COMP9120

Week 3: Logical Database Design

Semester 1, 2022

Dr Mohammad Polash
School of Computer Science

THE UNIVERSITY OF SYDNEY

Contents developed by Dr Lijun Chang

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 Burramattagal people and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.

Copyright warning

Outline of Today

- Logical Database Design
  - Relational model (relation and schema)
  - Data definition language (DDL)
  - Integrity constraints
  - Mapping E-R diagrams to relations

Relational Data Model

Relational Data Model

- 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.*

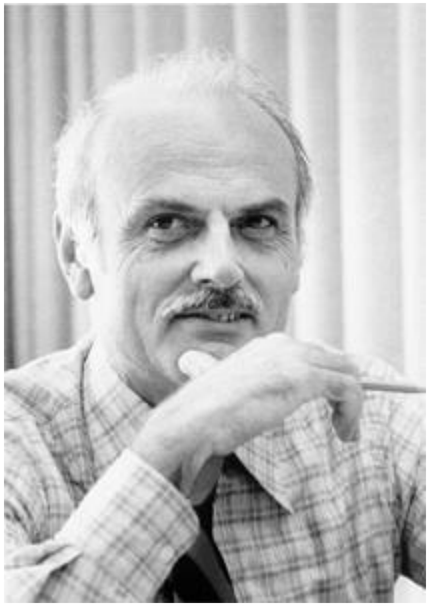


Photo of Edgar F. Codd

- Before 1970
  - Various ad-hoc models: hierarchical model and network model
  - Writing queries is a very elaborate task
- Since 1970
  - Relational model dominants and is the foundation for the leading DBMS products
  - Simple data representation and easy to express complex queries

Relational Data Model

- A database is a collection of one or more **relations**, where each relation is a table with rows and columns
  - This simple tabular representation enables even novice users to understand the contents of a database.
  - It permits the use of simple, high-level languages to query the data.
- The relational model of data is based on the mathematical concept of ***Relation***.
  - Studied in Discrete Mathematics
- Querying relational database has a theoretical foundation: **relational algebra**.

Informal Definition of a Relation

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

Attributes (also: columns, fields)

Tuples (rows, records)

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

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

- Formally, a relation R consists of a relation schema and a relation instance
- 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** is a **set** of tuples (*a table*) for a schema
  - Each tuple has the same number of fields as 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

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)

Example

› Is this a correct relation?

<table>
<tr><th>name</th><th>name</th><th>gender</th><th>address</th><th>phones</th></tr>
<tr><td>Peter</td><td>Pan</td><td>M</td><td>Neverland</td><td>0403567123</td></tr>
<tr><td rowspan="2">Dan</td><td rowspan="2">Murphy</td><td rowspan="2">M</td><td rowspan="2">Alexandria</td><td>0267831122</td></tr>
<tr><td>0431567312</td></tr>
<tr><td>Jin</td><td>Jiao</td><td>F</td><td>Darlington, Sydney</td><td></td></tr>
<tr><td>Sarah</td><td>Sandwoman</td><td>F</td><td>Glebe</td><td>0287898876</td></tr>
<tr><td>Peter</td><td>Pan</td><td>M</td><td>Neverland</td><td>0403567123</td></tr>
</table>

RDBMS Table Extends Mathematical Relation

- RDBMS **table** extends mathematical **relation**
  - 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 and 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

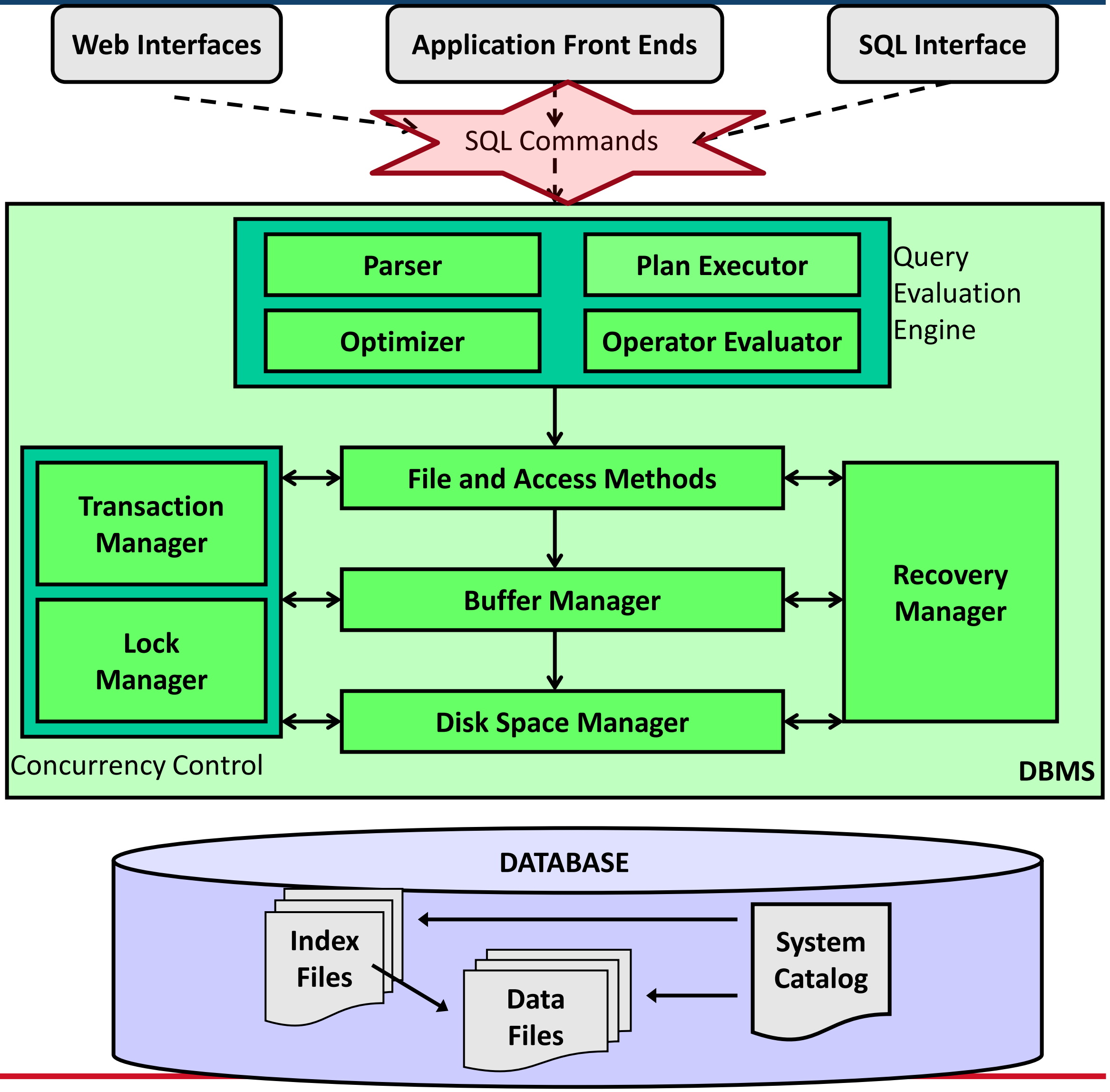
- Con:
  NULL causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effects later

Data Definition Language

THE UNIVERSITY OF SYDNEY

SQL for Interacting with RDBMS

- SQL (structured query language) is the standard language for interacting with RDBMS
  - **Data definition language (DDL)**
    - the 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 specified and enforced by the DBMS whenever tuples are added or modified.

- Several base data types available in ANSI 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
  - Always check the documentation https://www.postgresql.org/docs/9.5/static/datatype.html

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

Example: 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(10)
);
```

Table Deletion

› Deletion of tables:

DROP TABLE ***name ;***

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

DROP TABLE Student ;

Table Modification

- Existing schemas can be changed

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

  - Huge variety of vendor-specific options; see online documentation
    https://www.postgresql.org/docs/9.5/static/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(2,2),
                      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*] ;
  - Example: **DELETE FROM Student WHERE name = ‘Smith’ ;**

Integrity Constraints

Integrity Constraints

- Integrity Constraint (IC): facilities to specify a variety of rules to maintain the integrity of data when it is manipulated
  - A condition that must be true for *any* instance of the database; e.g., *domain constraints.*

- ICs are 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.
  - Stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Key & Foreign Key

- **Key**: the minimal set of attributes in a relation that can uniquely identify each row of that relation
  - Examples include employee id, social security 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 :
  - 1. No two distinct tuples in a legal instance can have the same values in all key fields,

  and

  - 2. This is not true for any subset of the key.
  - Part 2 false? A ***superkey***.
- E.g., *sid* is a key for Student.
  - What about *name*?
  - And the set {*sid, name*}? This is a superkey.

Key & Foreign Key

- If there's at least one key for a relation, we call each of them a ***candidate key***, and one of the keys is chosen (by DBA) to be the ***primary key (PK)***.
  - If we just say **key**, we typically mean ***candidate key***

- **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
  - 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	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'), 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)

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

  CONSTRAINT Student_PK PRIMARY KEY (sid)

Example: Primary & Foreign Keys

Student	
sid	name

Enrolled		
sid	ucode	semester

UoS		
ucode	title	credit_pts

```
CREATE TABLE Student ( sid INTEGER, name VARCHAR(20) ,
    CONSTRAINT Student_PK PRIMARY KEY (sid)
);
CREATE TABLE UoS  ( ucode CHAR(8), title VARCHAR(30), credit_pts INTEGER ,
    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)
);
```

Key Constraint

- No two distinct tuples can have the same values in all key attributes
- Careful: If 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) );
```

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

vs.

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

Foreign Key Constraint

› **Foreign Key Constraint (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. Enrolled(*sid*: integer, ucode: string, semester: string)
  sid is a foreign key referring to Student:

sid	ucode	semester
1234	COMP9120	2020S2
3456	COMP9120	2020S2
5678	COMP9120	2021S1
5678	COMP9007	2020S2

Q: What can we say about the Student relation?

ICs to Avoid 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
  - 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

How to Avoid Duplicate Rows?

- If at least one 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.

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 important to specify that no value in a given column can be NULL
  - E.g., the value can't be unknown, the concept can't be inapplicable

- 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
  - Up to one per table, and must be unique
  - Automatically disallow 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 NULL values
  - If there must be a parent tuple, then must combine with NOT NULL constraint

Let’s take a break!

THE UNIVERSITY OF SYDNEY

Mapping E-R Diagrams to Relations

THE UNIVERSITY OF SYDNEY

Mapping E-R Diagrams to Relations

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

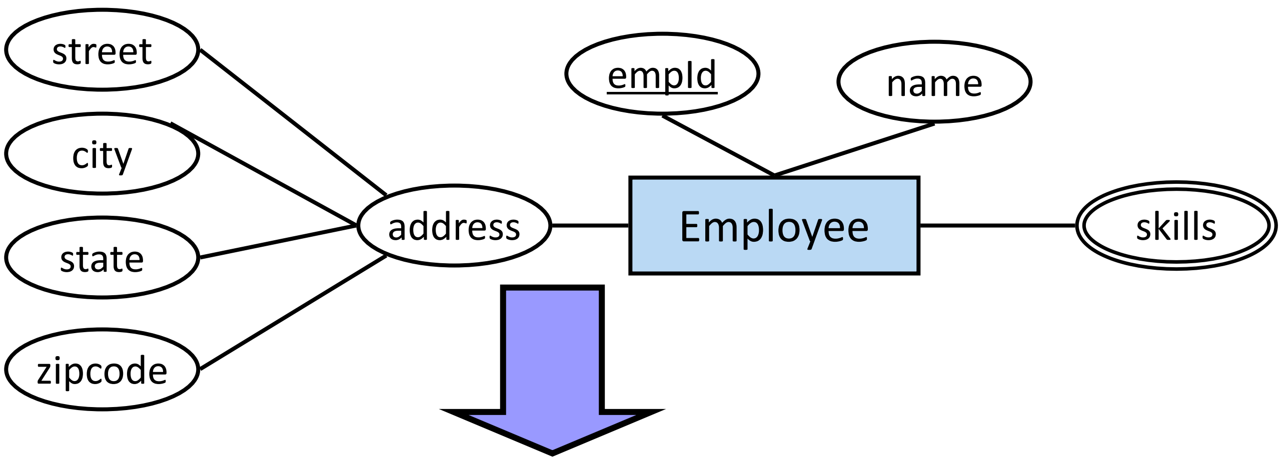
Mapping Strong Entity Sets

- Each **entity set** becomes a relation
  - Columns correspond with attributes
  - Rows correspond with 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 superior entity

Example: Mapping Strong Entity Sets

- Employee entity set with composite/multi-valued attributes



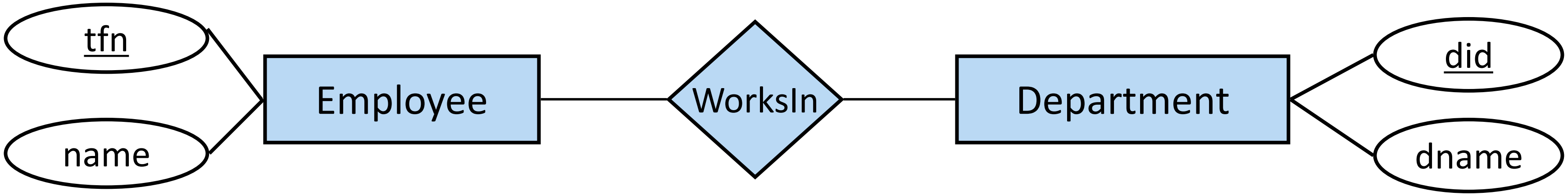
Employee					
empId	name	street	city	state	zipcode

Skills	
empId	skill

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? **Yes!**

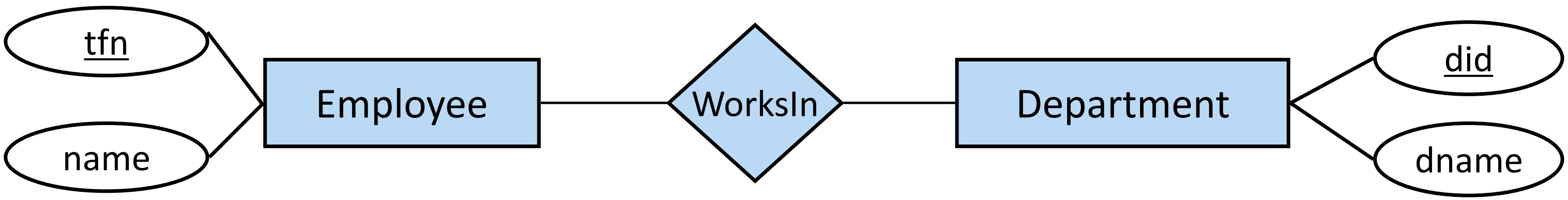
› Looking on each relationship side: this is a ***many-to-many*** *relationship*

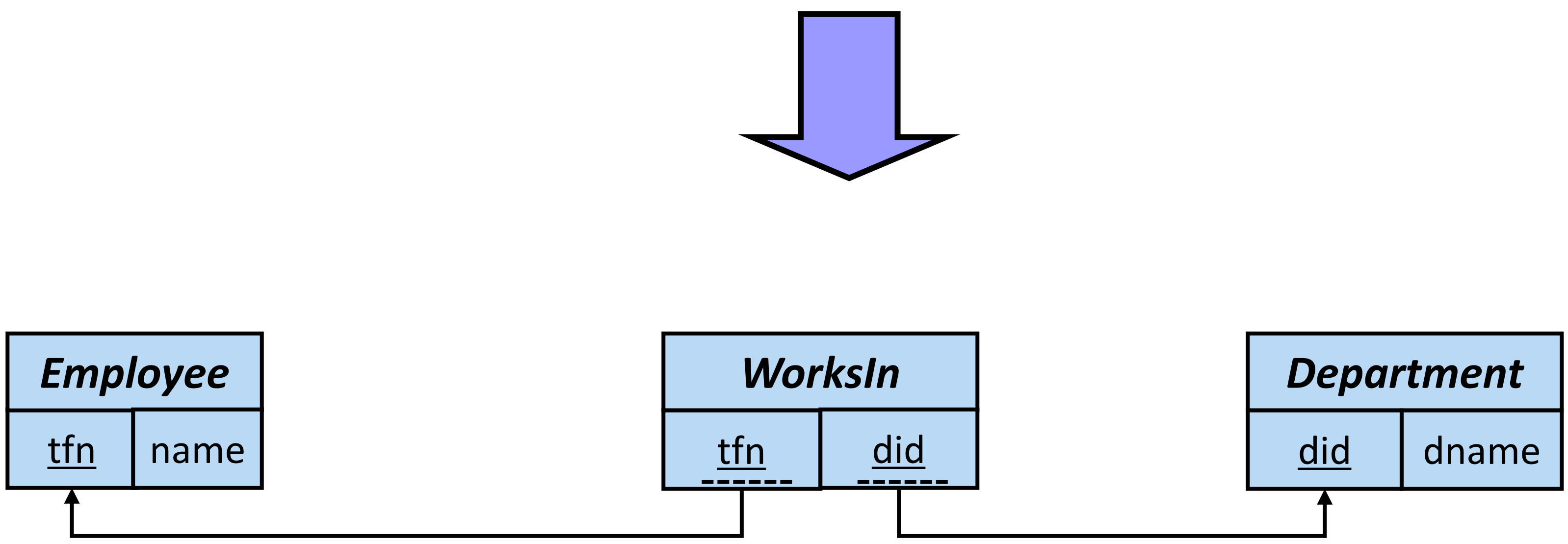
- 1 Employee can work in 0 to many Departments
- 1 Department can have 0 to many Employees

› **Mapping relationship types w/o constraints** - Create a ***new relation*** with the primary keys of the two participating entity sets as its primary key

- Relationship attributes placed on this new relation

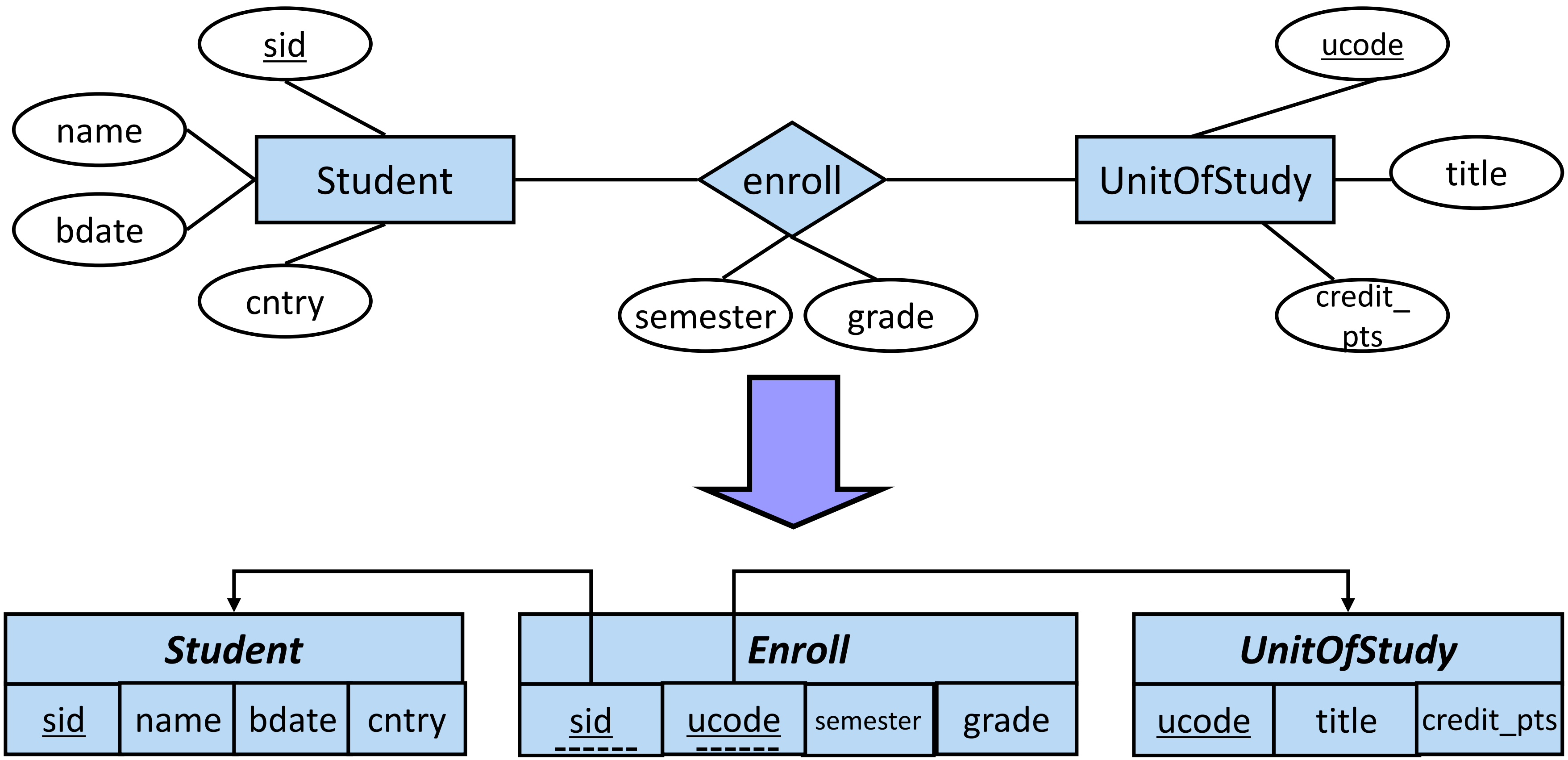
Mapping Relationship Types without Constraints



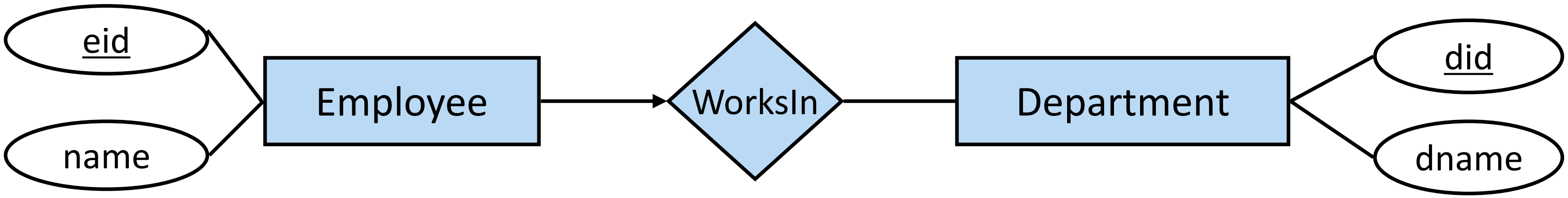


Example: Mapping Relationship Types without Constraints

- General relationship between Student & UnitOfStudy



Mapping Relationship Types with Key Constraints

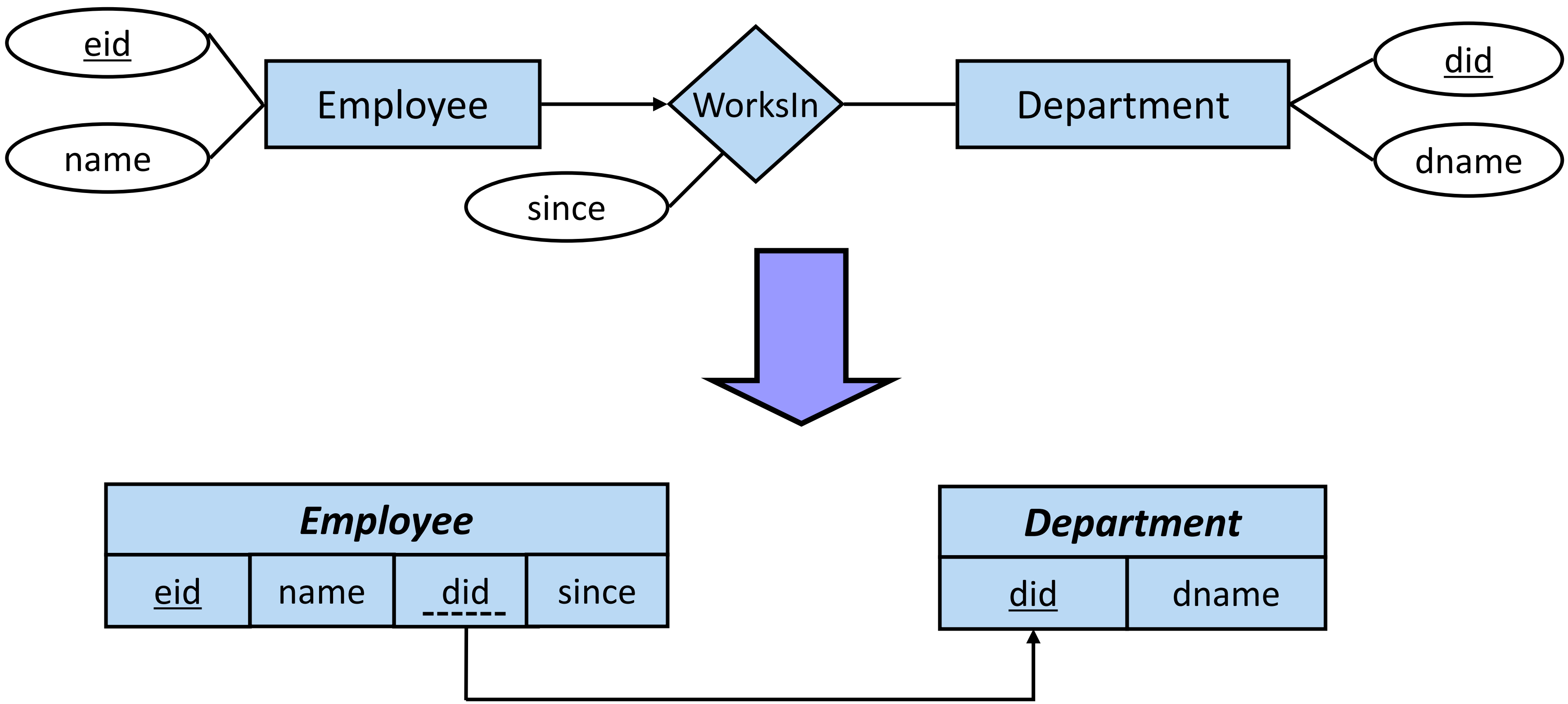


- Looking on each relationship side:
  - 1 Employee works in at most 1 Department
  - 1 Department can have 0 to Many Employees
- The primary key of Employee is a candidate key of WorksIn

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

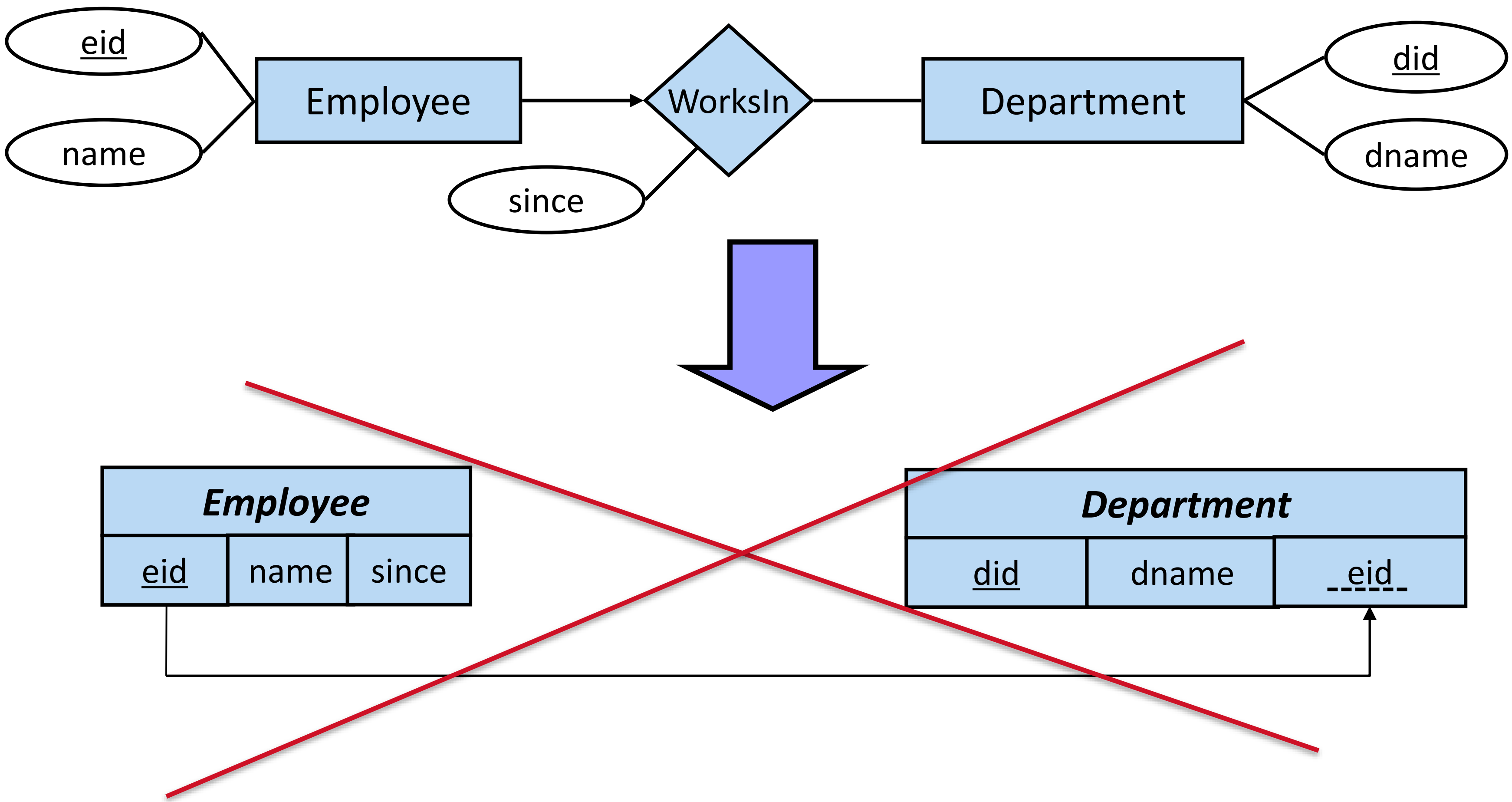
Example: Mapping Relationship Types with Key Constraints

› **Relationship with Key Constraint**



Example: Mapping Relationship Types with Key Constraints

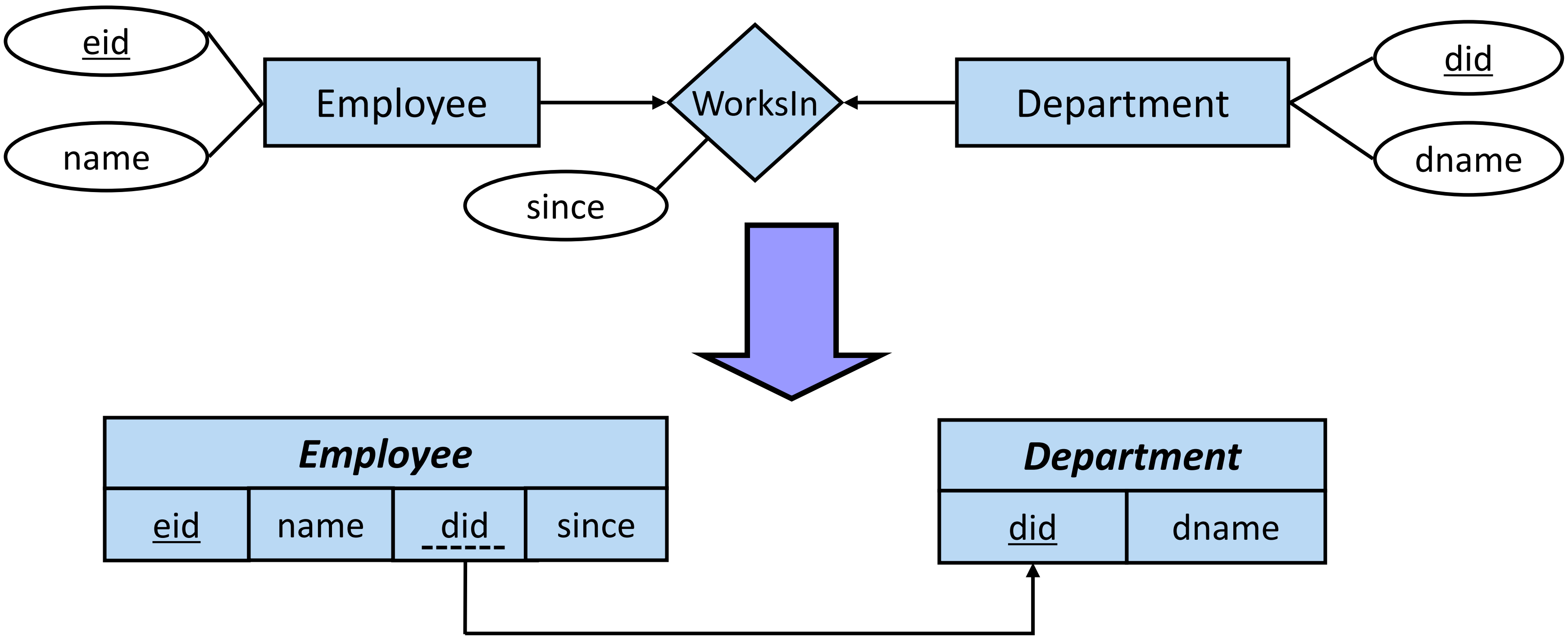
› **Relationship with Key Constraint**



Can we do this?

Example: Mapping Relationship Types with Key Constraints

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

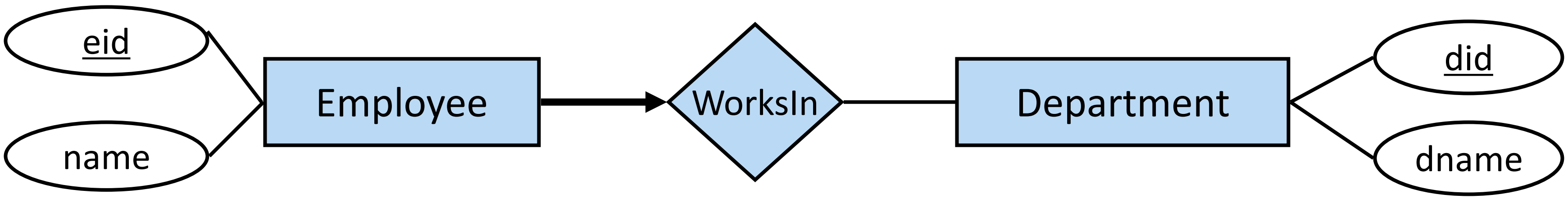


› Each Employee works in at most one Department

› Each Department has at most one Employee

- Add uniqueness constraint to foreign key

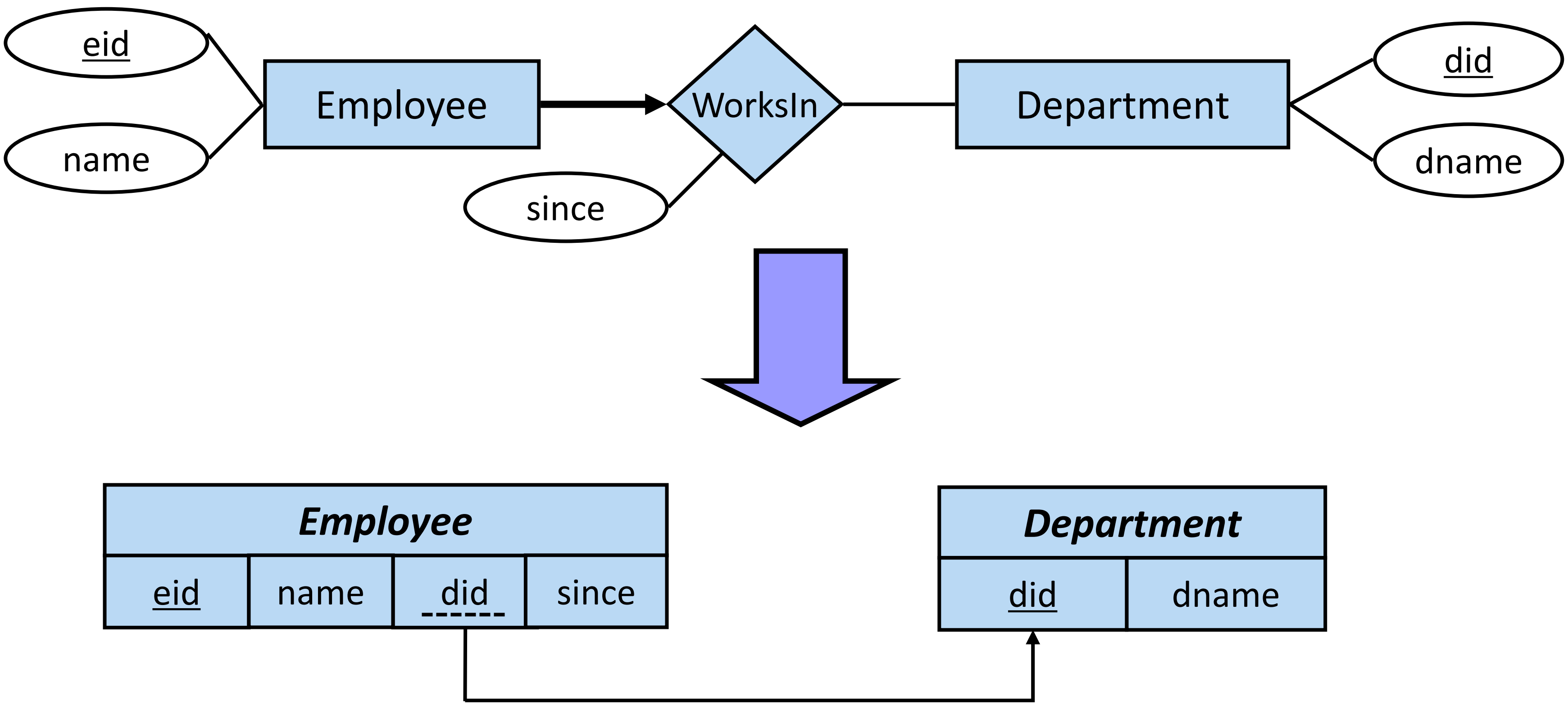
Example: Mapping Relationship Types with Key & Participation Constraints



- Looking 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
- The primary key of Employee is a candidate key of WorksIn
- Each Employee should work in one Department

Example: Mapping Relationship Types with Key & Participation Constraints

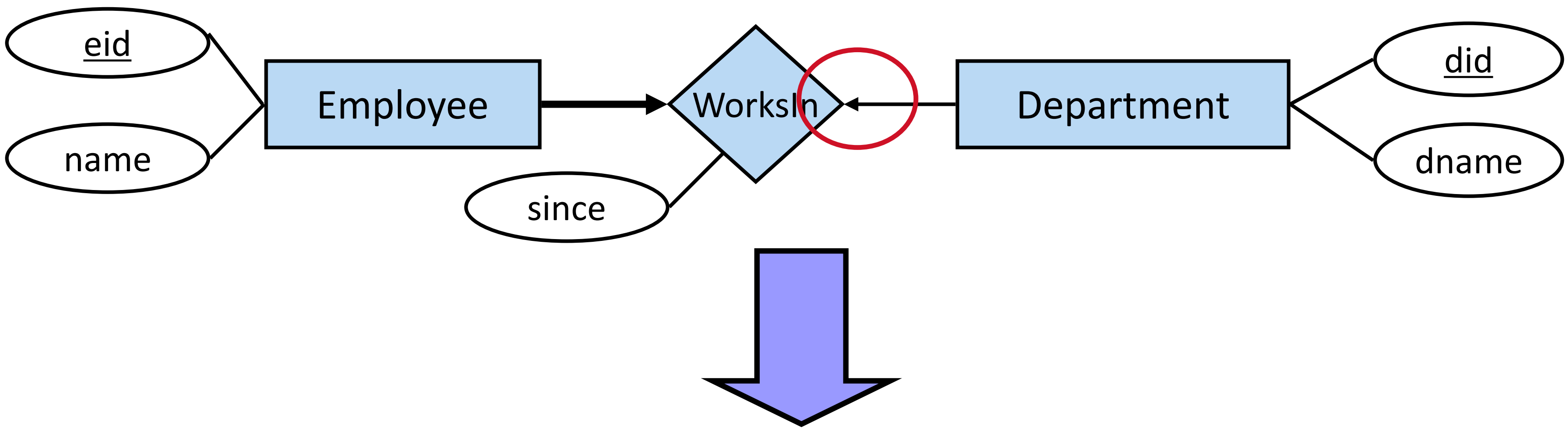
› **Relationship with Key & Participation Constraints**

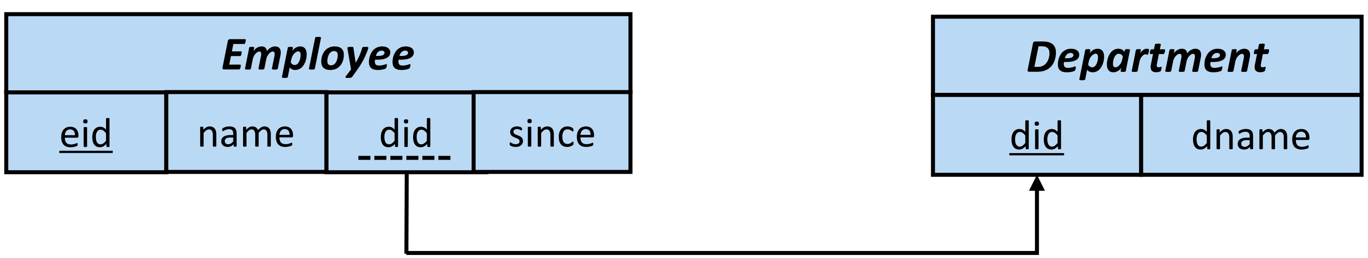


› **Key & Participation Constraint (thick arrow):** NOT NULL on foreign key

Example: Mapping Relationship Types with Key & Participation Constraints

- **Relationship with Key & Participation Constraints**

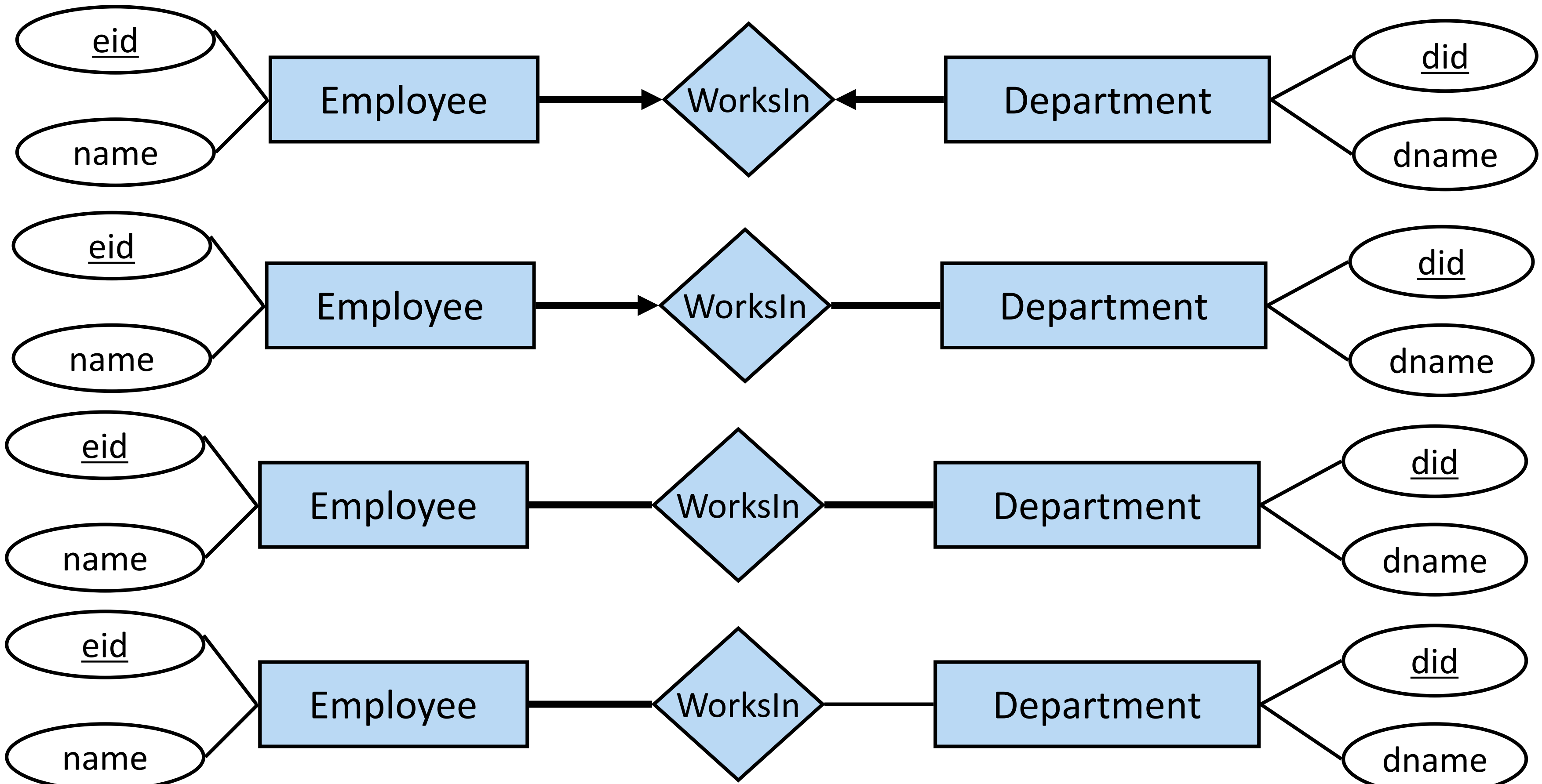




- **Key & Participation Constraint (thick arrow):** NOT NULL on foreign key
- Add uniqueness constraint to foreign key

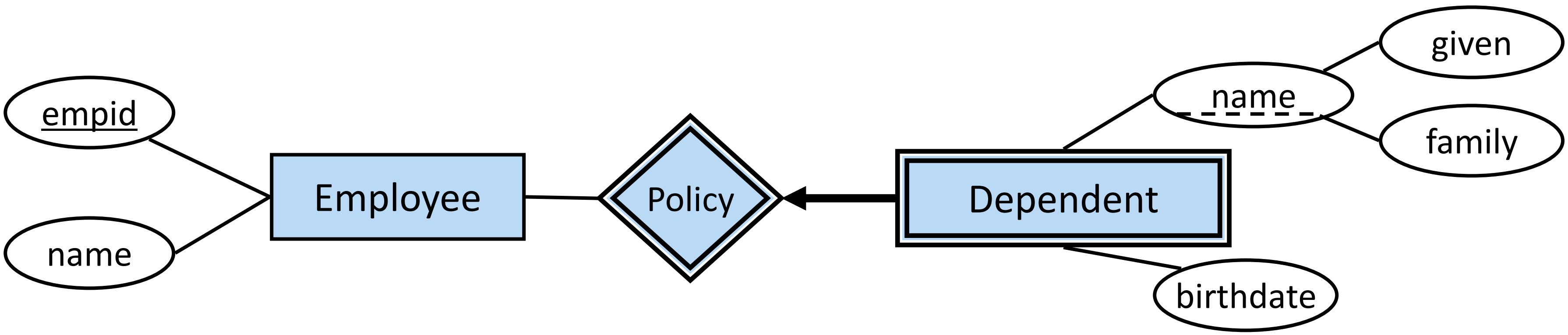
Enforcing Constraints

- Sometimes need more computationally expensive assertions or table constraints (and it may not always be worth doing this)



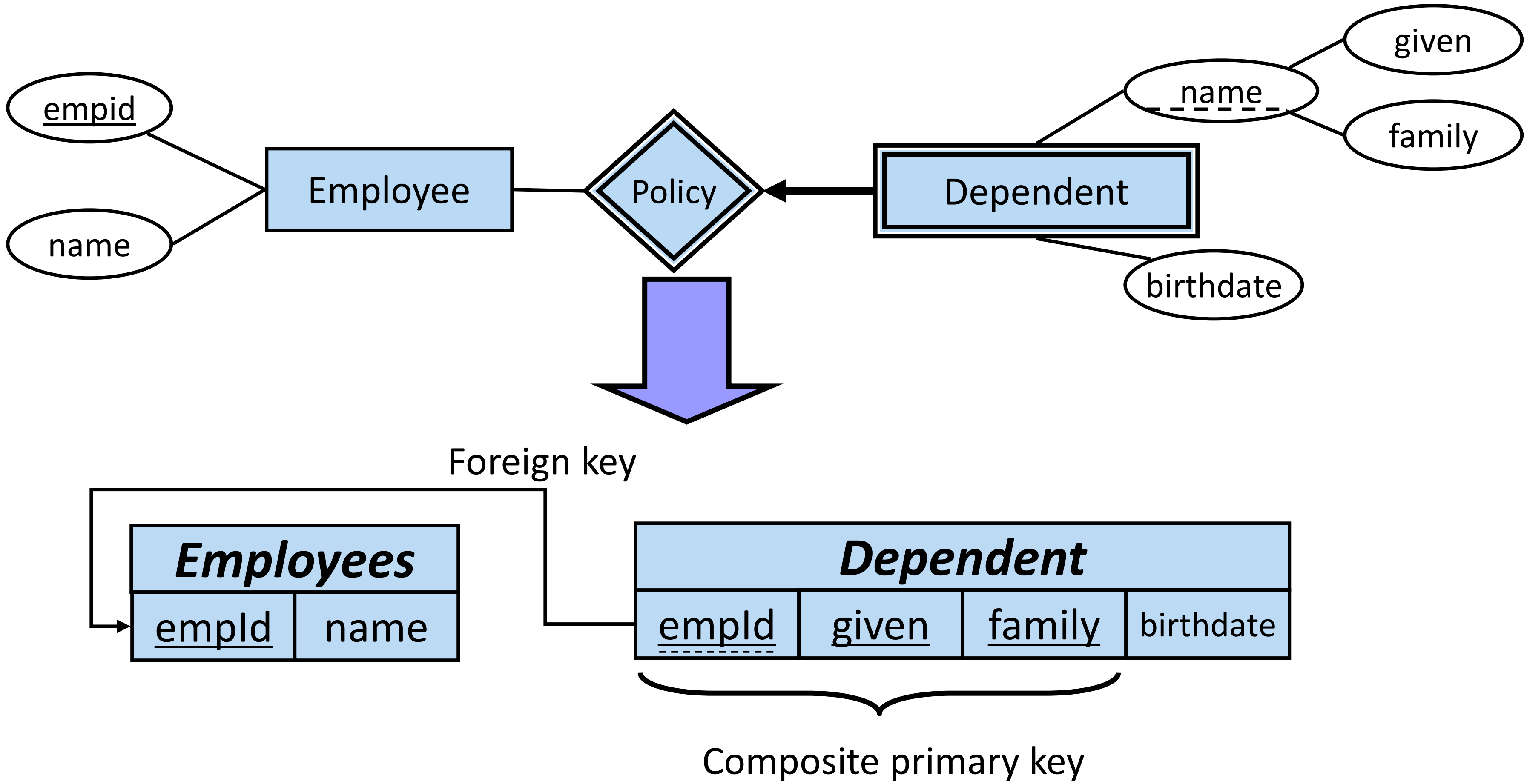
Mapping Weak Entity sets

- **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

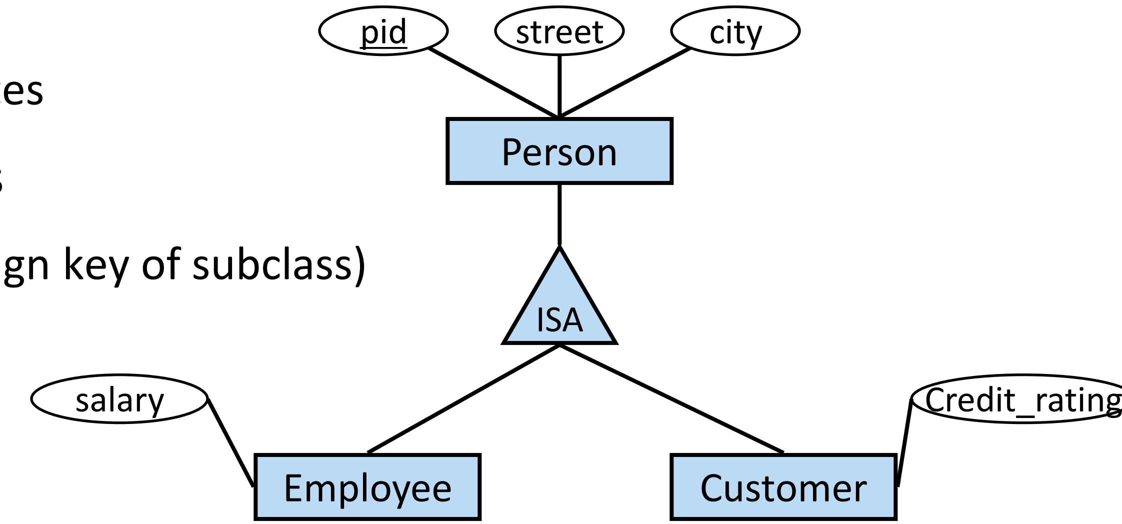


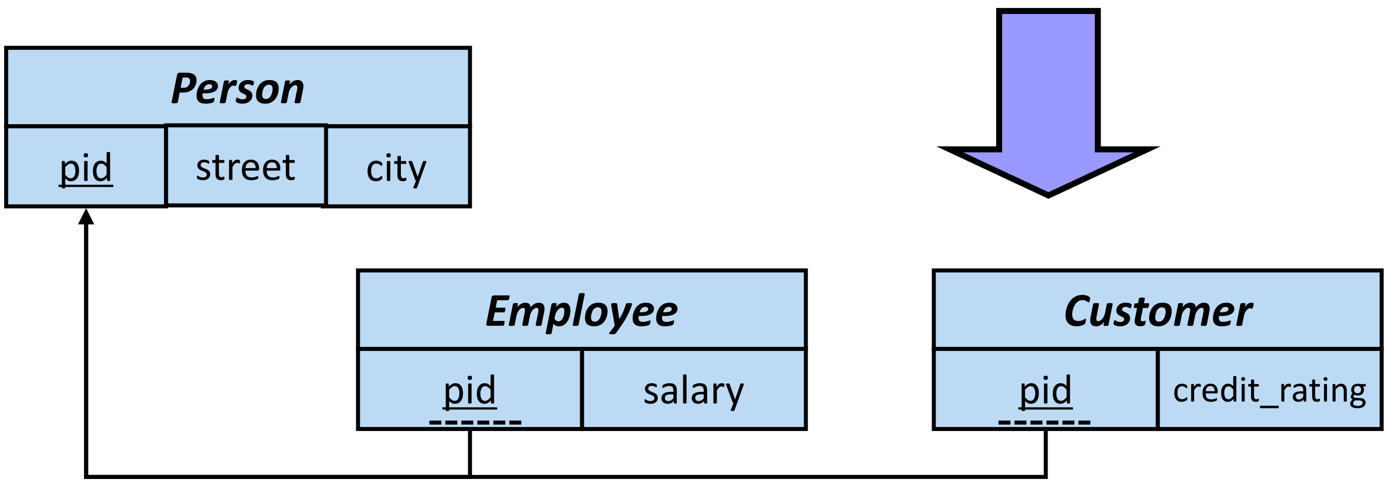
Mapping IsA-Hierarchy

- 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 also foreign key of subclass relation

Example: Mapping IsA-Hierarchy

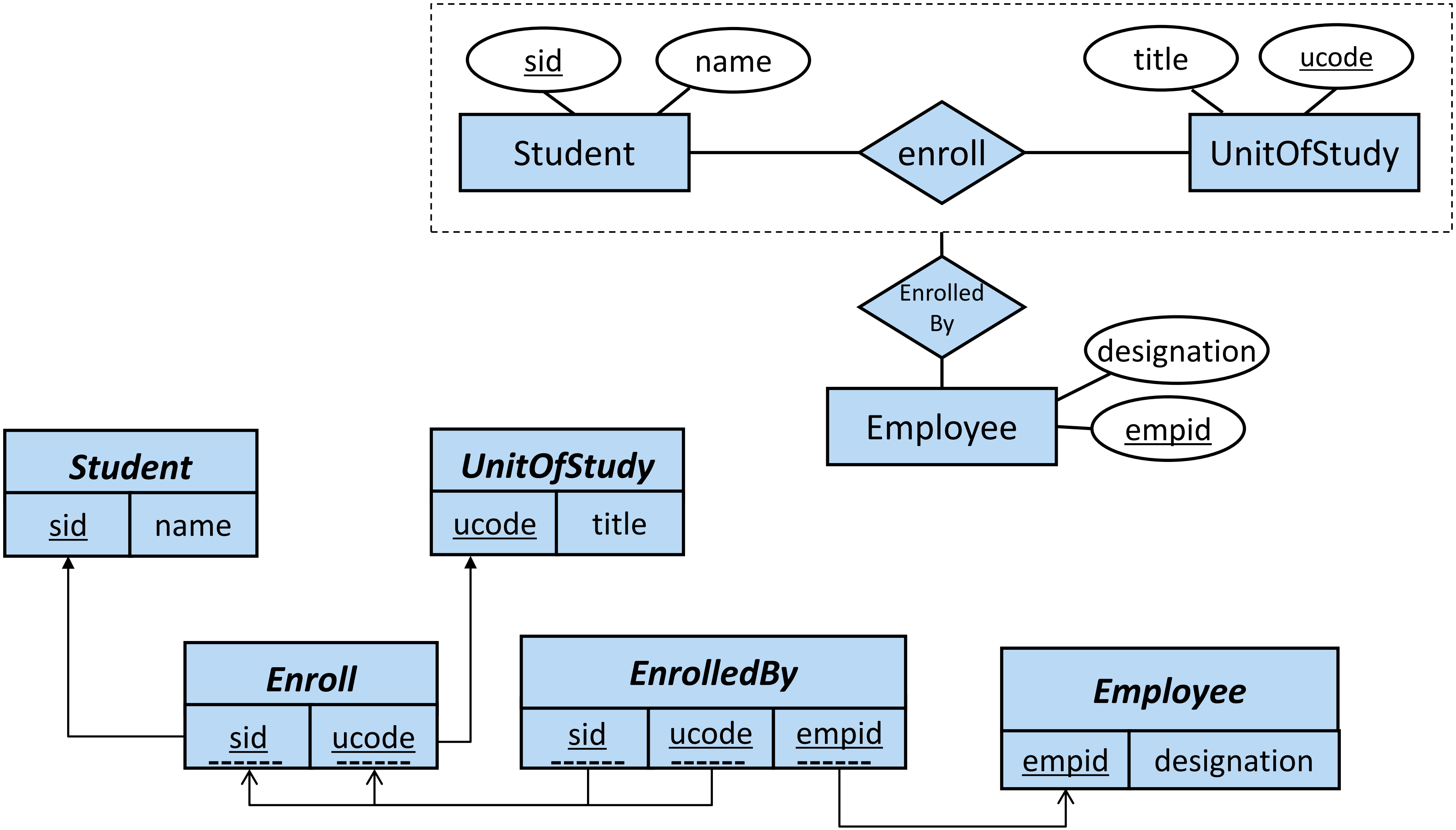
- Table for superclass
  - Primary key + common attributes
- Separate tables for subclasses
  - Primary key of superclass (foreign key of subclass) + specific attributes





Example: Mapping Aggregations

- 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, arity)**
  - **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*
- *PostgreSQL 9.5 Language Reference*
  - https://www.postgresql.org/docs/9.5/static/index.html

Next Week

- The Structured Query Language (SQL)
- 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