Databases 1

Daniel POP

Week 8

Database Design. Practice

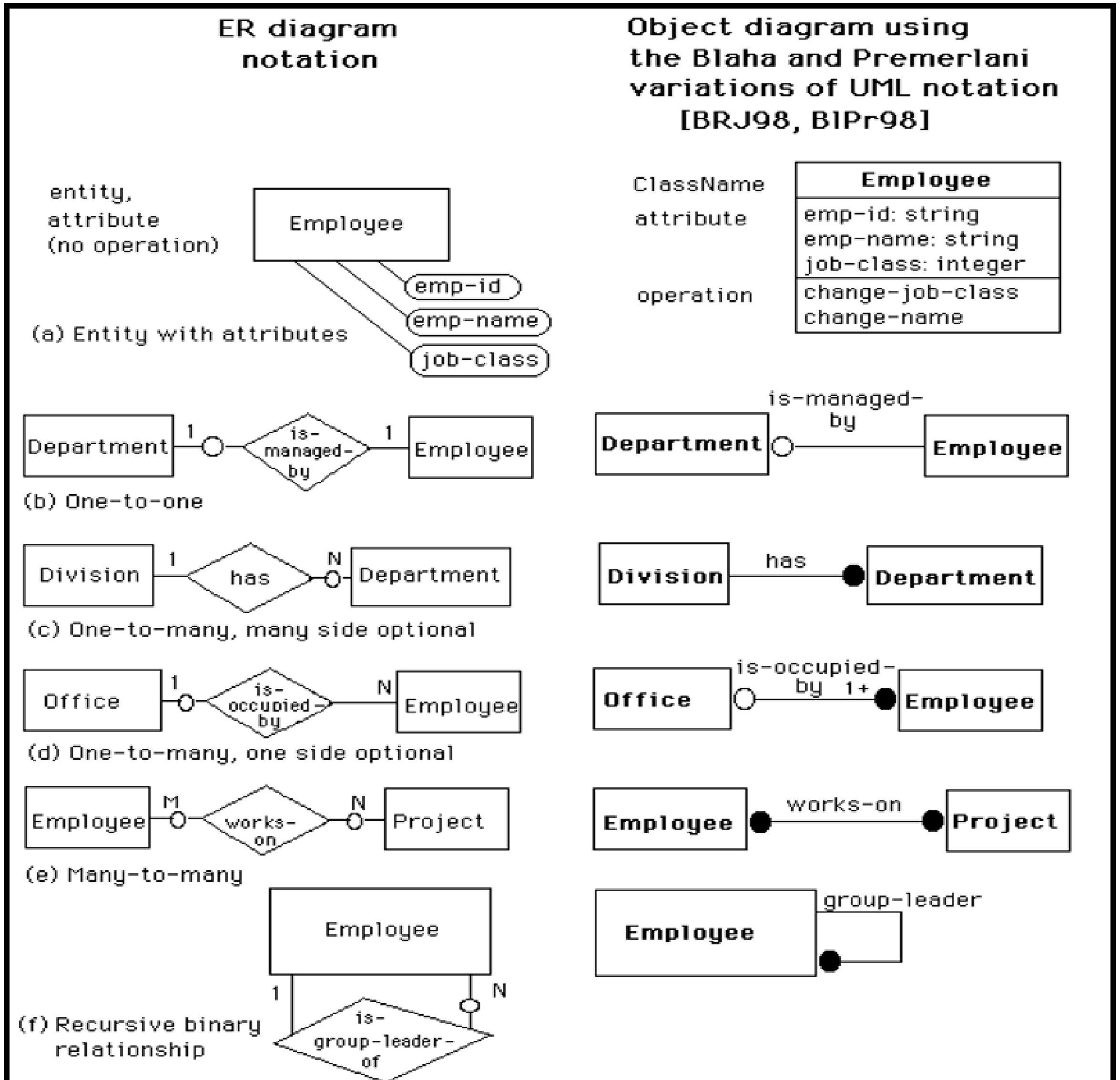
Agenda

- Entity/Relationship modelling
  - Entities / Entity Sets
  - Attributes
  - Relationships
- N-ary relationships
- Weak entity sets
- Mapping E-R diagrams to relational model
  - Mapping inheritance
- Case study 1. Geo
- Case study 2. Trains

Database Design

- Understand the real-world domain being modelled
- Specify it using a database design model
  - More intuitive and convenient for schema design
  - But not necessarily implemented by DBMS
  - A few popular ones: Entity/Relationship (E-R) model, Object Definition Language (ODL), UML (Unified Modeling Language)
- Translate specification to the data model of DBMS
  - Relational, XML, object-oriented, etc.
- Create DBMS schema

Multiple graphical languages



Conceptual modelling

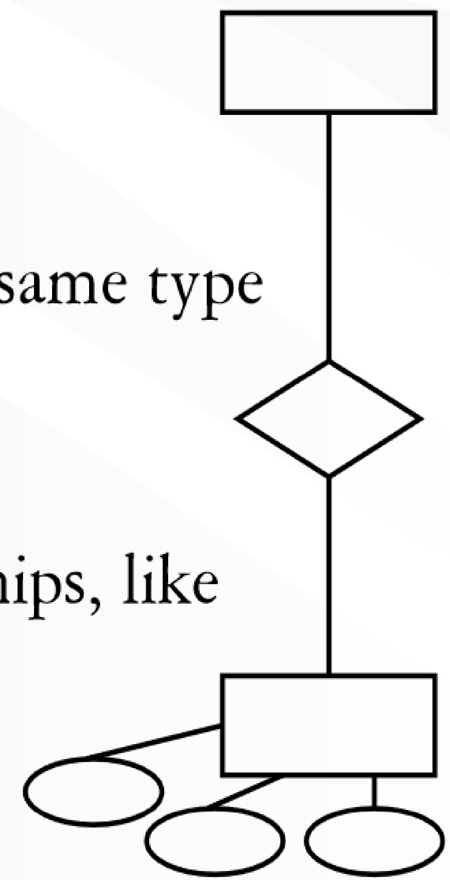
- Conceptual modelling (conceptual database design) is the process of constructing a model of the information use in an enterprise
- Model is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations.
- **This model is called a conceptual data model.**
- Conceptual models may also be referred to as logical models in the literature.
- The conceptual model is independent of all implementation details, whereas the logical model assumes knowledge of the underlying data model of the target DBMS.

Entity-relationship (E-R) model

- Historically and still very popular
  - Peter Chan 1976
- Graphical language
- Can think of as a “watered-down” object-oriented design model
- Primarily a design model—not directly implemented by DBMS
- Designs represented by E-R diagrams
  - We use the style of E-R diagram covered by GMUW; there are other styles/extensions
  - Very similar to UML diagrams
  - Crow’s Foot notation – Gordon Everest 1976 / Barker’s notation
    - Relationships need to be represented as tables/relations

E-R basics

- Entity: a "thing," like an object
- Entity set: a collection of things of the same type, like a relation of tuples or a class of objects
  - Represented as a rectangle
- Relationship: an association among entities
- Relationship set: a set of relationships of the same type (among same entity sets)
  - Represented as a diamond
- Attributes: properties of entities or relationships, like attributes of tuples or objects
  - Represented as ovals

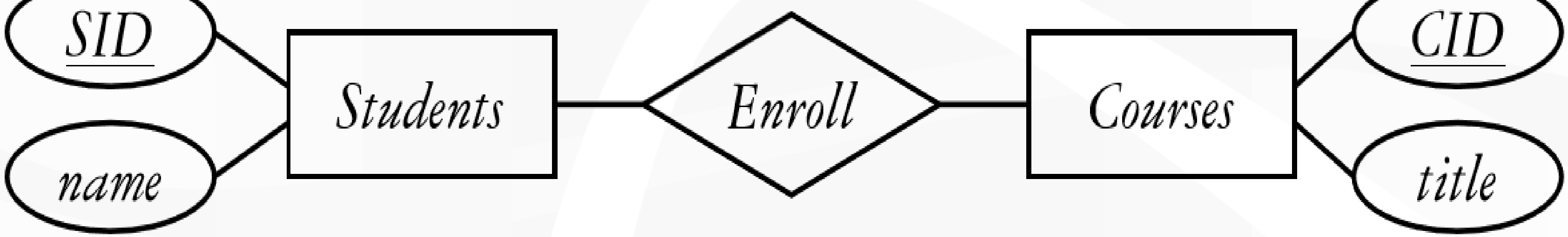


Entity sets. Attributes

- Entity Set has a name and a set of attributes
- An attribute has a name and a domain
  - data type (e.g., INT, VARCHAR)
  - constraints (e.g., allows, or not, NULL value)

Running example

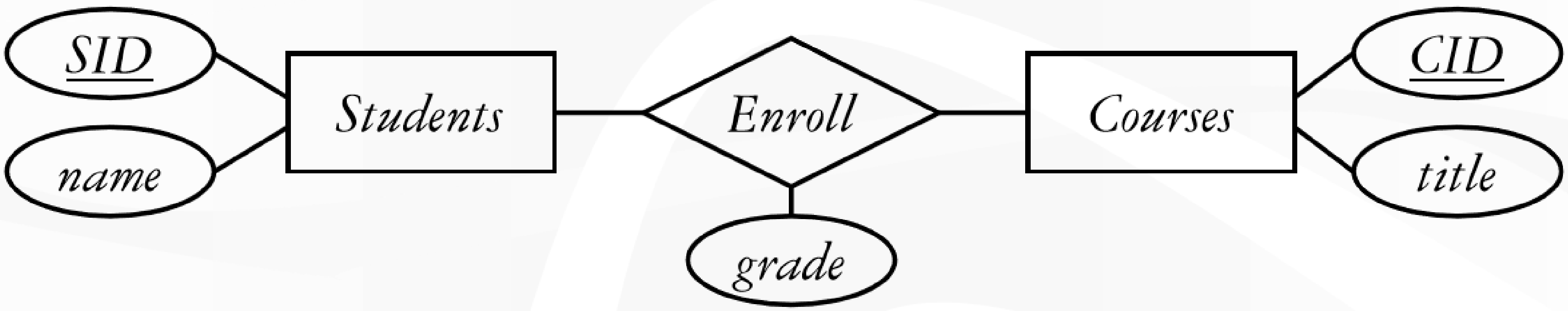
- Students enroll in courses



- A key of an entity set is represented by underlining all attributes in the key
  - A key is a set of attributes whose values can belong to at most one entity in an entity set—like a key of a relation

Relationships with attributes

- Example: students take courses and receive grades



- Where do the grades go?
  - With *Students*?
    - But a student can have different grades for multiple courses
  - With *Courses*?
    - But a course can assign different grades for multiple students
  - With *Enroll*!

Properties of relationships

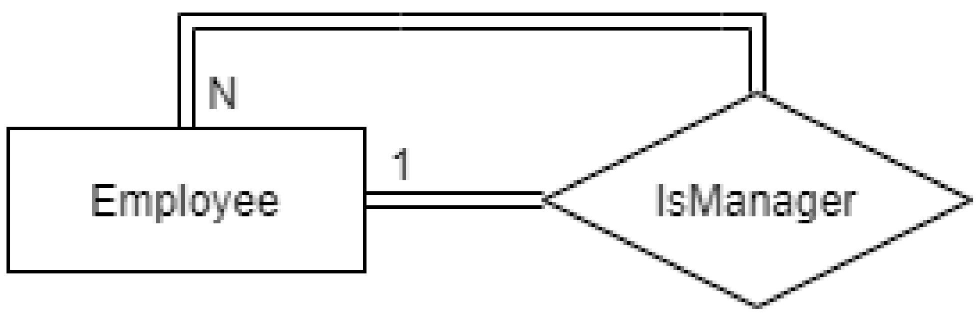
- Relationships can have attributes
- There could be multiple relationships between the same entity sets. Examples
  - (1) Students Enroll to Courses;
  - (2) Students are assigned to Teaching Assistant (TA) per Courses
- Properties of relationships
  - Degree
  - Multiplicity
  - Reflexivity

Properties of relationships

- **Degree** of a relationship = the number of participating entity sets in the relationship
  - Binary relations (degree = 2)
  - N-ary relations (N >= 3)
  - Example: Enroll is a binary relation because it connects 2 entity sets (Students and Courses);

Relationships classification

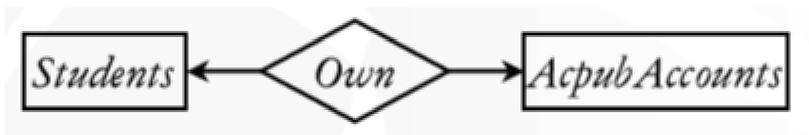
Reflexive relationships: entities of the same entity set are related to each other

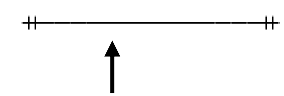


Multiplicity of relationships

Multiplicity - The multiplicity applies to the adjacent entity and it is independent of the multiplicity on the other side of the association. Let E and F be 2 entities.

One-one: Each entity in E is related to 0 or 1 entity in F and vice versa.





Crow's Foot notations

Many-one: Each entity in E is related to 0 or 1 entity in F, but each entity in F is related to 0 or more in E.

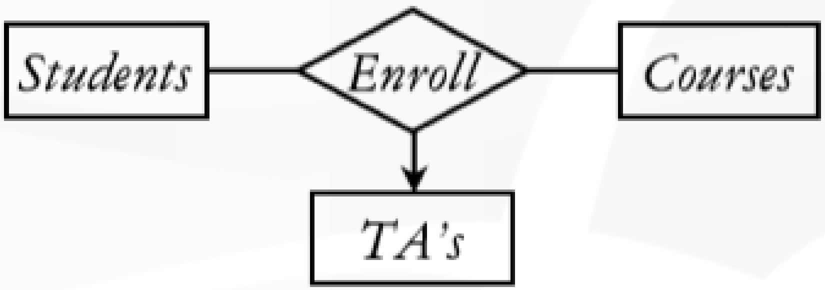
Many-many: Each entity in E is related to 0 or more entities in F and vice versa

"One" (0 or 1) is represented by an arrow.

"Exactly one" is represented by a rounded arrow.,

Modelling N-ary relationships

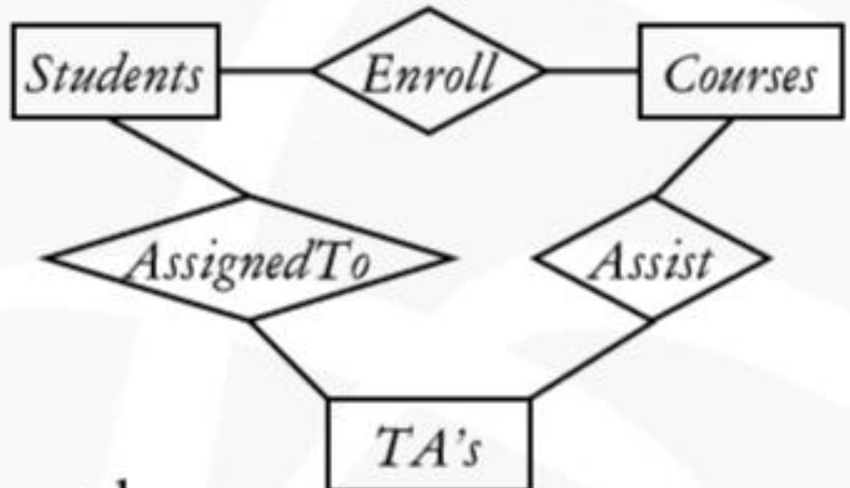
- Example: Each course has multiple TA's; each student is assigned to one TA



- Meaning of an arrow into *E*: Pick one entity from each of the other entity sets; together they must be related to either 0 or 1 entity in *E*

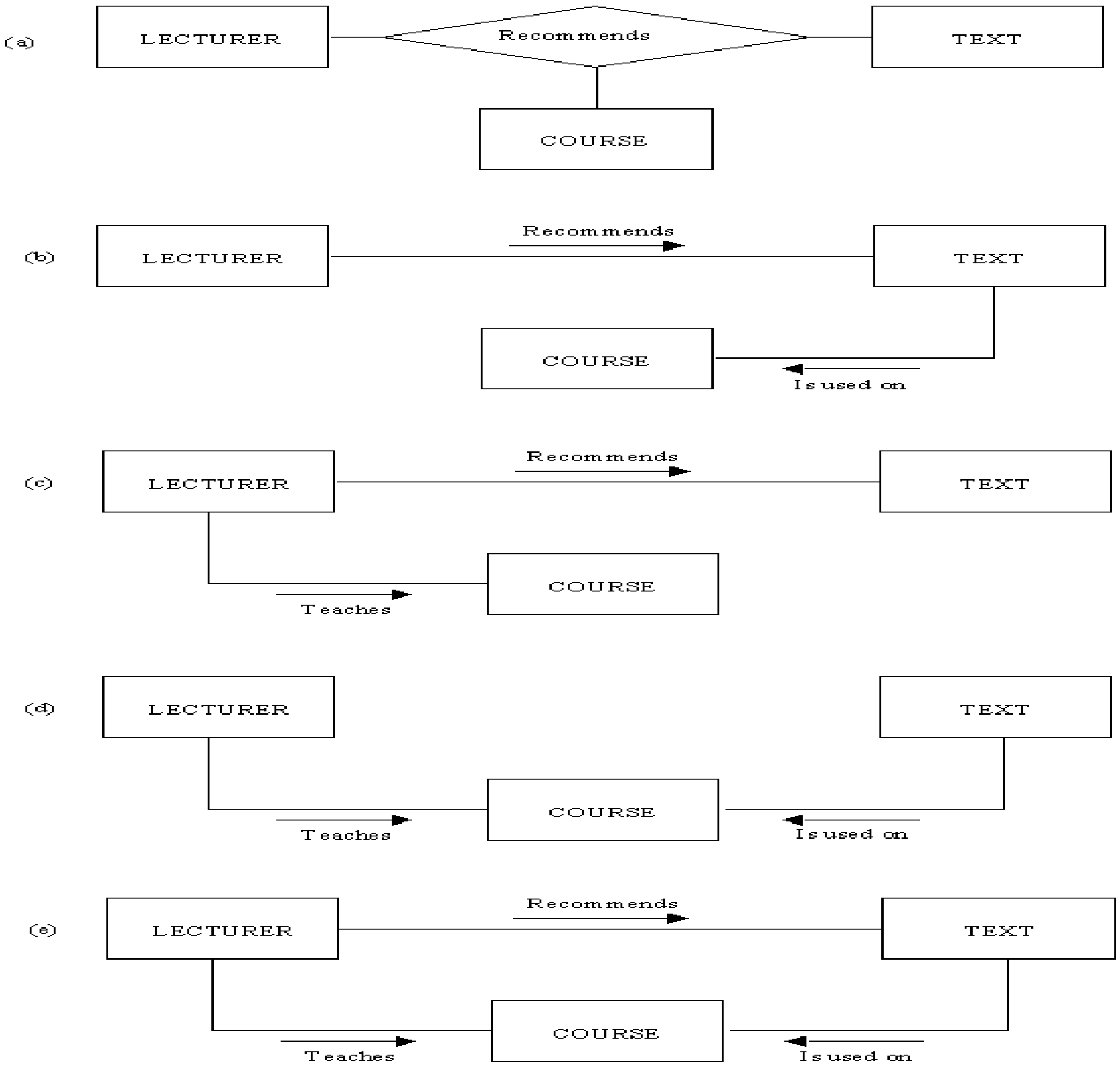
Modelling N-ary relationships

- Can we model *n*-ary relationships using just binary relationships?



- No; for example:
  - Bart takes CPS116 and CPS114
  - Lisa TA's CPS116 and CPS114
  - Bart is assigned to Lisa in CPS116, but not in CPS114

Modelling N-ary relationships. Exercise

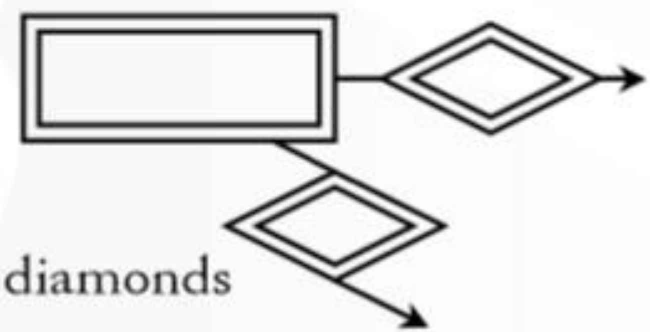


Which decompositions (b, c, d, e) of the n-ary relationship in (a) hold the same constraints as in (a)?

Weak Entity Sets

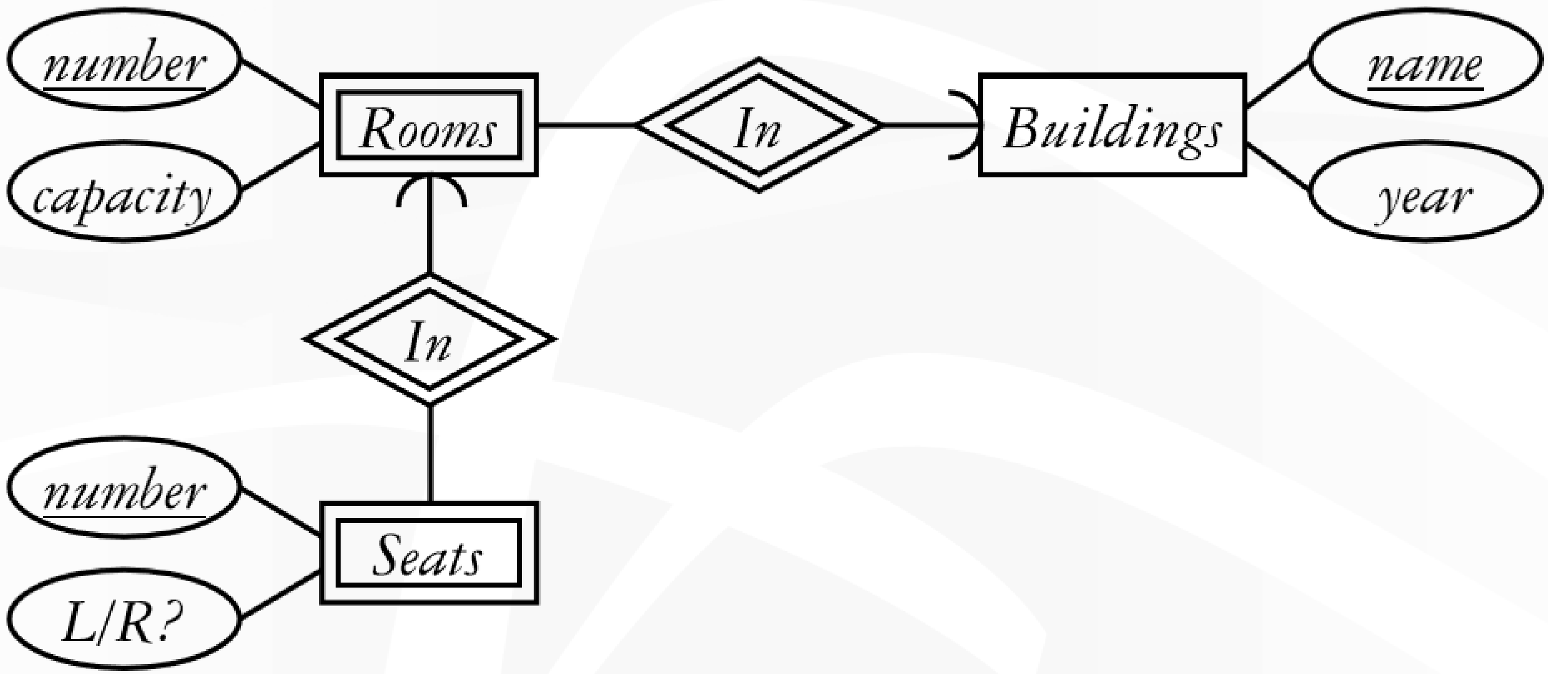
Sometimes, the key of an entity set *E* comes not completely from its own attributes, but from the keys of other (one or more) entity sets to which *E* is linked by many-one (or one-one) relationship sets

- Example: *Rooms* inside *Buildings* are partly identified by *Buildings*' name
- *E* is called a weak entity set
  - Denoted by double rectangle
  - The relationship sets through which *E* obtains its key are drawn as double diamonds



Weak Entity Sets. Example

Seats in rooms in buildings

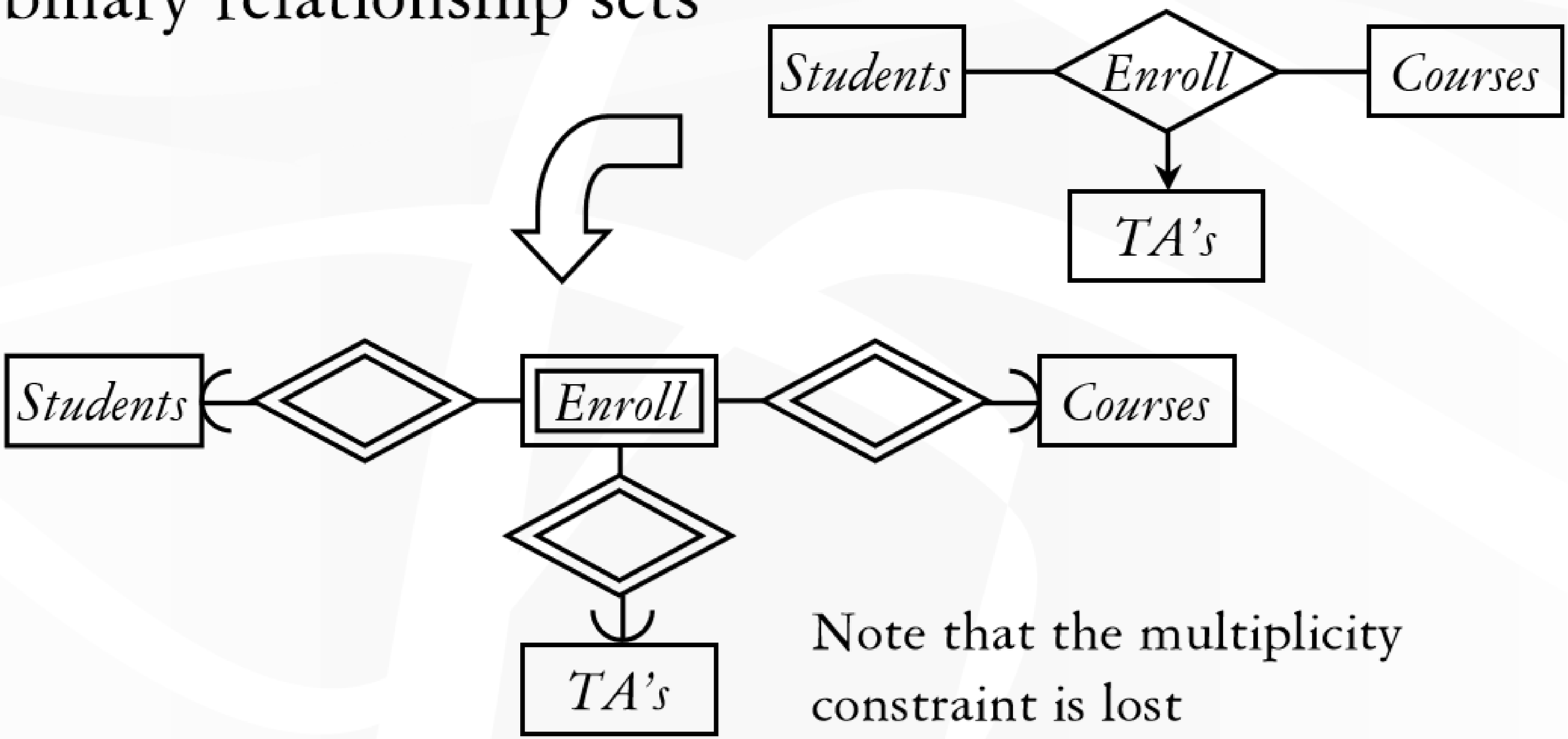


Why must double diamonds be many-one/one-one?

- With many-many, we would not know which entity provides the key value!

Modelling N-ary relationships

- An *n*-ary relationship set can be replaced by a weak entity set (called a connecting entity set) and *n* binary relationship sets

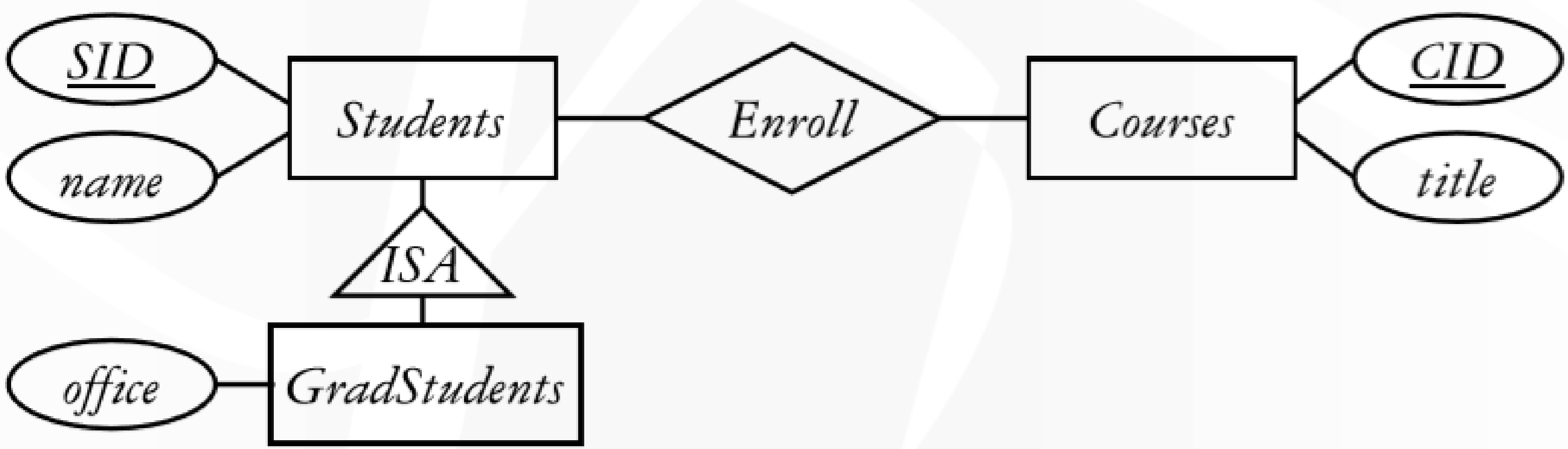


Note that the multiplicity constraint is lost

Example of lost constraints (multiplicity): a (student, course) pair has only one TA assigned

IS-A (Inheritance) relationships

- Similar to the idea of subclasses in object-oriented programming: subclass = special case, fewer entities, and possibly more properties
  - Represented as a triangle (direction is important)
- Example: Graduate students are students, but they also have offices



Translating E-R Diagram to Relational Model

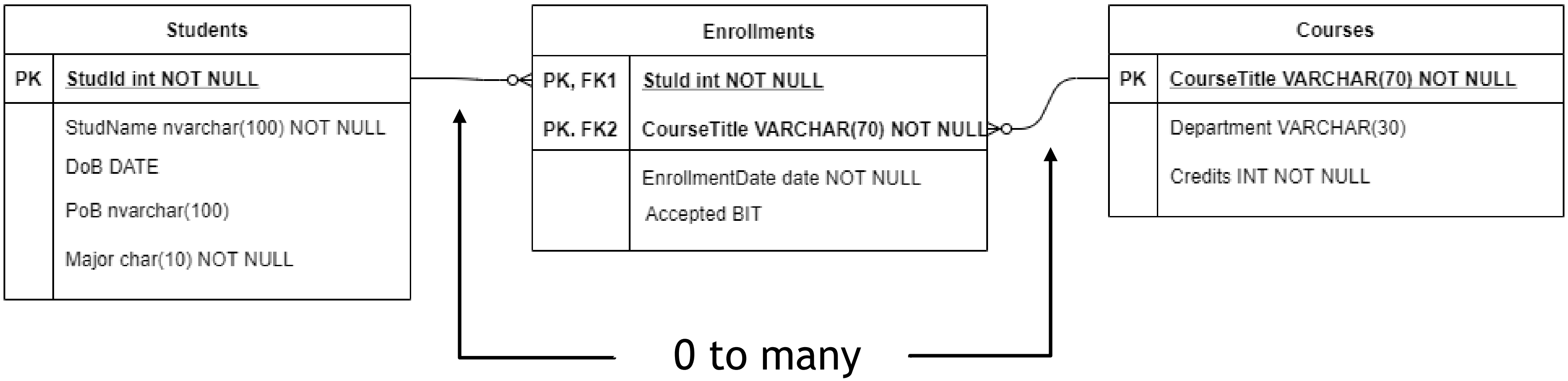
- An Entity Set directly translates to a table:
  - Attributes map to columns,
  - Key attributes become candidate keys
- A relationship translates to…. guess what? ….. a table (of course)
  - PK of connected entities become columns (FK)
  - Attributes of the relationship become columns
  - Choose the PK based on multiplicity of relationship
- A Weak Entity Set:
  - PK will be a composed PK of the connected entity sets primary key columns (which become FK)
  - Pay attention to name conflicts
- A double-diamond connecting one weak entity set to another entity – no need to translates since the keys migrate anyway (example: Seats-Rooms-Buildings)

Translating E-R Diagram to Relational Model. Examples

- Example:
  Students(SID, Name)
  Courses(CID, Title)
  Enroll(SID, CID, grade)

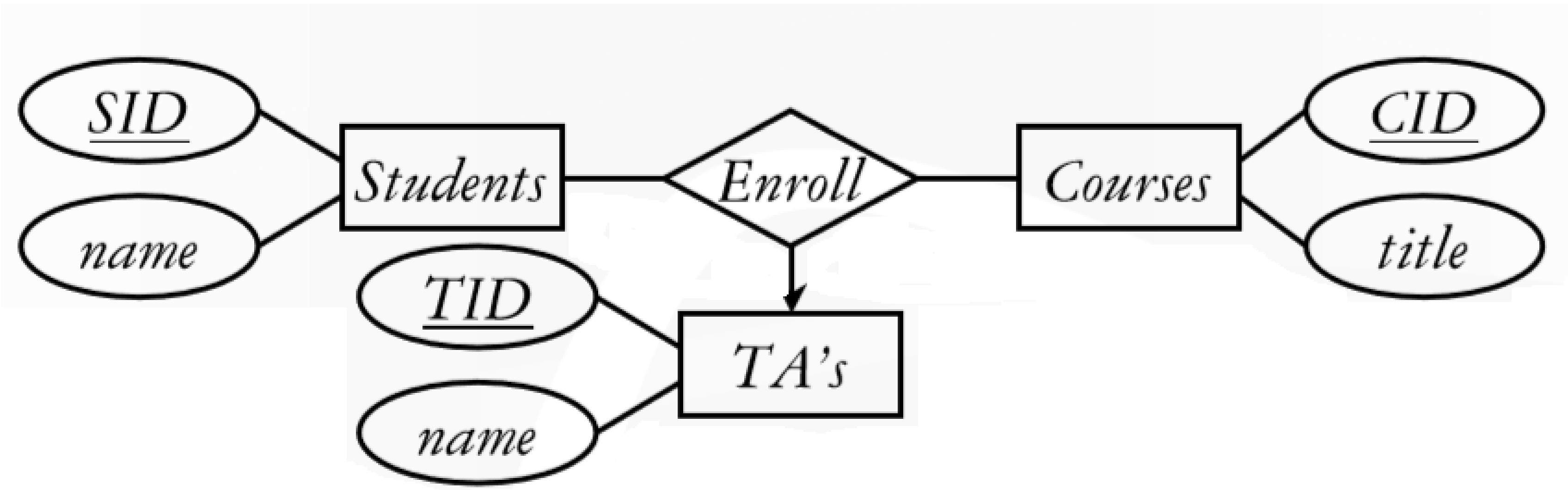
- Example:
  Buildings(name, year)
  Rooms(building_name, number, capacity)
  Seats(number, building_name, room_number, L_R)

Table Diagram. Examples



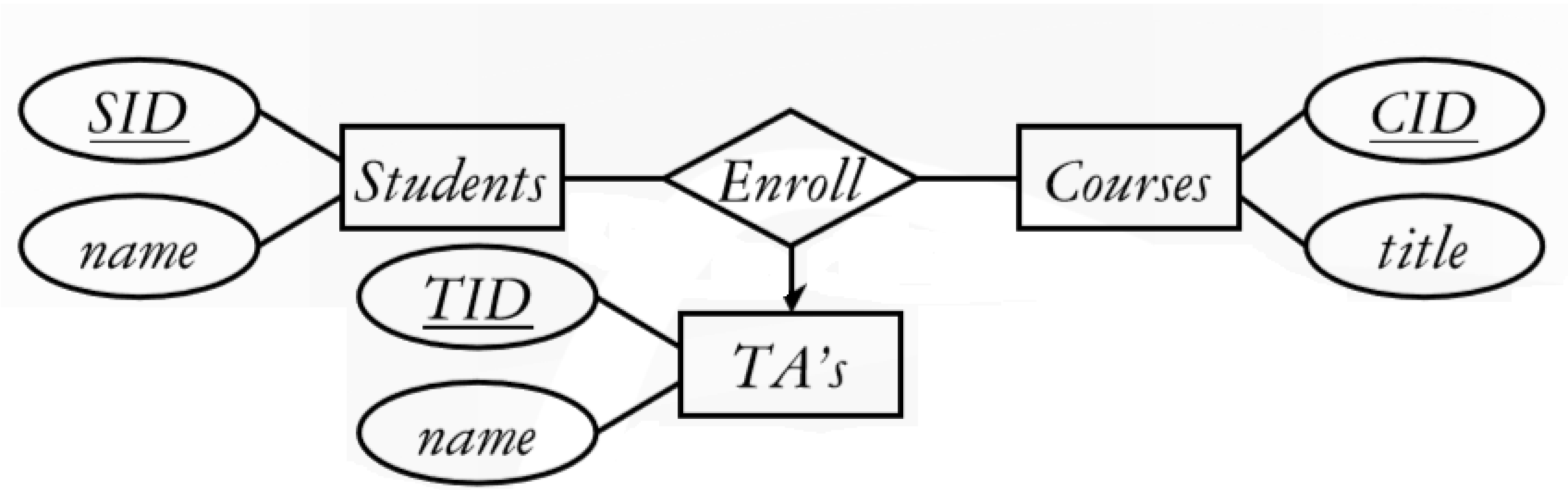
Exercise

- Translate the following diagram into relations. Identify the (primary) keys of the relations.



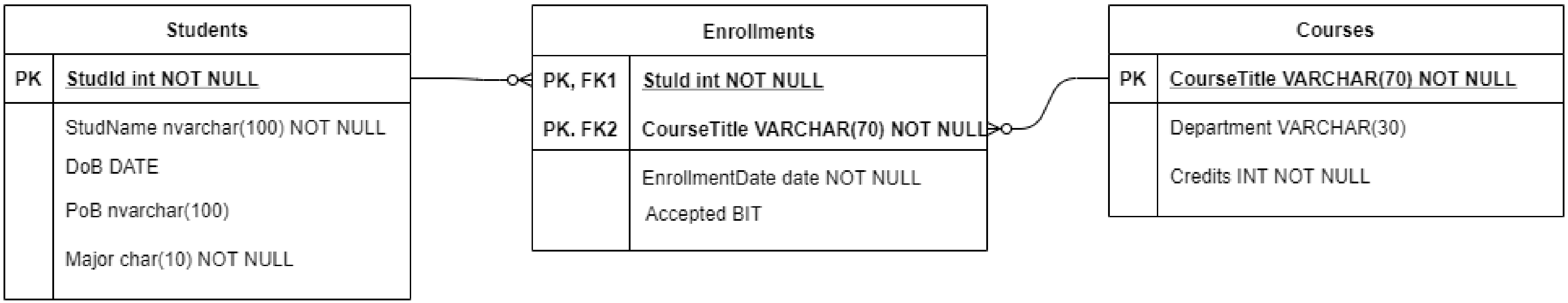
Exercise

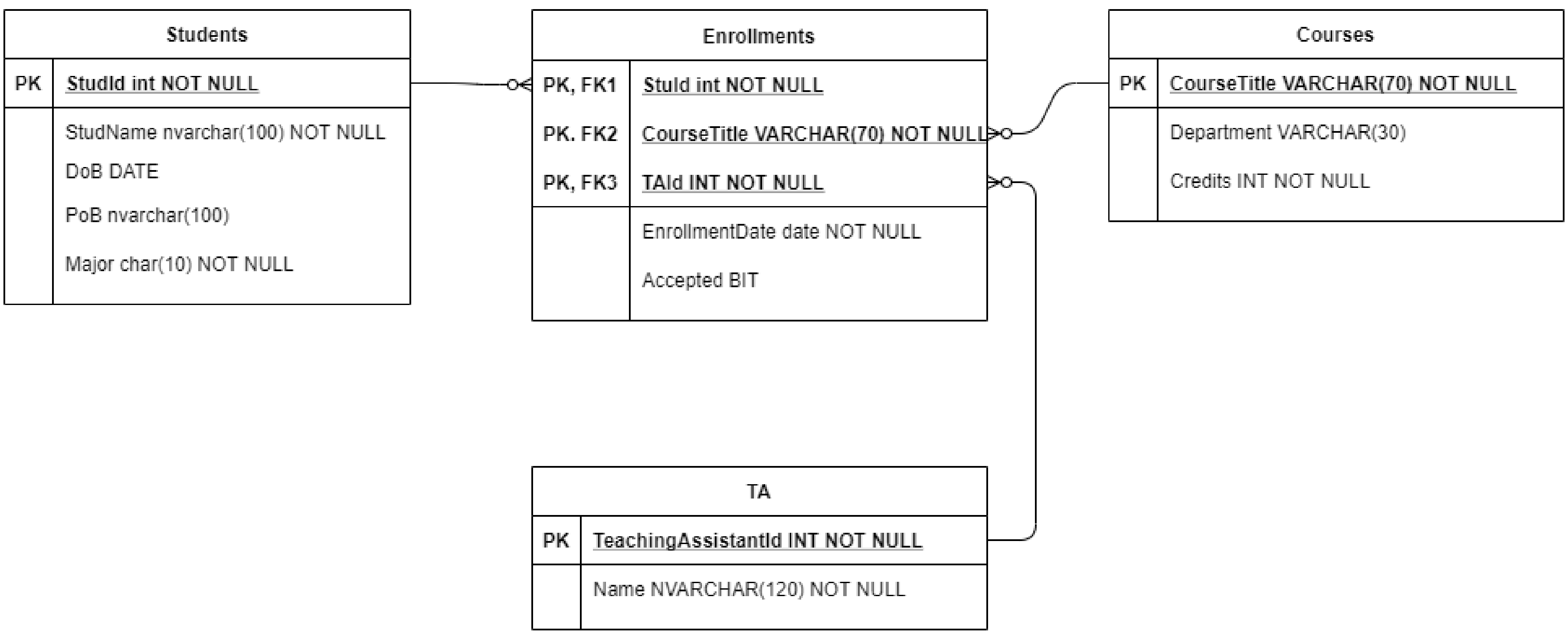
- Translate the following diagram into relations. Identify the (primary) keys of the relations.



Students(<u>SID</u>, Name)
Courses(<u>CID</u>, Title)
TAs(<u>TID</u>, Name)
Enroll(<u>SID</u>, <u>CID</u>, <u>TID</u>)

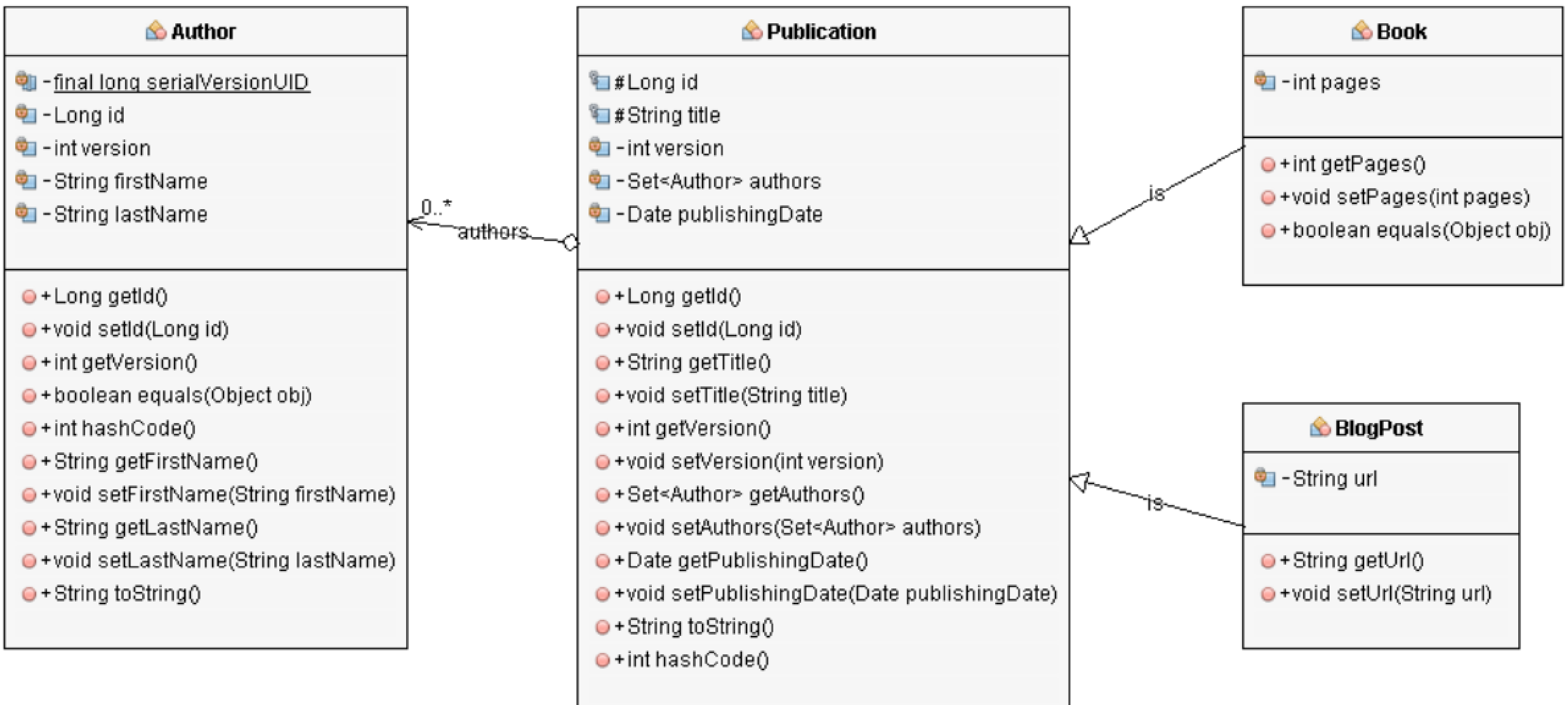
Table Diagram. Examples





Translating IS-A (Inheritance Mapping)

- Mapped Superclass
- Single Table
- Table per Class
- Joined Table

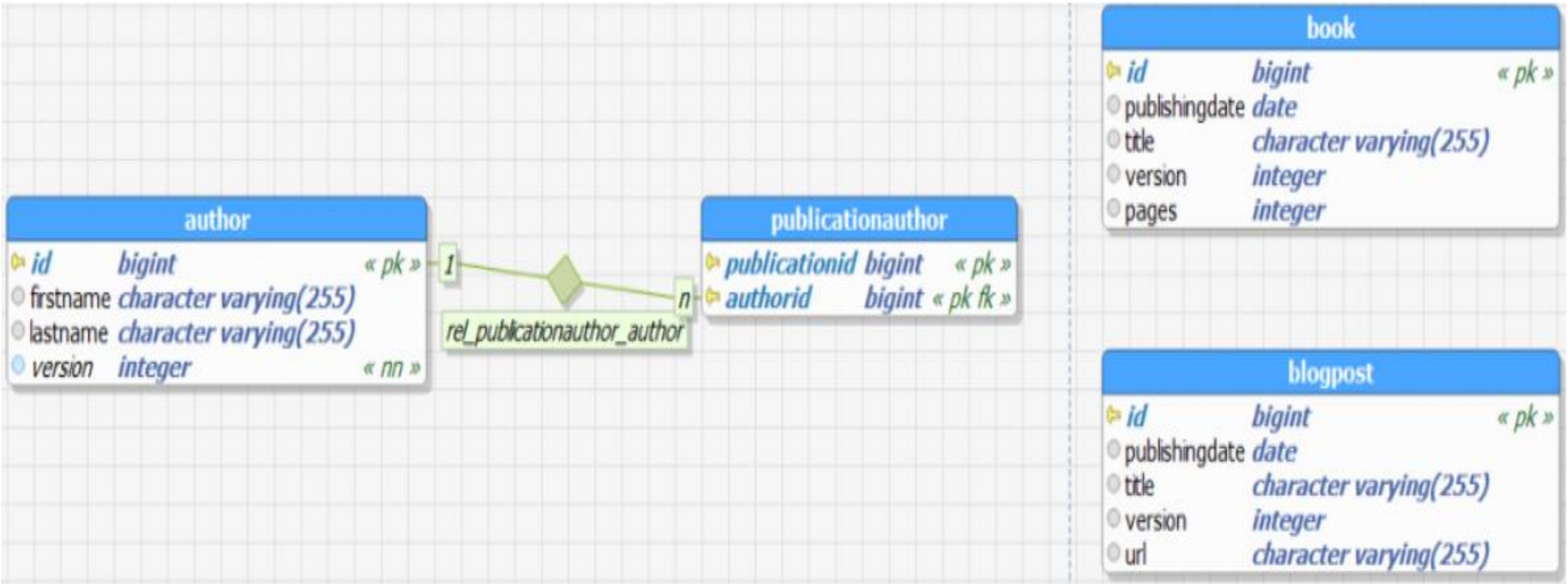


https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance Mapping. Mapped Superclass

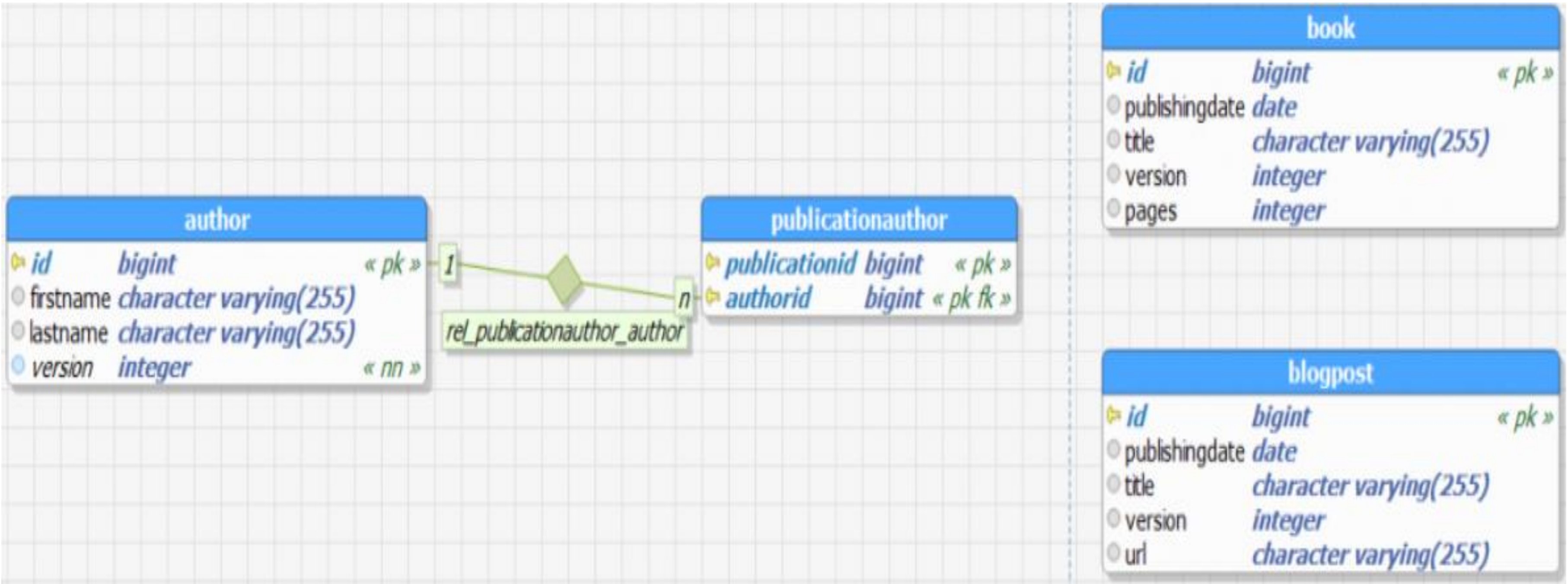
- Maps each subclass to its own table (includes attributes from superclass)
- There is no table for superclass
- Not possible to represent relationships for superclass (base class), e.g. author-publication relationship

Inheritance Mapping. Table per class



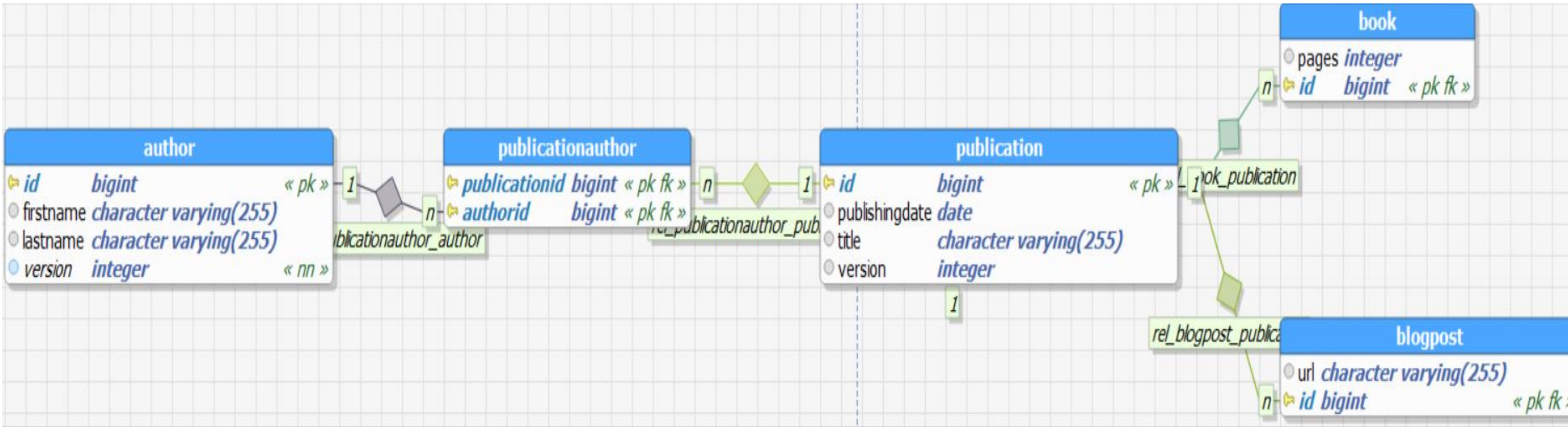
- Maps each subclass to its own relation (includes attributes from superclass)
- There is no table for superclass
- Maps the relationship Authors-Write-Publications to a relation
- Retrieval of authors-publications details rely on complex and expensive queries involving UNION

Inheritance Mapping. Table per class



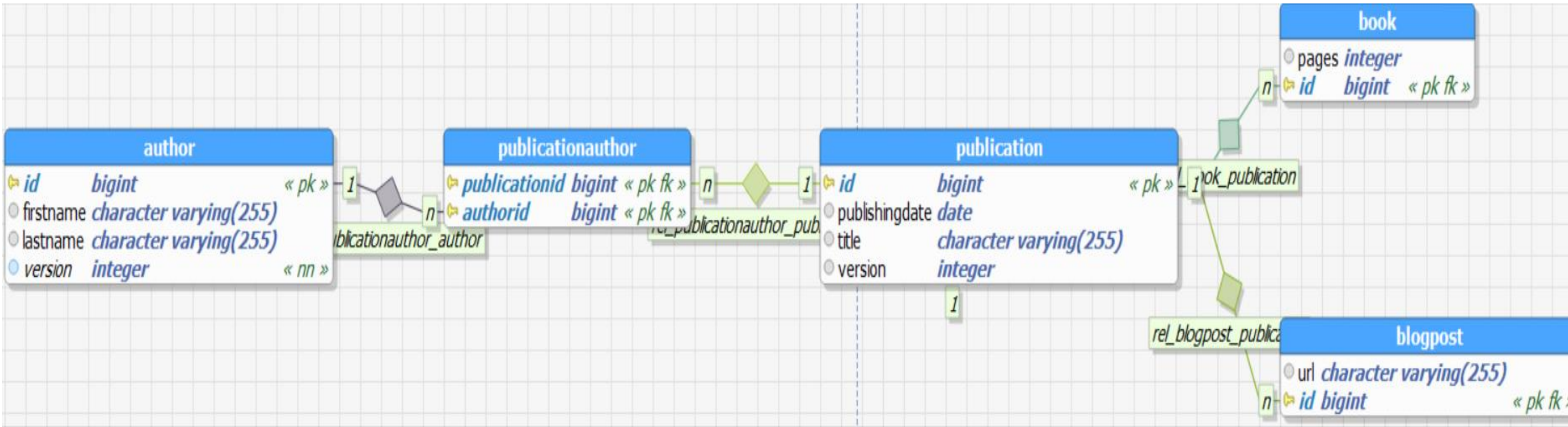

```
SELECT
    PA.*, P.*
FROM
    PublicationAuthor PA
    INNER JOIN
        (SELECT *, 1 as Type from Book  UNION ALL
        SELECT *, 2 as Type from BlogPost) P ON PA.publicationId = P.id
WHERE PA.authorId=?
```

Inheritance Mapping. Joined table



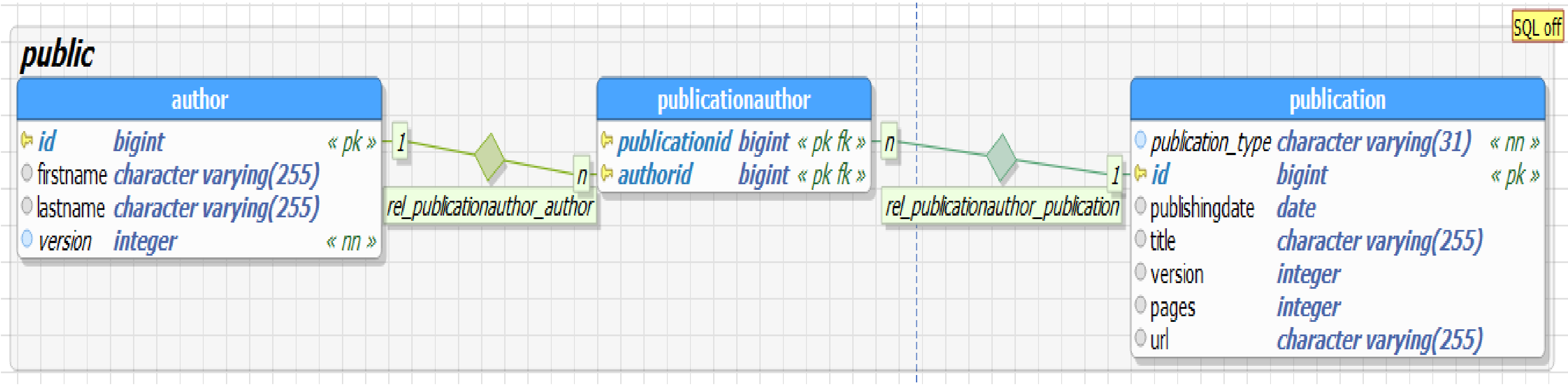
- Maps each subclass to its own relation (does not include attributes from superclass) + FK to parent's relation
- Maps the superclass to a relation as well
- Maps the relationships to a relation
- More joins are required as attributes are split between super class and subclasses
- Queries exhibit a better performance, but still complex

Inheritance Mapping. Joined table



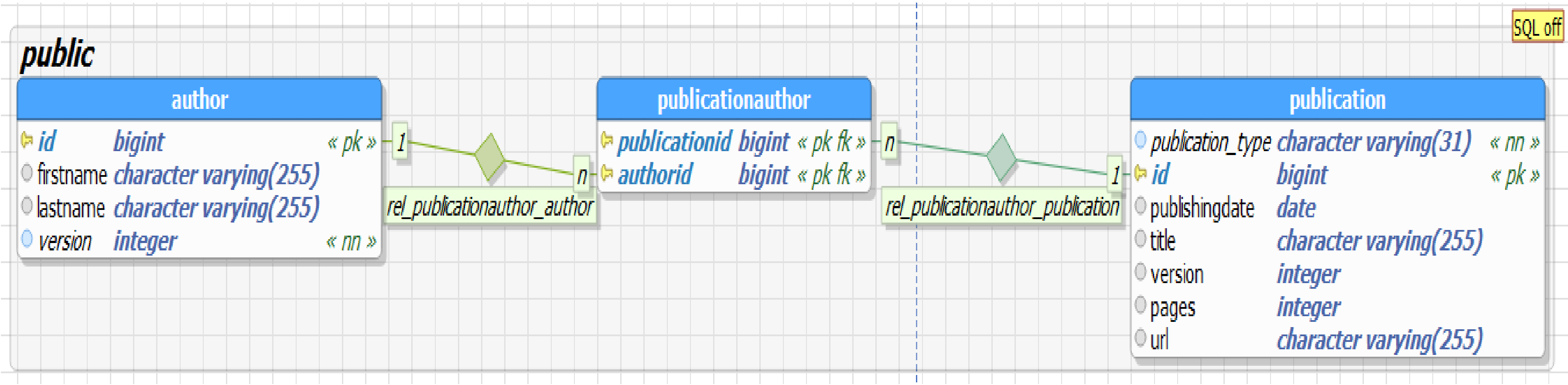

```
SELECT
    PA.*, P.id, P.publishingDate, P.title, P.version, B.pages, BP.url
FROM
    PublicationAuthor PA
    INNER JOIN Publication P ON PA.publicationId = P.id
    LEFT OUTER JOIN Book B ON P.id = B.id
    LEFT OUTER JOIN BlogPost BP ON P.id = BP.id
WHERE
    PA.authorId = ?
```

Inheritance Mapping. Single table



- Maps all entities of the inheritance structure to the same relation
- Easy to include in relationships; queries have best performance
- Drawbacks: lots of NULLs => data integrity may break
- An additional discriminator column is needed for the type

Inheritance Mapping. Single table




```
SELECT
    PA.*, P.*
FROM
    PublicationAuthor PA
    INNER JOIN Publication P ON PA.publicationId = P.id
WHERE
    PA.authorId = ?
```

Comparison of the four approaches

	Mapped superclass	Table per class	Single table	Joined
Attributes are scattered	No	No	No	Yes
Entity instances are scattered	Yes	Yes	No	Yes
Polymorphic queries / Does the superclass gets its own table	No	Yes	Yes	Yes

Choosing an approach

- If you require the best performance and need to use relationships, you should choose the single table strategy. But be aware, that you can't use not null constraints on subclass attributes which increase the risk of data inconsistencies.
- If data consistency is more important than performance and you need relationships, the joined strategy is probably your best option.
- If you don't need relationships, the table per class strategy is most likely the best fit. It allows you to use constraints to ensure data consistency and provides an (inefficient) option to express relationships.
- Use Mapped Superclass when the superclass factors out common properties of otherwise unrelated entities (e.g. auditing details – createdBy, createdAt, modifiedBy, modifiedAt, version)

IS-A (Inheritance) relationships

- Similar to the idea of subclasses in object-oriented programming: subclass = special case, fewer entities, and possibly more properties
  - Represented as a triangle (direction is important)
- Example: Graduate students are students, but they also have offices

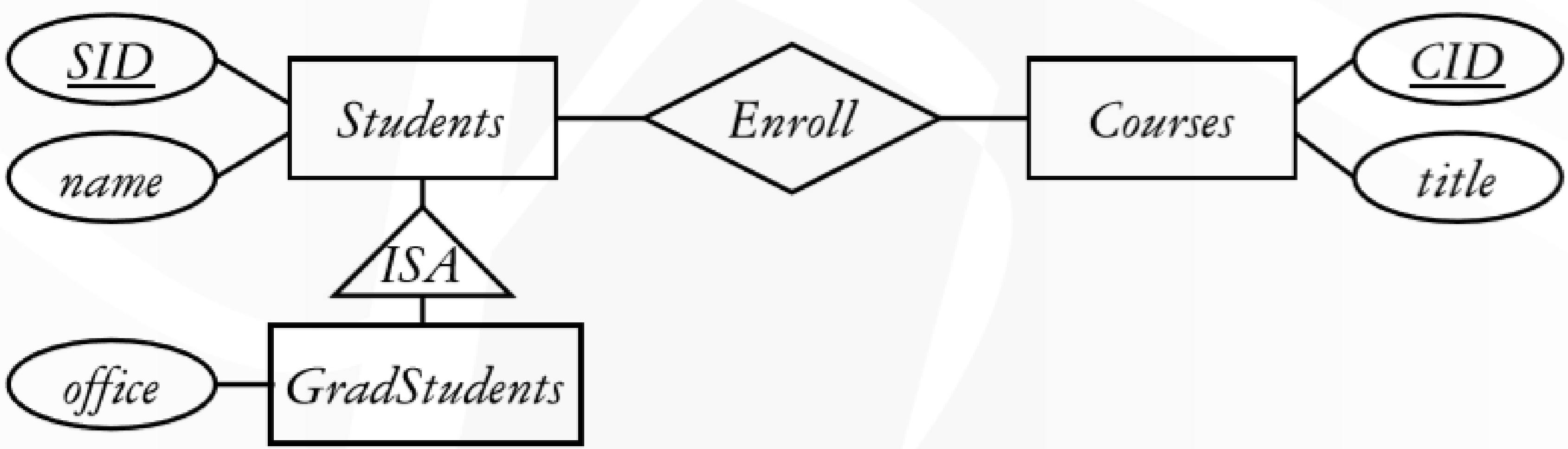
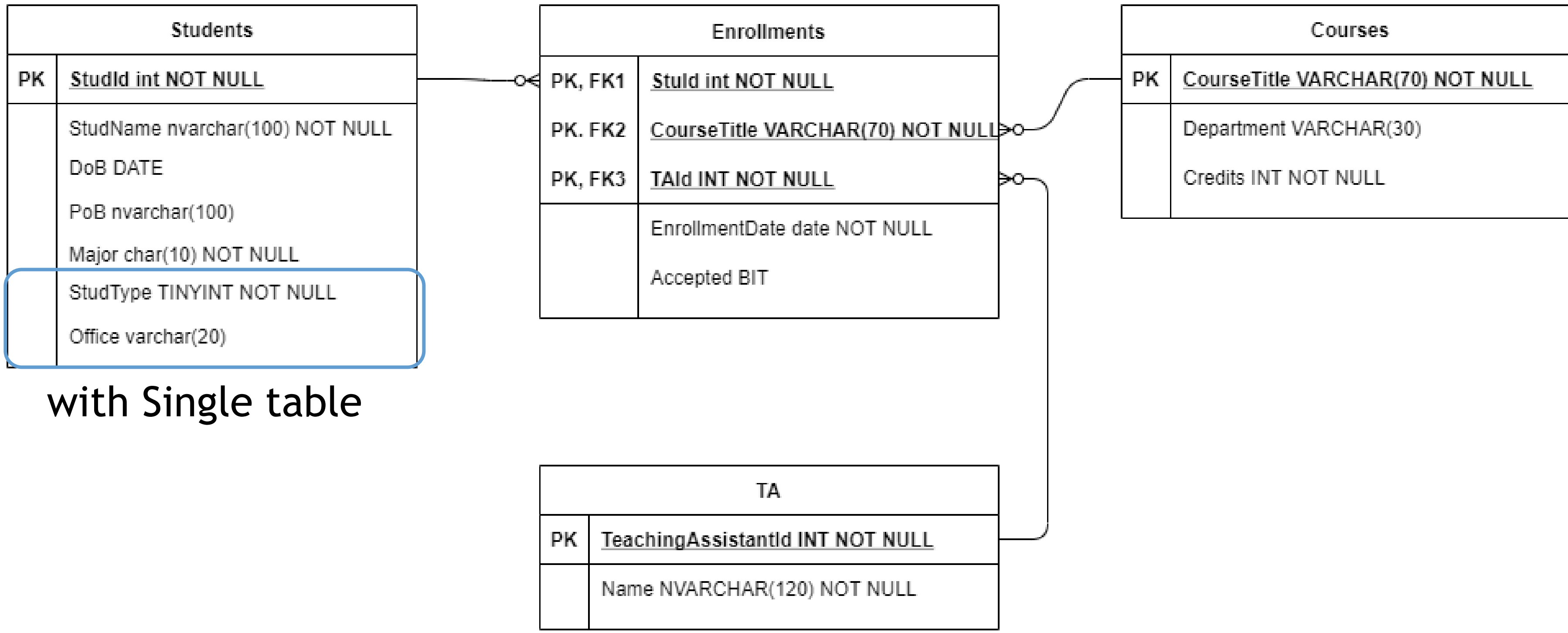


Table Diagram. Examples



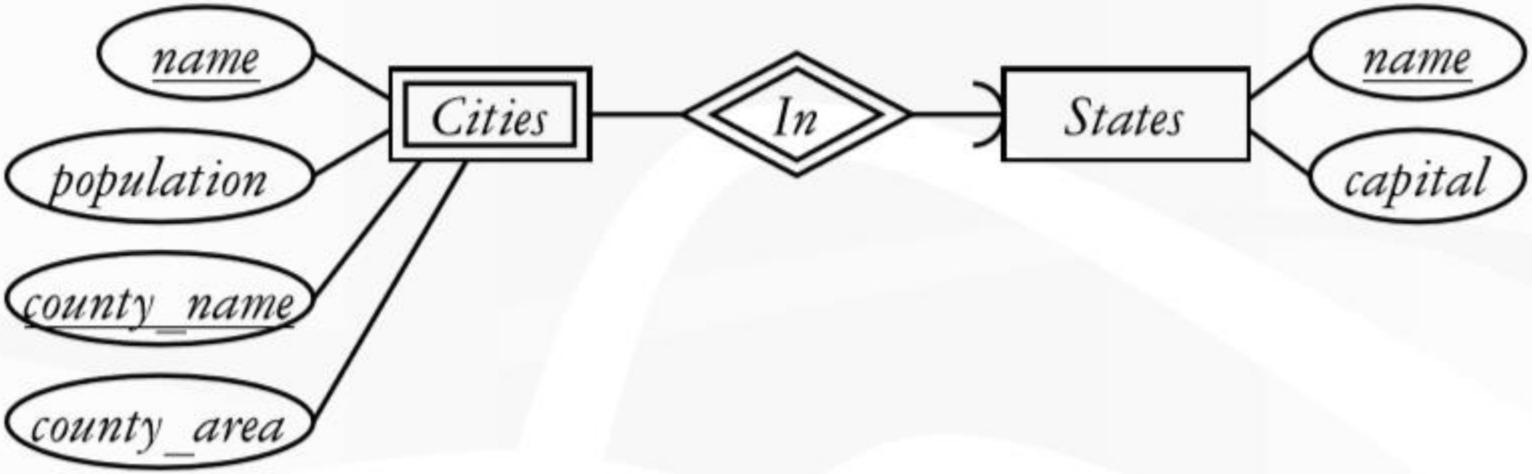
with Single table

Time for a Quiz

Case study 1

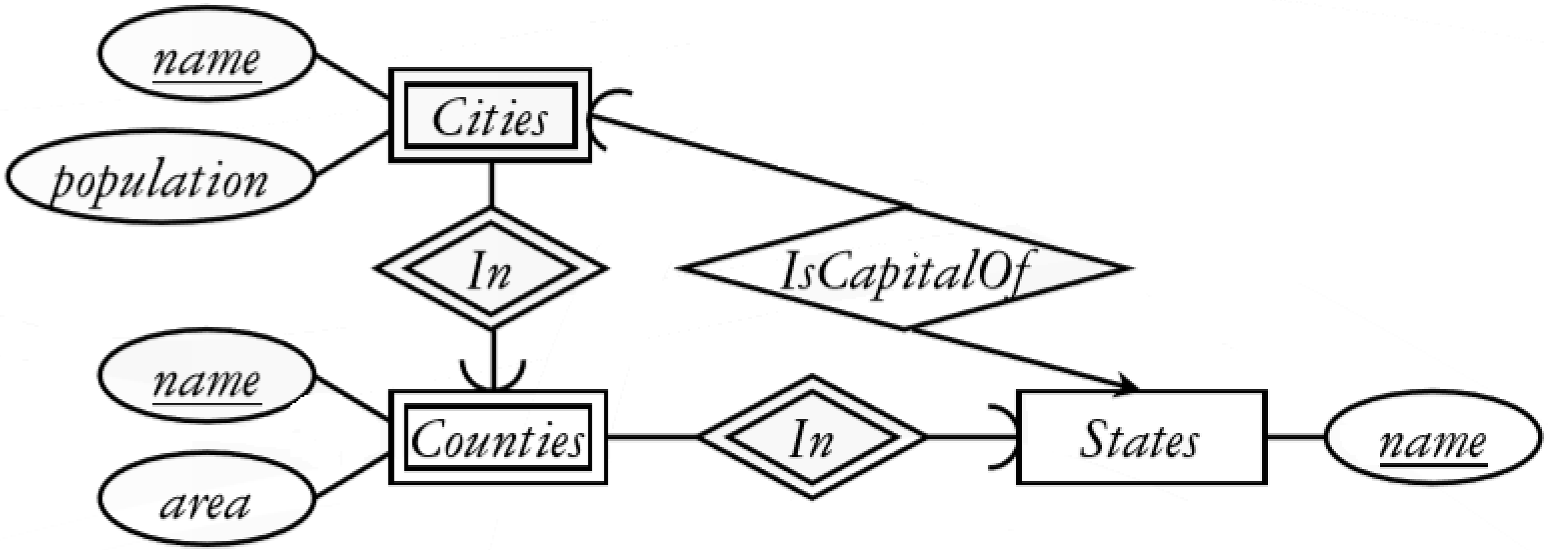
- Design a database representing cities, counties, and states
  - For states, record name and capital (city)
  - For counties, record name, area, and location (state)
  - For cities, record name, population, and location (county and state)
- Assume the following:
  - Names of states are unique
  - Names of counties are only unique within a state
  - Names of cities are only unique within a county
  - A city is always located in a single county
  - A county is always located in a single state

Case study 1. First design



- County area information is repeated for every city in the county
  - ☞ Redundancy is bad (why?)
- State capital should really be a city
  - ☞ Should "reference" entities through explicit relationships

Case study 1. Second design



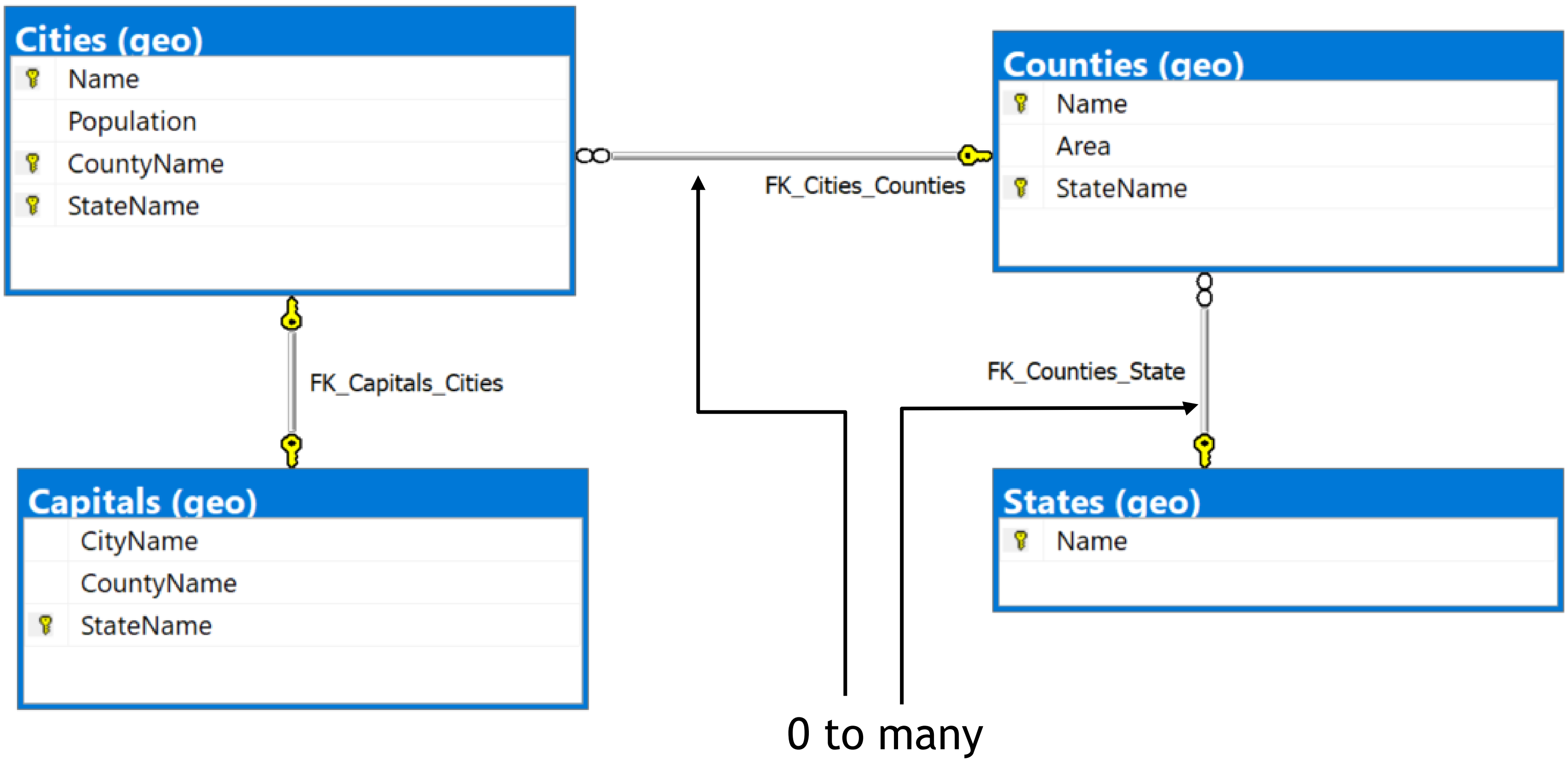
Case study 1. Second design discussion

- Database Schema
  States(Name)
  Counties(Name, Area, StateName)
  Cities(Name, Population, CountyName, StateName)
  CapitalOf(CityName, CountyName, StateName)

- Which NF?

- Is it anomalies free?

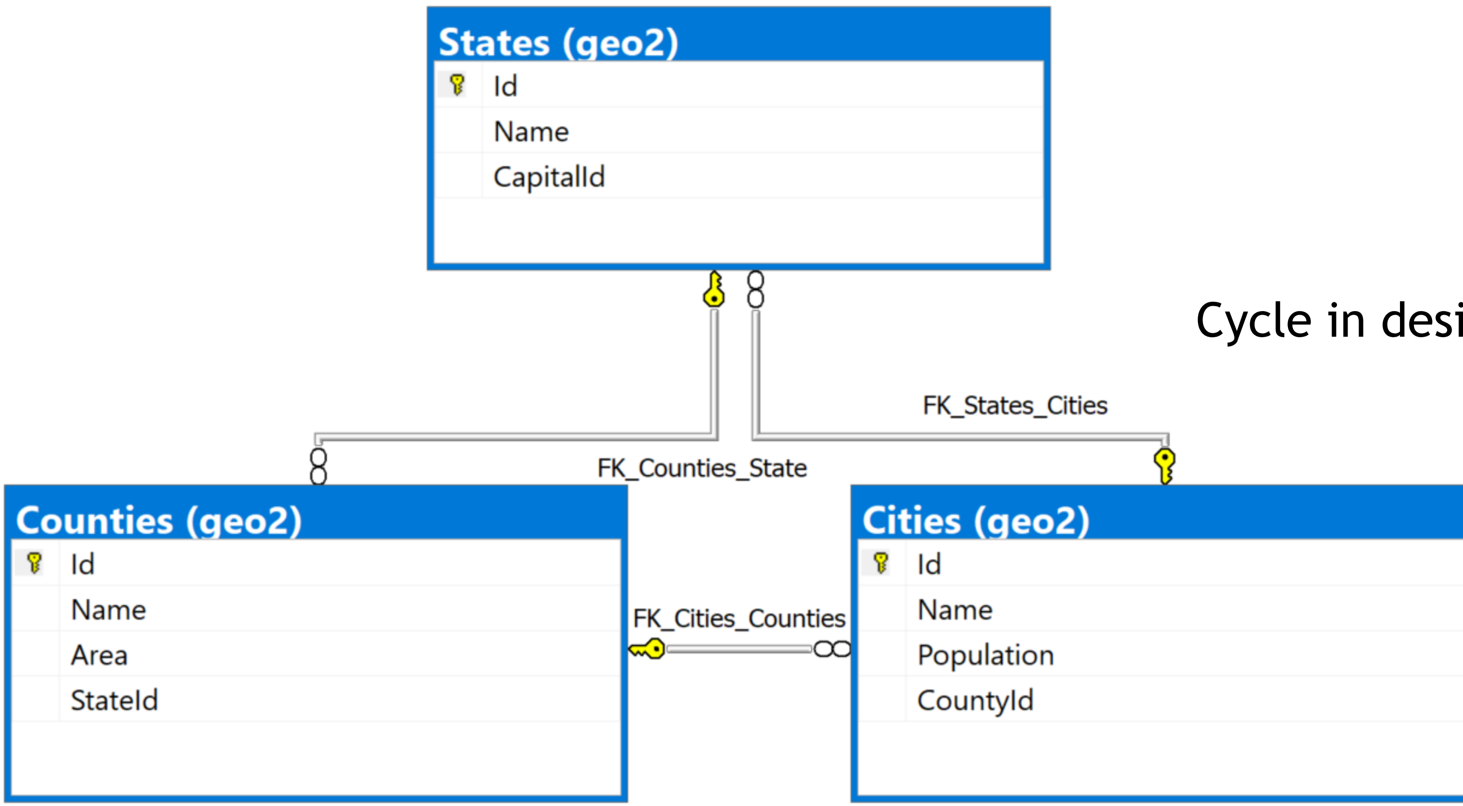
Case study 1. Second design discussion



This diagram has been exported from Microsoft SQL Server Studio and the notation differs a bit comparing to E-R symbols.

Case study 1. Third design

- Introduce surrogate PK
- Represent Capital in States



Cycle in design

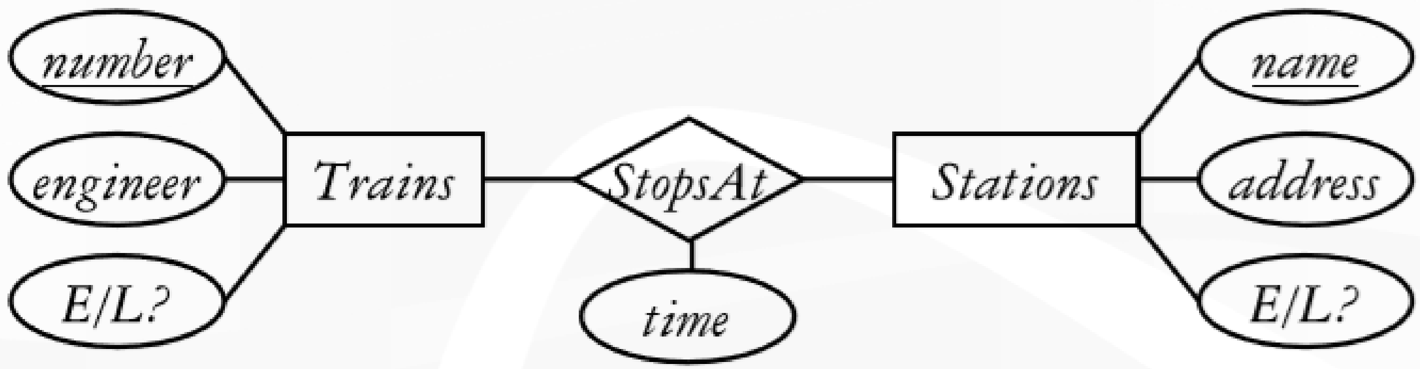
This diagram has been exported from Microsoft SQL Server Studio and the notation differs a bit comparing to E-R symbols.

Case study 2

Design a database consistent with the following:

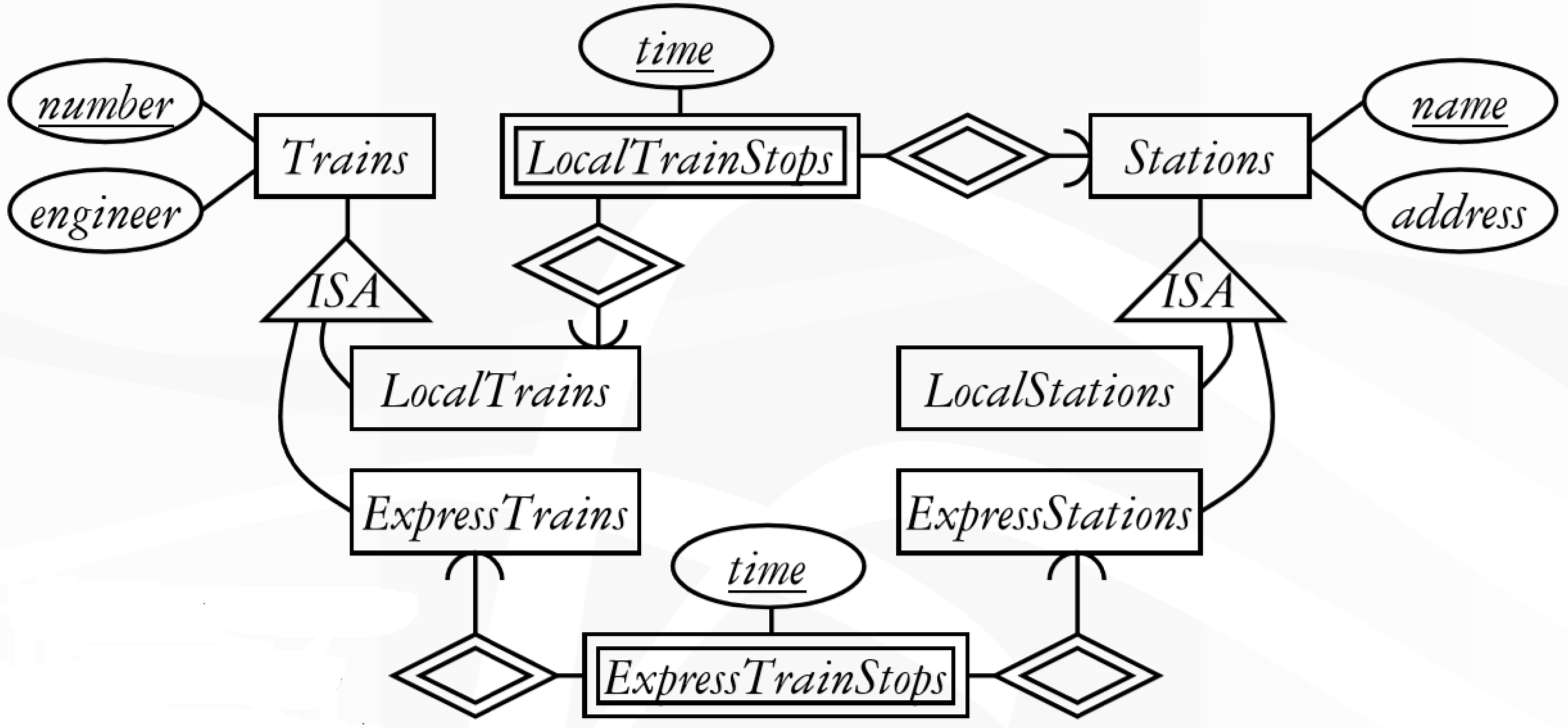
- A station has a unique name and an address, and is either an express station or a local station
- A train has a unique number and an engineer, and is either an express train or a local train
- A local train can stop at any station
- An express train only stops at express stations
- A train can stop at a station for any number of times during a day
- Train schedules are the same everyday

Case study 2. First design

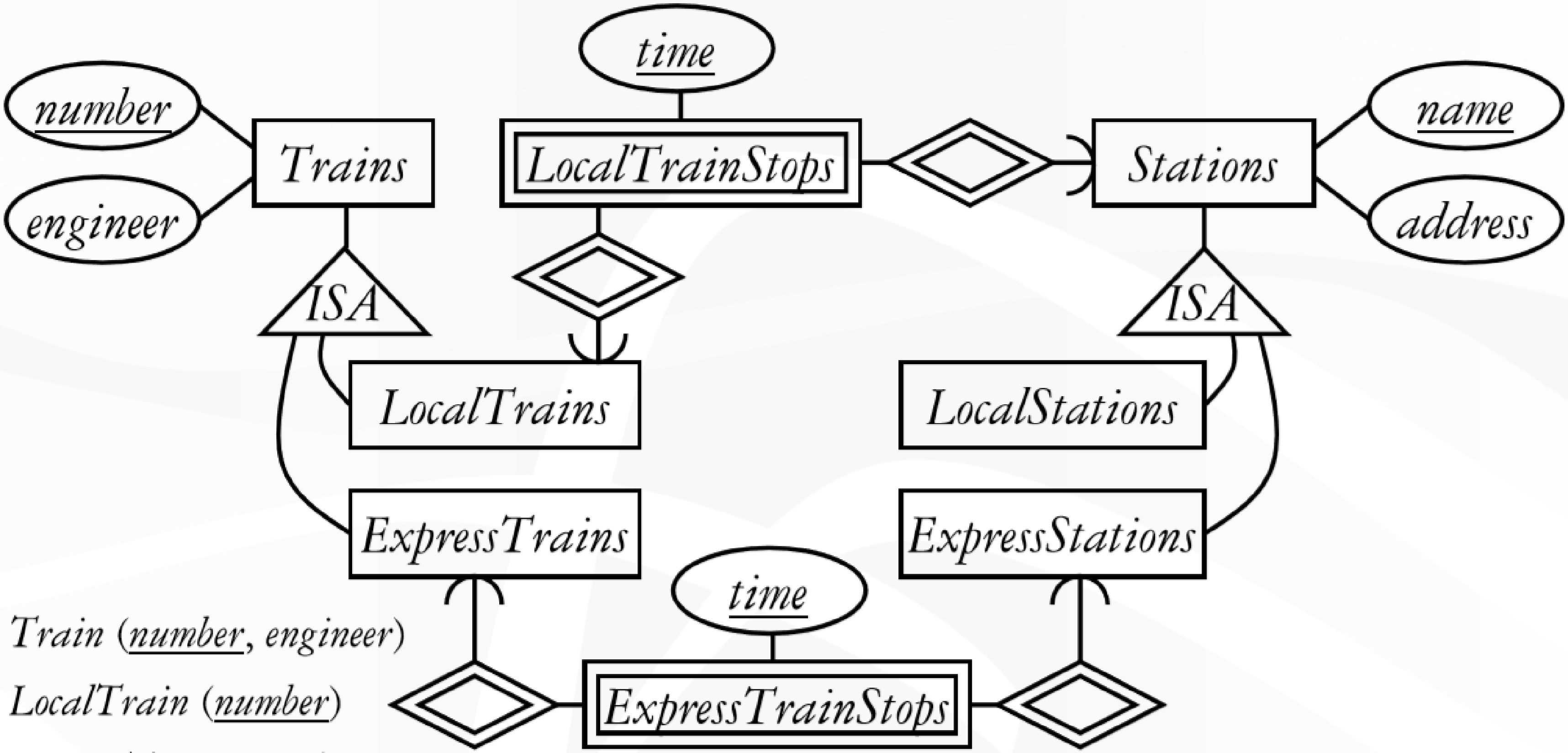


- Nothing in this design prevents express trains from stopping at local stations
  - ☞Should capture as many constraints as possible
- A train can stop at a station only once during a day
  - ☞Should not introduce constraints

Case study 2. Second design



Case study 2. Second design relational mapping



Train (number, engineer)

LocalTrain (number)

ExpressTrain (number)

Station (name, address)

LocalStation (name)

ExpressStation (name)

LocalTrainStop (local_train_number, station_name, time)

ExpressTrainStop (express_train_number, express_station_name, time)

Note that keys for *Local/ExpressTrainStop* come from assumptions not encoded in the E/R design

Case study 2. Second design refinement

Train (<u>*number*</u>, *engineer*), *LocalTrain* (<u>*number*</u>), *ExpressTrain* (<u>*number*</u>)
Station (<u>*name*</u>, *address*), *LocalStation* (<u>*name*</u>), *ExpressStation* (<u>*name*</u>)
LocalTrainStop (<u>*local_train_number*</u>, *station_name*, <u>*time*</u>)
ExpressTrainStop (<u>*express_train_number*</u>, *express_station_name*, <u>*time*</u>)

- Eliminate *LocalTrain* table
  - Can be computed as $\pi_{number}(Train) - ExpressTrain$
  - Slightly harder to check that *local_train_number* is indeed a local train number
- Eliminate *LocalStation* table
  - It can be computed as $\pi_{number}(Station) - ExpressStation$

Case study 2. Third design

Train (*number*, *engineer*, *type*)

Station (*name*, *address*, *type*)

TrainStop (*train_number*, *station_name*, *time*)

- Encode the type of train/station as a column rather than creating subclasses
- Some constraints are no longer captured
  - Type must be either "local" or "express"
  - Express trains only stop at express stations

  ☞ Fortunately, they can be expressed/declared explicitly as database constraints in SQL

☞ Arguably a better design because it is simpler!

Practical Design Recommendations

Practical Design Recommendations

- Avoid redundancy
- Everything should depend on the entire key and nothing but the key
- Carefully design the keys => BIG impact on performance, hence use integer type, introduce surrogates if necessary; capture natural keys as unique constraints
- Capture essential constraints; don’t introduce unnecessary ones
- Manage indexes
- Choose data types carefully
- Code style

Data types in selected DBMS

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

Note: Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**

SQL Data Types

- Exact numerics
  - BIGINT/INT/SMALLINT/TINYINT/BIT
  - SMALLMONY/MONEY - precision
  - DECIMAL/NUMERIC(p, s) - fractions
- Approximate numerics – floating point numeric data
  - FLOAT/REAL
- Character strings
  - CHAR(n)/VARCHAR(n)/TEXT – ASCII characters
    (VAR = variable-size)
  - *Collation* controls the code page that is used to store the character data
  - n defines the **string length in bytes** not the number of characters
- Character strings
  - National character strings
  - NCHAR(n)/NVARCHAR(n)/NTEXT - for everything else
  - n defines the **string length in byte-pairs** not the number of characters
- Date and tine
  - DATE, TIME, TIMESTAMP
- Binary strings
  - BINARY/VARBINARY/IMAGE
- Other data types
  - UNIQUEIDENTIFIER – 16-byte GUID, replication, hide next key, performance
  - XML – subset of XQuery language
  - Spatial geometry/geography types

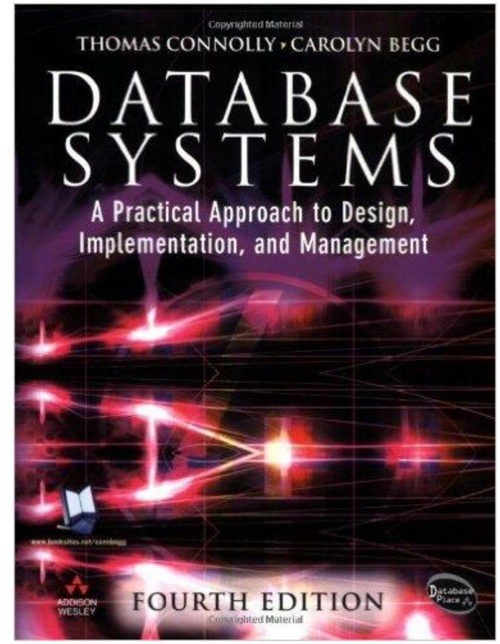
Code Conventions

- Avoid using reserved words for naming tables, fields, constraints (even variables
- Use schema to group the tables and stored procedures of a specific part of the application (e.g. instead of using [dbo].[SalesCustomer], use [Sales].[Customer])

- **Table names**
  - use singular nouns (e.g. use employee instead of employees)
  - use a single word that describes the table (if it is possible)
- **Field names**
  - do not use a table name into field names
  - keep them as short as possible
- **Constraints (PK / FK / etc)**
  - preferably use id to name a single Primary Key or a word to describe its unicity
  - use the name of the tables in a Foreing Key name (FK_<TargetTable>_<SourceTable>)
  - the name of a composite FK should contain all keys

Bibliography (recommended)

Teoria generala a bazelor de date, I. Despi, G. Petrov, R. Reisz, A. Stepan, Mirton, 2000
Cap 3



Database Systems - A Practical Approach to Design, Implementation, and Management (4th edition) by Thomas Connolly and Carolyn Begg, Addison-Wesley, 2004
Chapter 11, 12

References

- Mapping inheritance to Relational Model
  - https://www.thoughts-on-java.org/complete-guide-inheritance-strategies-jpa-hibernate/
- SQL Coding Style & Best Practices
  - https://www.red-gate.com/simple-talk/sql/t-sql-programming/sql-code-smells/
  - https://www.red-gate.com/simple-talk/sql/t-sql-programming/basics-good-t-sql-coding-style/