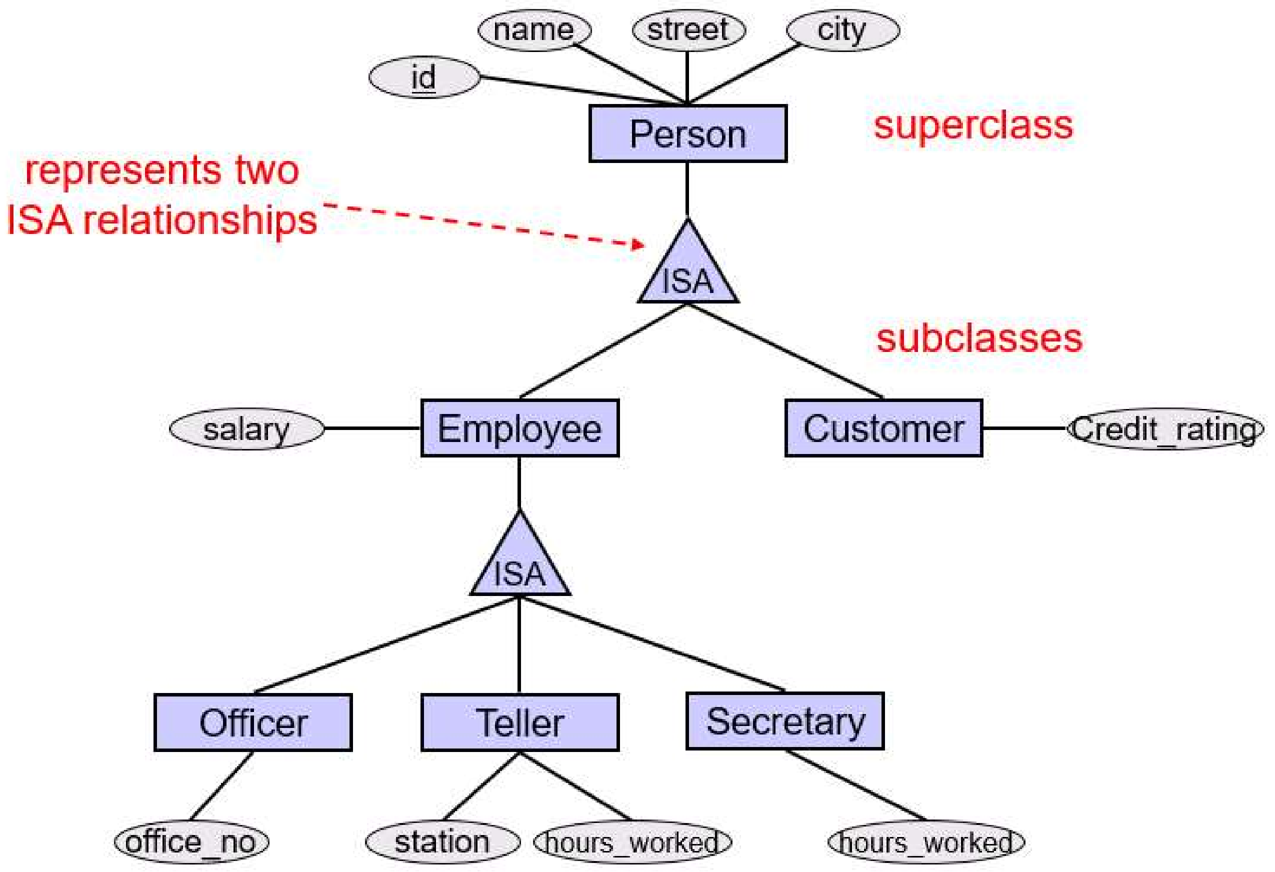
Kifer/Bernstein/Lewis - Chapter 3)

Based on material by Dr. Bryn Jeffries

THE UNIVERSITY OF SYDNEY

Flashback Question

Superclass / Subclass Example



How do we uniquely identify a Teller?

Data Model vs Schema

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

Relational Data Model

- 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.*

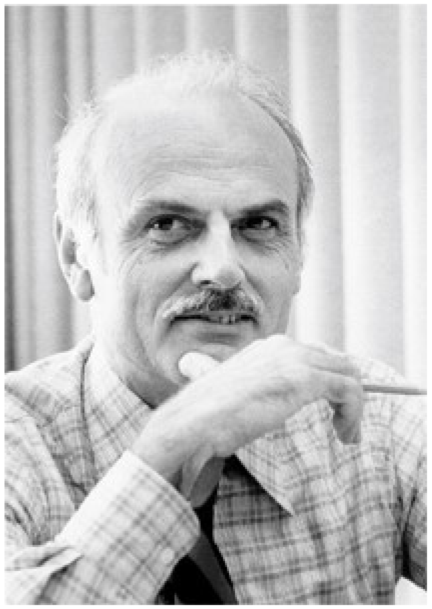


Photo of Edgar F. Codd

- 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.

Definition of a Relation

› **Informal Definition:**
A ***relation*** is a named, two-dimensional table of data

- Table consists of rows (record) and columns (attribute or field)

Attributes (also: columns, fields)

Student

sid	name	login	gender	address
5312666	Jones	ajon1121@cs	m	123 Main St
5366668	Smith	smith@mail	m	45 George
5309650	Jin	ojin4536@it	f	19 City Rd

Tuples (rows, records)

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

Relation Schema vs Relation Instance

- A relation *R* has a **relation schema:**
  - specifies name of relation, and name and data type (domain) of each attribute.
    - $A_1, A_2, \ldots, A_n$ are ***attributes,*** *each having a* ***domain***
      - $D_1, D_2, \ldots, D_n$ are the domains
      - each attribute corresponds to one domain:
        $dom(A_i) = D_i\ ,\ 1 <= i <= n$
    - $R = (A_1, A_2, \ldots, A_n)$ is a ***relation schema***
    - e.g. `Student(sid: string, name: string, login: string, addr: string, gender: char)`
- A **relation instance**: a set of tuples (*a table*) for a schema
  - each tuple has same number of fields as there are attributes defined in schema
  - Values of a field in a tuple must conform to domain defined in schema
  - Relation instance often abbreviated as just relation
  - #rows = ***cardinality***,
    #fields = ***degree*** (or ***arity***) of a relation

Terminology

Degree 5

Schema

Student

<u>sid</u>	name	login	gender	address
5312666	Jones	ajon1121@cs	m	123 Main St
5366668	Smith	smith@mail	m	45 George
5309650	Jin	ojin4536@it	f	19 City Rd

(one possible) instance

Cardinality 3

Relation Dimensions

› How many distinct tuples are in a relation instance with cardinality 22 and degree 12?

1. 22
2. 12
3. Up to 22
4. Up to 12
5. At least 22
6. At least 12

Some Remarks

- Not all tables qualify as a relation.
- Requirements:
  - Every relation must have a unique name.
  - Attributes (columns) in a relation 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 multi-valued, not composite).
  - A relation is a ***set*** of tuples (rows), so:
    - 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).**
  - (Normal forms covered more in another lecture)

THE UNIVERSITY OF SYDNEY

Example

› Is this a correct relation?

name	name	gender	address	phones
Peter	Pan	M	Neverland	0403 567123
Dan	Murphy	M	Alexandria	02 67831122
				0431 567312
Jin	Jiao	F	Jkdsafas sdf asdjf st	
Sarah	Sandwoman	F	Glebe	02 8789 8876
Peter	Pan	M	Neverland	0403 567123

RDBMS Tables vs Mathematical Relations

- RDBMS allows duplicate rows
- RDBMS support an order of tuples or attributes
- RDBMS allows null ‘values’ for unknown information
  - Codd later added NULLs to relational mathematics

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

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

THE UNIVERSITY OF SYDNEY

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 Student (   sid INTEGER,
                        name VARCHAR(20),
                       login VARCHAR(20),
                      gender CHAR,
                     address VARCHAR(50) );
```

› Deletion of tables (relations):

```
        DROP TABLE name ;
```

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

```
                  DROP TABLE Students ;
```

THE UNIVERSITY OF SYDNEY

Base Data Types of ANSI 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)

Exercise 1: Create Table in SQL

Student	
sid	name

Enrolled		
sid	ucode	semester

UnitOfStudy		
ucode	title	credit_pts

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

Table Modifications

- Existing schemas can be changed

  ALTER TABLE *name* **ADD COLUMN** … | **ADD CONSTRAINT**…| …

  - Huge variety of vendor-specific options; see online documentation
  - Eg:

  Rename column:

```
ALTER TABLE customers RENAME COLUMN credit_limit TO credit_amount;
```

  Add columns:

```
ALTER TABLE countries ADD
      (duty_pct     NUMBER(2,2),
                 visa_needed  VARCHAR2(3));
```

http://docs.oracle.com/cd/E16655_01/server.121/e17209/statements_3001.htm#CJAHHIBI

THE UNIVERSITY OF SYDNEY

Modifying Relations 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 VALUES (12345678, ‘Smith’);

INSERT INTO Student (name, sid) VALUES (‘Smith’, 12345678);
```

› 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’ ;
```

More details on those in the SQL lectures.

Relational Database Features

› **Data Structure:** A relational database is a set of relations (tables) with tuples (rows) and fields (columns) - a simple and consistent structure.

› **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.

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.
    - Declared ICs are checked when 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

```
CREATE TABLE Student (
           sid       INTEGER NOT NULL,
           name      VARCHAR(20),
           login     VARCHAR(20) NOT NULL,
           gender    CHAR,
           birthdate 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
    - Waste of storage
    - Huge danger of inconsistencies if we miss duplicates during updates
  - 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

Key Constraints

- **Primary keys** : the minimal set of attributes in a relation that can uniquely identify each row of that 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*

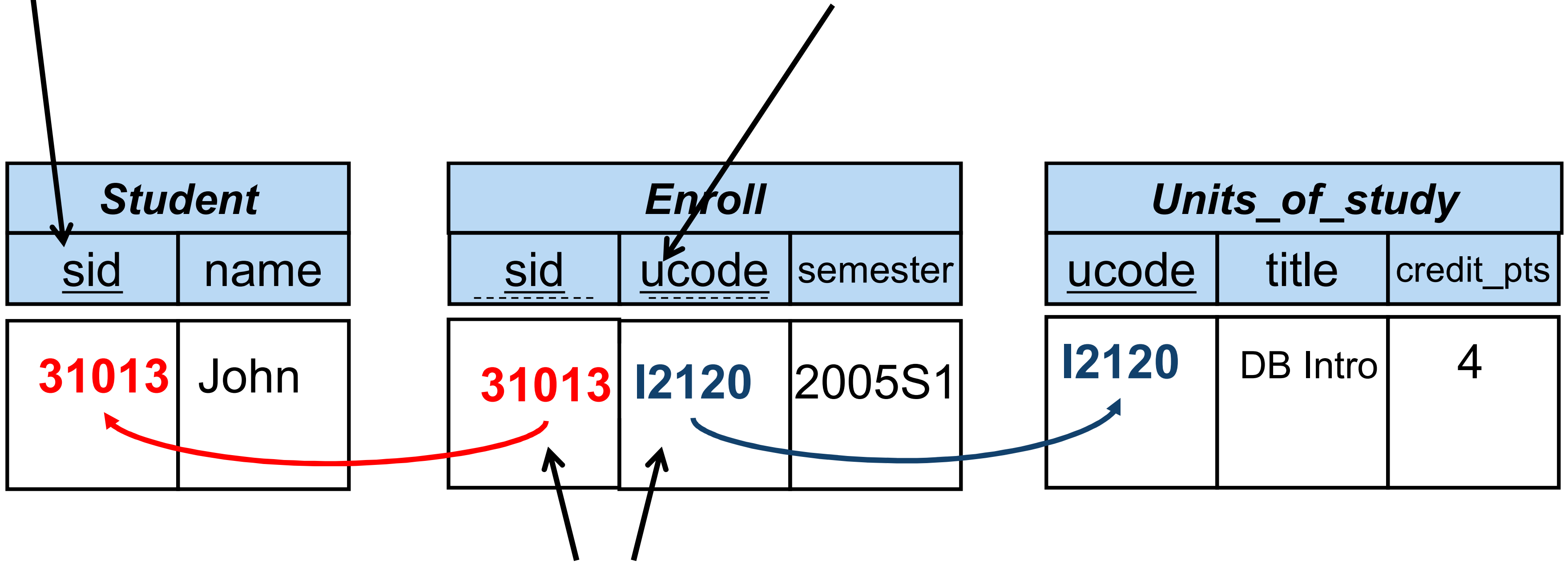
- **Foreign keys** are identifiers that enable a dependent relation to refer to its parent relation
  - Must refer to a candidate key of the parent relation
  - Like a `logical pointer'

- 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

sid	name
31013	John

Enroll

sid	ucode	semester
31013	I2120	2005S1

Units_of_study

ucode	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, gender*}? This is a superkey.

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, semester: string)

sid	**ucode**	**semester**
1234	COMP5138	2012S1
3456	COMP5138	2012S1
5678	COMP5138	2012S2
5678	COMP5338	2013S1

Q: What can we say about the Student relation?

THE UNIVERSITY OF SYDNEY

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
```

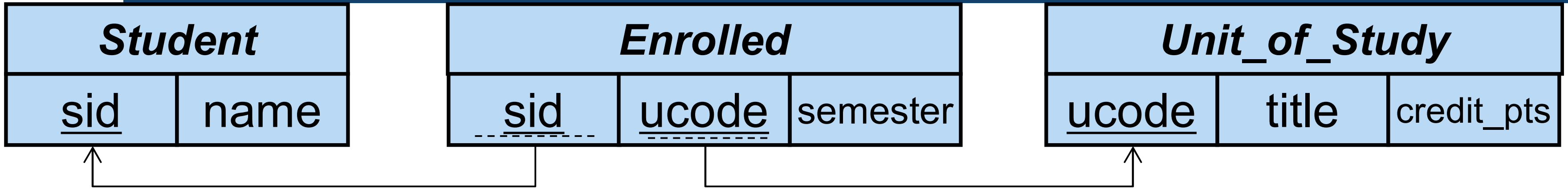
- Reference columns in the referenced table can be explicitly specified
  - but *must be declared as primary or candidate keys*

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

- Tip: Name them using **CONSTRAINT** clauses

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

Exercise 2: Primary & Foreign Keys



```
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), semester VARCHAR,
    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:

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

“For a given student and course, there is a single grade.”

“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.”

Keys and NULLs

- PRIMARY KEY
  - Up to one per table
  - 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 DBMSs 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

Correspondence with ER Model

- Relations (tables) correspond to entity types (entity types)
- Rows correspond with entities
- Columns correspond with attributes
- 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
    - Many-to-one, One-to-many, Many-to-many, One-to-one
    - Unary Relationships
    - Ternary Relationships
  - ISA Hierarchies
  - Aggregations

- We will concentrate in the lecture on typical examples…

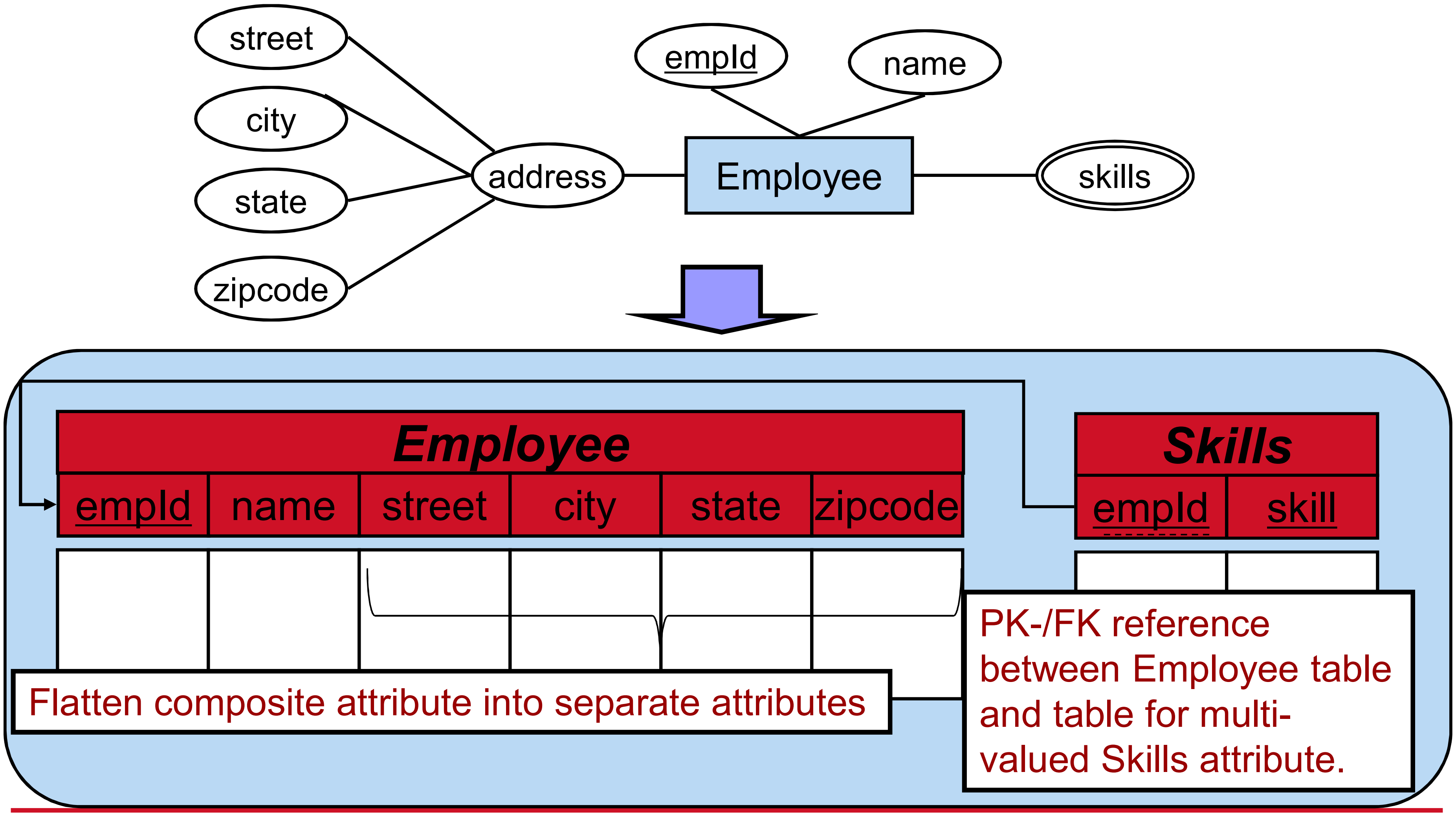
Mapping Strong Entity Types into Relations

› 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

Mapping Strong Entity Types

› Employee entity type with composite/multi-valued attributes



Flatten composite attribute into separate attributes

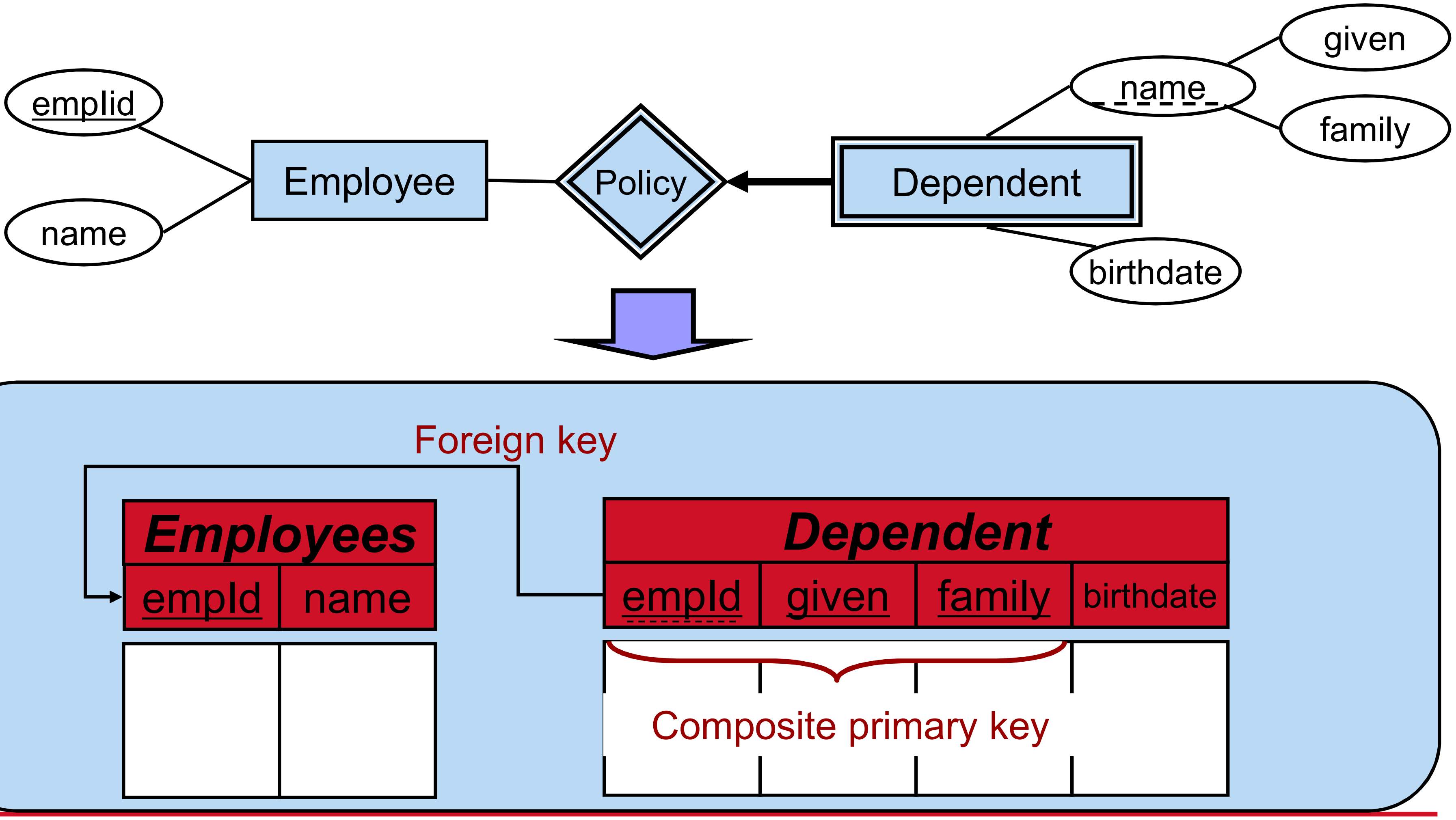
PK-/FK reference between Employee table and table for multi-valued Skills attribute.

Mapping of Weak Entity Types

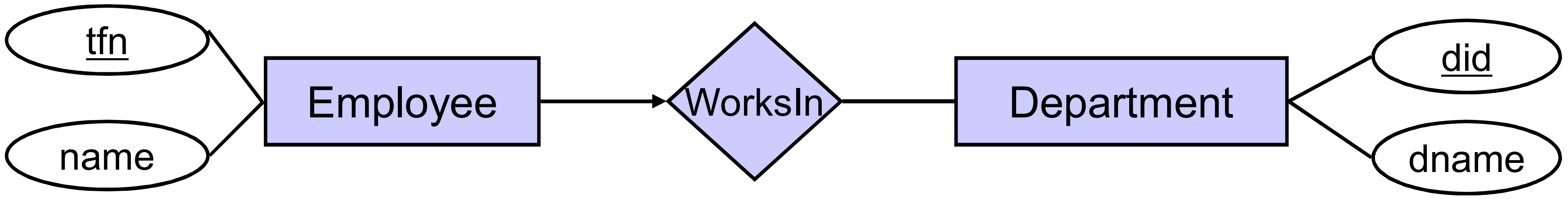
› **Weak Entity Types**
  - become a separate relation with a foreign key taken from the identifying owner entity
  - primary key composed of:
    - Partial key (discriminator) of weak entity
    - Primary key of identifying relation (strong entity)
  - Mapping of attributes of weak entity as shown before

Example: Mapping of Weak Entity

- Weak entity set ‘Dependent’ with composite partial key



Example Multiplicities

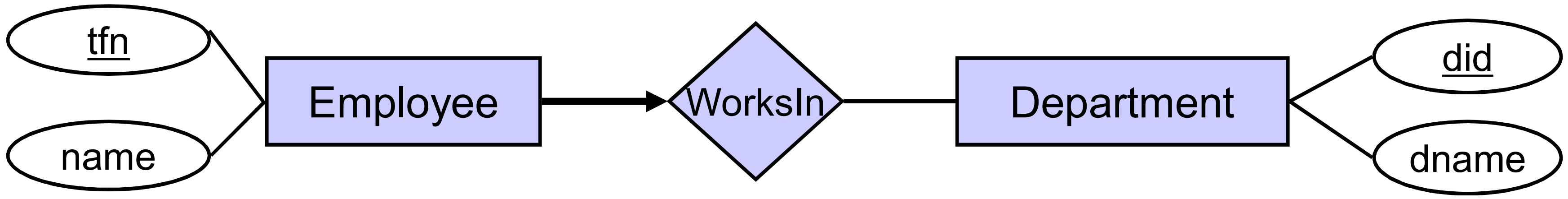


Looking on each *relationship side*:

- 1 Employee works in at most 1 Department
- 1 Department can have 0 to Many Employees

Example of a ***many to one*** *relationship*

Example Multiplicities

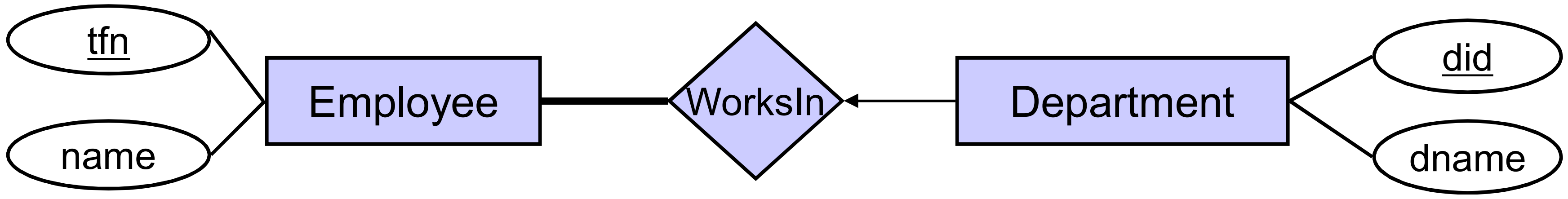


Looking on each *relationship side*:

- 1 Employee works in exactly 1 Department (***mandatory*** to have exactly 1 Dept.)
- 1 Department can have 0 to Many Employees

Example of a ***many to one*** *relationship*

Example Multiplicities



Looking on each *relationship side*:

- 1 Employee can work in 1 to many Departments (***mandatory*** to have at least 1 Dept.)
- 1 Department can have at most 1 employee

Example of a ***one to many*** *relationship*

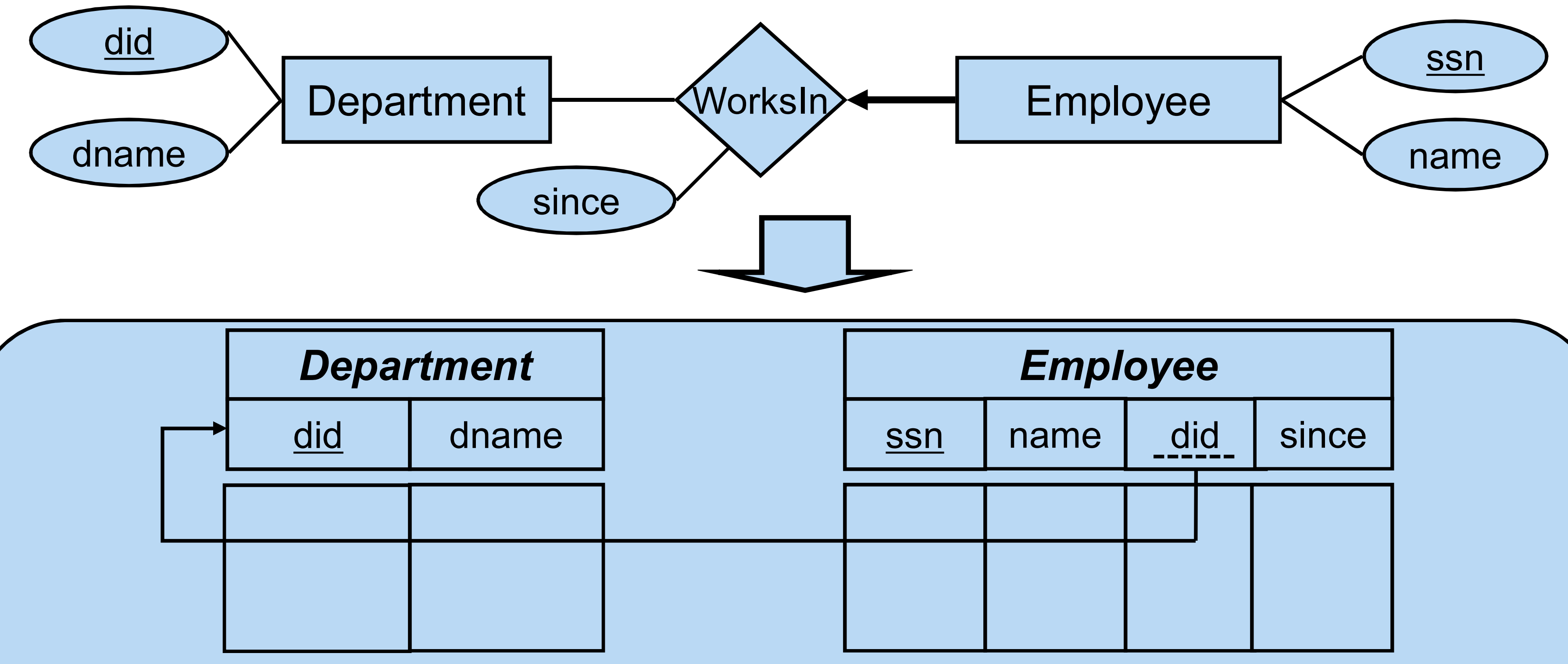
Example Mapping One to Many & Many To One

One-to-Many / Many-to-One - Primary key on the one side becomes a foreign key (call this FK1) on the many side

- If Participation Constraint on many side:
    - can use a NOT NULL constraint on the foreign key (FK1) on the many side
- Also move any relationship attributes to the many side

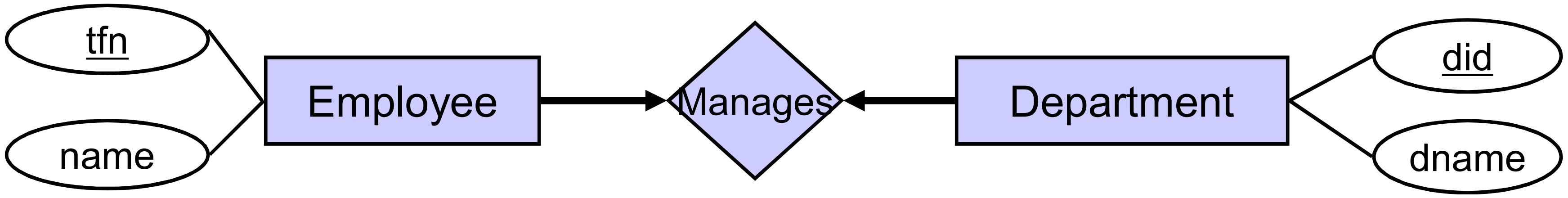
Example: Mapping of One-to-Many Relationships

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



› **Participation Constraint (thick arrow):** NOT NULL on foreign key

Example Multiplicities

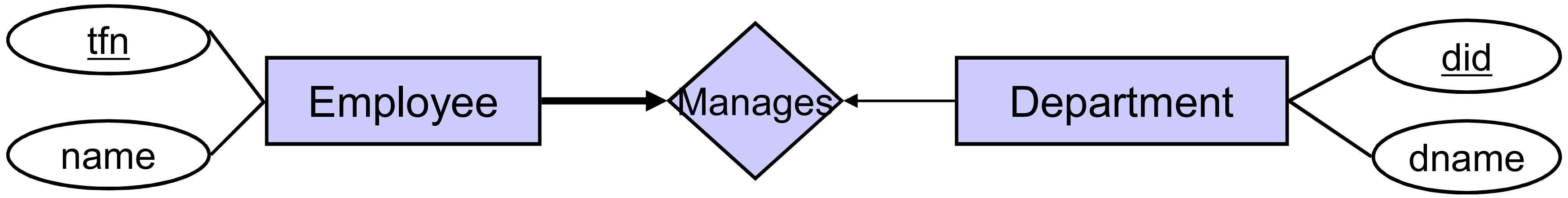


Looking on each relationship side:

- 1 Employee manages exactly 1 Department (***mandatory*** to have exactly 1 Dept.)
- 1 Department managed by exactly 1 Employee (***mandatory*** to have exactly 1 Emp.)

Example of a ***one to one*** *relationship*

Example Multiplicities

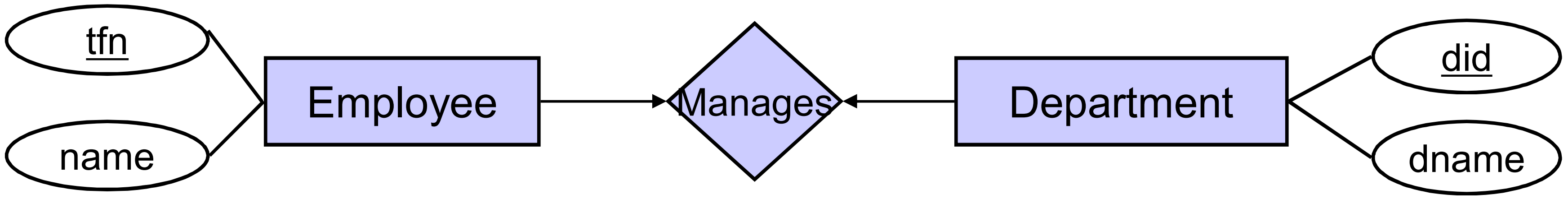


Looking on each relationship side:

- 1 Employee manages exactly 1 Department (***mandatory*** to have exactly 1 Dept.)
- 1 Department can be managed by at most 1 Employee

Example of a ***one to one*** *relationship*

Example Multiplicities



Looking on each relationship side:

- 1 Employee manages 0 or 1 Department
- 1 Department managed by 0 or 1 Employee

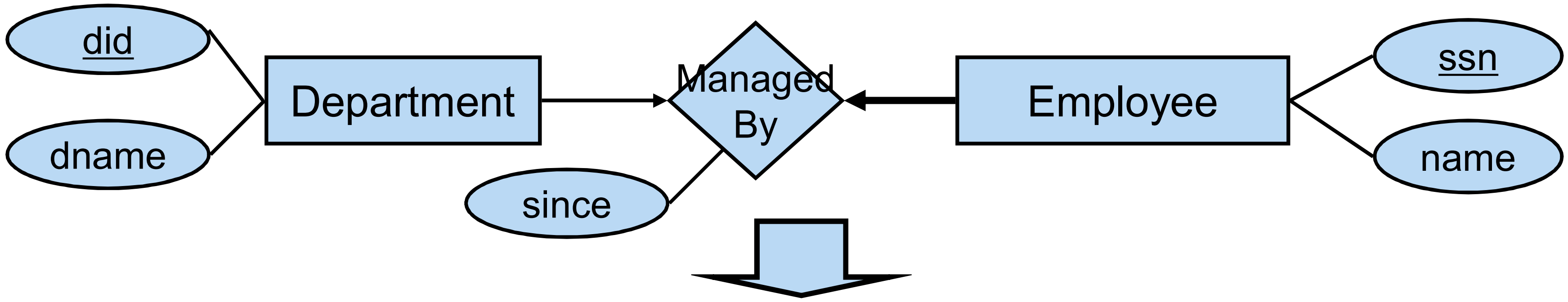
Example of a ***one to one*** *relationship*

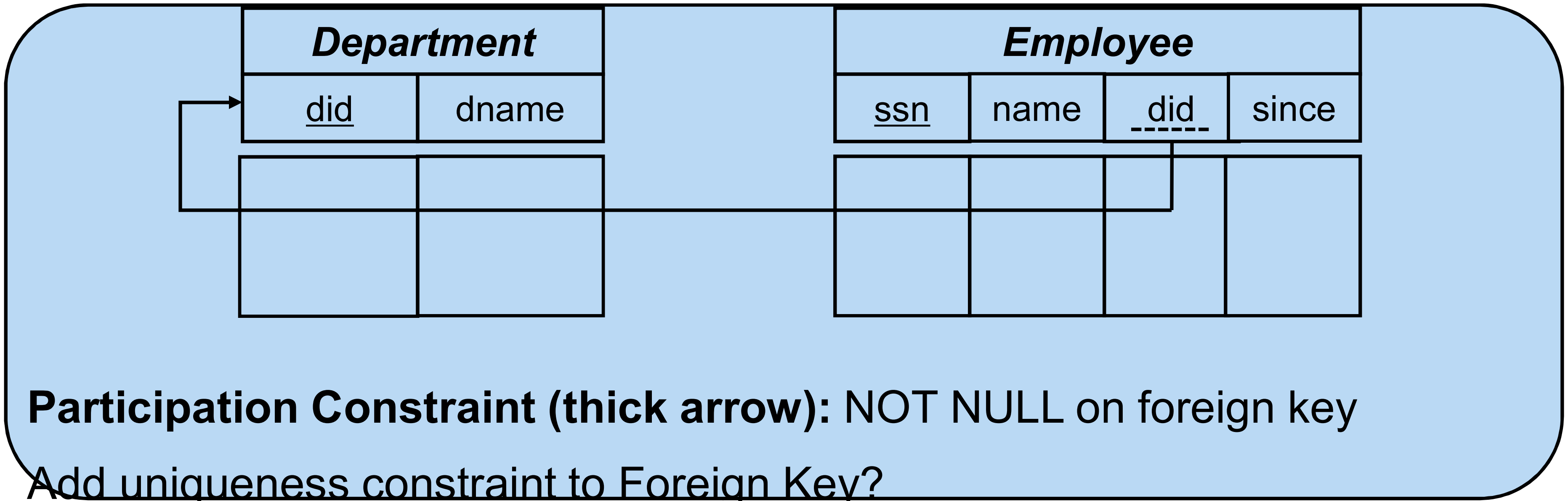
Example: Mapping One to One Relationships

- **One-to-One** – Primary key on one side becomes a foreign key on the other side.
  - If one of the sides is mandatory, place the foreign key on a mandatory side & use a not null constraint as shown on next slide.
  - Also move any relationship attributes to same side

Example: Mapping of One-to-One Relationships

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

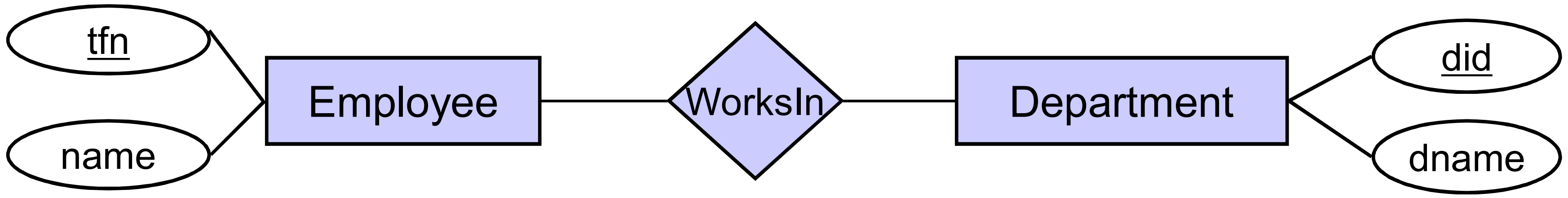




Participation Constraint (thick arrow): NOT NULL on foreign key

Add uniqueness constraint to Foreign Key?

Example Multiplicities

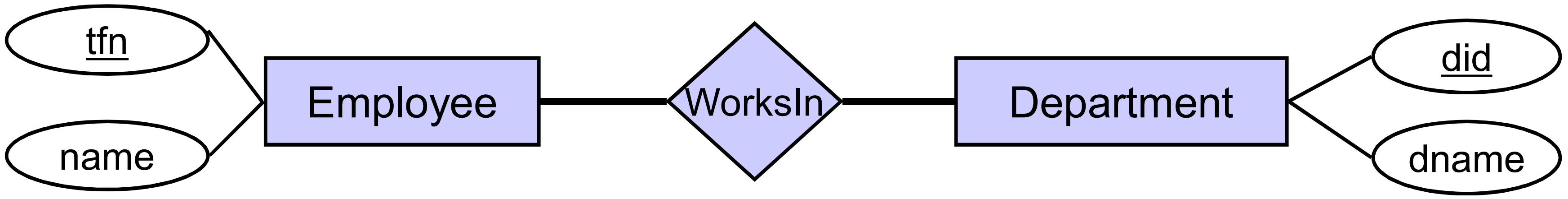


Looking on each relationship side:

- 1 Employee can work in 0 to many Departments
- 1 Department can have 0 to many Employees

Example of a ***many to many*** *relationship*

Example Multiplicities



Looking on each relationship side:

- 1 Employee can work in 1 to many Departments (***mandatory*** to have at least 1 Dept.)
- 1 Department can have 1 to many Employees (***mandatory*** to have at least 1 Emp.)

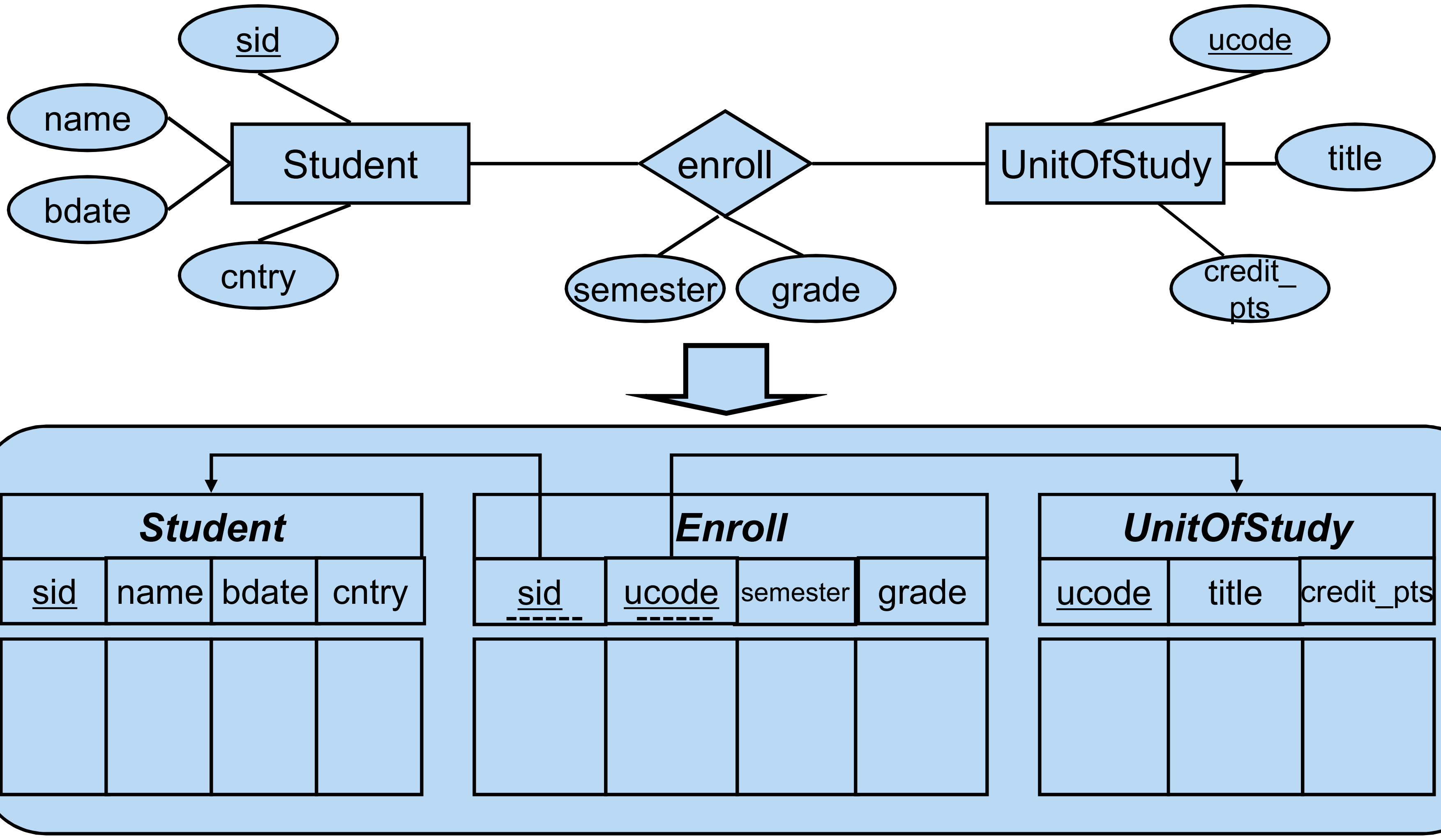
Example of a ***many to many*** *relationship*

Mapping Many to Many Relationships

- **Many-to-Many** - Create a ***new relation*** with the primary keys of the two entity types as its primary key
  - Relationship attributes placed on this new relation

Exercise: Mapping Many-to-Many Relationship Types

› Many-to-many relationship between Student & UnitOfStudy



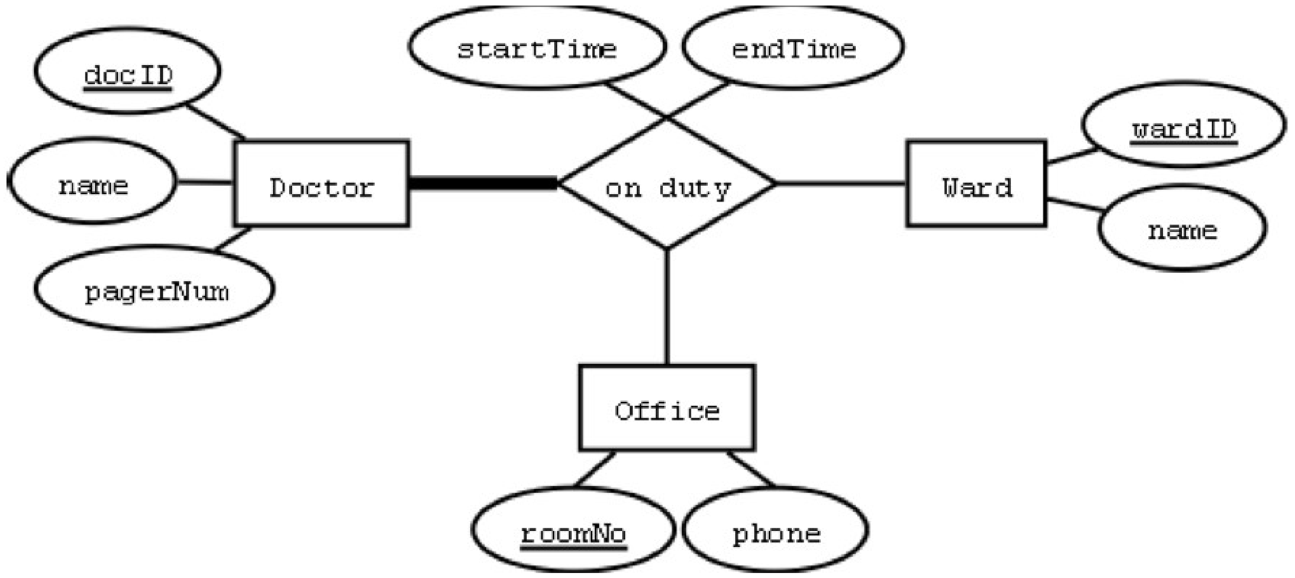
Enforcing Constraints

- › Can sometimes easily enforce in relational model
  - Eg: enforce participation on the many side of a one to many relationship by using a NOT NULL constraint
- › Sometimes need more computationally expensive assertions or table constraints (and it may not always be worth doing this)
  - Eg: Mandatory both sides of a many to many relationship

Example

› Which is the correct relation mapping of the "on duty" relationship?

1. OnDuty(<u>dutyID</u>, startTime, endTime)
2. OnDuty(<u>dutyID</u>, startTime, endTime, docID, wardID, roomNo)
3. OnDuty(<u>docID</u>, wardID, roomNo, startTime, endTime)
4. OnDuty(docID, <u>wardID</u>, <u>roomNo</u>, startTime, endTime)
5. OnDuty(<u>docID</u>, <u>wardID</u>, <u>roomNo</u>, startTime, endTime)
6. OnDuty(<u>startTime</u>, <u>endTime</u>, docID, wardID, roomNo)

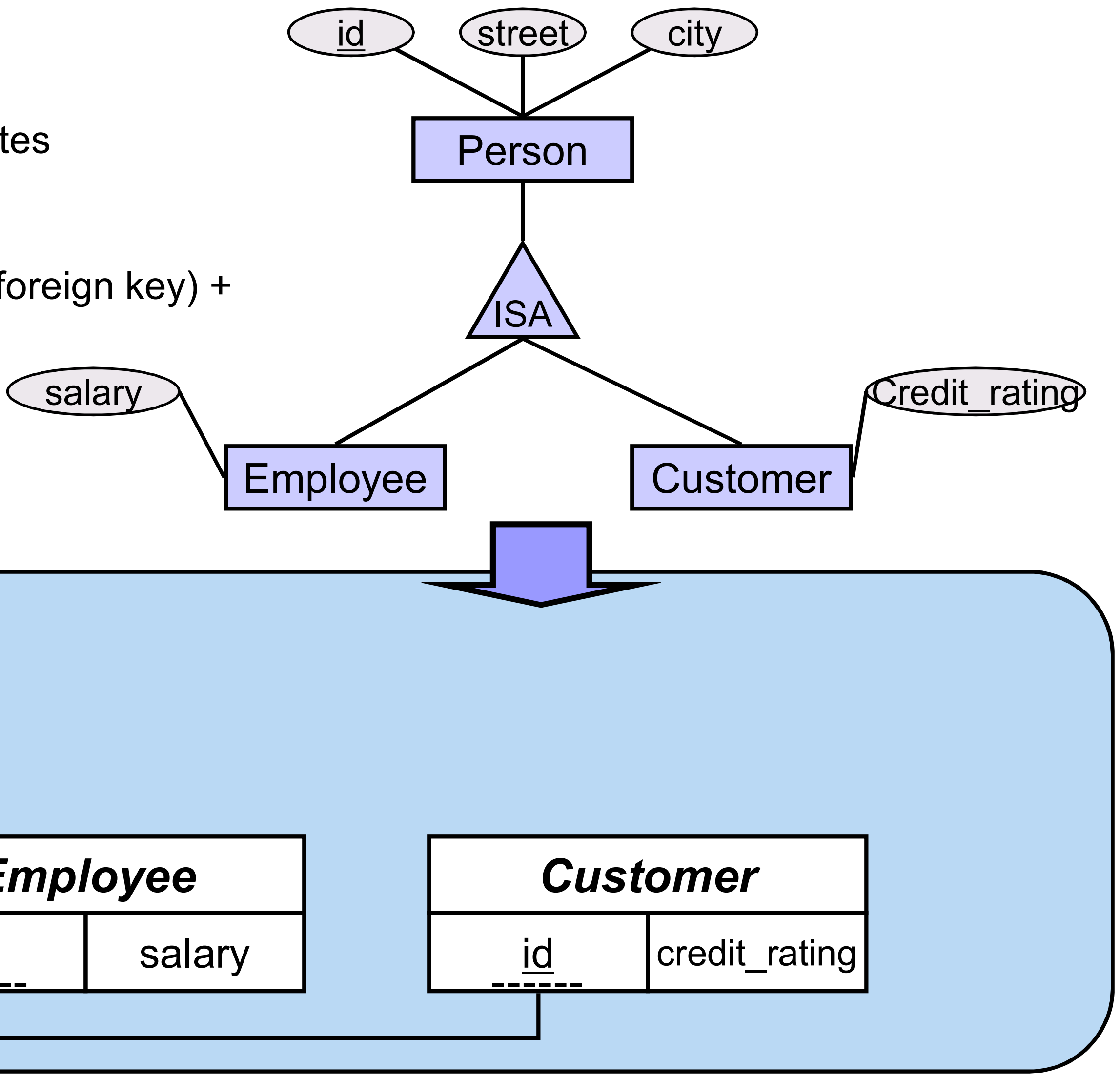


Example Mapping of ISA-Hierarchies

› Standard way (works always, not all constraints enforced):
  - Distinct relations for the superclass and for each subclass
  - Superclass attributes go into superclass relation
  - Subclass attributes go into each sub-relation; primary key of superclass relation also becomes primary key of subclass relation
  - 1:1 relationship established between superclass and each subclass, with superclass as primary table
    - I.e. primary keys of subclass relations become also a foreign key referencing the superclass relation

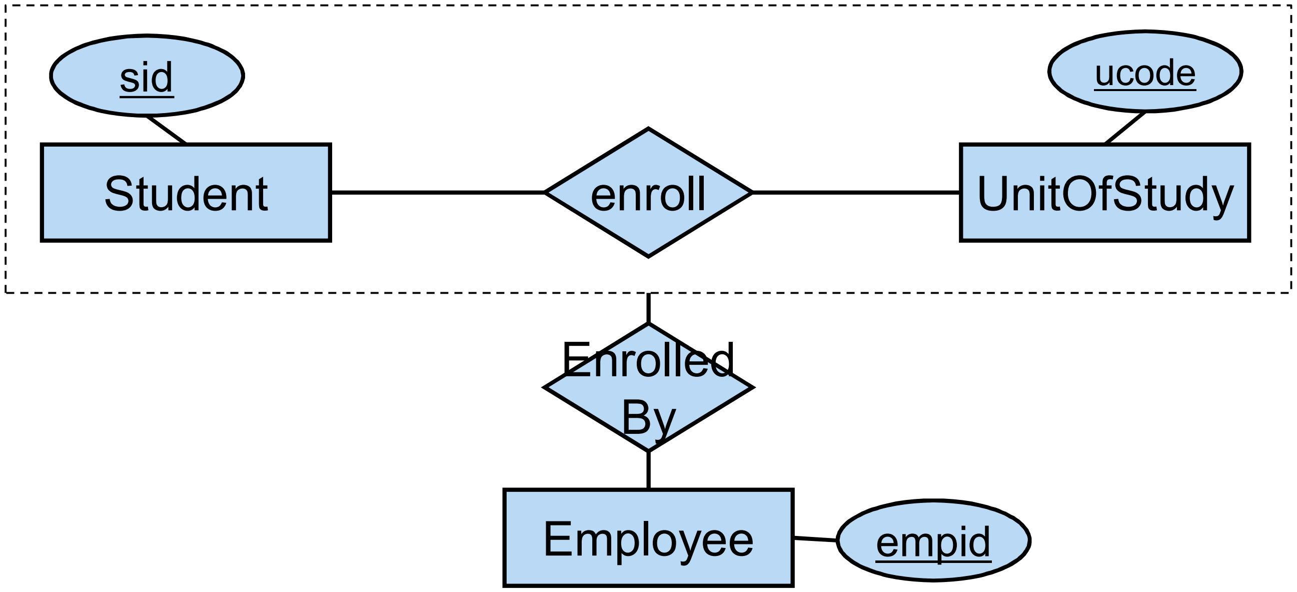
Example: Mapping of ISA-Herarchy

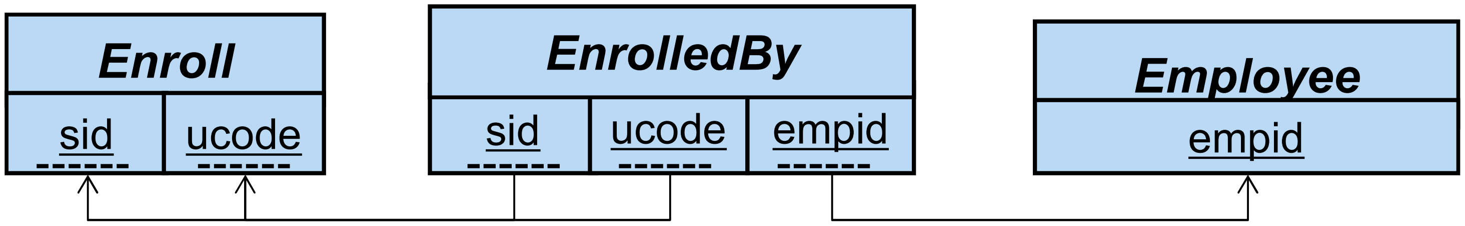
- Table for superclass
  - Primary key + common attributes
- Separate tables for subclasses
  - Primary key of superclass (==foreign key) + specific attributes



Example Mapping of Aggregations

- Foreign key to aggregation is key of aggregated relationship





Learning Outcomes

- **The Relational Model**
  - Design a relational schema for a simple use case
  - 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
  - Map an ER diagram to a relational database schema
- **Key topics:**
  - **Relations (schemas, instances, cardinality, degree)**
  - **NULL values**
  - **Integrity constraints**
    - **Keys (candidate, primary, foreign, super, composite keys)**
    - **Domain constraints (NOT NULL, data types)**
  - **SQL DDL (CREATE/DROP TABLE)**

References

- Ramakrishnan/Gehrke (3rd edition - the ‘Cow’ book)
  - Chapter 3.1-3.5, plus Chapter 1.5
- Kifer/Bernstein/Lewis (2nd edition)
  - Chapter 3
  - Chapter 4.5 for ER-diagram mappings
- Molina/Ullman/Widom (2nd edition)
  - Chapter 2.1 - 2.3, Section 7.1 – 7.3
  - Chapter 4.5 – 4.6 for ER-diagram mappings
  - *foreign keys come later, instead relational algebra is introduced very early on; also briefly compares RDM with XML*
- *Oracle 12c SQL Language Reference*
  - http://docs.oracle.com/cd/E16655_01/server.121/e17209/toc.htm

Next Week

- The Structured Query Language
- Relational Algebra
- Foundations of Declarative Querying
  - Relational Algebra
    - a formal query language for the relational data model

- Readings (choose one):
  - Ramakrishnan/Gehrke
    - Chapter 5.1-5.6 & Section 4.2
  - Kifer/Bernstein/Lewis
    - Chapter 5
  - Molina/Ullman/Widom
    - Chapter s 5.1-5.2 and 6.1 – 6.2

TO BE CONTINUED....