COMP9120

Week 3: Logical Database Design

Semester 1, 2025

Professor Athman Bouguettaya
School of Computer Science

THE UNIVERSITY OF SYDNEY

Contents co-developed by Dr Lijun Chang

Breaking the ice

THE UNIVERSITY OF SYDNEY

Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal people of the Eora nation and pay my respects to their Elders, past, present and emerging.

Copyright warning

Outline

- Logical Database Design
  - Relational model (relation and schema)
  - Data Definition Language (DDL) - Subset of SQL
  - Integrity constraints (domain, key and null constraints)
  - Mapping E-R diagrams to relations: rules

THE UNIVERSITY OF SYDNEY

Database Design Sequence

1. Requirements Analysis	Understand… ▸ what data needs to be stored ▸ what applications must be built ▸ what operations are most frequent
2. Conceptual Design	Develop… ▸ high-level description of the data closely matching how users think of the data ▸ Works as communication vehicle
3. Logical Design	Convert… ▸ conceptual design into a logical database schema **Today**
4. Schema Refinement	Refine… ▸ Identify problems in current schema & refine
5. Physical Design	Convert… ▸ logical schema into a physical schema for a specific DBMS and tuned for app.
6. App & Security Design	Determine who plays what roles, in what workflows, with security settings … ▸ What roles are played by different system entities in system processes, and what permissions should be given to these roles?

Relational Data Model

THE UNIVERSITY OF SYDNEY

Relational Data Model

› First proposed by the late 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** (equivalent to the Nobel Prize) in **1981**.*

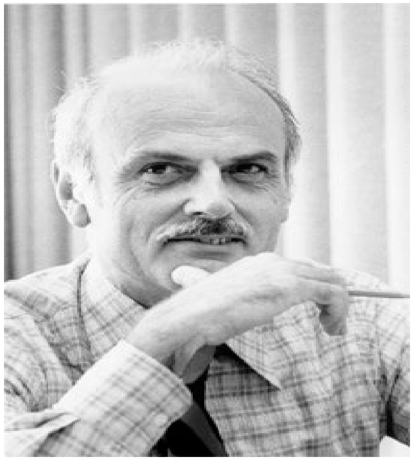


Photo of Edgar F. Codd

› Before 1970

- Various *computer hardware-centric models*: *hierarchical* and *network* models
- Writing queries was a very *elaborate* task that required *expertise* in *computer hardware*

› Since 1970

- Using a database has far more *general user-friendly:* **Relational model**
  - *Simple* data representation targeted at *non-experts* in computer hardware
    - *Powerful* model: Provides the ability to express *complex queries*
  - Has a *solid mathematical* foundation: *set theory* and *mathematical functions*
  - Adopted by the *major DBMS vendors*

Informal Definition of a Relation

- A ***relation*** is a *named, two-dimensional table* of data
  - Table consists of **rows** (called *tuple* or *record*) and **columns** (called *attribute* or *field*)

Relation name (also: Table name)

Attributes (also: columns, fields)

Tuples (rows, records)

Student

sid	name	login	gender	address
5312666	Jones	ajon1121	m	123 Main St
5366668	Smith	smith554	m	45 George
5309650	Jin	ojin4536	f	19 City Rd

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

THE UNIVERSITY OF SYDNEY

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).
    - The *restriction* of *atomic attributes* is also known as **First Normal Form (1NF).**
      - (Normal forms will be covered in Week 8)
  - A *relation* is a ***set*** of tuples (rows), so:
    - *every row is unique* (*cannot* have *two rows* with *exactly the same* values for all their fields)
    - the *order* of the rows is *immaterial/irrelevent*

Example

› Is this a correct relation? If not, how many errors are there?

name	name	gender	address	phones
Peter	Pan	M	Neverland	0403567123
Dan	Murphy	M	Alexandria	0267831122 0431567312
Jin	Jiao	F	Darlington, Sydney	
Sarah	Sandwoman	F	Glebe	0287898876
Peter	Pan	M	Neverland	0403567123

RDBMS Table Extends Mathematical Relation

- Relational DBMS (RDBMS) **implementations** of the relational model modify the mathematical **relation,** called **table** as follows:
  - RDBMS allows ***duplicate*** rows
  - RDBMS support ***ordering*** tuples and attributes
  - RDBMS allows *‘**null**’* values for *unknown* information

The Special NULL 'Value'

- RDBMS allows a special value ***NULL*** in a column to represent facts that are *not relevant/applicable*, or *not yet known*
  - Not known yet: e.g., a new student has *not* chosen their courses yet
  - Not relevant/applicable: e.g., the attribute *annual_salary* may *not* be *meaningful* or *relevant* for *adjunct lecturers*

lname	fname	annual_salary	birth	hired
Jones	Peter	35000	1970	1998
Smith	Susan	null	1983	1996
Smith	Alan	35000	1975	2000

THE UNIVERSITY OF SYDNEY

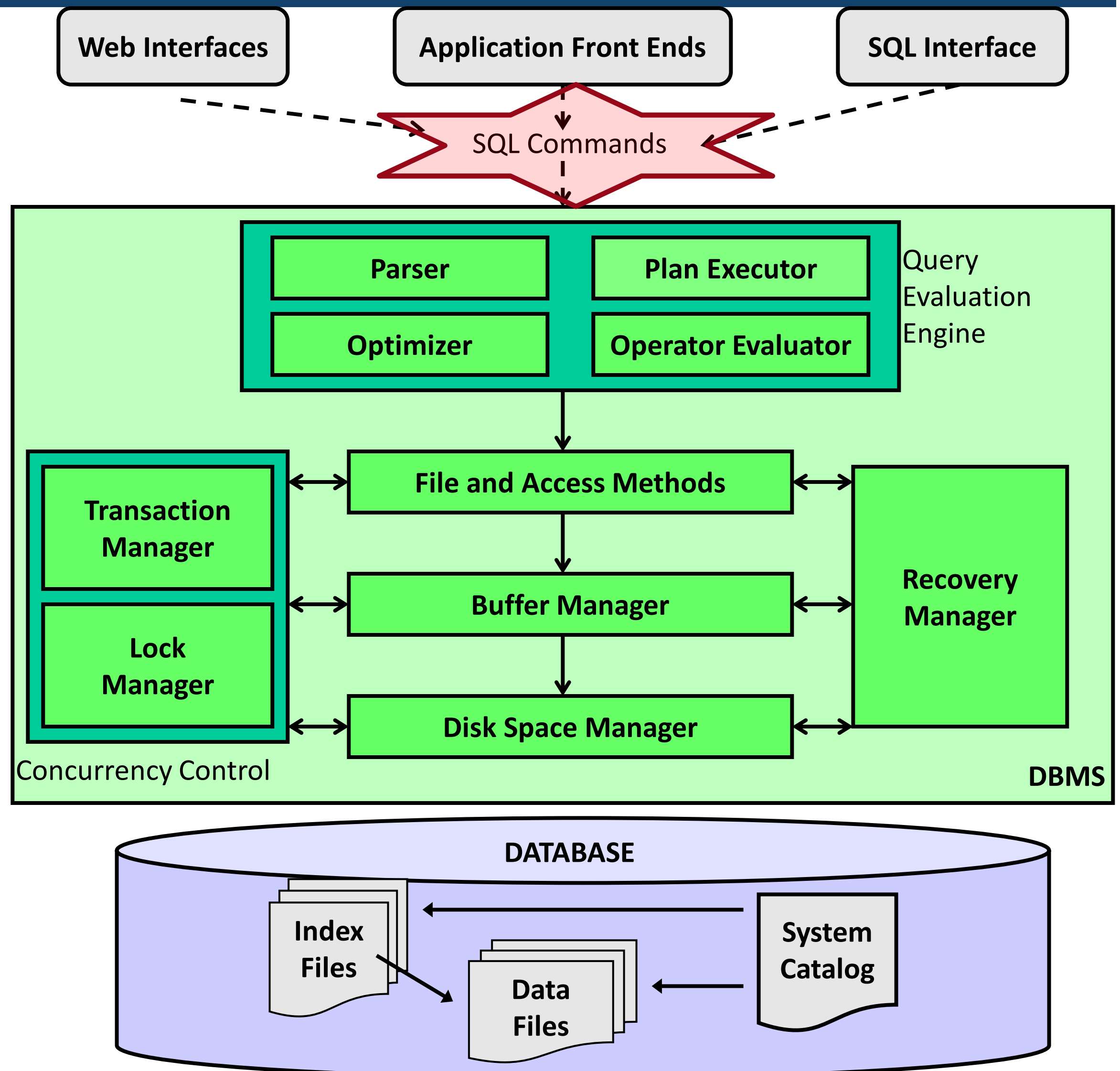
Advantage/Disadvantage of NULL values

- Advantage:
  - NULL can be useful because using an ordinary value with a specific meaning may not always work.
    - E.g., if we set salary = -1 to indicate that the salary is “unknown” as in the previous example, then calculating the *average* of salaries would *not* be correct.
- Disadvantage:
  - NULL causes may cause *complications* in the definition of many operations
  - We shall *ignore* the effect of null values for *now* and will consider their effects *later*

Data Definition Language

SQL for Interacting with RDBMS

- SQL (Structured Query Language) is the standard language for interacting with RDBMS. SQL consists of:
  - **Data Definition Language (DDL)**
    - subset of SQL that supports the ***creation***, ***deletion*** and ***modification*** of tables.
  - Data Manipulation Language (DML)
  - Data Control Language (DCL)



Creating Tables in SQL

- Creation of tables / relations:

  CREATE TABLE *name* (*list-of-columns*);

  - Example: Create the *Student* table

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

  - This specifies the **schema information**
  - Note that the type (domain) of each field is *enforced* by the DBMS whenever tuples are *added* or *modified*.

CHAR	VARCHAR
Used to store strings of fixed size	Used to store strings of variable length
Pads spaces to the right when storing strings less than the maximum size length	No padding needed as it is variable in size

Basic Data Types of ANSI SQL

Base Datatypes	Description	Example Values
SMALLINT (2 bytes) INTEGER (4 bytes) BIGINT (8 bytes)	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

https://www.postgresql.org/docs/current/datatype.html

Example: Creating Tables in SQL

Student	
sid	name

Enrolled		
sid	ucode	semester

UnitOfStudy		
ucode	title	credit_pts

```
CREATE TABLE Student (
    sid    INTEGER,
    name VARCHAR(20)
);
CREATE TABLE Enrolled (
    sid INTEGER, ucode CHAR(8), semester VARCHAR(10)
);
CREATE TABLE UnitOfStudy (
    ucode CHAR(8),
    title VARCHAR(30),
    credit_pts INTEGER
);
```

Table Deletion

› *Deletion* of tables:

DROP TABLE *name ;*

- *Both* the schema information and the tuples are deleted.
- Example: Delete the *Student* relation

DROP TABLE Student ;

Table Modification

› Existing tables can be changed

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

- Huge variety of vendor-specific options; see online documentation
  https://www.postgresql.org/docs/current/ddl-alter.html

Rename column:

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

Add columns:

ALTER TABLE countries **ADD COLUMN** duty_pct **NUMERIC**(4,2),
ADD COLUMN visa_needed **VARCHAR**(3);

Inserting tuples

- *Insertion* of tuples 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);

Modifying tuples

- *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*] ;
    - If **WHERE** clause is omitted, *all tuples are deleted*!
  - Example: **DELETE FROM Student WHERE name = ‘Smith’ ;**

Introduction to Integrity Constraints

Integrity Constraints

- Integrity Constraints (IC): they specify a *set of rules* to maintain the *integrity of the data* (i.e., *correctness*) when it is *manipulated*
  - An IC is a condition that must **hold true** for *any* instance of the database.
    - E.g., *domain constraints*

- ICs are *declared* in the schema
  - They are specified when a schema is *defined*.
  - Declared ICs are *checked* when relations are *modified*.

- A *legal* instance of a relation satisfies *all* specified ICs.
  - If ICs are *declared*, DBMS will *not* allow *illegal* instances.
    - Stored data is more *faithful* to real-world meaning.
    - Avoids *data entry errors*, too!

Key & Foreign Key

- **Key**: Refers to the ***minimal*** set of attributes in a relation that ***uniquely*** identify *each row* of that relation
  - Examples: *employee id*, *tax file numbers*, etc. This is how we can guarantee that all *rows* are *unique*.
  - Keys can be **simple** (*single* attribute) or **composite** (*multiple* attributes)

- A set of attributes is a ***key*** for a relation *if and only if*:
  1. *No two* distinct tuples in a legal instance can have the *same values* in *all* key fields, and *uniqueness*
  2. This is *not true* for *any subset* of the key *minimality*

If 2. is *not true* (i.e., false), then the key is called a ***superkey***!

- E.g., *sid* is a *key* for Student. However,
  - Is *name* a key? No
  - How about the set {*sid, name*}? This *is* a superkey. Why?

    Because the set {*sid, name*} is *not* minimal!

ICs for Avoiding Duplicate Rows

› In an 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*

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

How to Avoid Duplicate Rows?

Two scenarios:

- If a key is specified for a table,
  - is it possible for the table to contain two identical rows?
    - No
- If no key is specified for a table,
  - specify the entire set of attributes as a candidate key by the **UNIQUE** clause.

Key & Foreign Key

- If there are more than one key for a relation, we call each one of them a ***candidate key***. The *candidate key* chosen by the database administrator (DBA) is called the ***primary key (PK)***.
  - If we just say **key**, this usually refers to ***candidate key.*** This will depend on the context.

- **Foreign keys (FK)** are identifiers that enable a <u>*dependent relation*</u> to refer to its <u>*parent relation*</u>
  - Must refer to a *candidate key* of the *parent* relation
  - Acts like a `logical pointer’

Example for Key & Foreign Key

Primary key identifies each tuple of a relation, underlined by a solid line

Composite Primary Key consisting of more than one attribute.

Student	
sid	name
31013	John

Enroll		
sid	ucode	semester
31013	C9120	2023S2

Units_of_study		
ucode	title	credit_pts
C9120	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'), underlined by a dashed line

Key & Foreign Key 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 **UNIQUE** clause lists attributes that comprise a *candidate key*.
  - The **FOREIGN KEY** clause lists the attributes that comprise the *foreign key* and the name of the relation referenced by the foreign key.

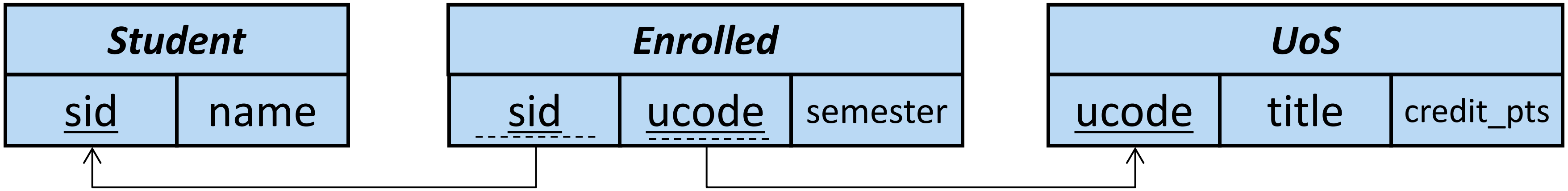
- *By default*, a *foreign key* references the *primary key* attributes of the *referenced table*

  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)

Example: Primary & Foreign Keys




```
CREATE TABLE Student ( sid INTEGER, name VARCHAR(20) ,
    PRIMARY KEY (sid)
);
CREATE TABLE UoS  ( ucode CHAR(8), title VARCHAR(30), credit_pts INTEGER ,
    PRIMARY KEY (ucode)
);
CREATE TABLE Enrolled ( sid INTEGER, ucode CHAR(8), semester VARCHAR,
    FOREIGN KEY (sid) REFERENCES Student,
    FOREIGN KEY (ucode) REFERENCES UoS,
    PRIMARY KEY (sid,ucode)
);
```

THE UNIVERSITY OF SYDNEY

Foreign Key Constraint

› **Foreign Key Constraint (Referential Integrity)**:
For each tuple in the referring relation whose foreign key value is **A**, there must be a tuple in the referred relation with a candidate key that also has value **A**

- e.g. Enrolled(*sid*: integer, ucode: string, semester: string)
  where *sid* is a *foreign key* referring to Student:

Student

sid	name
31013	John

Enrolled

sid	ucode	semester
31013	C9120	2023S2

Units_of_study

ucode	title	credit_pts
C9120	DB Intro	4

NOT NULL constraint

- RDBMS by default allows a special entry ***NULL*** in a column to represent facts that are not relevant, or not yet known

- For certain applications, it is essential to specify that *no value* in a given column can be *NULL*
  - E.g., the value can't be unknown (i.e., should be known) or the concept can't be inapplicable (i.e., it is applicable)

- In SQL

```
CREATE TABLE Student (
        sid        INTEGER NOT NULL,
        name       VARCHAR(20) NOT NULL,
        login      VARCHAR(20) NOT NULL,
        gender     CHAR,
        birthdate  DATE
);
```

Keys and NULLs

- PRIMARY KEY
  - *At most one* per table, and must be unique
  - Automatically disallows NULL values
- CANDIDATE KEY
  - May be *more than one* per table, and all must be declared as UNIQUE
  - May have NULL values
- FOREIGN KEY
  - By default, allows NULL values
  - If every tuple must have a parent tuple, then must combine the FK with the NOT NULL constraint

Let’s take a break!

Let us also play game!

Mapping E-R Diagrams to Relations

THE UNIVERSITY OF SYDNEY

Rules for Mapping E-R Diagrams to Relational Model

- Recall that an E-R diagram consists for
  - Strong entity sets
  - Weak entity sets
  - Relationship types
    - Key constraints
    - Participation constraints
  - IsA Hierarchies
  - Aggregations

Mapping Strong Entity Sets

- In the **ER-RM** (Relational Model) **mapping**, each **entity set** becomes a **relation**
  - **Columns** correspond to **attributes**
  - **Rows** correspond to **entities**

- Attributes
  - **Simple attributes**

    E-R attributes map directly onto the relation
  - **Composite attributes**

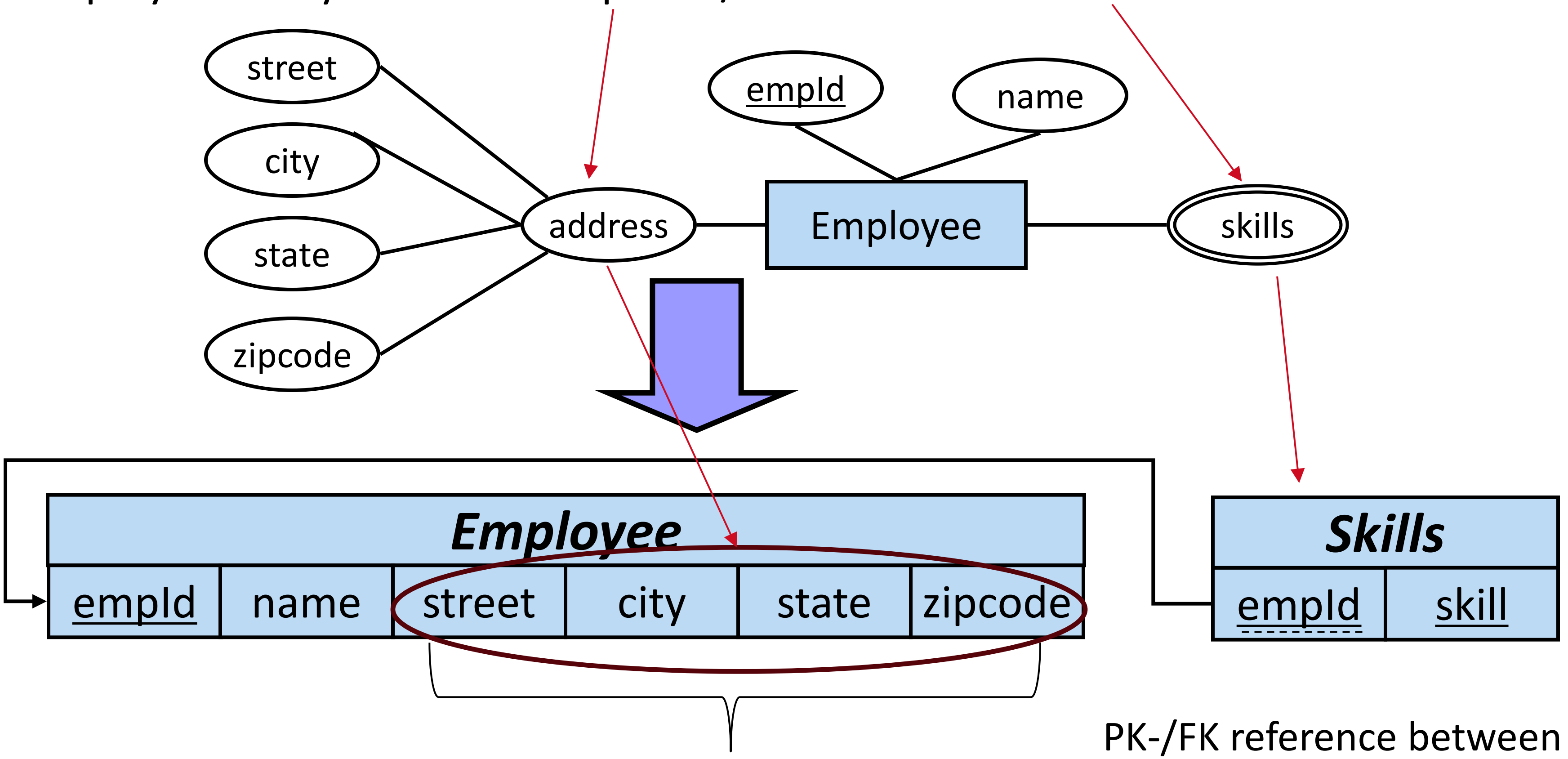
    Composite attributes are *flattened* out by creating a *separate field* for *each component attribute*

    => We use only their simple, component attributes
  - **Multi-valued attribute**

    Becomes a *separate relation* with a *foreign key* taken from the containing entity

Example: Mapping Strong Entity Sets

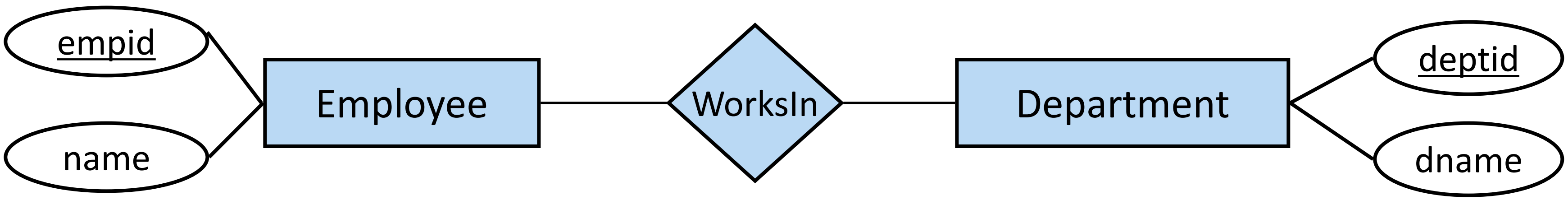
› Employee entity set with composite/multi-valued attributes



Flatten composite attribute into separate attributes

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

Mapping Relationship Types *without* Constraints

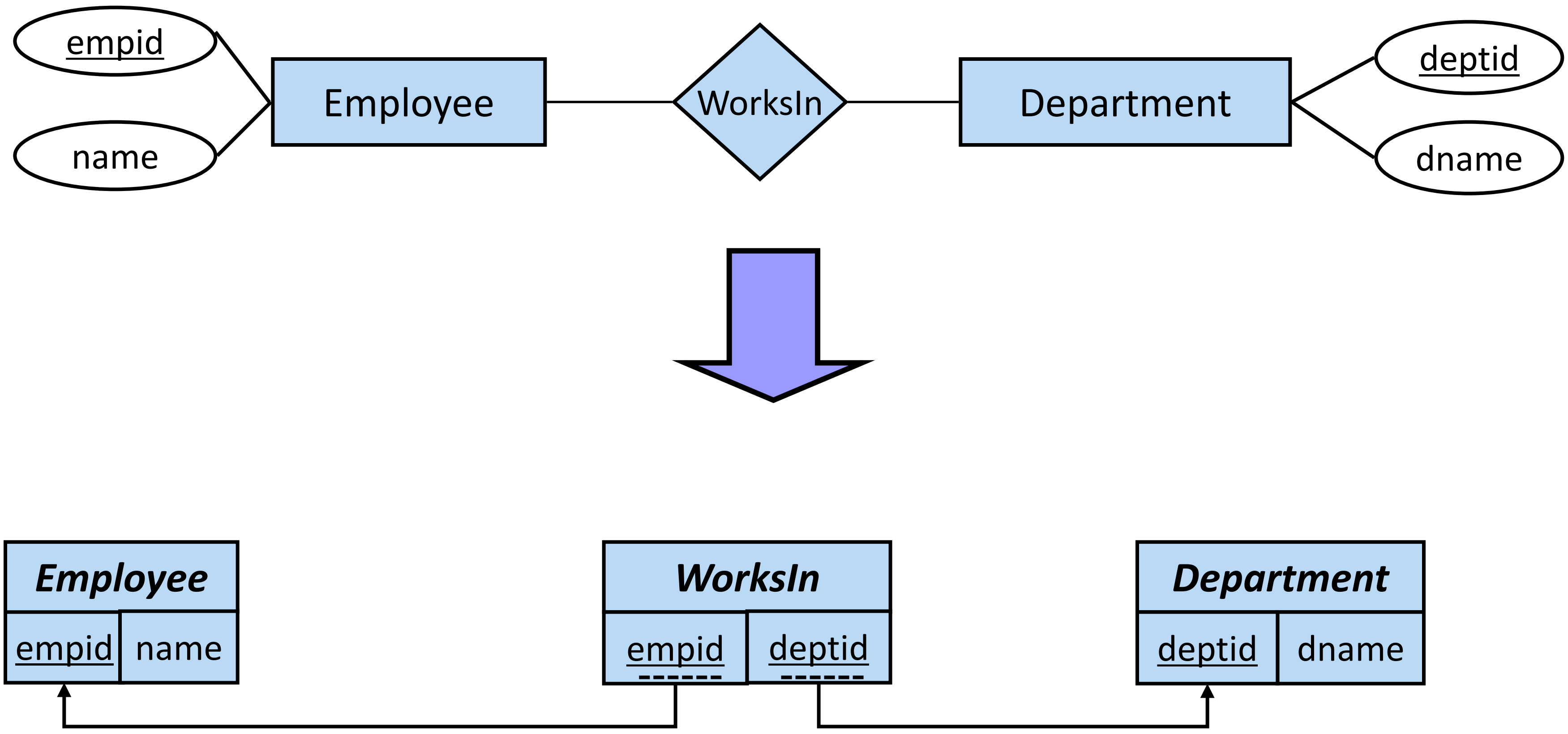


- E-R Model: the *combination* of the *primary keys* of the *participating* entity sets form a **superkey** of a relationship.
  - Is this a **candidate key**?
    - Looking at each relationship side: this is a ***many-to-many*** *relationship*
      - 1 Employee can work in 0 to many Departments
      - 1 Department can have 0 to many Employees

Yes!

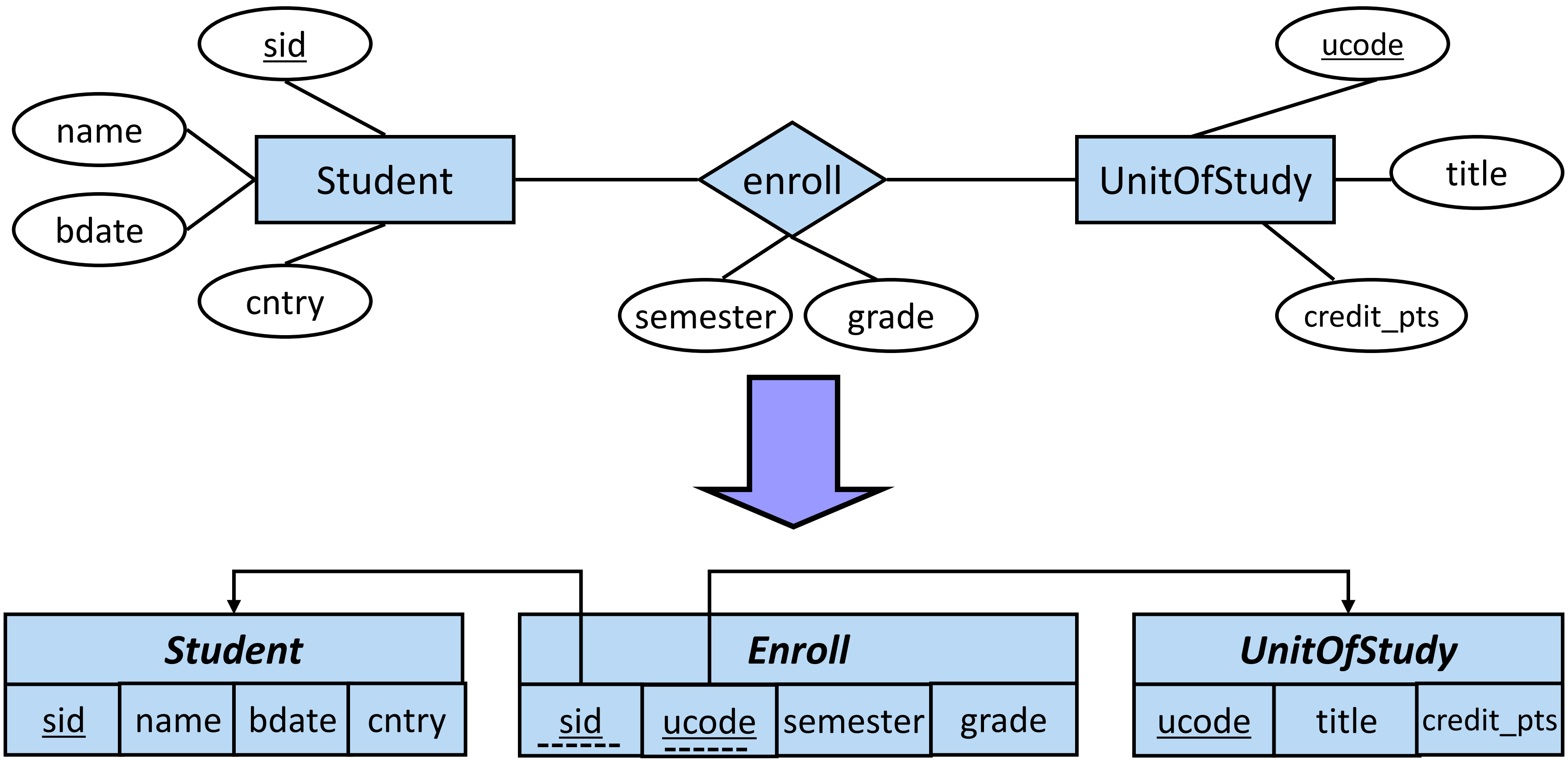
- **Mapping relationship types without constraints** - Create a ***new relation*** with the primary keys of the two participating entity sets as **its primary key**
  - All *relationship attributes* become attributes in this *new relation*

Mapping Relationship Types *without* Constraints

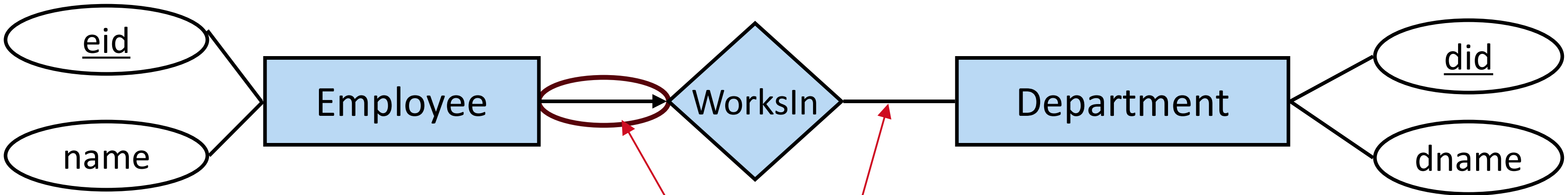


Example: Mapping Relationship Types *without* Constraints

› General relationship between Student & UnitOfStudy



Mapping Relationship Types *with* Key Constraints

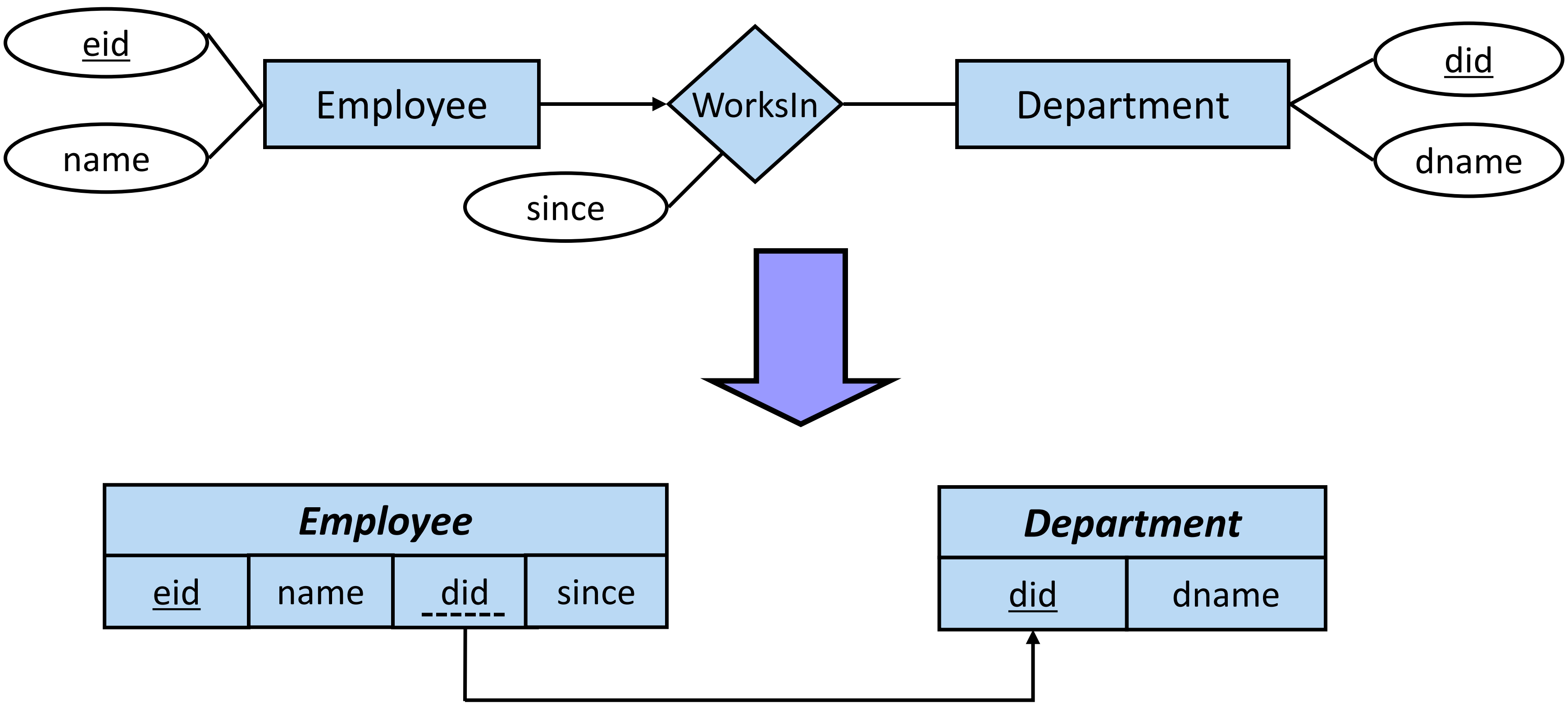


- Looking at each relationship side:
  - 1 Employee works in at most 1 Department
  - 1 Department can have 0 to Many Employees
- The *primary key* of *Employee* is also a *(candidate) key* of WorksIn

- One approach is doing the *same* as mapping relationship types *without* constraints, but *choosing* the correct *primary key*
- A *better approach,* however, is ***combining*** the <u>*relation of the entity set*</u> that *participates* in the key constraint and the <u>*relation of the relationship type*</u>

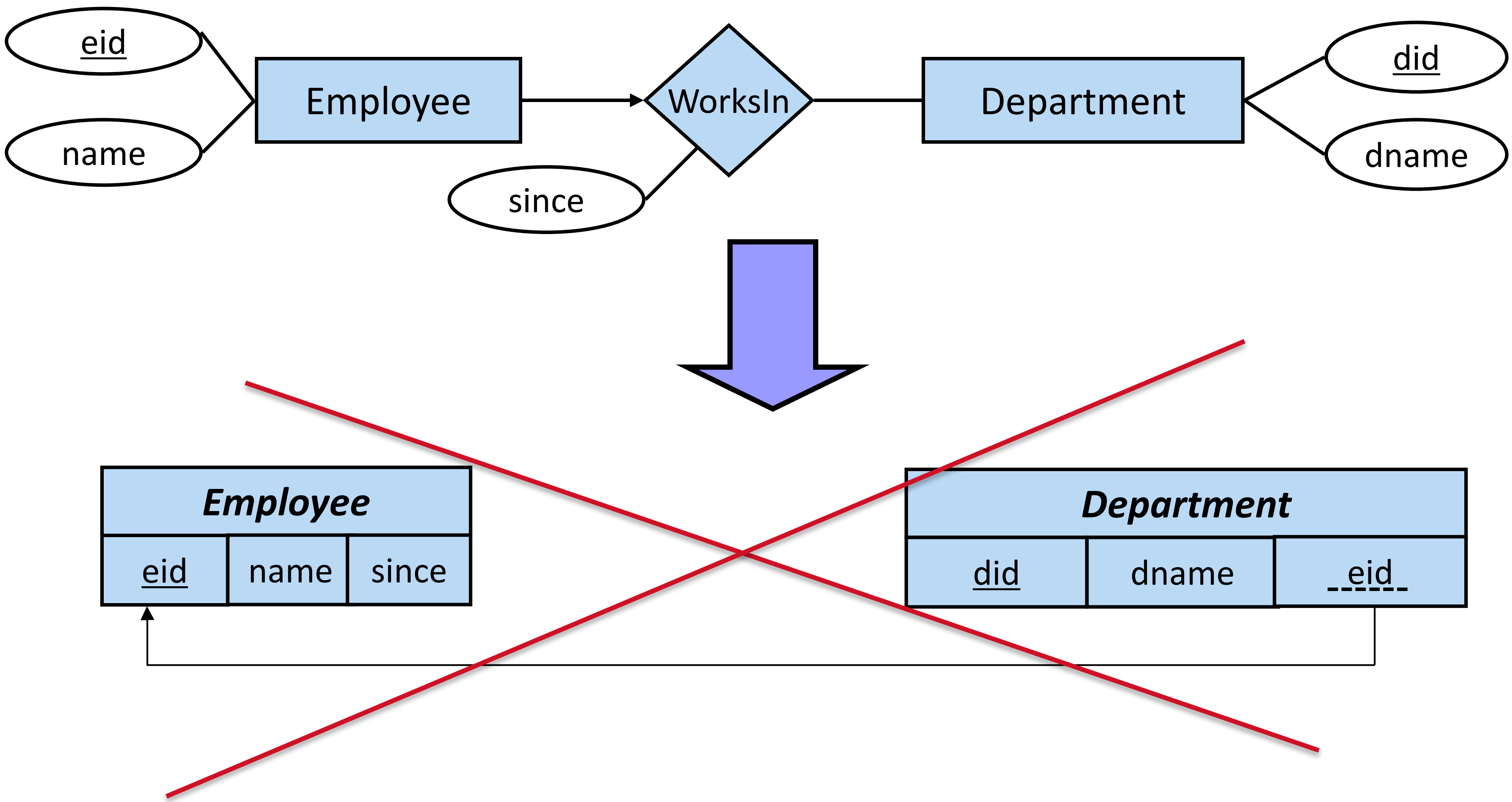
Example: Mapping Relationship Types *with* Key Constraints

› **Relationship with Key Constraint**



Example: Mapping Relationship Types *with* Key Constraints

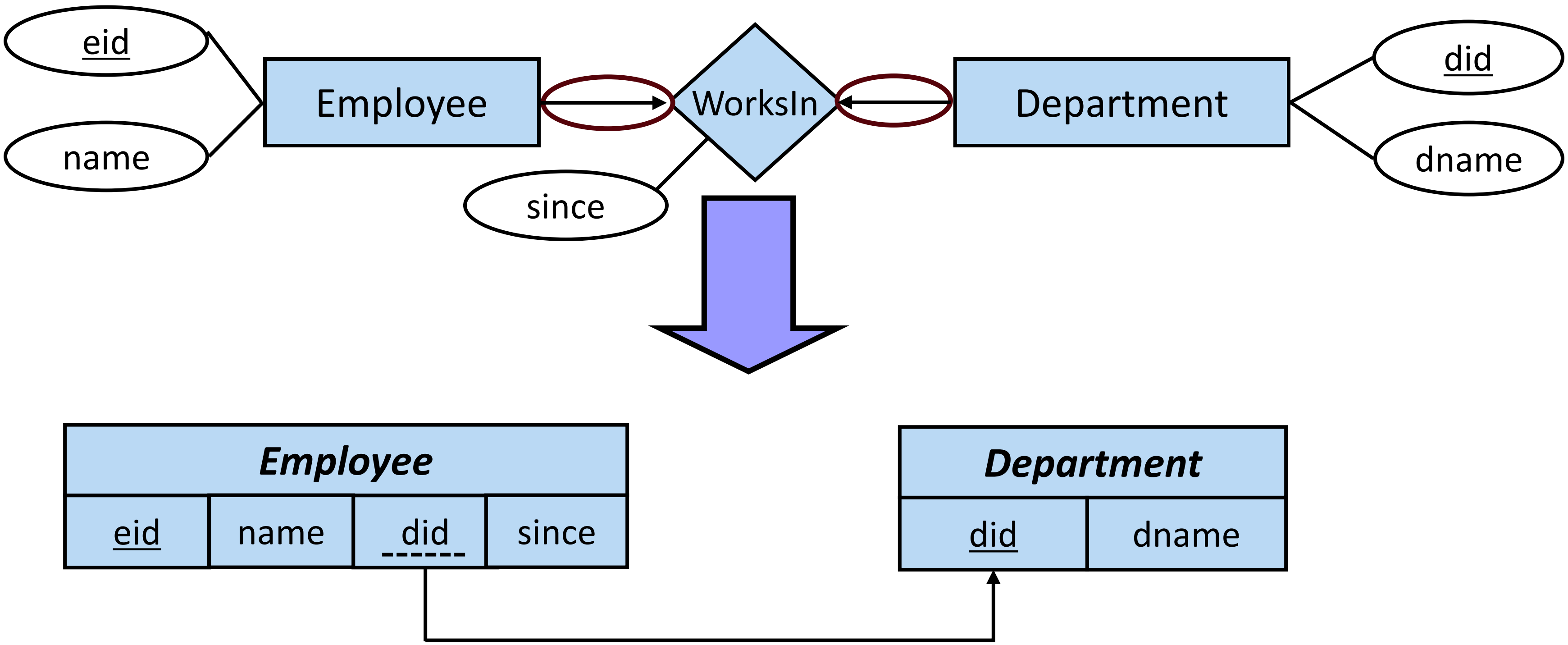
› **Relationship with Key Constraint**



Can we do this?

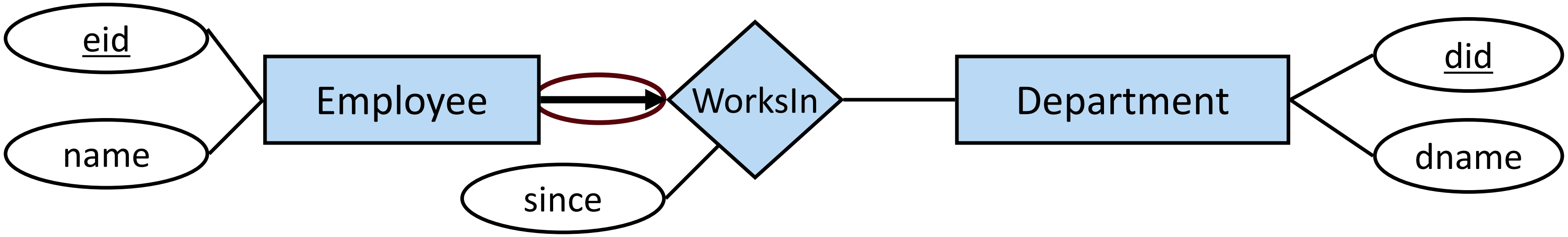
Example: Mapping Relationship Types *with* Key Constraints

- **Relationship with Key Constraints on *both sides***



- Each Employee works in at most one Department
- Each Department has at most one Employee
  - Add ***uniqueness*** constraint to *foreign key*

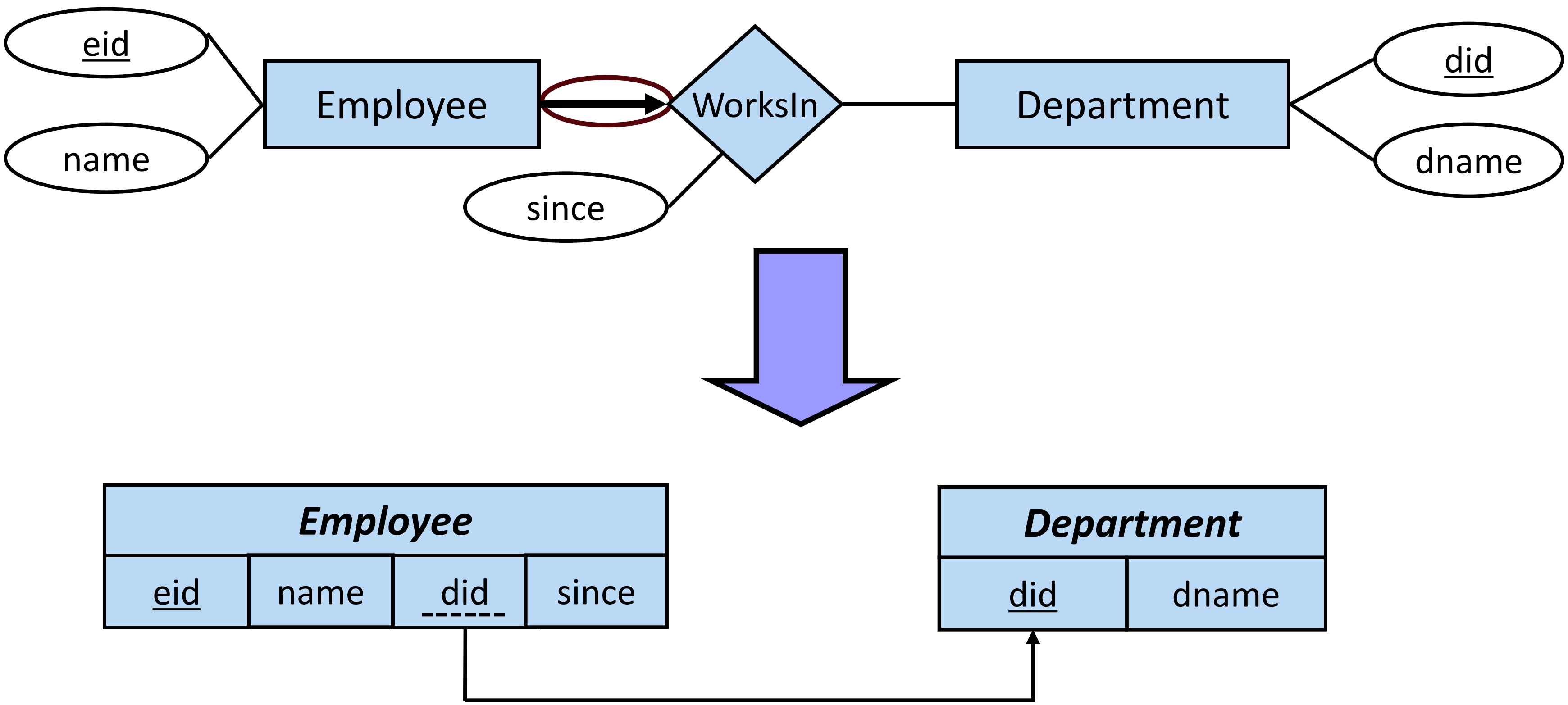
Example: Mapping Relationship Types *with Key* & *Participation* Constraints



- Looking at each relationship side:
  - 1 Employee works in exactly 1 Department (***mandatory*** to have exactly 1 Department.)
  - 1 Department can have 0 to Many Employees
- The primary key of Employee is a (candidate) key of WorksIn

Example: Mapping Relationship Types with *Key* & *Participation* Constraints

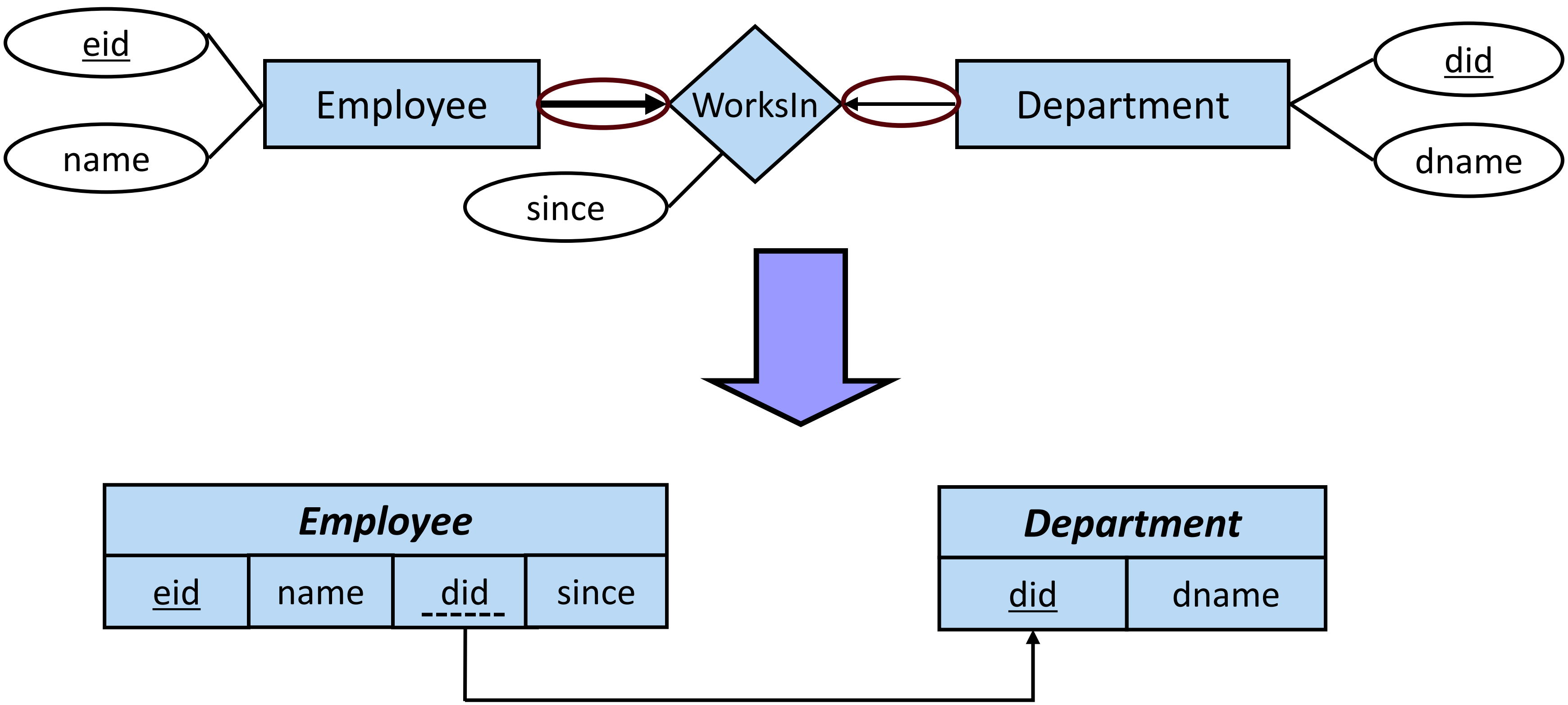
› **Relationship with Key & Participation Constraints**



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

Example: Mapping Relationship Types with *Key* & *Participation* Constraints

› **Relationship with Key (on both sides) & Participation (on one side) Constraints**

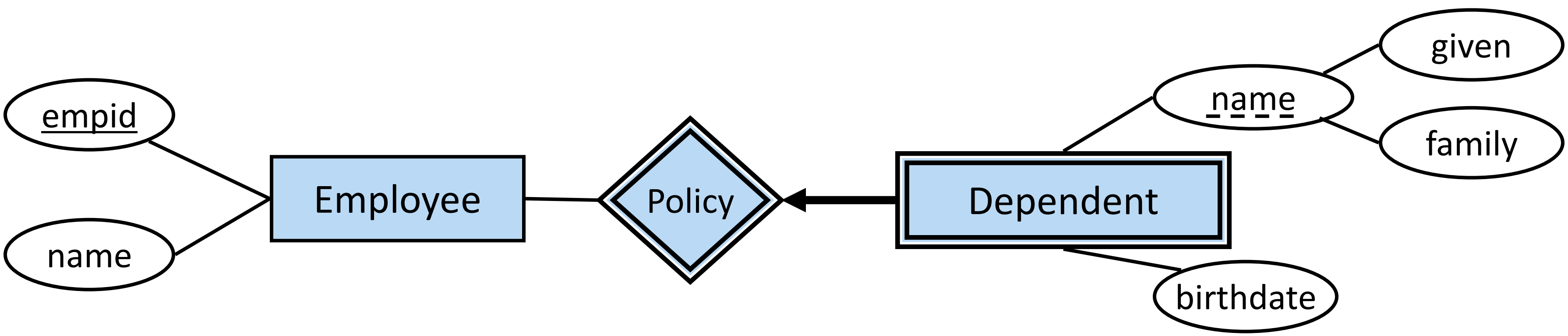


› **Key & Participation Constraint (thick arrow): *NOT NULL*** on foreign key

› Add ***uniqueness*** constraint to foreign key

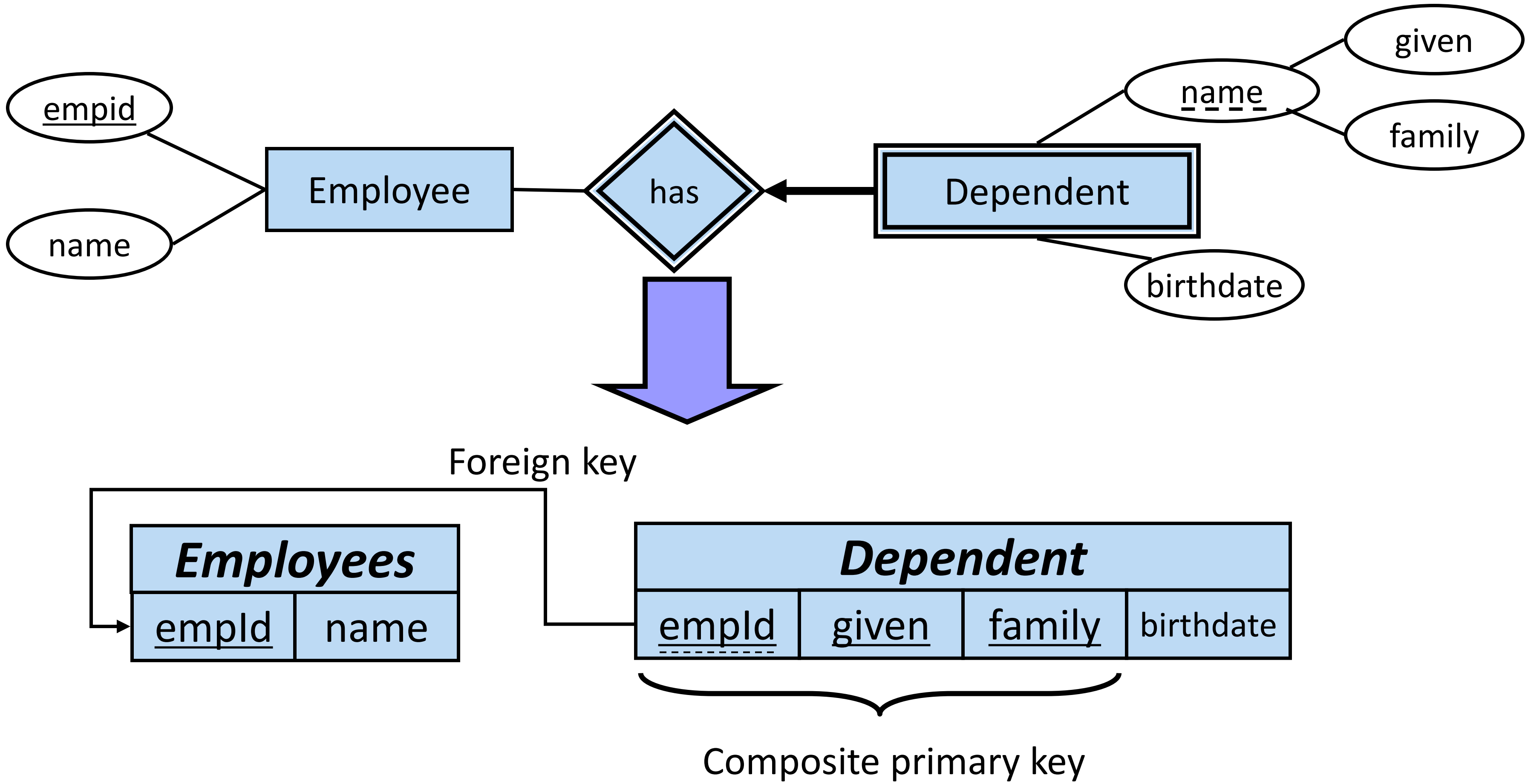
Mapping *Weak Entity* sets

- **Weak Entity Sets** 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



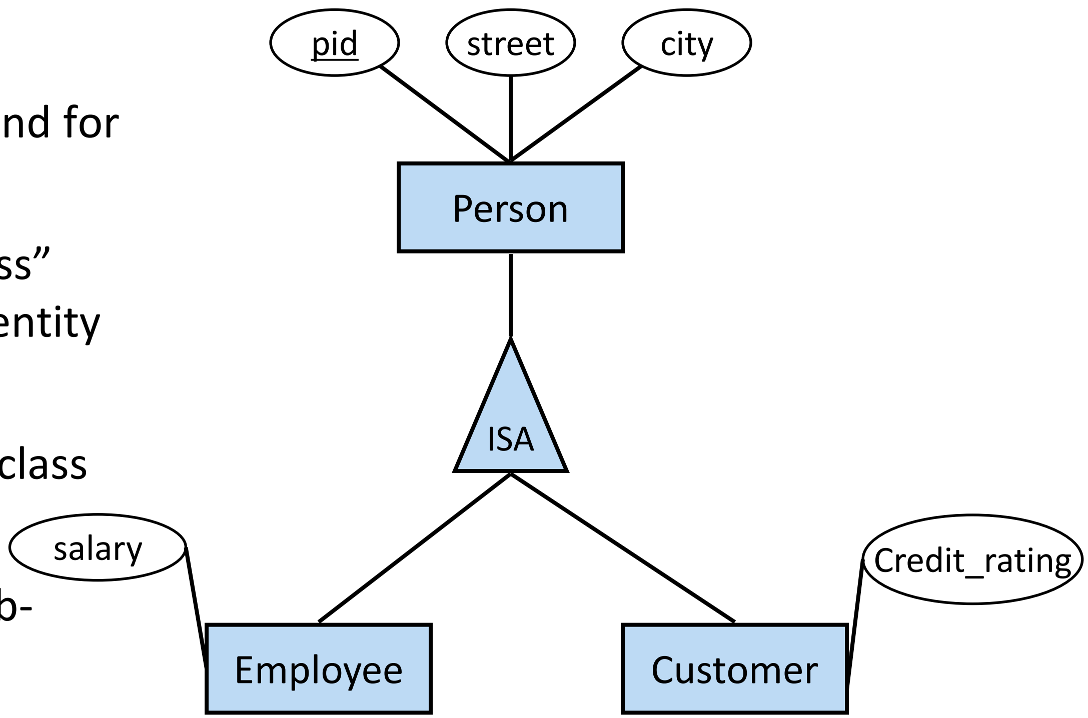
Example: Mapping Weak Entity Sets

- Weak entity type *Dependent* with composite partial key



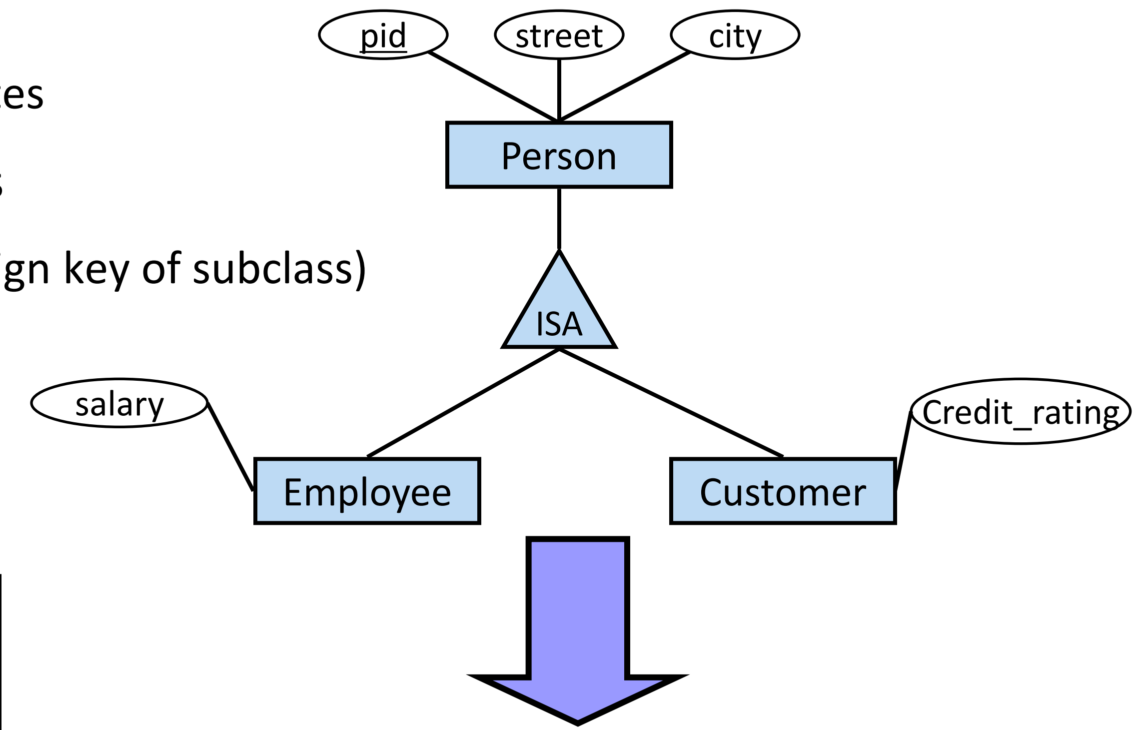
Mapping IsA-Hierarchy

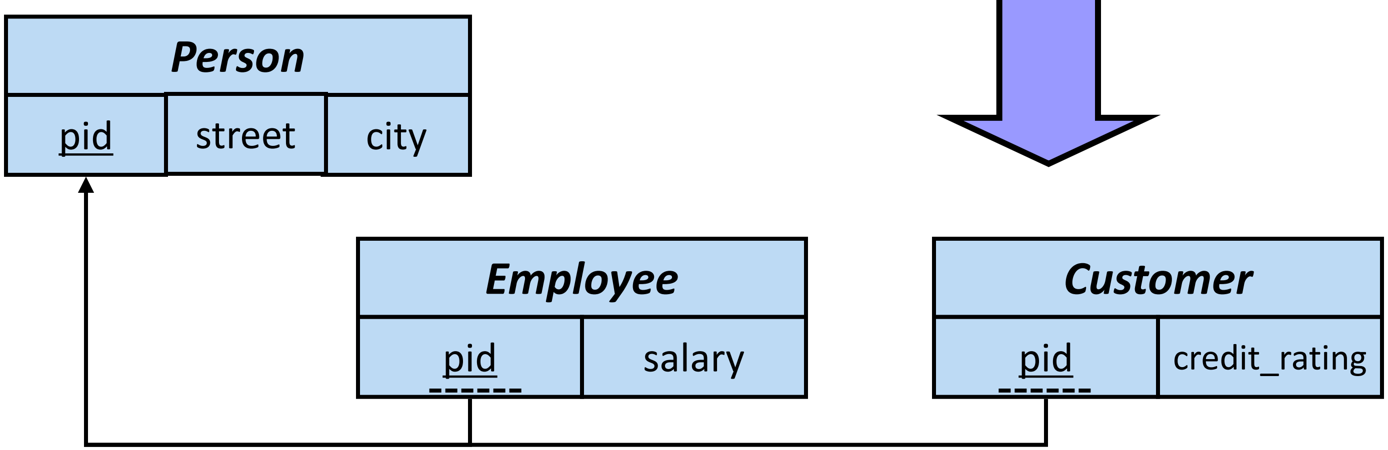
- Standard way: works always, *not all* constraints enforced
  - *Distinct relations* for the superclass and for each subclass
  - Consider each “subclass IsA superclass” separately, in a similar way to weak entity set but without partial key
    - Superclass attributes go into superclass relation
    - Subclass attributes go into each sub-relation; primary key of superclass relation becomes primary key and foreign key of subclass relation



Example: Mapping IsA-Hierarchy

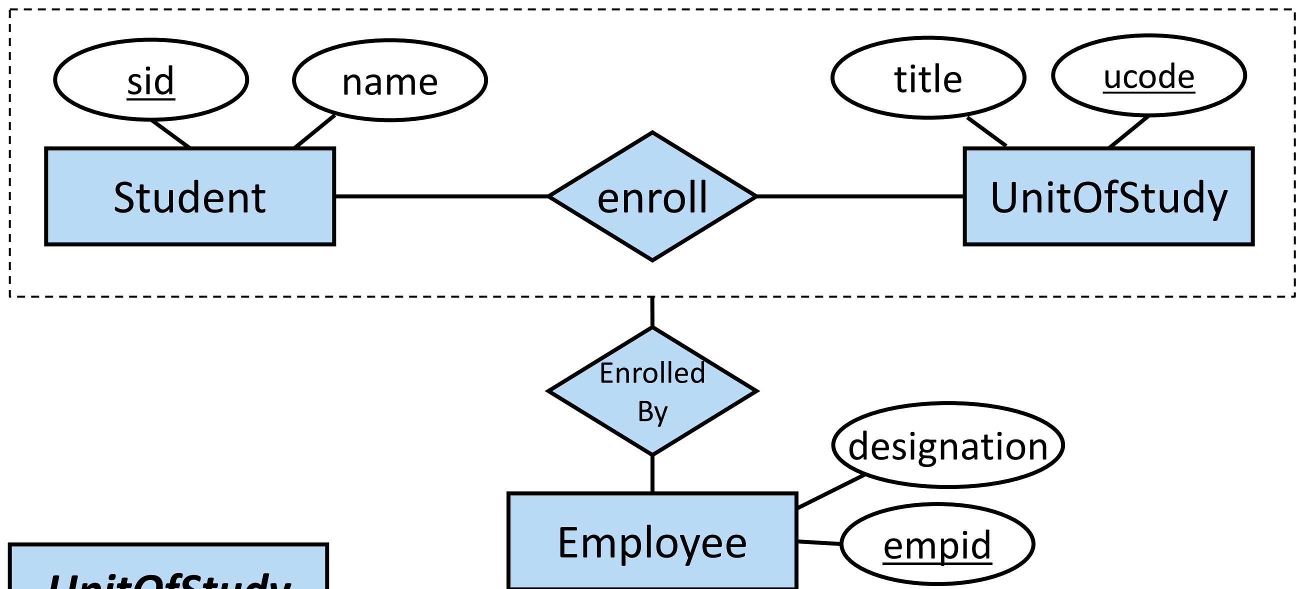
- Table for superclass
  - Primary key + common attributes
- Separate tables for subclasses
  - Primary key of superclass (foreign key of subclass) + specific attributes

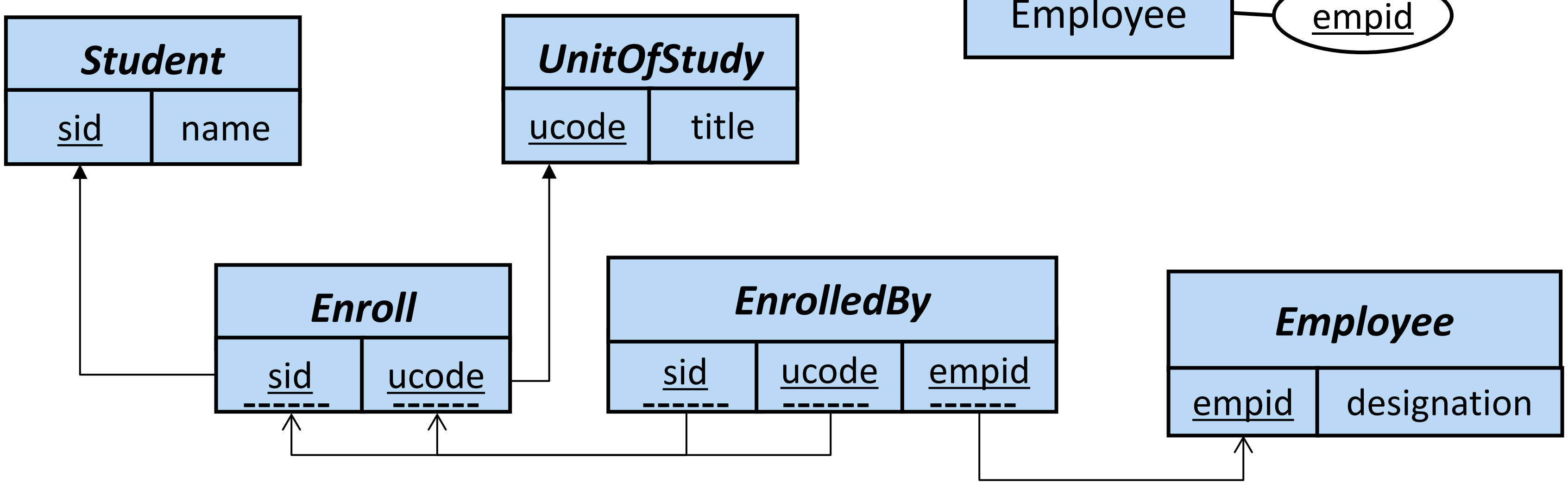




Example: Mapping Aggregations

› *Keys of* aggregation and entity type form the *key* and *foreign keys* of the aggregated relationship.





Learning Outcomes

- **Key topics:**
  - Relations (schemas, instances, etc)
  - NULL values
  - Integrity constraints
    - Keys (candidate, primary, foreign, superkey, composite keys)
    - Domain constraints (NOT NULL, data types)
  - SQL DDL (CREATE/DROP TABLE)
- **The Relational Model**
  - Design a relational schema for a given UoD
  - 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

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*
- *PostgreSQL 17 Language Reference*
  - https://www.postgresql.org/docs/17/index.html

Next Week

- Relational algebra and SQL
  - Relational Algebra
    - A formal query language for the relational data model
  - Data Manipulation Language: The Structured Query Language (SQL)

- Readings:
  - Ramakrishnan/Gehrke
    - **Chapter 5.1-5.6 & Section 4.2**
  - Kifer/Bernstein/Lewis
    - Chapter 5
  - Molina/Ullman/Widom
    - Chapters 5.1-5.2 and 6.1 – 6.2

See you next week!