Web and Database Technology

The Relational Model

Christoph Lofi

TU Delft WIS Web Information Systems

Exam

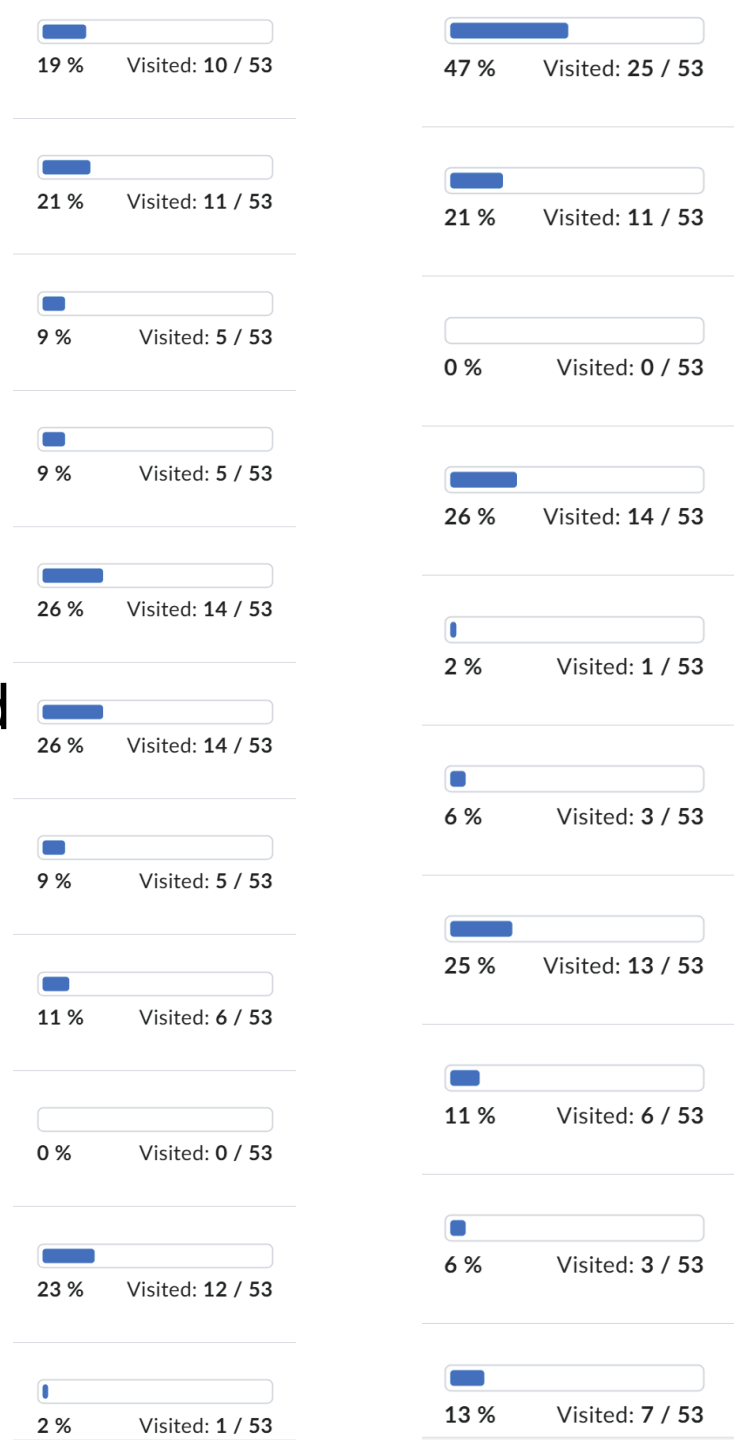
- Friday 12th December
  - Hybrid Digital / Paper
  - Comparable to previous years in “spirit”
  - All topics covered:
    - Schema Design (conceptual / logical / c->l)
    - SQL
    - Functional Dependencies
    - Open Questions and discussions
    - (not detours….)

Exam

- DO THE LABS!!!
  - Discuss with your peers!
- Check the old exams (but do not overfit on them)
- Study the slides
- Check out the mandatory and optional material on Brightspace

- Students clicking on things in Brightspace…

Check out the mandatory and material on Brightspace



Web and Database Technologies – Midterm Exam Resit

CSE 1500 - Q3 2023/24 - 12.04.2023 - Task Sheet

Christoph Lofi – Ujwal Gadiraju

Important Information:

- **On this sheet you find the exam tasks. You write your responses on the accompanying answer sheet that has been distributed to you!**
- You may use the distributed scratch paper as you see fit, but do not use it for your final solutions.
- The overall time is **90 minutes**. Please briefly assess all tasks before starting and **prioritize the order** of your work per your strengths and weaknesses!
- **After the exam time is up, stop writing and stay seated (!!!).** We will collect the exam at your place. **Please wait until we announce that all exams have been collected**, and only leave your place after that!
- The only external items you may use are pens.

Task	max
1 Conceptual Modelling - Open Questions - EER Modelling	37
2 Logical Relational Schemas	20
3.1 FDs & Normalization - Multiple Choice	9
3.2 Normalization - Normalization	12
3.3 Functional Dependencies - FD modelling	8
4 SQL	20
5 Miscellaneous Questions	18
Overall	124

TU Delft Web Information Systems

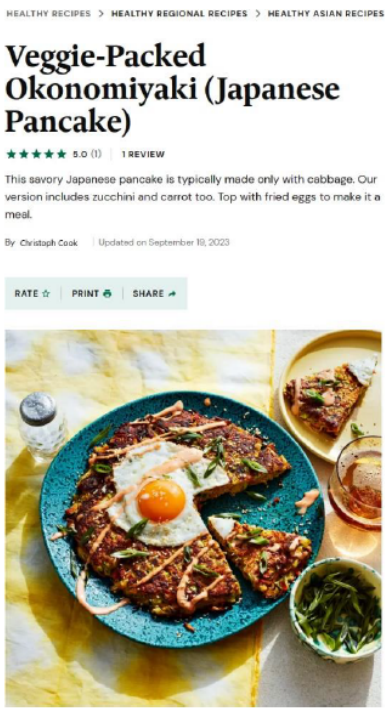
1 Conceptual Modelling (7+30 = 37 points)

Scenario:

You are tasked to design a new cooking website to promote more healthy living. At its core, the webpage serves as a digital repository for recipes, cooking tips, and culinary inspiration, but also gives insights into the nutritional values of food. The following description was provided by the client to inform the requirement engineering process.

The client describes the website by using a mock-up example of the user interface with the following notes. Read the following client's requirement description!

On page 5, you will find tasks relating to this requirement description.



1: Recipe Summary at the top of each page

Each recipe has an author.

Authors are also users of the site but will have additional functionality in their profile page like links to all the recipes they posted and can see some simple statistics for each of their recipes like number of ratings, average rating, number of reviews, and number of page views. Authors can also delete their recipes on their profile page, which also deletes all associated statistics and ratings.

Recipes can be rated by users with 1-5 stars.

In addition to the star rating, users can also review a recipe by writing a short review text, but they can also provide only a rating without additional review text.

The counter shown in the example on the left for "number of reviews" simply counts how many ratings are provided in a review text.

Users can also print the recipe, or forward it to popular social media like Instagram, Facebook, etc.

Each recipe is associated with one or more categories (the shown example recipe is in the "healthy Asian recipes" category.).

Categories can be sub-categories of other categories, e.g., "healthy regional recipes" is the super-category of "healthy Asian recipes".

The picture of the dish is stored as an URL reference.

1.1 Schema Design Decisions (7 points)

a) 2 points: The client suggests using email address as a primary key for users. Provide at least 1 strong pro argument supporting this idea, and 1 strong con argument speaking against this idea.
b) 3 points: For the cooking directions (description part 4) of the requirements, the client insists on the system being able to reuse the texts of individual cooking steps between recipes. (e.g.: " 'Whisk mayonnaise, Siracha, and sesame oil in a small bowl.' is step 1 in this recipe but can be step 5 in another recipe.")
Provide at least two strong arguments of why this requirement is not a good idea and should be dropped, and briefly what you would recommend doing instead to achieve a better system design.
c) 2 points: Should "ratings" be modelled as a weak entity? Provide at least one strong argument for or against this. If you decide that ratings should be a weak entity, from which strong entity(s) should ratings borrow part of its identity?

1.2 Conceptual Schema Design (30 points)

Design a **Conceptual EER data schema** satisfying the description on page 2 to 4. Use **Chen EER Notation!**
Try to be as complete as possible. Minimize redundancy! Provide all primary keys and cardinalities! Comment on your most important design choices and explain which described data requirements or constraints could not be represented in your diagram (if any).

Also, the client refused to listen to any of your arguments you might have provided for task 1.1, and wants the schema designed to match their description on page 2-4. (to clarify: task 1.2 is fully independent from 1.1)

2 Logical Relational Schema (20 points)

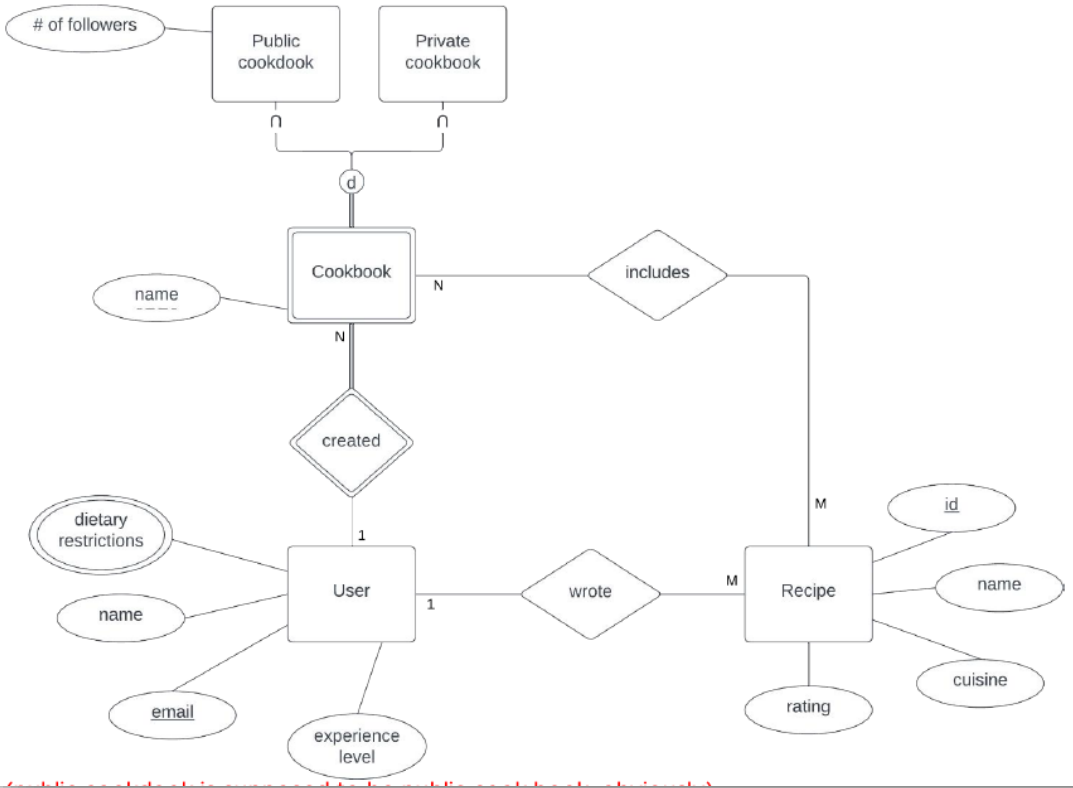
You are given the following conceptual EER schema, describing a simple cookbook recipe collection app. Users can collect recipes in their own virtual cookbook (which is either private or public), but can also write their own recipes.

Your task is to translate this conceptual EER schema to a logical relational schema. Be aware that there might be multiple correct solutions, but you should opt for the one with less data redundancy, and be as efficiently as possible!

For each relation, specify attributes, primary keys, and foreign key constraints. You are allowed and encouraged to enrich your answer with comments that clarify the assumptions you made.

As a reminder, you are required to use the following logical relational schema notation:

```
BOOK (ID, Title, Publisher -> PUBLISHER(Name))
PUBLISHER (Name, Address, Phone)
```



b) **(3 points)** Given the following functional dependencies and the relation R(A, B, C, D), which answers apply?

{A} → {B}
{B} → {C}
{A, C} → {D}

A. {A} → {D} is a valid FD.
B. {A, B} is a candidate key of R.
C. {A, B, D} is a superkey of R.
D. Relation R is in 3NF.

c) **(3 points)** Given the following functional dependencies and the relation R(A, B, C, D, E), which answers apply?

{A, B, C} → {D, E}
C → E
E → {A, B}
B → D

A. {C} is a candidate key for R.
B. {A, B, C} is a superkey for R.
C. Relation R is in 3NF.
D. {A, B} cannot functionally determine all other attributes of R.

3.2 Normalization (12 points)

Consider the relation `Refrigerator(Model_Id, Year, Price, Factory, Color)` storing data on refrigerators. We abbreviate this as `R(M, Y, P, F, C)`.

Also, the following set of functional dependencies are given:

`M → F,`

`{M, Y} → P`

`F → C`

The diagram of functional dependencies is the following:

a) Normalize the above relation into 1NF and (very) briefly explain why your result is correct. What is your chosen candidate key?
b) Normalize the above relation into 2NF (but not yet into 3NF) and briefly explain why your result is correct. Mention keys.
c) Normalize the above relation into 3NF and briefly explain why your result is correct. Mention keys.

We do not consider follow-up errors for this task.

4 SQL (20 points)

The following schema describes recipes. Recipes have an id, a name, a preparation and a cook time, a difficulty (e.g., easy/medium/hard) a cuisine (e.g., French) and some instructions. A recipe can have one or more prerequisite recipes (e.g., a sauce). Apart from this, each recipe can also has some ingredients. The recipe ingredients have some preparation instructions and the number of units needed for that ingredient. Units abstract different measurements units like gram or liter. Ingredients have a name, calories per unit, and a Boolean field describing whether they are vegetarian.

```
PREREQUISITE (recipe_id → RECIPE(id), required_recipe_id → RECIPE(id))
RECIPE (id, name, prep_time, cook_time, difficulty, cuisine,
   instructions)
RECIPE_INGREDIENT (recipe_id → RECIPE(id),
   ingredient_id → INGREDIENT(id),units, prep_instructions)
INGREDIENT (id, name, calories_per_unit, is_vegetarian)
```

Write SQL queries that fulfill the following tasks, based on the schema and information provided above.

4.1. **(4 Points)** List the average number of calories of all vegetarian and non-vegetarian ingredients. The result schema should be (is_vegetarian, average_number_of_calories_per_unit), and the result will have exactly 2 rows.

4.2. **(5 Points)** For each 'Easy' recipe, list the total calories (i.e. the sum of the calories of each ingredient in their used quantities).
You should NOT consider the prerequisite recipes in this calculation!
The result schema should be (recipe_id, recipe_name, total_calories)

4.3. **(5 Points)** List all ingredients which are used in at least 10 and at most 15 different recipes. The result schema should be (ingredient_id, ingredient_name).

4.4. **(6 Points)** List all vegetarian recipes (i.e. recipes which use only vegetarian ingredients for themselves and their prerequisites) which have at least one prerequisite.
It is guaranteed by the application that prerequisite recipes are not having prerequisites themselves.
The result schema should be (recipe_id, recipe_name).

5 Miscellaneous Questions (18 points)

Briefly answer the following questions:

5.1. **(3 Points)** Consider the logical schema from Task 4. Modify the following query such that it yields the same results **without** using the `DISTINCT` keyword:

```
SELECT DISTINCT r.name FROM recipe r ORDER BY r.name DESC
```

5.2. **(3 Points)** Consider the logical schema from Task 4. Briefly describe the result of the following query:

```
SELECT * FROM Ingredient i, Recipe r
INTERSECT
SELECT * FROM Recipe r
```

5.3. **(2 Points)** Briefly define what Data Models are (such as for example the Relational Model or Document Model).

5.4. **(2 Points)** Briefly define what an SQL Injection Attack is.

5.5. **(3 Points)** Given the schema `R(a, b)` and `S(c → R(a), d)`, briefly explain the differences or commonalities of the result sets of the following two queries:

```
(1)   SELECT * FROM R, S WHERE R.a = S.c

(2)   SELECT * FROM R JOIN S ON S.c = R.a
```

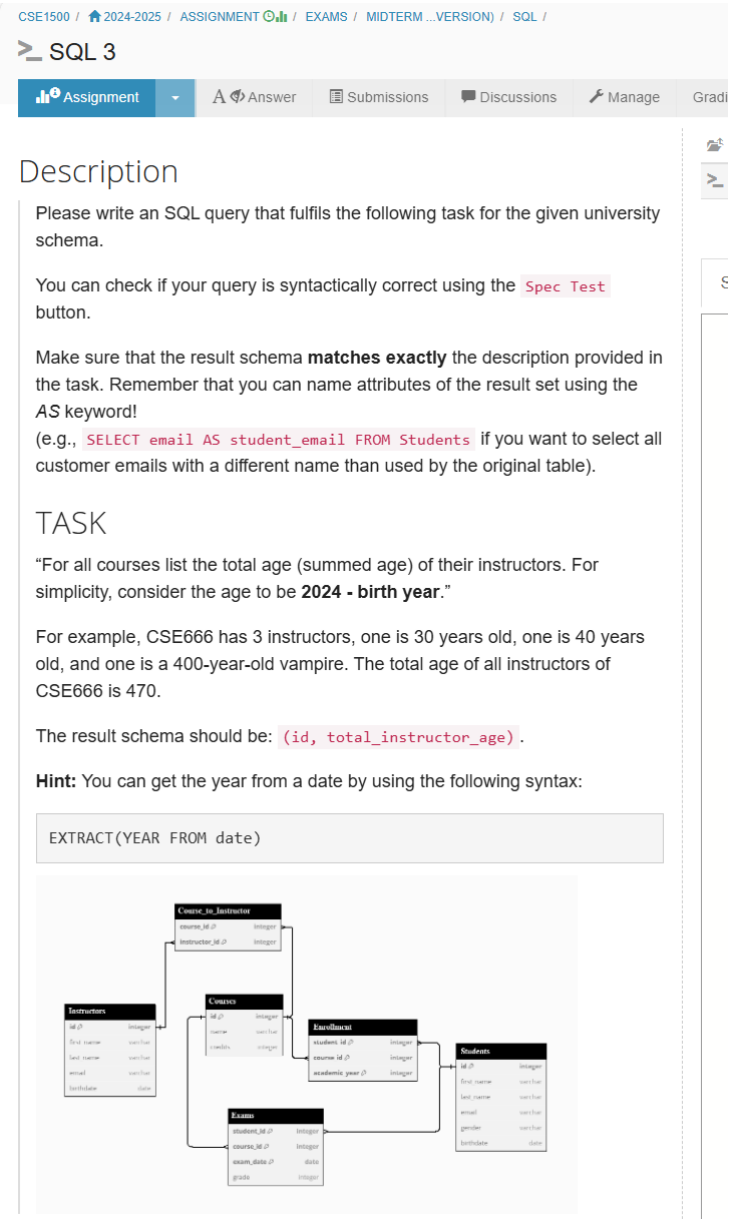
5.6. **(3 Points)** For the schema from task 4, briefly describe the result of the following query:

```
SELECT r.id, ri.recipe_id FROM recipe r
LEFT JOIN recipe_ingredient ri ON ri.recipe_id=r.id
GROUP BY ri.recipe_id, r.id
HAVING r.cook_time > 5
UNION
SELECT r.id, ri.recipe_id FROM recipe r, recipe_ingredient ri
```

5.7. **(2 Points)** Provide at least 2 strong arguments for why it is good practice to first design a Conceptual Schema and then transform that into a Logical Schema, as compared to only designing a Logical Schema (skipping the Conceptual Schema altogether)

SQL in Weblab

- You **will get 0 points** if
  - Your query has syntax errors
  - Returns an empty result
  - Weblab will tell you that you have syntax errors or empty results!
- You will get full points if your query produced the correct result
- You might get partial points if your query does fit the defined error cases
  - Some of these cases are predefined, as for example, “wrong order of results”
  - Most partial point cases will be created during grading
- The only feedback you get during the exam is:
  - Syntax Error / Empty Result: 0 points
  - Perfect: full points
  - Predfined partial point case: some points
  - “potential partial points (or not), wait for grading”



Goal for Today

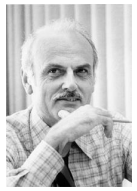
- RECAP (again…): Data Models
- From sets to relations: Intro to Relational Model

A brief history of databases

1964
1st database

DB
DB.io
Database of Databases
Integrated Data Store

1970
Relational model

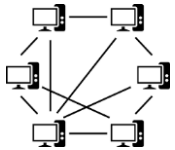


Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

1990s
Distributed databases,
Data integration,
Object-oriented database,
Multimedia database



2010s
Big data,
NoSQL databases,
Cloud computing

2024
?

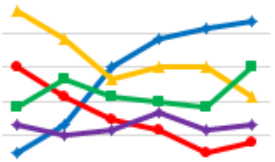
Time

Why the relational model?

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.



trend chart

416 systems in ranking, November 2023

Rank Nov 2023	Rank Oct 2023	Rank Nov 2022	DBMS	Database Model	Score Nov 2023	Score Oct 2023	Score Nov 2022
1.	1.	1.	Oracle	Relational, Multi-model	1277.03	+15.61	+35.34
2.	2.	2.	MySQL	Relational, Multi-model	1115.24	-18.07	-90.30
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	911.42	+14.54	-1.09
4.	4.	4.	PostgreSQL	Relational, Multi-model	636.86	-1.96	+13.70
5.	5.	5.	MongoDB	Document, Multi-model	428.55	-2.87	-49.35
6.	6.	6.	Redis	Key-value, Multi-model	160.02	-2.95	-22.03
7.	7.	7.	Elasticsearch	Search engine, Multi-model	139.62	+2.48	-10.70
8.	8.	8.	IBM Db2	Relational, Multi-model	136.00	+1.13	-13.56
9.	9.	↑ 10.	SQLite	Relational	124.58	-0.56	-10.05
10.	10.	↓ 9.	Microsoft Access	Relational	124.49	+0.18	-10.53

https://db-engines.com/en/ranking

Recap: Data Models / Theory

- A **data model** (theory) consists of three parts
  - Structure
    - **data structures** are used to create databases representing the modeled objects
  - Integrity
    - rules expressing the **constraints** placed on these data structures to ensure structural integrity
  - Manipulation
    - operators that can be applied to the data structures, to **update** and **query** the data contained in the database

Data Model Theories

- There are several common data models
  - **Relational Model**
  - Key-Value Model
  - Document-centered Model
  - Graph Model
  - Etc.

Relational Model Example

- Core Idea: We store data in Tables!!

movie

m_id	title	year
1	Terminator 2 – Judgement Day	1991
2	Twins	1988

Primary key: m_id

actor

a_id	f_name	l_name
1	Arnold	Schwarzenegger
2	Linda	Hamilton
3	Danny	DeVito

Primary key: a_id

cast

m_id	a_id	role
1	1	T-800
1	2	Sarah Connor
2	1	Julius Benedict
2	3	Vincent Benedict

Primary key: (m_id, a_id)
- Note: This allows each actor to only have one role per movie. Primary key should be (m_id, a_id, role) to allow multiple roles per movie.

Foreign key constraints: (m_id-> movie.m_id, a_id-> actor.a_id)

Relational Model Example

Columns (or also “attribute”)

movie

m_id	title	year
1	Terminator 2 – Judgement Day	1991
2	Twins	1988

actor

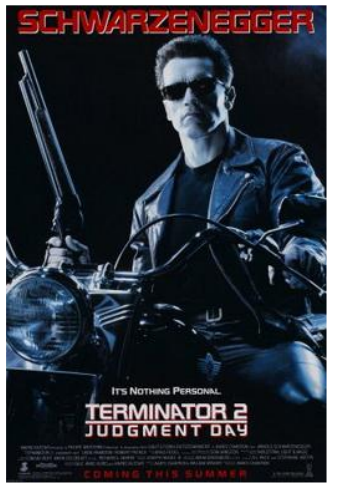
a_id	f_name	l_name
1	Arnold	Schwarzenegger
2	Linda	Hamilton
3	Danny	DeVito

cast

m_id	a_id	role
1	1	T-800
1	2	Sarah Connor
2	1	Julius Benedict
2	3	Vincent Benedict

Rows
(or also “tuples”)

Table (or also: “Relation”)



Relational Model Example

movie

m_id	title	year
1	Terminator 2 – Judgement Day	1991
2	Twins	1988

Primary key: m_id

actor

a_id	f_name	l_name
1	Arnold	Schwarzenegger
2	Linda	Hamilton
3	Danny	DeVito

Primary key: a_id

Foreign Key Constraint
(just the graphical representation of it)

cast

m_id	a_id	role
1	1	T-800
1	2	Sarah Connor
2	1	Julius Benedict
2	3	Vincent Benedict

Primary key: (m_id, a_id)

- Note for later: This allows each actor to only have one role per movie. Primary key should be (m_id, a_id, role) to allow multiple roles per movie.
- Foreign key constraints: (m_id-> movie.m_id, a_id-> actor.a_id)

Document Model Example

Movie-Centered Documents

```
"terminator2":

{ "title" : "Terminator 2 : Judgement Day",
  "year"  : 1991,
  "director" : {
    "first_name" : "James",
    "last_name"  : "Cameron" }
  "actors": [
    {"first_name" : "Arnold", "last_name" : "Schwarzenegger" },
    {"first_name" : "Linda", "last_name" : "Hamilton" },
    {"first_name" : "Edward", "last_name" : "Furlong" }
  ]
}
```

```
"twins":
{ "title" : "Twins",
  "year"  : 1988,
  "director" : {
    "first_name" : "Ivan",
    "last_name"  : "Reitmann" }
  "actors": [
    {"first_name" : "Arnold", "last_name" : "Schwarzenegger" },
    {"first_name" : "Danny", "last_name" : "DeVito" },
    {"first_name" : "Kelly", "last_name" : "Preston" }
  ]
}
```

Actor-Centered Documents

```
"arni":
{ "first_name" : "Arnold",
  "last_name" : "Schwarzenegger",
  "movies" : [
    {
       "title" : "The Terminator 2",
       "year"  : "1991" }
    {  "title" : "Twins",
       "year"  : "1988" }
  ]
}
```

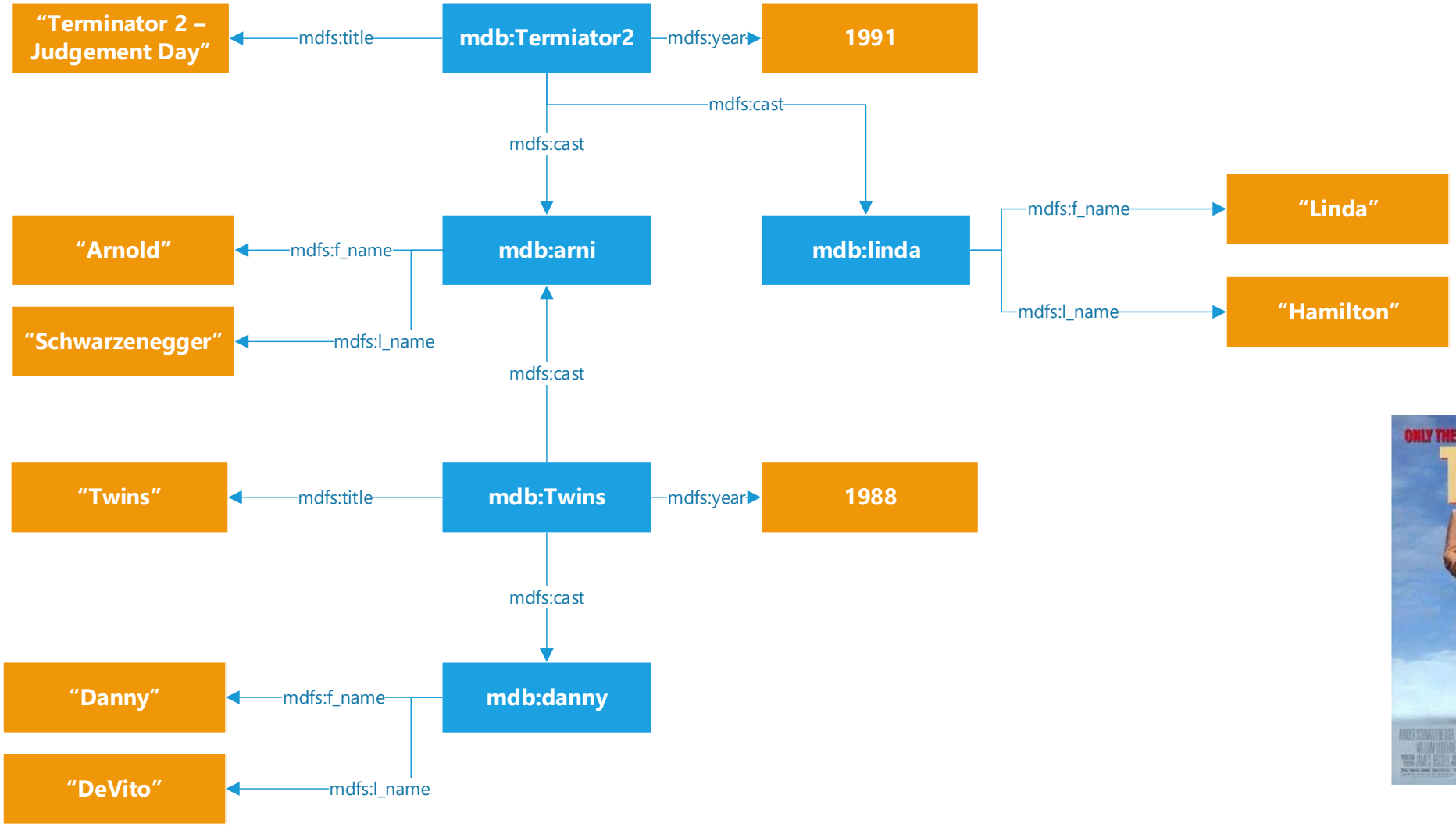
```
"linda":
{ "first_name" : "Linda",
  "last_name" : "Hamilton",
  "movies" : [
    {
       "title" : "The Terminator 2",
       "year"  : "1991" }
  ]
}
```

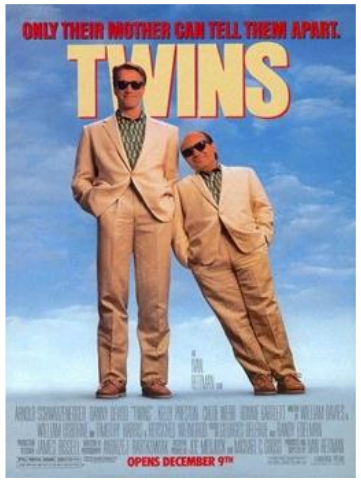
Key-Value Store Example

Terminator 2

Twins 2

Graph Model Example





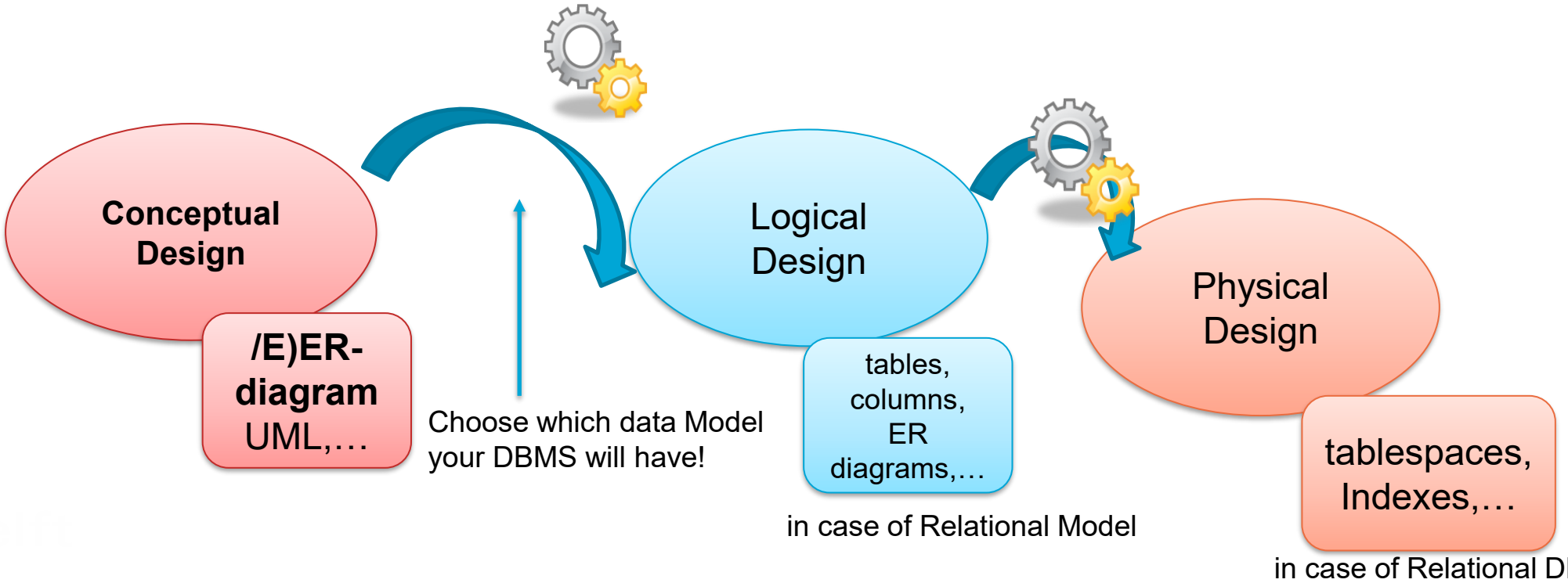
Data Models

- A **Data Model** describes data objects, operations and their effects
- Data Definition Language (**DDL**)
  - How to structure data?
    - Relational Databases use SQL for that
- Data Manipulation Language (**DML**)
  - How to add, edit, delete data?
    - Relational Databases *also* use SQL for that

  - **DML** and **DDL** are usually clearly separated, since they handle **data** and **meta-data**, respectively

Recap: Towards Schema Design !!

- Modeling the data involves three design phases
  - result of one phase is input of the next phase
  - often, automatic transition is possible with some additional designer feedback



Recep: Schemas and Instances

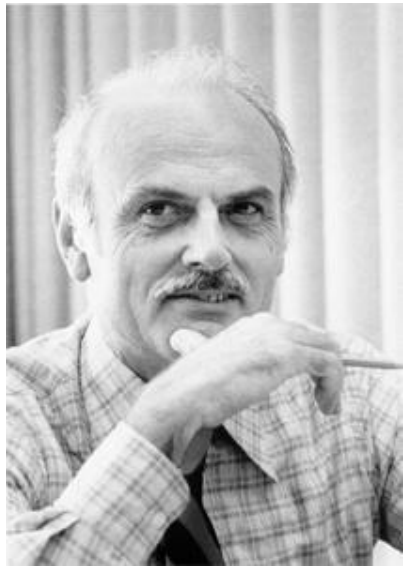
- Later in modeling, we will again introduce 3 layers of schemas
  - Schemas abstract the data to be stored about the real world
  - **Conceptual Schema**
    - What entities are important, and what are their relationships?
      - “I want to store movies and actors. Each movie can have multiple actors. “
  - **Logical Schema**
    - Transforms the conceptual schema with respect to the chosen Data Model
    - Relational Logical Schema: Describes tables and their attributes, keys and constraints
      - “Actor(id, firstname, lastname);
        Movie(id, title, year);
        a2m(a_id->Actor(id), m_id->Movie(id), role)”
  - **Physical Schema**
    - Describes HOW data is organized on disks

Recap: Terms

- **Database**: “A collection of related data represented (using a data model and a defined data schema)”
- **Database Management System** (DBMS): “A software system managing and maintaining a database”
- **Data Model**: “A formal definition on how to represent data (in general) and the available data operations.”
- **Data Schema**: “A definition of the structure of a specific database”

Relational Model Concepts

- A Relation is a mathematical concept based on the ideas of sets



- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper:
  - "A Relational Model for Large Shared Data Banks“, Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

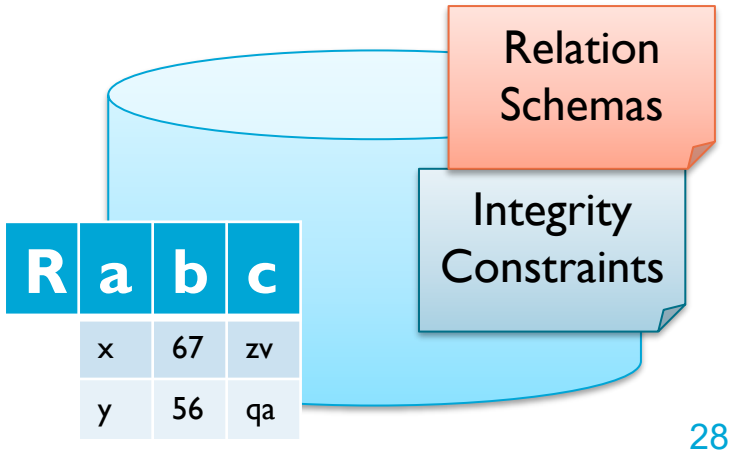
Relational?

In short!

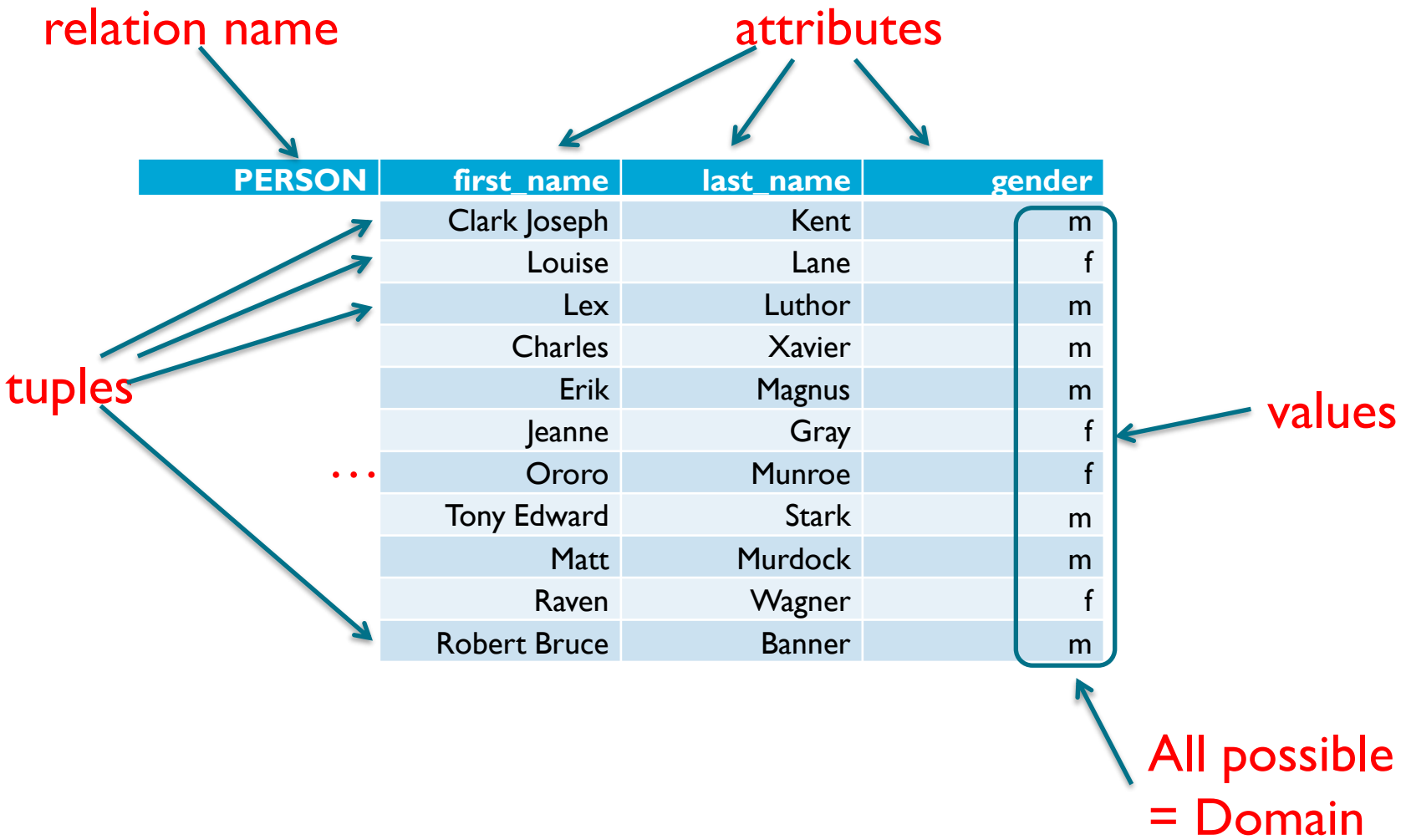
- **Data Organized into Tables**: Data is structured in tables, also known as relations, which consist of rows and columns.
- **Rows Represent Records**: Each row, or tuple, in a table represents a single record or entry.
- **Columns Represent Attributes**: Each column, or attribute, in a table represents a specific piece of data related to each record.
- **Unique Identifiers**: Each table has a primary key, a unique identifier that ensures each record can be uniquely distinguished from others.
- **Foreign Keys**: Foreign keys link tables together by referencing primary keys in other tables, establishing relationships between the data. (more in later classes)
- **Relational Integrity Rules**: Constraints like primary keys, foreign keys, and unique constraints ensure the accuracy and consistency of the data.
- **SQL**: Structured Query Language (SQL) is used to interact with and manage the data within the relational model.
- **Normalization**: The process of organizing data to reduce redundancy and improve data integrity through various normal forms. (more in later classes)

Relational Model

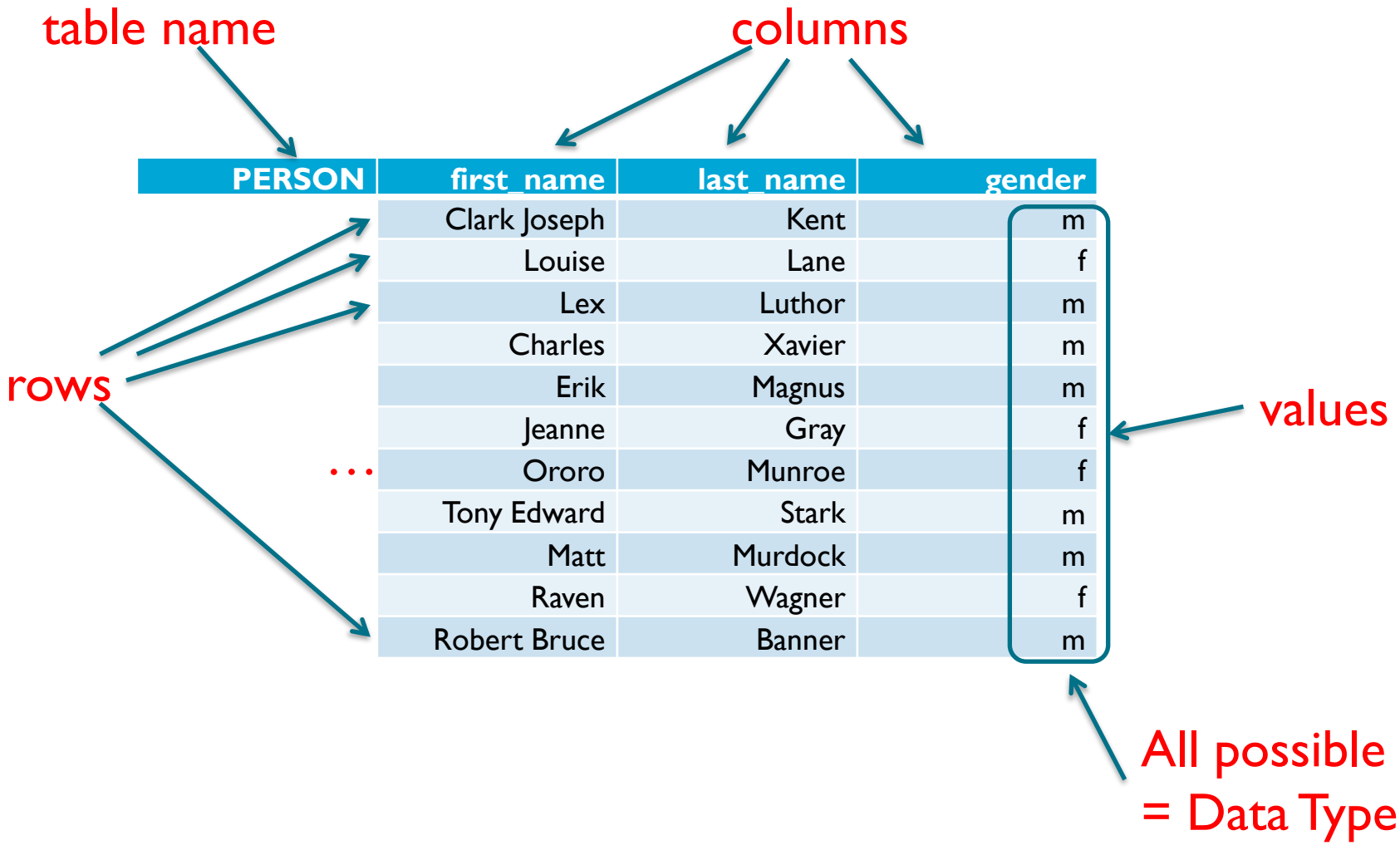
- A **relational database schema** consists of
  - a set of relation schemas
  - a set of integrity constraints
- A **relational database instance** (or state) is
  - a set of **relations** adhering to the respective schemas and respecting all integrity constraints



Relational Model - Formal



Relational Model – See it as a table



What is a “Relation”?

Relation (mathematics)

4 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about basic notions of (homogeneous binary) relations in mathematics. For a more in-depth treatment, see Binary relation. For relations on any number of elements, see Finitary relation.

In mathematics, a **relation** denotes some kind of *relationship* between two objects in a set, which may or may not hold.[1] As an example, "*is less than*" is a relation on the set of natural numbers; it holds, for instance, between the values 1 and 3 (denoted as $1 < 3$), and likewise between 3 and 4 (denoted as $3 < 4$), but not between the values 3 and 1 nor between 4 and 4, that is, $3 < 1$ and $4 < 4$ both evaluate to false. As another example, "*is sister of*" is a relation on the set of all people, it holds e.g. between Marie Curie and Bronisława Dłuska, and likewise vice versa. Set members may not be in relation "to a certain degree" – either they are in relation or they are not.

Formally, a relation R over a set X can be seen as a set of ordered pairs (x,y) of members of X.[2] The relation R holds between x and y if (x,y) is a member of R. For example, the relation "*is less than*" on the natural numbers is an infinite set R_{less} of pairs of natural numbers that contains both $(1,3)$ and $(3,4)$, but neither $(3,1)$ nor $(4,4)$. The relation "*is a nontrivial divisor of*" on the set of one-digit natural numbers is sufficiently small to be shown here: $R_{\text{dv}} = \{ (2,4), (2,6), (2,8), (3,6), (3,9), (4,8) \}$; for example 2 is a nontrivial divisor of 8, but not vice versa, hence $(2,8) \in R_{\text{dv}}$, but $(8,2) \notin R_{\text{dv}}$.

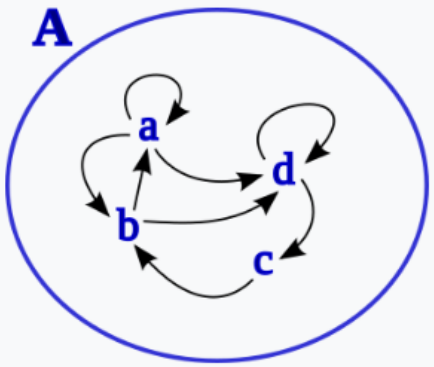
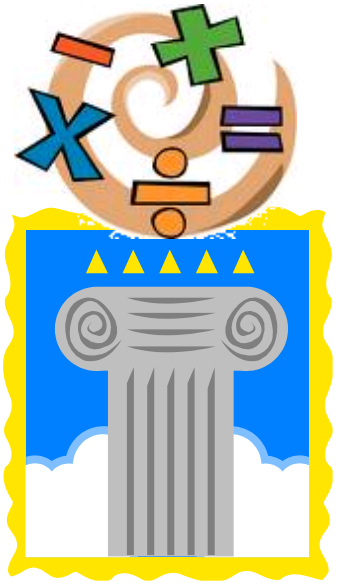


Illustration of an example relation on a set $A = \{ a, b, c, d \}$. An arrow from x to y indicates that the relation holds between x and y. The relation is represented by the set $\{ (a,a), (a,b), (a,d), (b,a), (b,d), (c,b), (d,c), (d,d) \}$ of ordered pairs.

Basic Set Theory

- We need set theory for this….
- **Set theory** is the foundation of mathematics
  - you probably all know these things from your math courses, but repeating never hurts
  - the **relational model** is based on set theory; understanding basic math will help a lot

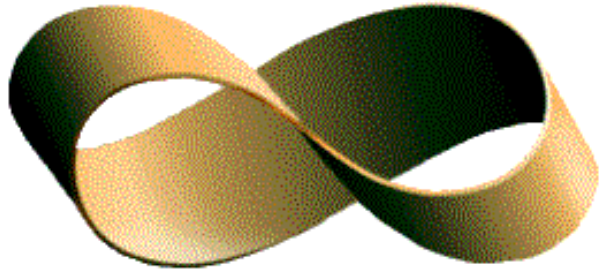


Sets

- A **set** is a mathematical **primitive,** and thus has no formal definition
- A set is a **collection** of objects (called *members* or *elements* of the set)
  - objects (or entities) have to be understood in a very broad sense, can be anything from physical objects, people, abstract concepts, other sets, …
- Objects **belong** (or do **not belong**) to a set (alternatively, *are* or *are not* in the set)
- A set **consists** of all its elements

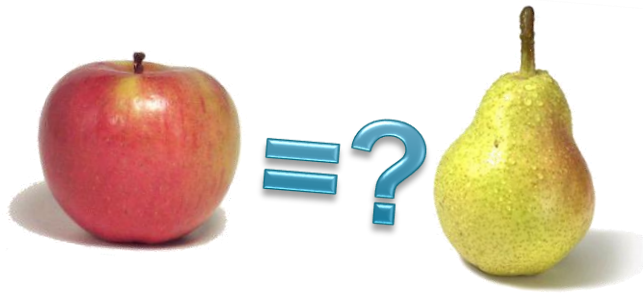
Sets

- Sets can be specified **extensionally**
  - Count / list all its elements
  - e.g. A = {delft, 42, Christoph, Count von Count}
- Sets can be specified **intensionally**
  - provide a criterion deciding whether an object belongs to the set or not (**membership criterion**)
  - examples:
    - $A = \{x \mid x > 4 \text{ and } x \in \mathbb{Z}\}$
    - $B = \{x \in \mathbb{N} \mid x < 7\}$
    - C = {all facts about databases you should store}
    - D = {all people who do not shave themselves}
- Sets can be either **finite** or **infinite**
  - set of all super villains is finite
  - set of all numbers is infinite

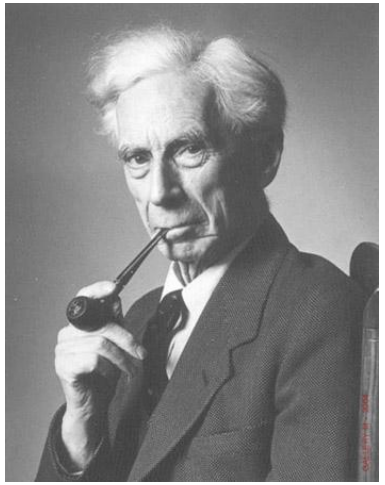


Sets

- Sets are different, iff they have different members
  - $\{a, b, c\} = \{b, c, a\}$
  - duplicates are not supported in standard set theory
    - $\{a, a, b, c\} = \{a, b, c\}$
- Sets can be empty (written as {} or ∅)
- Notations for set membership
  - $a \in \{a, b, c\}$
  - $e \notin \{a, b, c\}$



Sets



- Defining a set by its **intension**
  - intension must be **well-defined** and **unambiguous**
  - there is always is a clear **membership criterion** to determine whether an object belongs to the set (or not)
  - Example for an invalid definition (Russell's paradox 1901):

> In a small town, there is just one male barber.
> He shaves all and only those men in town
> who do not shave themselves.

- **does the barber shave himself?**
  - Reading suggestion: Logicomix, Doxiadis &Papadimitriou, 2008

Sets

- Still, the set's **extension** might be unknown (however, there is one)
- Example
  - *All students in this room who are older than 22.*
  - well-defined, but not known to me …
  - but (at least in principle) we can find out!
- Why should we care? Because:
  - **Intensional set** ≈ database query
  - **Extensional set** ≈ result of a query, table

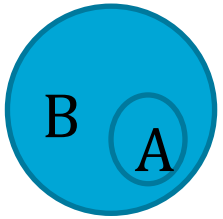
Sets

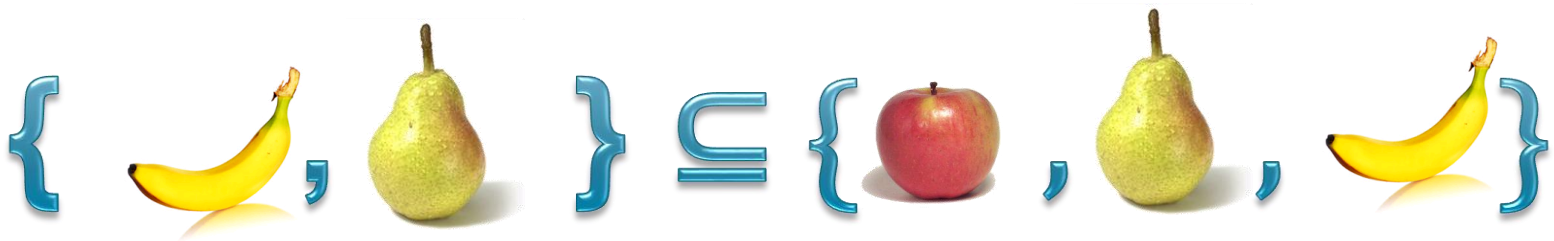
- For every set and object, there is an accompanying definition of **equality** (or equivalence)
  - is $x = y$?
- However, you could have two different descriptions of the same element
  - example: the set of all 26 *standard* letters
    - ‘ö’ is not contained in this set
    - ‘m’ = ‘M’ and both reflect a single element of the set
      - ‘m’ and ‘M’ are different descriptions of the **same** object (at least we can see it as such)
  - example: the set of all 59 letters and umlauts in German
    - ‘ö’ is element of the set
    - ‘m’ ≠ ‘M’ and are both elements of the set (two different objects)

Sets

- Sets have a cardinality (i.e., number of elements)
  - denoted by $|A|$
  - $|\{a, b, c\}| = 3$
- Set A is a **subset** of set B, denoted by $A \subseteq B$, iff every member of A is also a member of B
- B is a **superset** of A, denoted by $B \supseteq A$, iff $A \subseteq B$

$A \subseteq B$





Tuples

- A **tuple** (or vector) is a sequence of objects
  - length 1: Singleton
  - length 2: Pair
  - length 3: Triple
  - length *n*: *n*-tuple
- In contrast to sets…
  - tuples can contain an object more than once
  - the objects appear in a certain **order**
  - the length of the tuple is **finite**
- Written as $\langle a, b, c \rangle$ or (a, b, c)

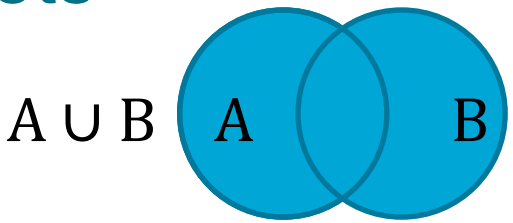
Tuples

- Hence
  - $\langle a, b, c\rangle \neq \langle c, b, a\rangle$, whereas $\{a, b, c\} = \{c, b, a\}$
  - $\langle a_1, a_2\rangle = \langle b_1, b_2\rangle$ iff $a_1 = b_1$ and $a_2 = b_2$

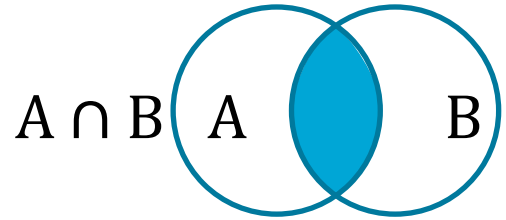
- ***n*-tuples** ($n > 1$) can also be defined as a cascade of ordered pairs:
  - $\langle a, b, c, d\rangle = \langle a, \langle b, \langle c, d\rangle\rangle\rangle$

Set Operations

- Four binary **set operations**
  - union, intersection, difference and cartesian product
- Union: ∪
  - creates a new set containing all elements that are contained in (at least) one of two sets
  - $\{a, b\} \cup \{b, c\} = \{a, b, c\}$

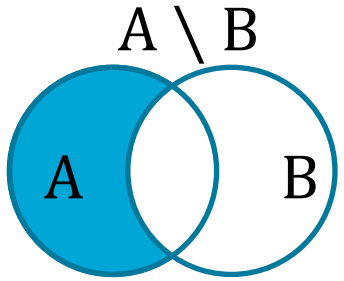


- Intersection: ∩
  - creates a new set containing all elements that are contained in both sets
  - $\{a, b\} \cap \{b, c\} = \{b\}$



Set Operations

- Difference: \
  - creates a set containing all elements of the first set without those also being in the second set
  - $\{a, b\} \setminus \{b, c\} = \{a\}$

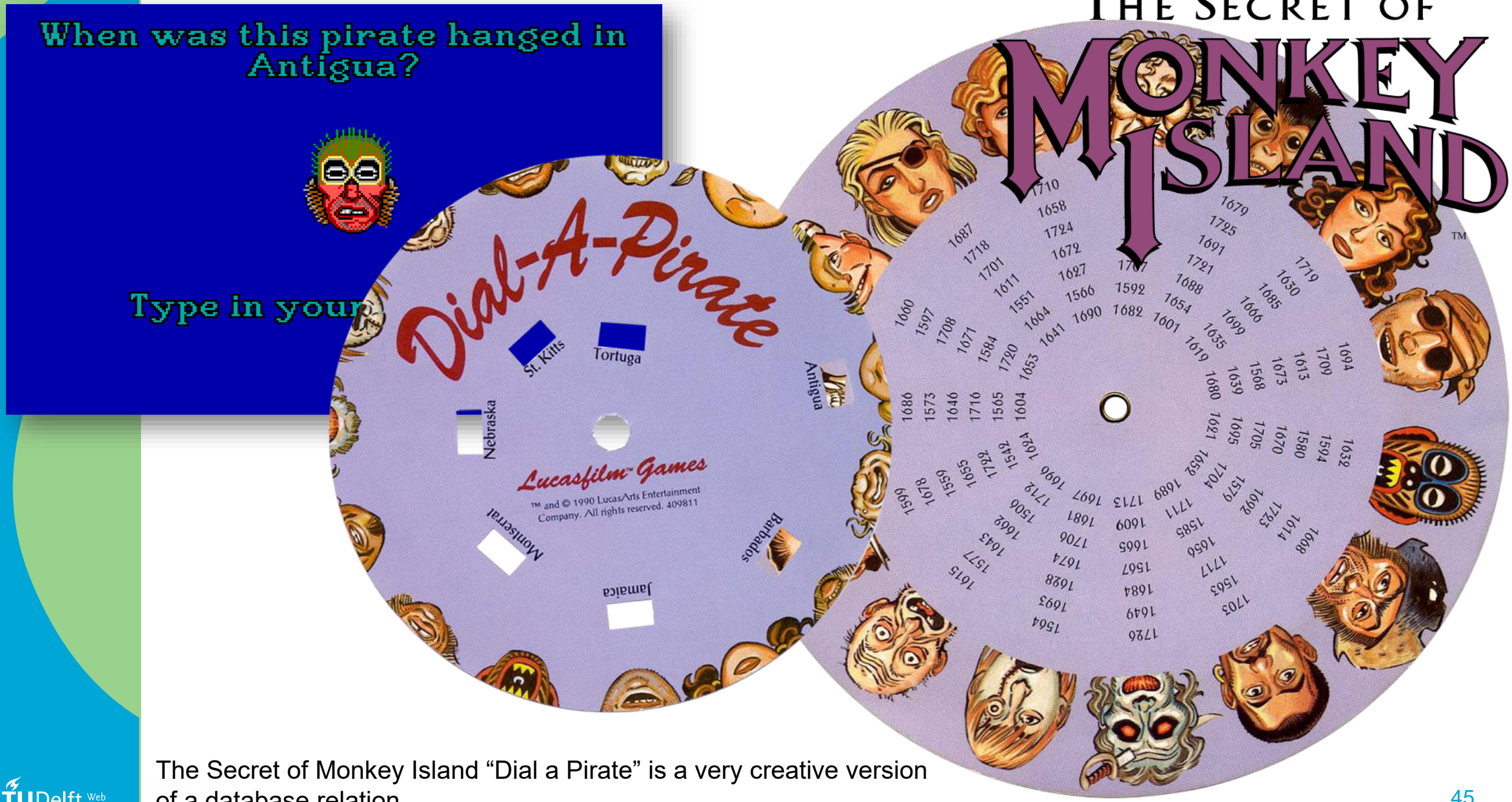


Set Operations

- Cartesian Product: $\times$
  - the cartesian product is an operation between two sets, creating a new set of pairs such that:

    $$A \times B = \{\langle a, b\rangle \mid a \in A \text{ and } b \in B\}$$
  - named after René Descartes
- Example
  - {a, b} × {b, c} = {⟨a, b⟩, ⟨a, c⟩, ⟨b, b⟩, ⟨b, c⟩}
  - Color = { red, white }
  - $\mathrm{Building}$ = { house, palace }
  - $\mathrm{Cleverness} \times \mathrm{Character}$ = {⟨red, house⟩, ⟨white, house⟩, ⟨red, palace⟩, ⟨white, palace⟩}

- The cartesian product can easily be extended to higher dimensionalities: $A \times B \times C$ is a set of triples



The Secret of Monkey Island “Dial a Pirate” is a very creative version of a database relation…

Relations

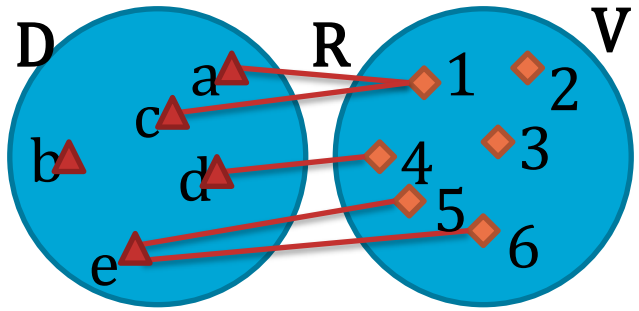
- A **relation** R over some sets $D_1, \ldots, D_n$ is a **subset** of their **cartesian** product
  - $R \subseteq D_1 \times \ldots \times D_n$
  - the elements of a relation are **tuples**
  - the D_i are called **domains**
  - each D_i corresponds to an **attribute** of a tuple
    - *n*=1: Unary relation or **property**
    - *n*=2: Binary relation
    - *n*=3: Ternary relation
    - …

Relations

- Some important properties
  - relations are sets (of tuples) in the mathematical sense, thus **no duplicate tuples** are allowed
  - the **set of tuples** is **unordered**
  - the **list of domains** is **ordered**
    - And so are the values of each tuple
  - relations can be modified by…
    - **inserting** new tuples,
    - **deleting** existing tuples, and
    - **updating** (that is, modifying) existing tuples.

Relations

- A special case: Binary relations
  - $\mathbf{R \subseteq D_1 \times D_2}$
    - D_1 is called **domain,** D_2 is called **co-domain** (range, target)
  - relates objects of two different sets to each other
  - R is just a set of ordered pairs
  - $R = \{\langle a,1\rangle, \langle c,1\rangle, \langle d,4\rangle, \langle e,5\rangle, \langle e,6\rangle\}$
    - can also be written as a**R**1, c**R**1, d**R**4, …
  - imagine $\mathbf{Likes \subseteq Person \times Beverage}$
    - Asterios Likes Coffee, Christoph Likes Tea, …
  - For example, binary relations can naively be used to implement n:m relationship types in a logical data model
  - Functions are a special case of binary relations



Relations

- Example
  - Accessory = {spikes, butterfly helmet}
  - Material = {silk, armor plates}
  - Color = {pink, black}

Color × Material × Accessory =
{⟨pink, silk, butterfly helmet⟩,
⟨pink, silk, spikes⟩,
⟨pink, armor plates, butterfly helmet⟩,
⟨pink, armor plates, spikes⟩,
⟨black, silk, butterfly helmet⟩,
⟨black, silk, spikes⟩,
⟨black, armor plates, butterfly helmet⟩,
⟨black, armor plates, spikes⟩}

Relations

- Relation **FamousHeroCostumes** ⊆ **Color × Material × Accessory**

FamousHeroCostumes =
{⟨pink, silk, butterfly helmet⟩,
⟨black, armor plates, spikes⟩}

Relations

- Relation **FamousHeroCostumes** $\subseteq$ **Color × Material × Accessory**

FamousHeroCostumes =
{⟨pink, silk, butterfly helmet⟩,
⟨black, armor plates, spikes⟩}

Relational Model

- Well, that's all nice to know… but: we are here to learn about **databases!**
  - where is the connection?
- **Here it is…**
  - a **database schema** is a description of a database in terms of relations and attribute domains
    - Description of all possible instances
      - …but also intensional set of tuples
  - a **database instance** is a set of tuples having certain attribute values
    - An extensional set of tuples
      - Thus, a **Relation**

Relational Model

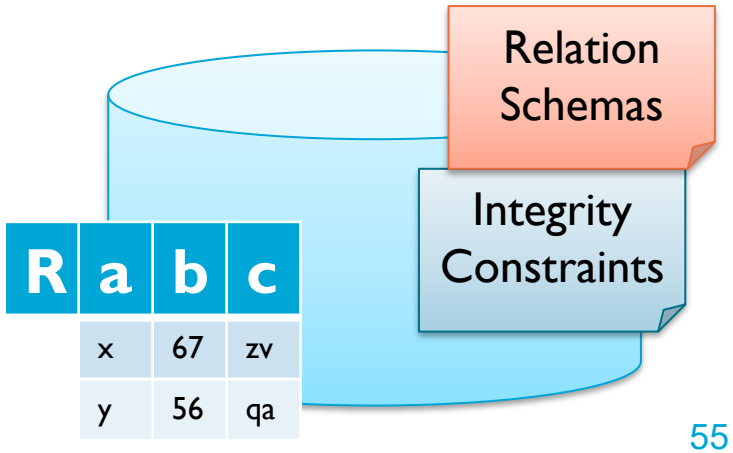
- OK, then…
  - designing a database schema (e.g., by ER modeling) determines entities and relationships, as well as their corresponding **sets of attributes** and associated **domains**
  - the **Cartesian product** of the respective domains is the set of all possible tuples (of each entity type or relationship type)
  - a **relation** extensionalizes the **actually existing** subset out of all possible instances

Relational Model

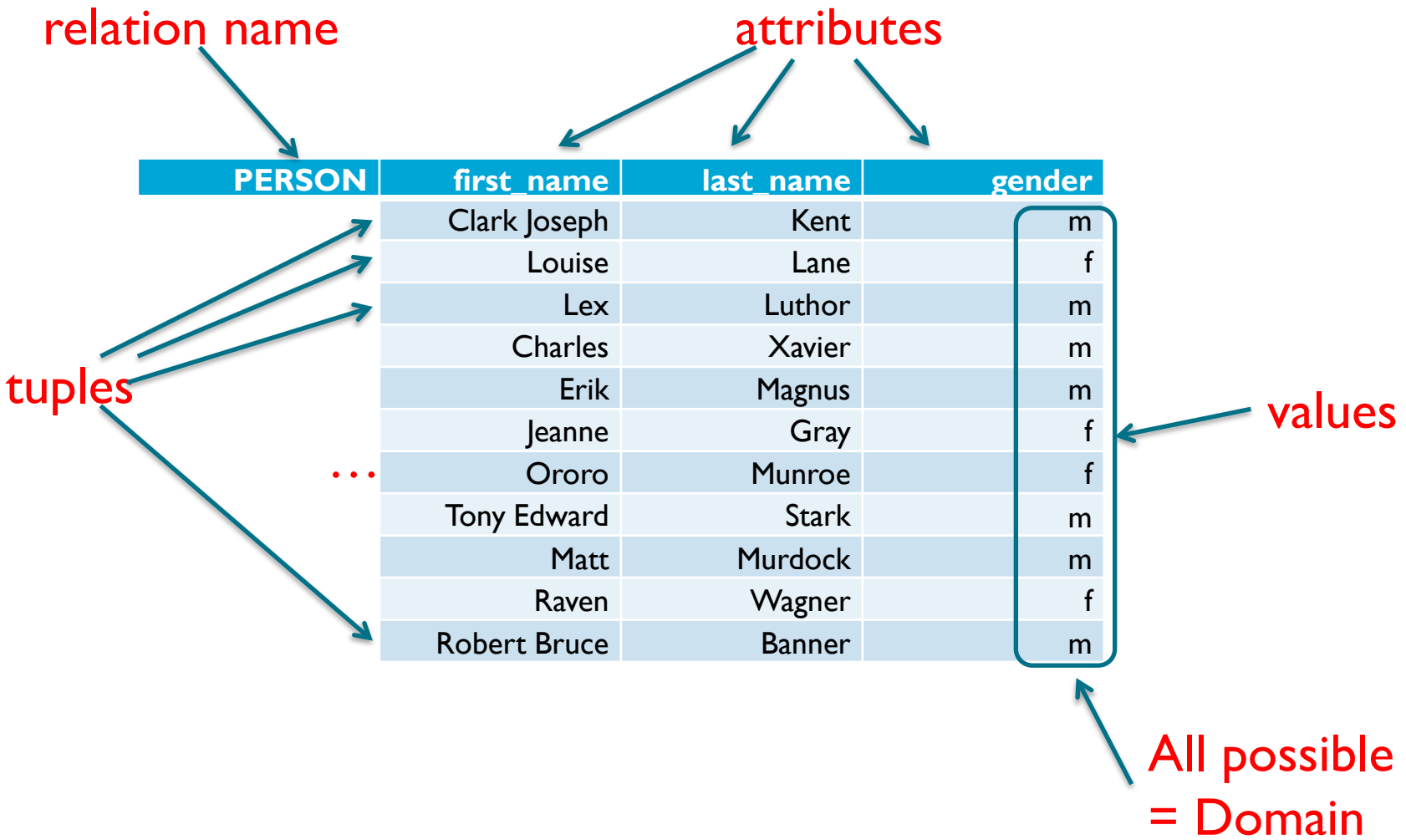
- Database schemas are described by **relation schemas** $R(A_1, ..., A_n)$
- Domains are assigned by the dom function
  - $\mathrm{dom}(A_1) = D_1, \mathrm{dom}(A_2) = D_2, ...$
  - Also written as: $R(A_1{:}D_1, ..., A_n{:}D_n)$
- The actual database instance is given by a set of matching **relations**
- Example
  - relation schema:
    $\mathrm{Cat}(\mathrm{name}: \mathrm{string}, \mathrm{age}: \mathrm{number})$
  - A possible relation:
    { (Blackie, 2), (Kitty, 1), (Fluffy, 4) }

Relational Model

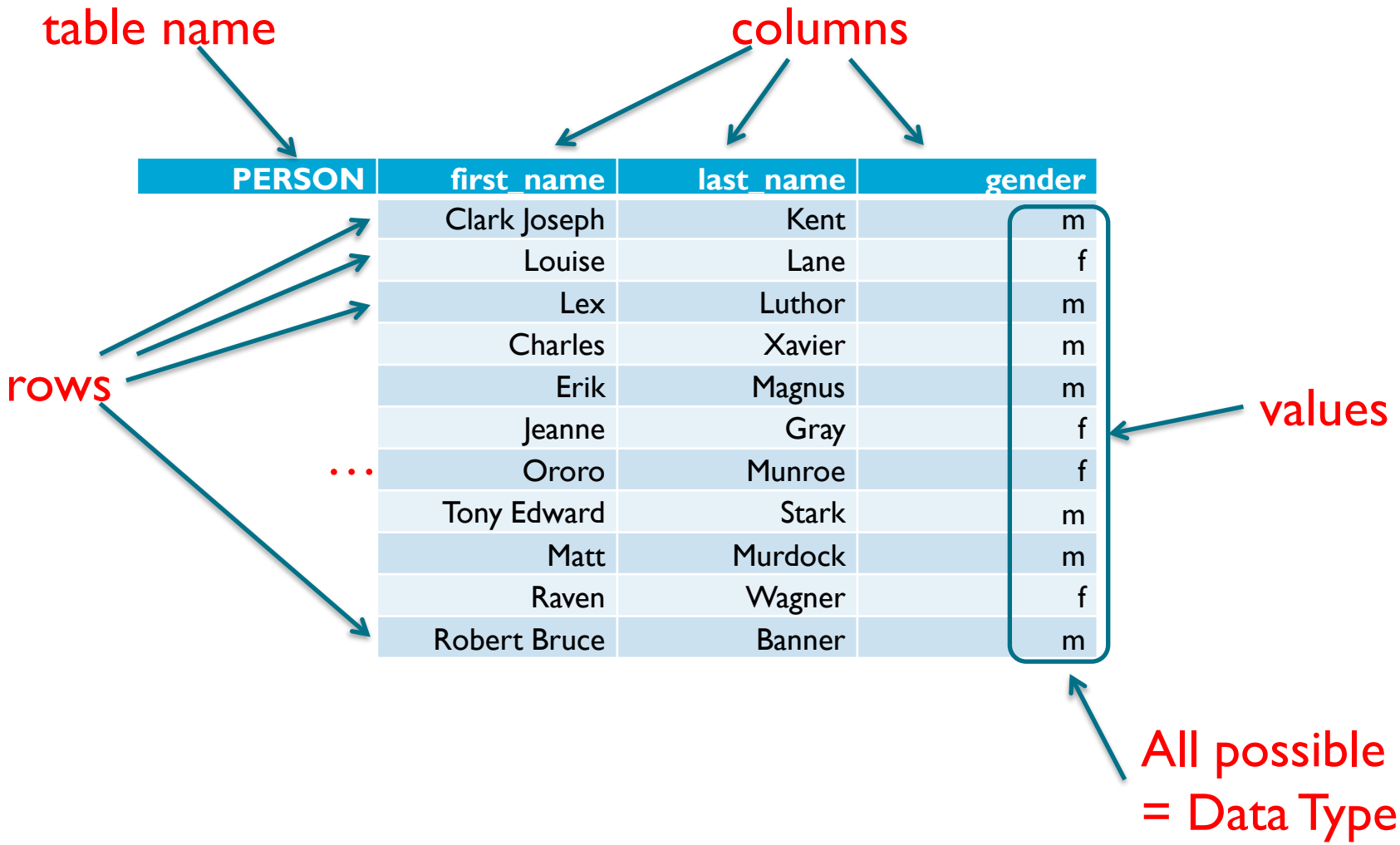
- A **relational database schema** consists of
  - a set of relation schemas
  - a set of integrity constraints
- A **relational database instance** (or state) is
  - a set of relations adhering to the respective schemas and respecting all integrity constraints



Relational Model - Formal



Relational Model – See it as a table



Relational Model

- Every relational DBMS needs a language to define its relation schemas (and integrity constraints)
  - **Data Definition Language** (DDL)
  - typically, it is difficult to formalize all possible integrity constraints, since they tend to be complex and vague
- A relational DBMS also needs a language to handle and manipulate tuples
  - **Data Manipulation Language** (DML)
- Today's RDBMS use **SQL** as both DDL and DML
  - Compare to XML: Here, DDL and DML are separated
    - XMLSchema and DTD vs XQuery and XPATH

Integrity Constraints

TU Delft WIS Web Information Systems

Database Integrity

- Database constraints are rules applied to data in a database to ensure **consistency**.
  - Consistency describes that data is correct and valid concerning the applied constraints and rules
  - Consistency does not necessarily mean that data is correct, but at least it's well-formed…

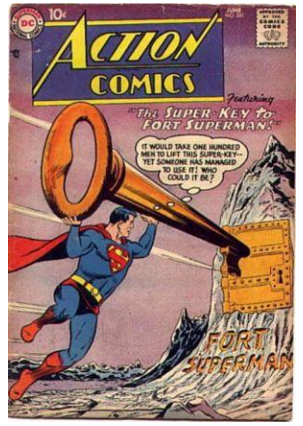
Common Constraints

- **Constraints** in the **Theoretical** Relational Model
  - **Primary Key Constraint**: Ensures that each tuple in a relation has a unique identifier that cannot be null.
  - **Foreign Key Constraint**: Ensures that values in a (set of) attributes match values in a related relations primary key, enforcing referential integrity between tables.
  - **Unique Constraint**: Ensures that all values in a (set of) attribute are unique across tuples of a relation

Common Constraints

- **Constraints** in the DBMS **implementation** of the Relational Model
  - **Not Null Constraint**: Ensures that a column cannot have null values, meaning every row must have a value for this column.
  - **Check Constraint**: Ensures that values in a column satisfy a specific condition, such as range constraints or specific formats.
  - **Default Constraint**: Provides a default value for a column when no value is specified during insertion.
  - **Replica Consistency Constraint**: Replicas (copies) of the same records have the same values (in distributed databases which have multiple copies of the same data for practical reasons)

Primary Key Constraint



- **Primary Key Constraint**
  - A relation is defined as a **set** of tuples
    - all tuples mustbe **distinct**, i.e., no two tuples can have the same combinations of values for all attributes
    - so-called **uniqueness (unique key) constraint** or primary key constraint
  - Therefore, we can define the **key** of a relation as a designated **subset of attributes** for which no two tuples have the same values (are **unique**)
    - It‘s a little bit more complex than that…see later lecture
  - Each relation will need a designated key
    - We will write this as for example **Hero**(alias, name, age, ...)

Primary Key Constraint

- How to write this?
  - If a primary key has a single attribute:
    - R1(<u>a</u>, b, c)
  - If a primary key has a multiple attribute:
    - R1(<u>a</u>, <u>b</u>, c)
    - “Composite Primary Key”

Foreign Key Constraint

- **Foreign Key Constraint**
  - Sometimes, we want to link tuples in different relations
    - This will be integral for realizing ER relationships in a database
  - A **foreign key constraint** can be defined between the key attributes of one relation and some attributes of another one
    - e.g.,
      movie(m_id, title, year)
      cast(m_id→movie, a_id → actor, role)

References the key 'm_id' of movie

  - Tuples of the referring relation can only have values for the referencing attribute which are the key of an existing tuple in the referenced relation
    - This is called **referential integrity**

Foreign Key Constraint

movie

m_id	title	year
1	Terminator 2 – Judgement Day	1991
2	Twins	1988

Primary key: m_id

actor

a_id	f_name	l_name
1	Arnold	Schwarzenegger
2	Linda	Hamilton
3	Danny	DeVito

Primary key: a_id

Foreign Key Constraint
(just the graphical representation of it)

cast

m_id	a_id	role
1	1	T-800
1	2	Sarah Connor
2	1	Julius Benedict
2	3	Vincent Benedict

Primary key: (m_id, a_id)
- Note for later: This allows each actor to only have one role per movie. Primary key should be (m_id, a_id, role) to allow multiple roles per movie.
- Foreign key constraints: (m_id-> movie.m_id, a_id-> actor.a_id)

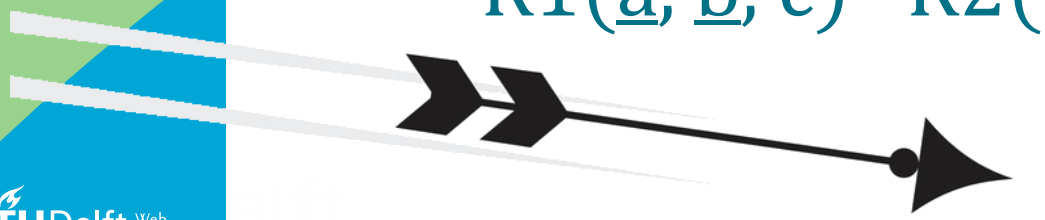
Foreign Key Constraint

- How to write this?:
  - If a key with a single attribute is referenced, we write this as:

$$R1(\underline{a}, b, c) \quad R2(\underline{d}, e \rightarrow R1)$$

  - If a composite key is referenced, we write this as, e.g.,

$$R1(\underline{a}, \underline{b}, c) \quad R2(d, \underline{e}, f, \mathbf{(d, f) \rightarrow R1})$$



Basic Constraints

- **NOT NULL Constraint**
  - Remember, a relation is defined as $\boldsymbol{R \subseteq D_1 \times \ldots \times D_n}$ with tuples $t \in R$
  - However, in a practical application its common that not always all attribute values are known
    - Therefore, it is usually assumed that there is a special NULL value in each domain, i.e. $NULL \in D_i$
  - Sometimes, this is not desired for certain attributes
    - Introduces the NOT NULL constraint
  - **Primary Key must never be NULL**
  - e.g. Address(<u>street</u>: string NOT NULL,
    <u>number</u>: numeric NOT NULL,
    <u>zip-code</u>: numeric NOT NULL,
    city: string NOT NULL
    postbox : string)

How do we write this in Text?

- Let's agree on the following text notation (e.g., for homework paper):
  - $\mathbf{R(\underline{a}, b, c)}$
    - Relation R with primary key a
  - $\mathbf{R(\underline{a}, \underline{b}, c)}$
    - Relation R with composite primary key (a, b)
    - You can also use $\mathbf{R(pk(a, b), c)}$ if you like. Also applies to other examples on this slide.
  - $\mathbf{R1(\underline{a}, b, c);\ \ R2(\underline{d}, e \rightarrow R1);}$
    - Foreign key from R2.e to R1.a; R1.a and R2.d are primary keys
    - You could also write this as $\mathbf{R1(\underline{a}, b, c);\ \ R2(\underline{d}, e, e \rightarrow R1);}$
  - $\mathbf{R1(\underline{a}, b, c);\ \ R2(\underline{d}, e \rightarrow R1(a));}$
    - Same as above, but with explicit mention of foreign key reference attribute (as used in SQL)
  - $\mathbf{R1(\underline{a}, \underline{b}, c)\ \ R2(d, \underline{e}, f, (d,f) \rightarrow R1)}$
    - R1 has a composite primary key (a,b), R2 uses the primary key e
    - Foreign key constraint from (d, f) to R1(a,b)
  - $\mathbf{R1(\underline{a}, \underline{b}, c)\ \ R2(d, \underline{e}, f, (d,f) \rightarrow R1(a,b))}$
    - Same as above, but with explicit mention of foreign key reference attributes (as used in SQL)

How do we write this in ASCII?

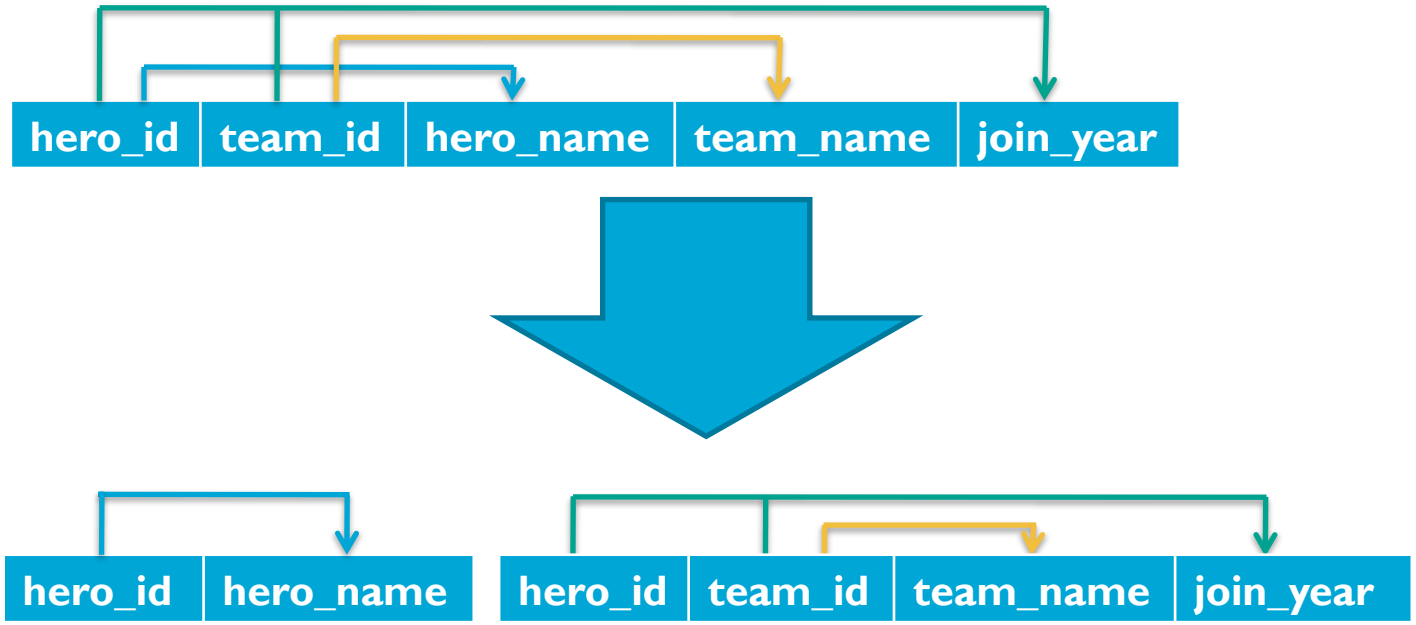
- Let's agree on the following ASCII notation (e.g., for Weblab):
  - `R(_a, b, c)`
    - Relation R with primary key a
  - `R(_a, _b, c)`
    - Relation R with composite primary key (a, b)
    - You can also use `R(pk(a, b), c)` if you like. Also applies to other examples on this slide.
  - `R1(_a, b, c);  R2(_d, e -> R1);`
    - Foreign key from R2.e to R1.a; R1.a and R2.d are primary keys
    - You could also write this as `R1(_a, b, c);  R2(_d, e, e -> R1);`
  - `R1(_a, b, c);  R2(_d, e -> R1(a));`
    - Same as above, but with explicit mention of foreign key reference attribute (as used in SQL)
  - `R1(_a, _b, c)  R2(d, _e, f, (d,f)-> R1)`
    - R1 has a composite primary key (a,b), R2 uses the primary key e
    - Foreign key constraint from (d, f) to R1(a,b)
  - `R1(_a, _b, c)  R2(d, _e, f, (d,f)-> R1(a,b))`
    - Same as above, but with explicit mention of foreign key reference attributes (as used in SQL)

Summary

- The Relation Model stores data in Relations
  - In this context, Relations are a subset of the Cartesian Product of a list of Domains…
  - Relations are defined in schemas
    - The **extension** of this schema is the stored data
      - "List all tuples which are there"
    - Queries can be modeled as **intentional** relations
      - "List all tuples which fulfill a given condition"
  - Constraints can / must be defined
    - Especially: Primary Key Constraints, Foreign Key Constraints
- In practical terms:
  - Relations ~~ tables; Tuples ~~ rows; Attributes ~~ columns

Next Week

- Normalization
- Functional Dependencies
- Normal Forms
  - 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, 6NF



Information and Data Modelling

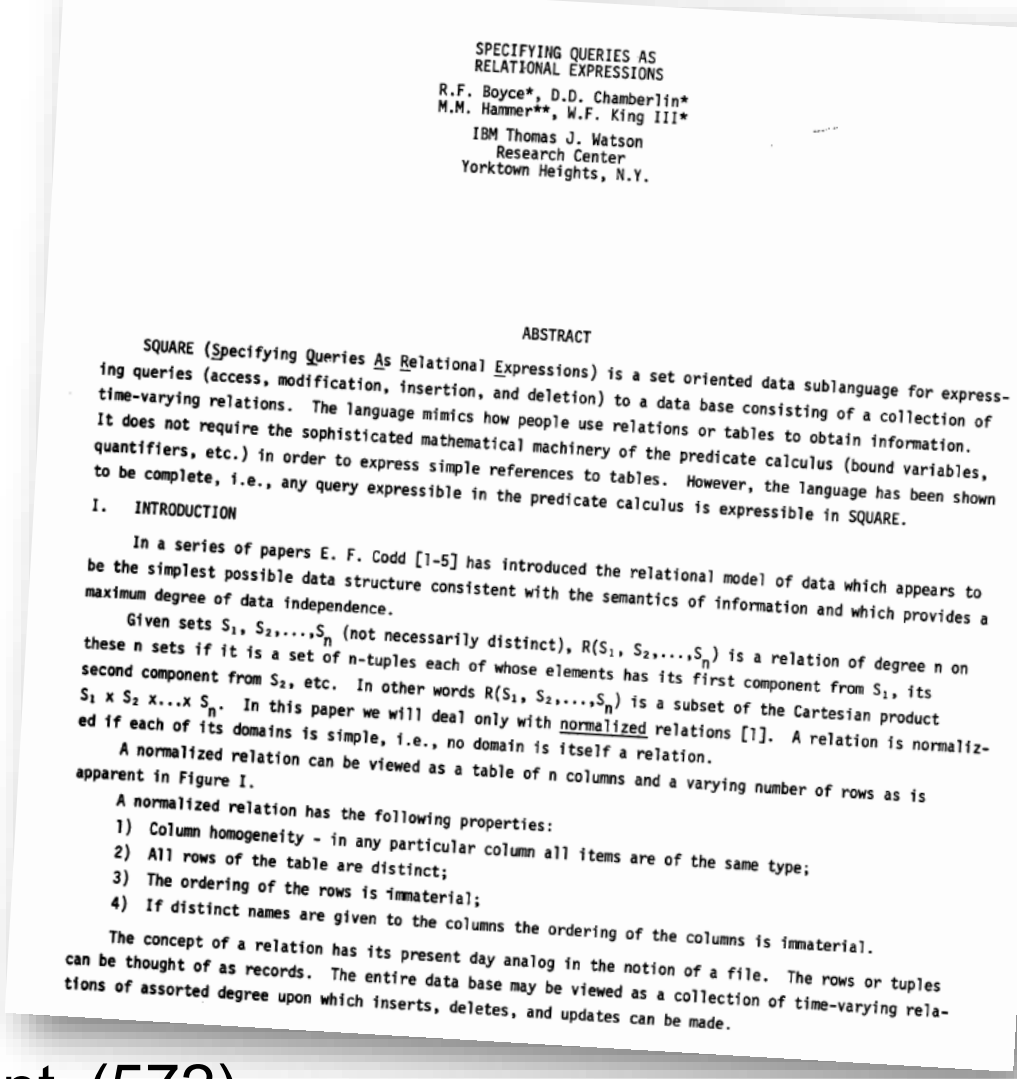
See you next time!

SQL and Square

From theory to practice

TU Delft WIS Web Information Systems

SQUARE



SPECIFYING QUERIES AS
RELATIONAL EXPRESSIONS

R.F. Boyce*, D.D. Chamberlin*
M.M. Hammer**, W.F. King III*

IBM Thomas J. Watson
Research Center
Yorktown Heights, N.Y.

ABSTRACT

SQUARE (Specifying Queries As Relational Expressions) is a set oriented data sublanguage for expressing queries (access, modification, insertion, and deletion) to a data base consisting of a collection of time-varying relations. The language mimics how people use relations or tables to obtain information. It does not require the sophisticated mathematical machinery of the predicate calculus (bound variables, quantifiers, etc.) in order to express simple references to tables. However, the language has been shown to be complete, i.e., any query expressible in the predicate calculus is expressible in SQUARE.

I. INTRODUCTION

In a series of papers E. F. Codd [1-5] has introduced the relational model of data which appears to be the simplest possible data structure consistent with the semantics of information and which provides a maximum degree of data independence.

Given sets $S_1, S_2, \ldots, S_n$ (not necessarily distinct), $R(S_1, S_2, \ldots, S_n)$ is a relation of degree n on these n sets if it is a set of n-tuples each of whose elements has its first component from S_1, its second component from S_2, etc. In other words $R(S_1, S_2, \ldots, S_n)$ is a subset of the Cartesian product $S_1 \times S_2 \times \ldots \times S_n$. In this paper we will deal only with normalized relations [1]. A relation is normalized if each of its domains is simple, i.e., no domain is itself a relation.

A normalized relation can be viewed as a table of n columns and a varying number of rows as is apparent in Figure I.

A normalized relation has the following properties:

1) Column homogeneity - in any particular column all items are of the same type;
2) All rows of the table are distinct;
3) The ordering of the rows is immaterial;
4) If distinct names are given to the columns the ordering of the columns is immaterial.

The concept of a relation has its present day analog in the notion of a file. The rows or tuples can be thought of as records. The entire data base may be viewed as a collection of time-varying relations of assorted degree upon which inserts, deletes, and updates can be made.

SQUARE: $_{name}Student_{id}(572)$ ← Conditions

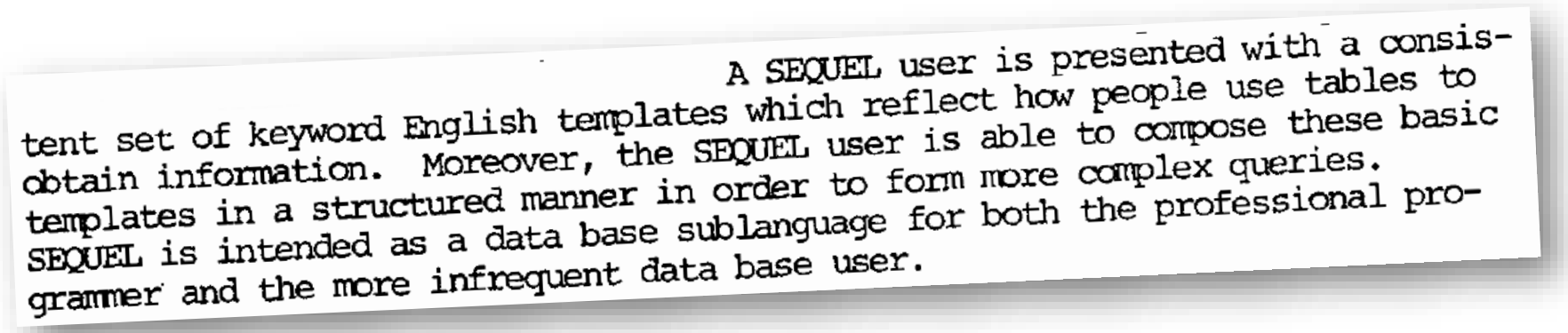
What part of the result tuples should be returned?

The range relation of the result tuples

Attributes with conditi

SEQUEL

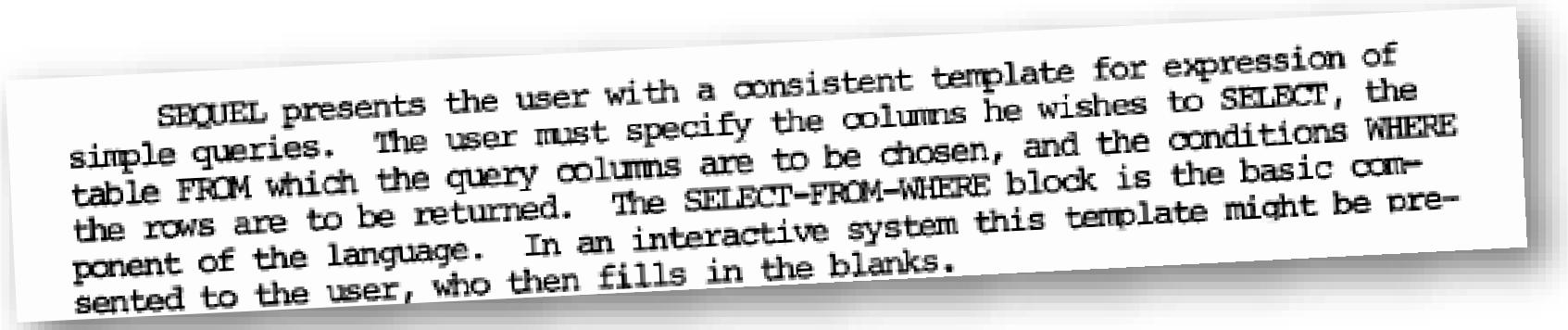
- In 1974, Chamberlin & Boyce proposed **SEQUEL**
  - ***S**tructured **E**nglish **Que**ry **L**anguage*
  - based on SQUARE
- Guiding principle
  - use natural **English keywords** to structure queries
  - supports ***fluent*** vocalization and notation



A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

SEQUEL

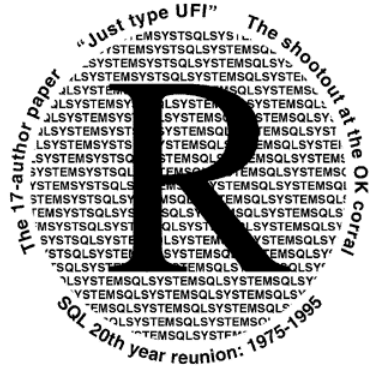
- Fundamental keywords
  - **SELECT:** what attributes should be retrieved?
  - **FROM:** what relations are involved?
  - **WHERE:** what conditions should hold?



SEQUEL presents the user with a consistent template for expression of simple queries. The user must specify the columns he wishes to SELECT, the table FROM which the query columns are to be chosen, and the conditions WHERE the rows are to be returned. The SELECT-FROM-WHERE block is the basic component of the language. In an interactive system this template might be presented to the user, who then fills in the blanks.

From Theory to Practice

- **System R** was the first working prototype of a relational database system (starting 1973)
  - most **design decisions** taken during the development of System R substantially **influenced** the design of **subsequent systems**
- **Questions**
  - how to store and represent data?
  - how to query for data?
  - how to manipulate data?
  - how do you do all this with good performance?



From Theory to Practice

- The challenge of the **System R** project was to create a **working prototype system**
  - theory is good
  - but developers were willing to sacrifice theoretical beauty and clarity for the sake of usability and performance
- **Vocabulary change**
  - mathematical terms were too unfamiliar for most people
  - **table** = relation
  - **row** = tuple
  - **column** = attribute
  - **data type** = domain

From Theory to Practice

- Naming Conventions

Real RDBMS	Relational Model
table	relation
row	tuple
column	attribute
data type	domain

- Nnotably, there are no ER-Relationships here
  - These get implemented by Integrity Constraints later…

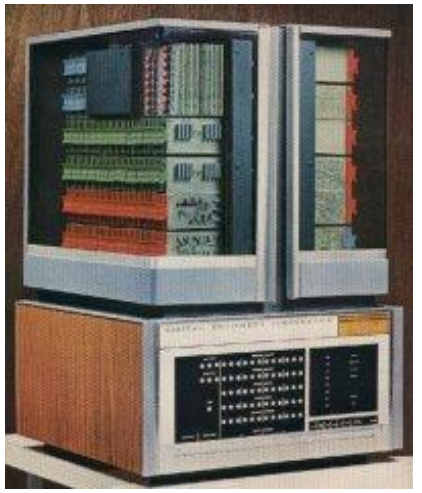
From Theory to Practice

- **Design decisions:**
  During the development of System R, two major and very controversial decisions had been made
  - allow **duplicate tuples**
  - allow **NULL values**
- Those decisions are still subject to discussions…

From Theory to Practice

- **Duplicates**
  - in a relation, there **cannot** be any **duplicate tuples**
  - also, query results cannot contain duplicates
    - the *relational algebra* and *relational calculi* all have implicit **duplicate elimination**

From Theory to Practice



- **Practical considerations**
  - you want to query for **name** and **birth year** of all **students** of TU Delft
  - the result returns roughly **25,000 tuples**
  - probably there are **some** duplicates
  - it's **1973,** and your computer has **16 kilobytes** of main memory and a very slow external storage device…
  - to eliminate duplicates, you need to **store** the result, **sort** it, and **scan** for adjacent duplicate lines
    - System R engineers concluded that this effort is not worth the effect
    - duplicate elimination in result sets happens only on-request

From Theory to Practice

- **Decision:** Don't eliminate duplicates in results
- What about the tables?
  - again: ensuring that no duplicates end up in tables requires some work
  - engineers also concluded that there is no real need in enforcing the no-duplicate policy
    - if the user wants duplicates and is willing to deal with all the arising problems – then that's fine
- **Decision:** Allow duplicates in tables
- As a result, the theory underlying relational databases shifted from **set theory** to **multi-set theory**
  - straightforward, only notation is more complicated

From Theory to Practice

- Sometimes, an attribute value is **not known** or an attribute does **not apply** for an entity
  - e.g. what value should the attribute *university_degree* take for the entity *Jos Kaas*, if *Jos Kaas* does not have any degree?
  - e.g. you regularly observe the weather and store temperature, wind strength, and air pressure every hour – and then your barometer breaks... what now?

From Theory to Practice

- Possible solution:
  For each domain, **define a value** indicating that data is not available, not known, not applicable, …
  - for example, use *none* for Jan de Berg's degree, use −1 for missing pressure data, ...
  - Problem:
    - you need such a special value for each domain or use case
    - you need special failure handling for queries, e.g. *compute average of all pressure values that are not −1*

From Theory to Practice

- Again, system designers chose the simplest solution (regarding implementation): **NULL values**
  - **NULL** is a special value which is usable in any domain and represents that data is just not there
    - there are many interpretations of what NULL actually means
      - Missing, inapplicable, invalid, unimportant, irrelevant, ….
  - Systems have some default rules how to deal with NULL values
    - aggregation functions usually ignore rows with NULL values (which is good in most, but not all cases)
    - three-valued logic
    - however, creates some strange anomalies

From Theory to Practice

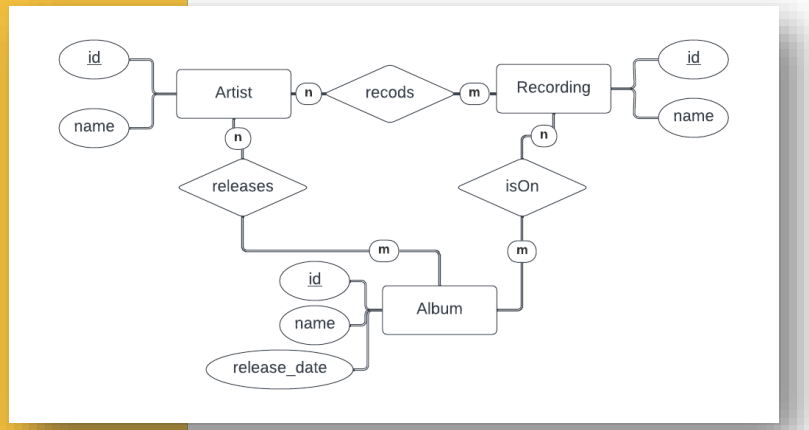
- Another tricky problem:
  How should users **query** the DB?
- Classical answer
  - Relational Algebra and Relational Calculi
  - **problem:** more and more **non-expert users**
- More *natural* query interfaces:
  - **QBE** (query by example)
  - **SEQUEL** (structured English query language)
  - **SQL:** the current standard; derived from SEQUEL

Revisit “Conceptual to Logical in WDT03”

TU Delft WIS Web Information Systems

Example: A logical schema of our music database using the relational model

Conceptual Schema



Focus on Entity Types and their relationships

Logical (Relational) Schema

Focus the resulting relations / tables and constraints

Artist

PK	id
	name

l_artist_recording

PK,FK	a_id
PK,FK	r_id

Recording

PK	id
	name
	rating
	length

l_artist_album

PK,FK	a_id
PK,FK	alb_id

Album

PK	id
	name
	release_date

l_album_recording

PK,FK	alb_id
PK,FK	r_id
	position

Artist

id	name
85	Madness
86	Qaballah Steppers
87	Steve Rossiter
88	Enya
89	Madonna
91	Bill Cosby
92	Yanni
93	Metallica
94	John Williams
96	The Bangles
98	Burl Ives
99	Bing Crosby
101	Frank Sinatra
104	[Disney]
110	Harry Connick, Jr.
111	Mariah Carey
116	Enigma
119	George Winston
121	The Chieftains
122	New Edition
123	Cliff Richard
125	Dan Fogelberg
127	James Taylor
129	Simon & Garfunkel

l_artist_recording

a.id	r.id
94	257208
94	334474
94	447670
94	453112
94	453118
94	453119
94	464602
94	464603
94	464604
94	496934
94	515742
94	515746
94	515748
94	532750
94	532751
94	532752
94	532754
94	532755
94	532757

Recording

record_id	record_name	rating...	length
257208	Superman: Main Theme	4	04:26
334474	Prologue	6	02:12
447670	Duel of the Fates	7	04:14
453112	Opening Titles	8	00:33
453118	Welcome to Jurassic Park	8	07:54
453119	My Friend, the Brachiosaurus	6	04:16
464602	Ah, Rats!!!	8	03:41
464603	Escape From Venice	6	04:24
464604	No Ticket	6	02:46
496934	E.T.'s New Home	4	01:38
515742	Yoda's Theme	6	03:29
515746	Yoda and the Force	6	04:02
515748	Lando's Palace	6	03:53
532750	Star Wars Main Title / Ambush on Corus…	8	03:46
532751	Across the Stars (Love Theme From Atta…	8	05:33
532752	Zam the Assassin / The Chase Through C…	10	11:07
532754	Departing Coruscant	6	01:44
532755	Anakin and Padmé	6	03:56
532757	The Meadow Picnic	8	04:14
532758	Bounty Hunter's Pursuit	8	03:23
532759	Return to Tatooine	8	06:56
532760	The Tusken Camp / The Homestead	6	05:54
532761	Love Pledge / The Arena	8	08:29
532762	Confrontation With Count Dooku / Finale	8	10:45

Artist

id	name
85	Madness
86	Qaballah Steppers
87	Steve Rossiter
88	Enya
89	Madonna
91	Bill Cosby
92	Yanni
93	Metallica
94	John Williams
96	The Bangles
98	Burl Ives
99	Bing Crosby
101	Frank Sinatra
104	[Disney]
110	Harry Connick, Jr.
111	Mariah Carey
116	Enigma
119	George Winston
121	The Chieftains
122	New Edition
123	Cliff Richard
125	Dan Fogelberg
127	James Taylor
129	Simon & Garfunkel

l_artist_recording

a.id	r.id
94	257208
94	334474
94	447670
94	453112
94	453118
94	453119
94	464602
94	464603
94	464604
94	496934
94	515742
94	515746
94	515748
94	532750
94	532751
94	532752
94	532754
94	532755
94	532757

Recording

record_id	record_name	rating...	length
257208	Superman: Main Theme	4	04:26
334474	Prologue	6	02:12
447670	Duel of the Fates	7	04:14
453112	Opening Titles	8	00:33
453118	Welcome to Jurassic Park	8	07:54
453119	My Friend, the Brachiosaurus	6	04:16
464602	Ah, Rats!!!	8	03:41
464603	Escape From Venice	6	04:24
464604	No Ticket	6	02:46
496934	E.T.'s New Home	4	01:38
515742	Yoda's Theme	6	03:29
515746	Yoda and the Force	6	04:02
515748	Lando's Palace	6	03:53
532750	Star Wars Main Title / Ambush on Corus…	8	03:46
532751	Across the Stars (Love Theme From Atta…	8	05:33
532752	Zam the Assassin / The Chase Through C…	10	11:07
532754	Departing Coruscant	6	01:44
532755	Anakin and Padmé	6	03:56
532757	The Meadow Picnic	8	04:14
532758	Bounty Hunter's Pursuit	8	03:23
532759	Return to Tatooine	8	06:56
532760	The Tusken Camp / The Homestead	6	05:54
532761	Love Pledge / The Arena	8	08:29
532762	Confrontation With Count Dooku / Finale	8	10:45

This table requires foreign key constraints:
e.g, there should be no a.id without a matching artist.id,
and no r.id without a matching recording.record_id!
This is called Referential Integrity

Artist

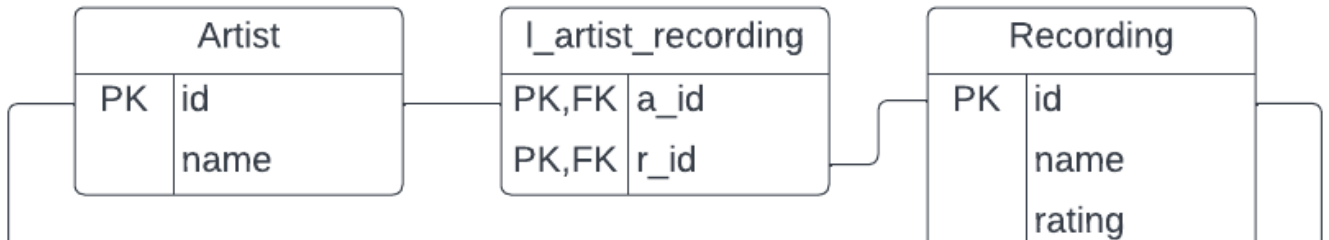
id	name
85	Madness
86	Qaballah Steppers
87	Steve Rossiter
88	Enya
89	Madonna
91	Bill Cosby
92	Yanni
93	Metallica
94	John Williams
96	The Bangles
98	Burl Ives
99	Bing Crosby
101	Frank Sinatra
104	[Disney]
110	Harry Connick, Jr.
111	Mariah Carey
116	Enigma
119	George Winston
121	The Chieftains
122	New Editi…
123	Cliff Ric…
125	Dan Fogel…
127	James Tay…
129	Simon & G…

l_artist_recording

a.id	r.id
94	257208
94	334474
94	447670
94	453112
94	453118
94	453119
94	464602
94	464603
94	464604
94	496934
94	515742
94	515746
94	515748
94	532750
94	532751
94	532752
94	532754

Recording

record_id	record_name	rating...	length
257208	Superman: Main Theme	4	04:26
334474	Prologue	6	02:12
447670	Duel of the Fates	7	04:14
453112	Opening Titles	8	00:33
453118	Welcome to Jurassic Park	8	07:54
453119	My Friend, the Brachiosaurus	6	04:16
464602	Ah, Rats!!!	8	03:41
464603	Escape From Venice	6	04:24
464604	No Ticket	6	02:46
496934	E.T.'s New Home	4	01:38
515742	Yoda's Theme	6	03:29
515746	Yoda and the Force	6	04:02
515748	Lando's Palace	6	03:53
532750	Star Wars Main Title / Ambush on Corus…	8	03:46
532751	Across the Stars (Love Theme From Atta…	8	05:33
532752	Zam the Assassin / The Chase Through C…	10	11:07
532754	Departing Coruscant	6	01:44
532755	Anakin and Padmé	6	03:56
532757	The Meadow Picnic	8	04:14
532758	Bounty Hunter's Pursuit	8	03:23
…	…ne	8	06:56
…	/ The Homestead	6	05:54
…	e Arena	8	08:29
…	th Count Dooku / Finale	8	10:45

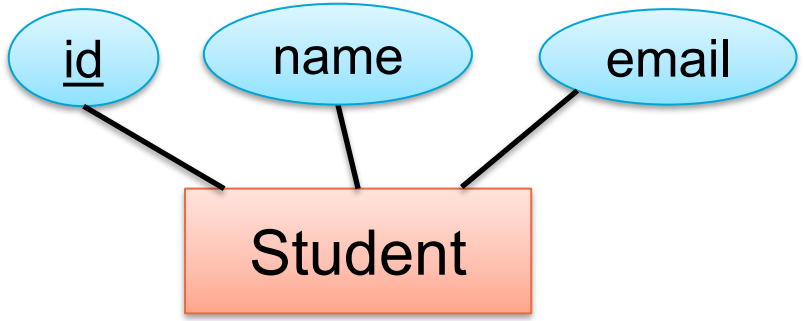


Conceptual To Logical

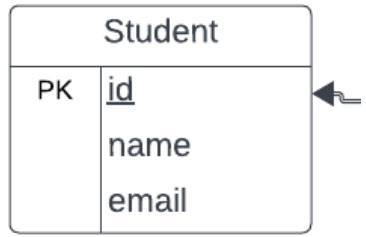
- Converting a simple Entity Type into a relation schema:

It's as simple as that

Conceptual Schema
(EER Diagram)



Logical Relational Schema
(ER Diagram)



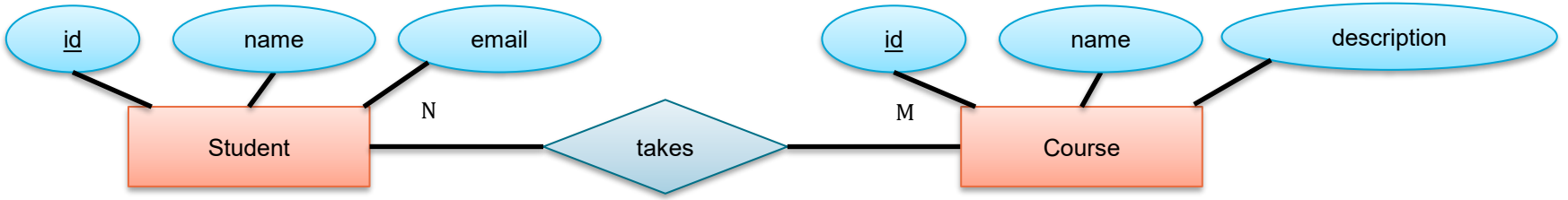
Student(id, name, email)

Logical Relational Schema
(Text Representation)

Conceptual To Logical

- Converting an N:M relationship type into a logical relation schema:
  - Relationship type becomes a separate table
    - Foreign key constraints ensure that the two entity types are properly linked
    - The new relationship table uses a composite primary key

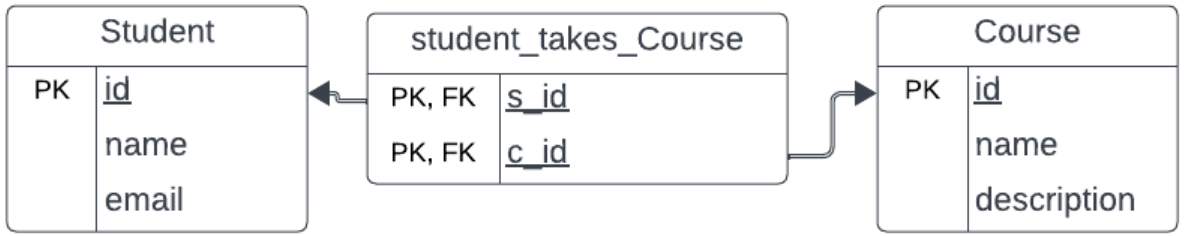
Conceptual Schema
(EER Diagram)



Student(id, name, email)
student_takes_course(s_id → Student(id), c_id→ Course(id))
Course(id, name, description)

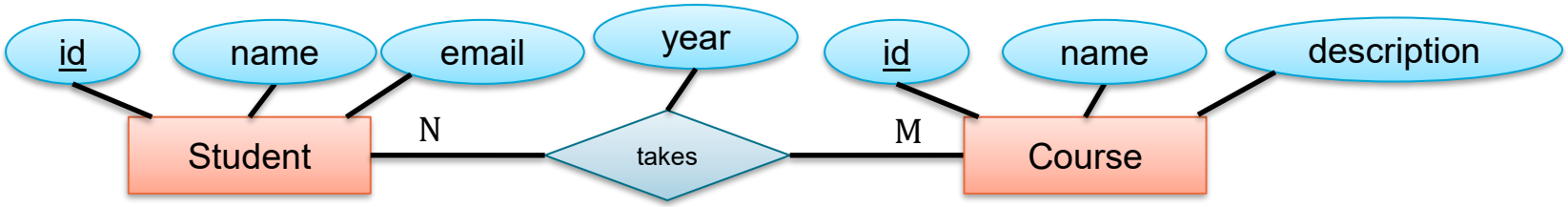
Logical Relational Schema
(Text Representation)

Logical Relational Schema
(ER Diagram)



Conceptual To Logical

- We can add additional attributes to relationships!

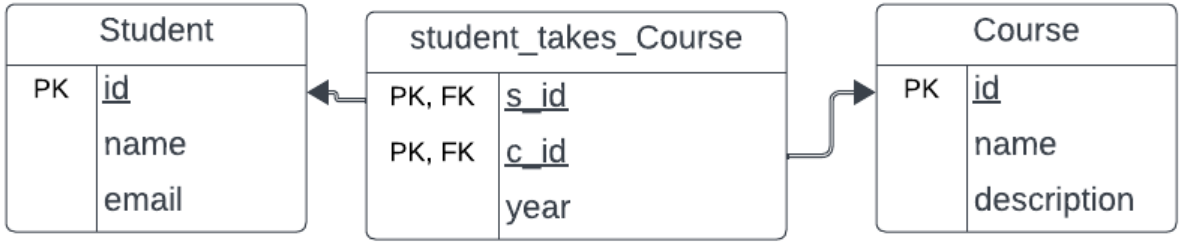


Conceptual Schema
(EER Diagram)

Student(id, name, email)
student_takes_course(s_id → Student(id), c_id→ Course(id), year)
Course(id, name, description)

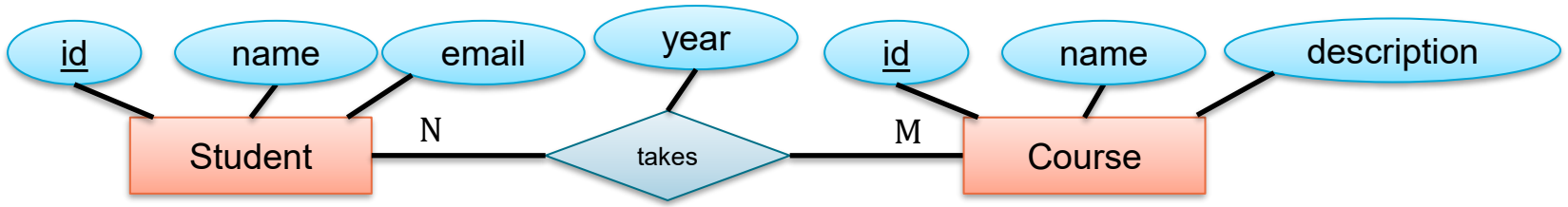
Logical Relational Schema
(Text Representation)

Logical Relational Schema
(ER Diagram)



Conceptual To Logical

- We can add additional attributes to relationships!
  - This does still not allow a student to take a course multiple times in different years
    - Primary Key is (Student.id, Course.id)!

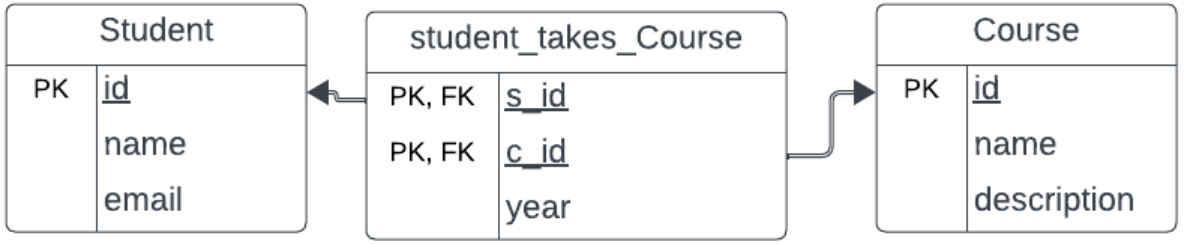


Conceptual Schema
(EER Diagram)

Student(id, name, email)
student_takes_course(s_id → Student(id), c_id→ Course(id), year)
Course(id, name, description)

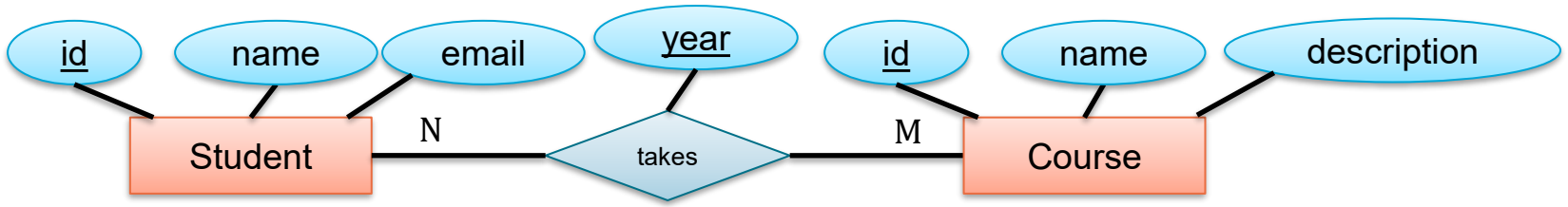
Logical Relational Schema
(Text Representation)

Logical Relational Schema
(ER Diagram)



Conceptual To Logical

- How can a student take a course multiple times?
  - Expand the primary key to include more attributes!
    - Taking a course is now identified by (Student.id, Course.id, year)!

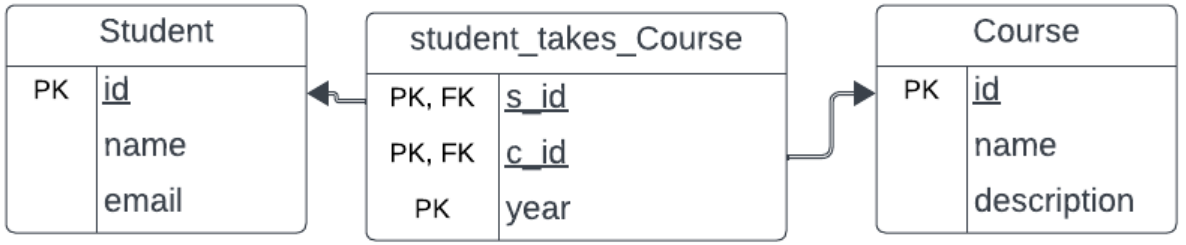


Conceptual Schema
(EER Diagram)

Student(id, name, email)
student_takes_course(s_id → Student(id), c_id→ Course(id), year)
Course(id, name, description)

Logical Relational Schema
(Text Representation)

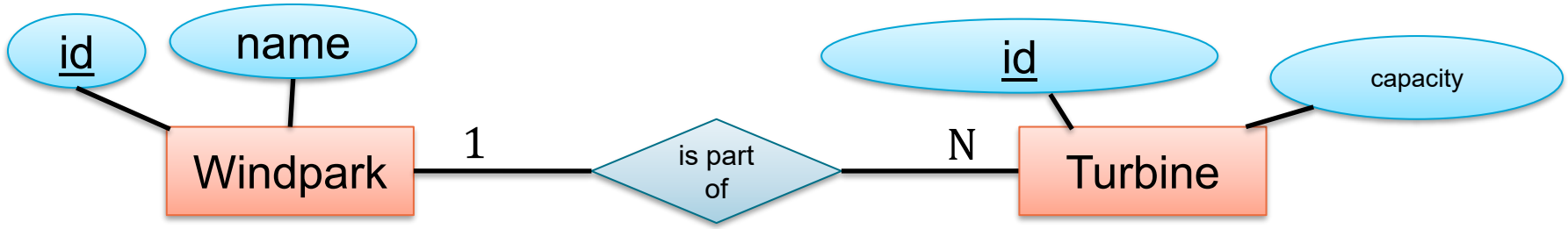
Logical Relational Schema
(ER Diagram)



Conceptual To Logical

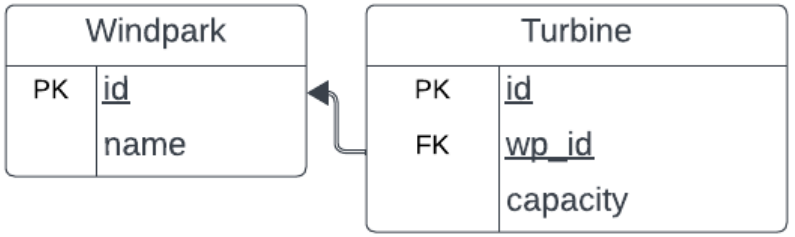
- Converting a 1:M relationship type a logical relation schema:
  - Entity Type at 1-side can only participate once at the relationship type
    - => Push relationship type to the 1-side

Conceptual Schema



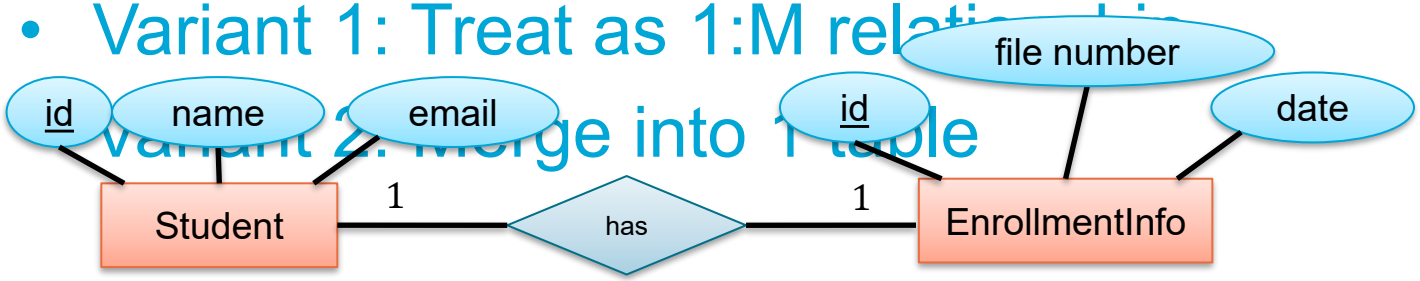
Logical (Relational) Schema

Windpark(id, name)
Turbine(id, capacity, **wp_id→ Windpark(id)**)



Conceptual To Logical

- Converting a 1:1 relationship type into a logical relation schema:
  - Many choices….
    - Variant 1: Treat as 1:M relationship
    - Variant 2: Merge into 1 table



Logical (Relational) Schema: Variant 1

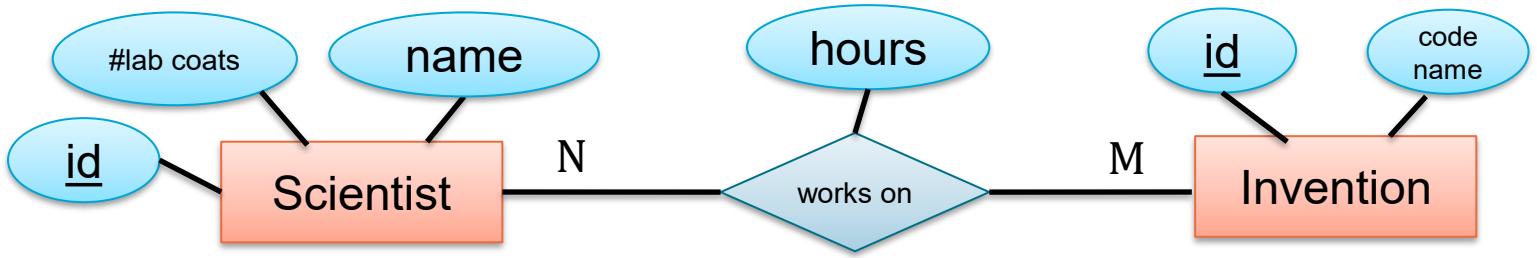
Student(id, name, email)
EnrollmentInfo(id, fileNumber, date, **s_id→ Student(id)**)

Logical (Relational) Schema: Variant 2

Student(id, name, email, enrollmentId, enrollmentFileNumber, enrollmentDate)

Conceptual To Logical

- How to deal with attributes attached to a relationship type
  - For N:M, just put:



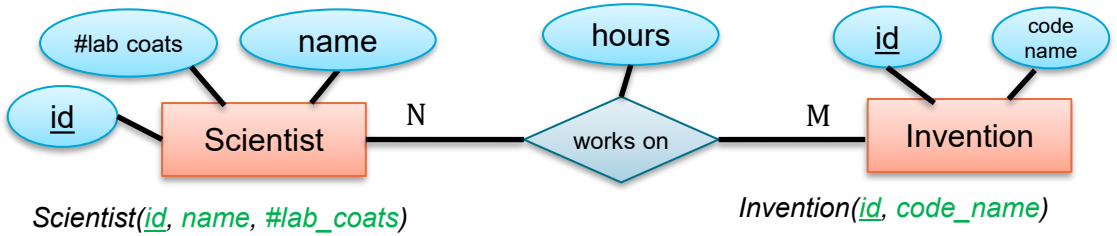
Scientist(id, name, #lab_coats)

Invention(id, code_name)

scientist_works_on_invention(
scientist → Scientist(id),
invention → Invention(id),
hours
)

Conceptual To Logical

- Don't do this!!

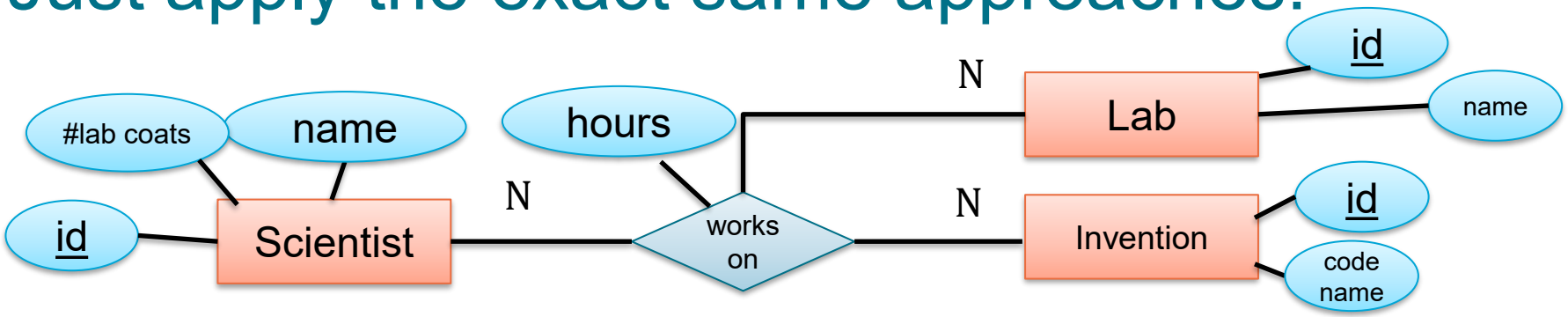


scientist_works_on_invention(
worksOnID,
scientist → Scientist(id),
invention → Invention(id),
hours
)

  - While this is not incorrect per se, the performance of this will be bad
    - Physical design heuristics (see IDM); messes up primary indexes

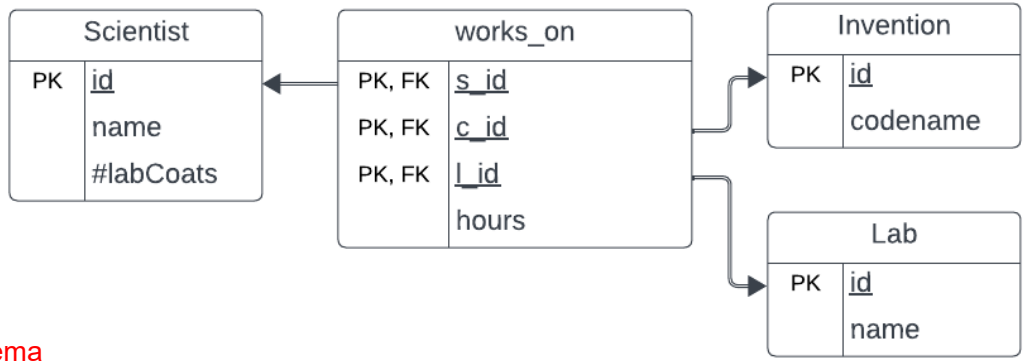
Conceptual To Logical

- What about n-ary relationship types? (n>2)
  - Just apply the exact same approaches:



Conceptual Schema
(EER Diagram)

works_on(
s_id → Scientist(id),
i_id → Invention(id),
l_id → Lab(id)
hours)



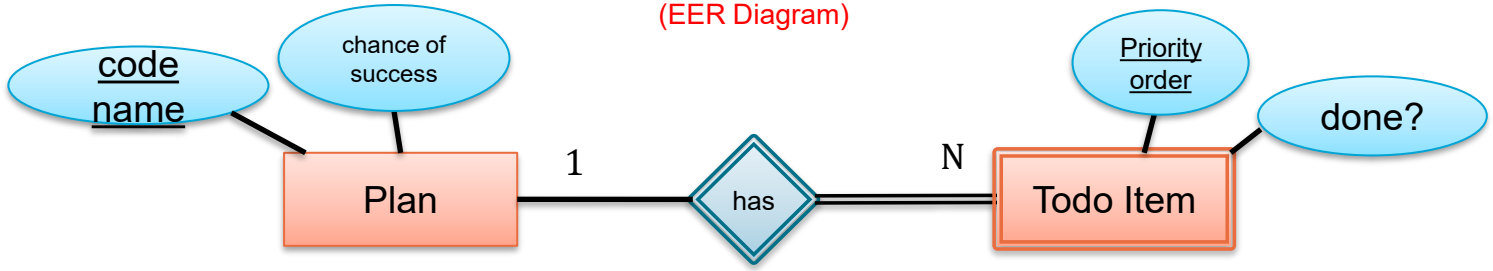
Logical Relational Schema

Conceptual To Logical

- Converting a weak entity into a relation schema:
  - Weak entities are only unique together with the entity at the **identifying relationship**

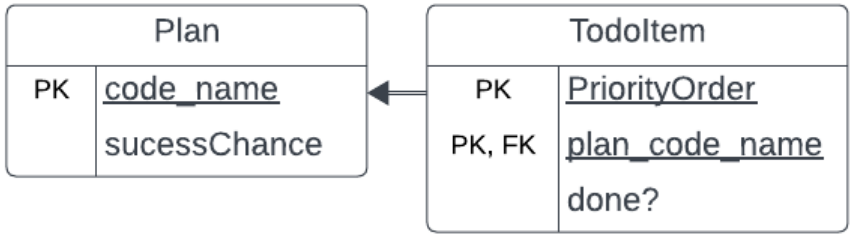
  => Follow identifying relationship and inherit respective foreign keys

Conceptual Schema
(EER Diagram)

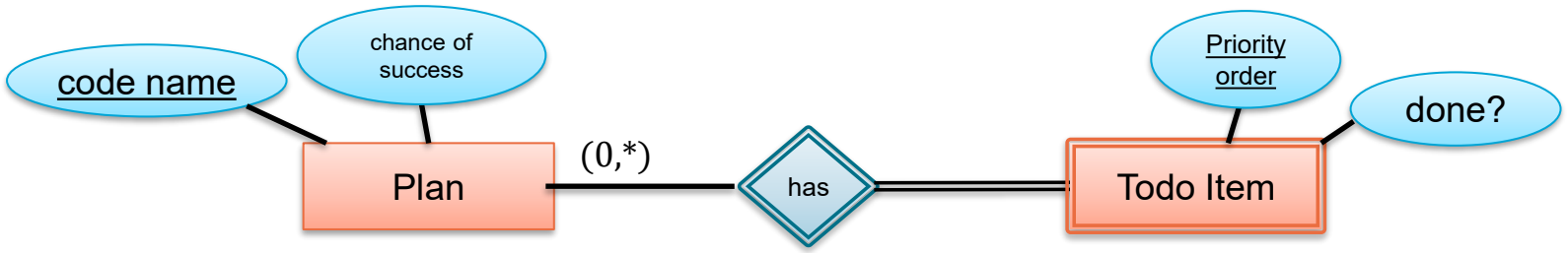


Logical Relational Schema

Evil_Plan(*code_name*, *chance_of_success*)
todo_item(priority_order, plan → Plan(codename), done)



Conceptual To Logical



Plan(*code_name*, *chance_of_success*)
todo_item(priority_order, plan_name → Plan(code_name), done)

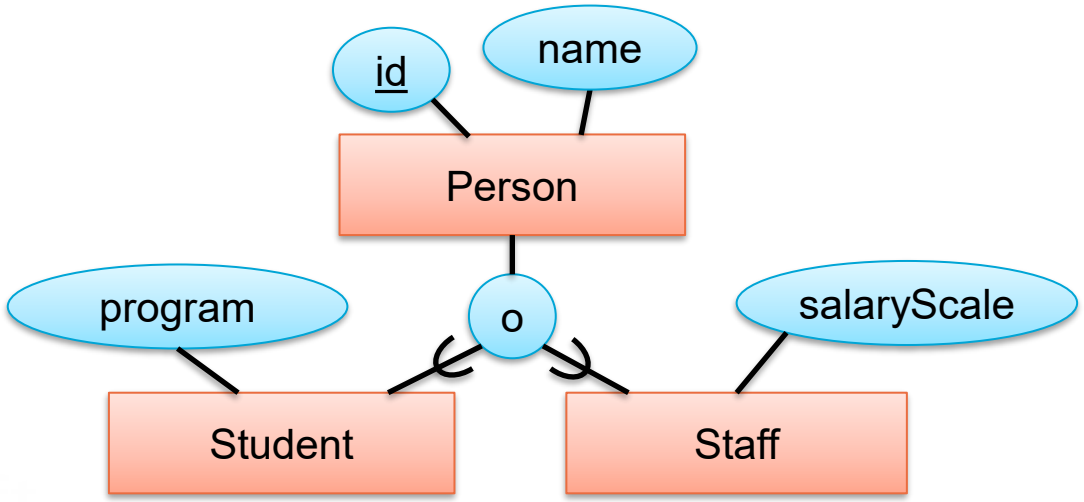
Vs:

Evil_Plan(*code_name*, *chance_of_success*)
todo_item(plan_name → Plan(code_name), priority_order, done)

- Spoiler IDM CSE1505: "2nd version is likely better as it will order all todo items on disc by evil plans; so, all todo items belong to a plan are close by
  - This is a bit of physical design…

Conceptual To Logical

- Converting types with inherited attributes/ relations into a relation schema:
  - Can be implemented in many ways
    - More than 5…

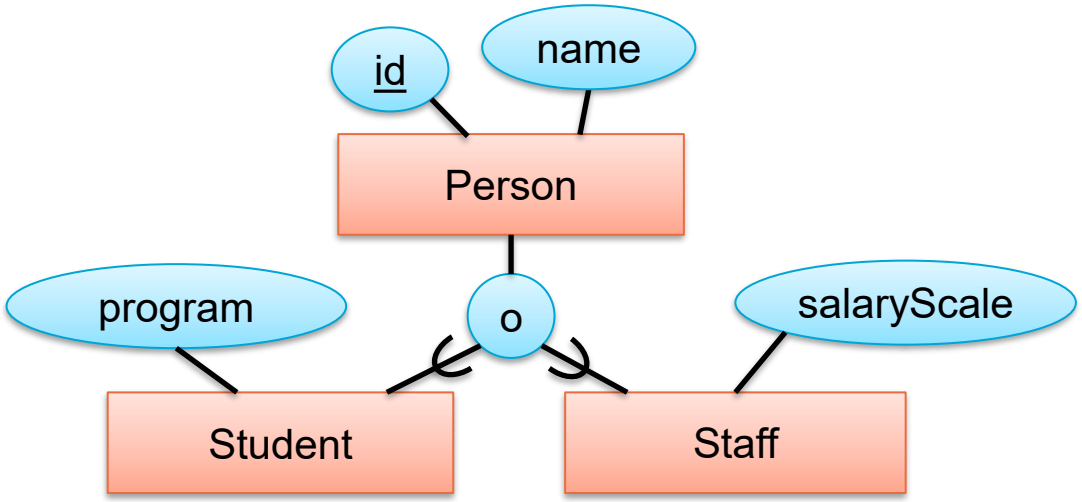


Conceptual To Logical

Overlapping non-total Inheritance:
Persons can be Students, or Staff, or both or neither.

- Version 1:

Person(id, name)
Student(id → Person(id), program)
Staff(id → Person(id), salaryScale)

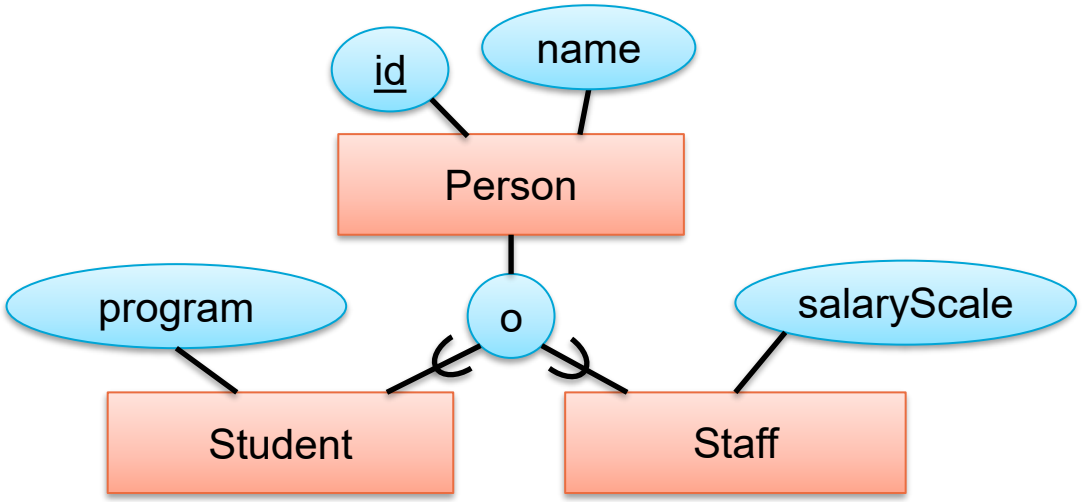


All Persons (including Students and Staff) have a row in **Person**,
Students have an extra row in **Student** referring to Person,
Staff have an extra row in **Staff** referring to Person
-> Clean but maybe clumsy?

Conceptual To Logical

Overlapping non-total Inheritance:
Persons can be Students, or Staff, or both or neither.

- Inheritance Version 2:



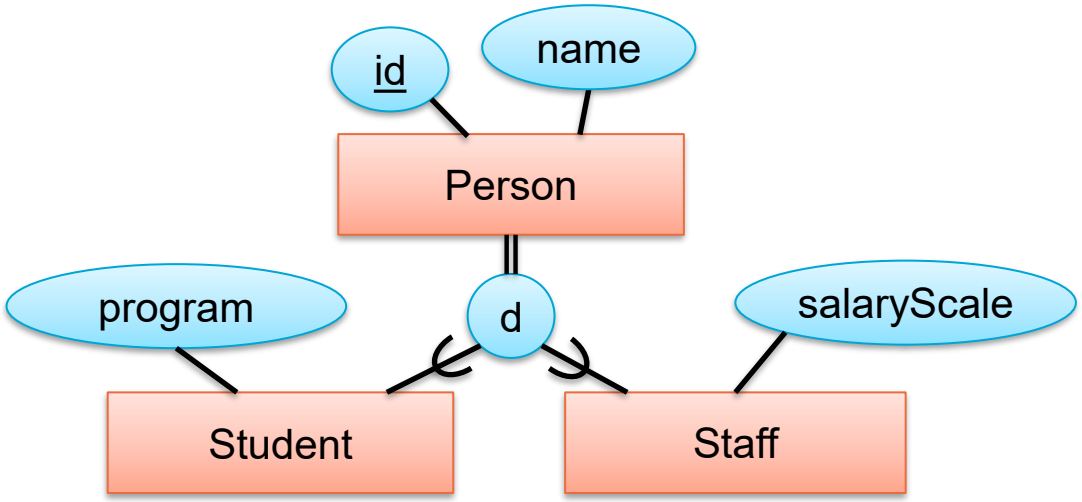
Person(id, name)
Student(id, name, program)
Staff(id, name, salaryScale)

Students have a row in Student table.
Staff have a row in Staff table.
Students who are also Staff have a row in both **Student and Staff**.
Person who are neither Students nor Staff have a row in **Person**.
-> Likely not a nice solution?

Conceptual To Logical

Disjoint total Inheritance:
Persons are Students or Staff, but not both nor neither.

- Inheritance Version 2: (now total disjoint!!)



Student(id, name, program)
Staff(id, name, salaryScale)

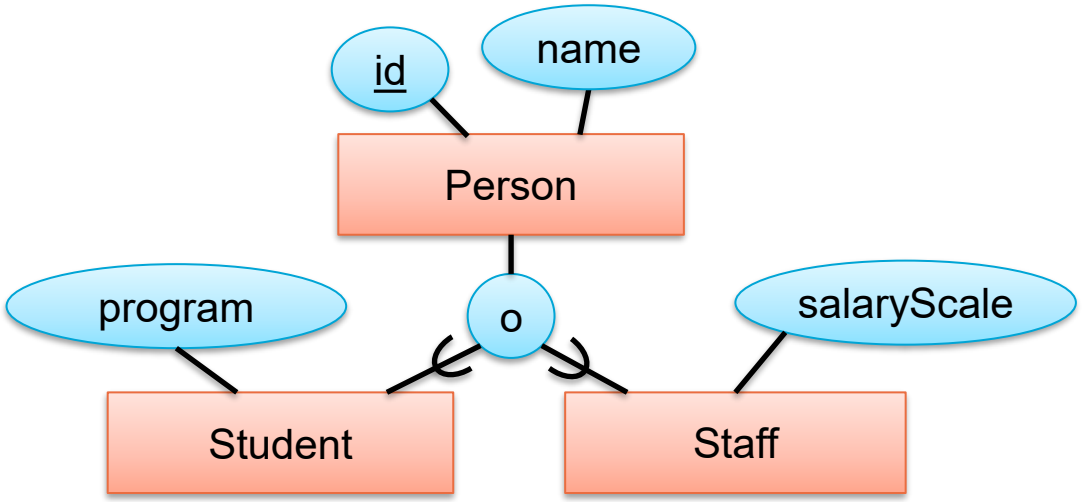
No need for Persons anymore (as everything is either Student xor staff)
Potential mess with ids (there could be a student with same id as a staff member), but likely okey?
-> Better…?

Conceptual To Logical

Overlapping Non-Total Inheritance:
Persons can be Students, or Staff, or both or neither.

- Version 3:

Person(<u>id</u>, name, program, salaryScale)



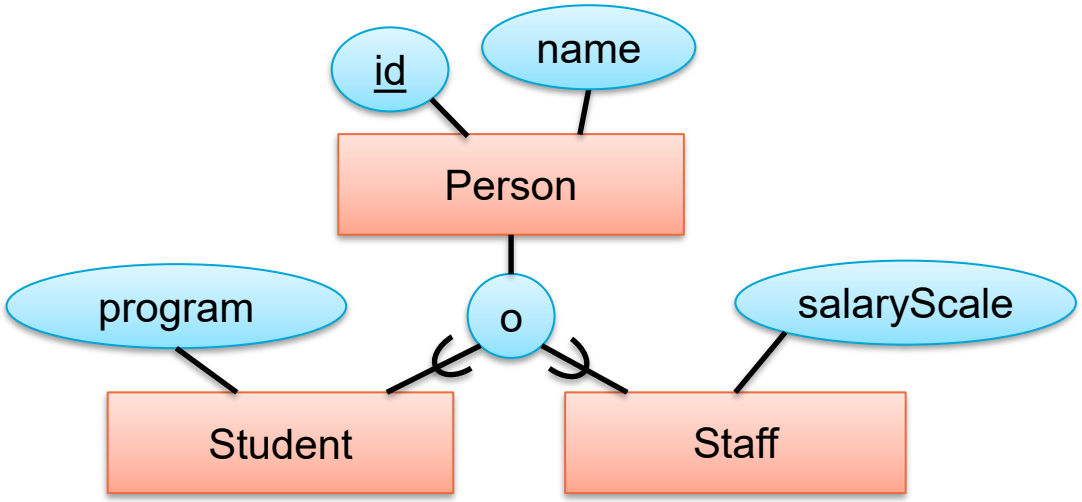
Only one big table!
Use program attribute iff Student, use salaryScale iff Staff.
Leave unused attributes empty/NULL.
No explicit type information in the logical schema.
It's somewhat ugly but also popular.

Conceptual To Logical

Overlapping Non-Total Inheritance:
Persons can be Students, or Staff, or both or neither.

- Version 3b:

Person(id, type, name, program, salaryScale)



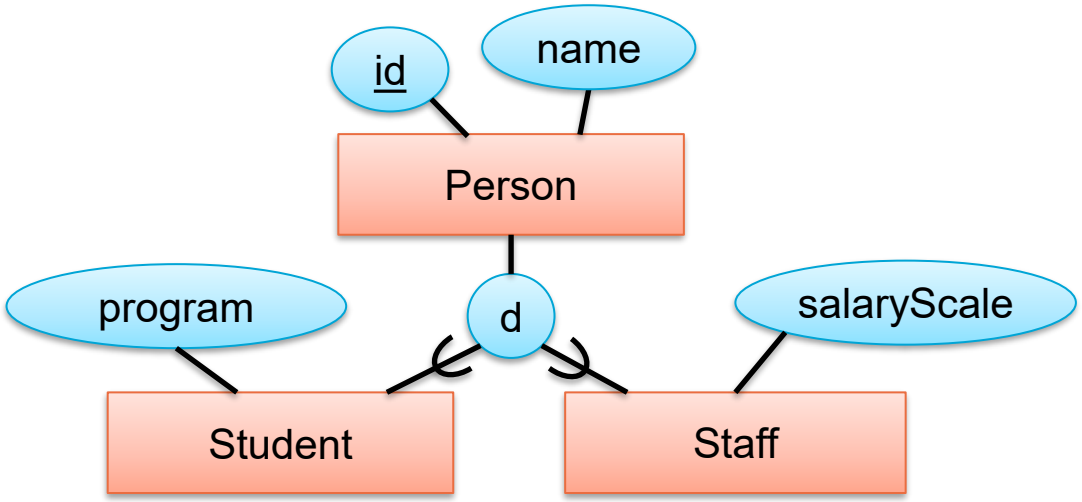
A variant of Version 3: Just add a new “type” attribute which stores what type a person is, e.g. “student”, “staff”, “both”, “neither”. Checking subtypes involves a lot of manual String comparisons.

Debatable if this is nice…

Conceptual To Logical

Disjoint Non-Total Inheritance:
Persons can be Students, or Staff, or neither.

- Version 3c:



Person(id, type->Type(tid), name, program, salaryScale)
Type(tid, type_name)

A variant of Version 3b: A second table holds information about all available types, e.g. “staff”, “student”, etc. Works only nicely for disjoint inheritance.