Database Design

Course Objectives

- Use entity-relational data modeling to design application storage schemas
- Use relational database concepts, SQL, Postgres, and supporting tools to create and modify schemas and data
- Use SQL to query data and estimate complexity of the query
- Design normalized and efficient database relations
- Decide when to avoid normalization and use non-relational data models
- Apply transactions to guard data consistency

Course Plan

Session 1	Introduction to databases, E-R model
Session 2	Relational model
Session 3	Working with PostgreSQL
Session 4	Joins, Nested queries, Aggregations
Session 5	Views, Functions, Transactions
Session 6	Normalization
Session 7	Non-relational DBMS

Resources

- Database System Concepts, Silberschatz (Theory)
- Postgres documentation https://www.postgresql.org/docs/
- Learn PostgreSQL, Luca Ferrari
- NoSQL Distilled, M. Fowler

Session 1

Introduction to Database Systems

- Data persistence
- Drawbacks of using files
- Aspects of data quality
- Database managements systems (DBMS)
- DBMS applications
- Data abstraction levels
- Data models
- Entity-relationship model
- Relational model and SQL

Data persistence

- RAM is typically used for computations while the program is running
- Persistent storage is used to **store** the result of computations in order to **load** when needed again
- Data in RAM is lost when computer is turned off
- Data in persistent storage remains until cleared to empty space
- Data is stored as files in storage devices

Drawbacks of using files

- Files are efficient to store and load sequential data
- However when working we data-intensive applications we face the following challenges:
  - Data redundancy which leads to inconsistency
  - Need to develop programs to access the data on files
  - No data isolation and non-standard data formats
  - Difficult to enforce data integrity
  - Difficult to enforce atomicity of updates
  - Concurrency issues
  - Security issues
- **If we tried to solve the above issues in a general system and make it standardized, we would end up with a Database Management System (DBMS)**

Example

- A university is organizing different events and conferences and wants to collect data about participants.
  The organizers print out forms with the following columns (fields) and distribute among the participants.

Fullname	Student? (Y/N)	University	Company	Age	Email Address

- What are some issues that might occur with data collection?

Aspects of data quality

- **Completeness**. Do we have all the required data?
- **Consistency**. Do the data match?

  Student field is empty / University field is empty

  Data from different conference rooms
- **Uniqueness**. Do we have multiple rows that point to the same data?

  WF, Webb, WebbFountaine, 08/10/1990, 10/08/1990
- **Timelessness**. Data should mean the same thing over time
- **Validity**. Data that makes sense and is usable

  no_one@my_imaginary_domain.com
- **Accuracy**. Data is as detailed as required

  08/10/1990, 1990

Aspect of data quality – cont’d

- In order to make sure that the data has the mentioned qualities you’ll need to write programs or scripts.
- In DBMS data qualities are enforced using **data integrity constraints** (in short, constraints)
- A DBMS has a standard mechanism to **define** and **follow** the integrity of data according to the constraints that you as the developer defined.

Exercise

- Imagine that we are developing a Phonebook application where you can store and search contacts and their phone number, email, etc.
- Define some data integrity constraints that you'd think are important

Data qualities

- Completeness
- Consistency
- Uniqueness
- Timelessness
- Validity
- Accuracy

Database management system (DBMS)

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both **convenient** and **efficient** to use

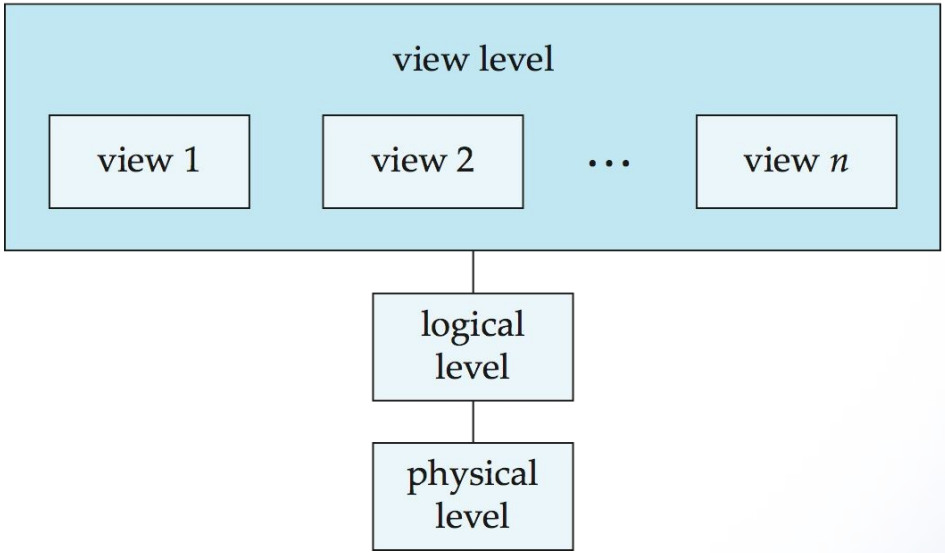
- Database application examples
  - Banking and financial transaction processing
  - Booking of airline tickets and hotels, Booking.com, AirBnb, etc.
  - Online stores
  - Manufacturing and management of resources, inventory, supply chain

Data abstraction levels

- Abstraction: When working with Java libraries to read/write data from/to files are you aware that the computer is working with a SSD or HDD?
- This is called “physical level abstraction”

- There are 3 levels of abstraction in DBMS
  - Physical level: Describes how a record is stored (record: customer, order, student, …)
  - Logical level: Describes the parts of data and their relationship from data design aspect
    Customer = name, email, credit card no, order history, ...
  - View level: Controls what and how should be displayed
    E.g. stored in USD but displayed in XOF, hide some fields for security

Data abstraction levels



A single piece of data can be displayed in different views.

Example: Phonebook

```
Person {
    string firstName,
    string lastName,
    PhoneNumber[] phoneNumbers
}
```

```
PhoneNumber {
    integer type,
    integer country,
    integer area,
    integer rawNumber
}
```

- How to store the data physically?
- Identify the physical, logical, and view abstraction levels.

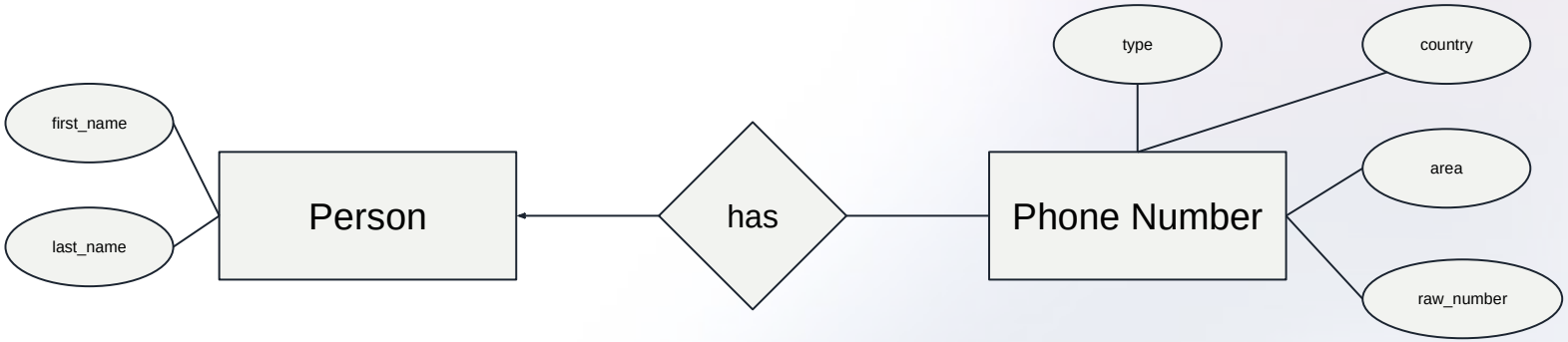
Data models

- In the previous examples we used a light-weight model to describe:
  - What are the pieces of data we have in the Phonebook application
  - What are the types of each piece of the data
  - How are the pieces related to each other (e.g. A person can have multiple phone numbers)
  - We could also add some constraints: Think of some?

- There are two main methods for data modeling in order to develop DBMS applications:
  - Entity-Relationship (ER) model and diagrams (ERD)
  - Relational model and its query language (SQL)

Entity-relationship model

- The application data of your domain is divided into two types 1) entity, 2) relationship
  - **Entity**: Is the “thing” that has some meaning in your application and you want to model it. Each entity is made of several “attributes”
  - **Relationship**: How are the entities related to each other
- Phonebook example: Two entities 1) Person, 2) Phone Number and a relationship between the two (“has”): One person has many phone numbers



sourcemind

Relational model

- In Relational model, there are relations and attributes only. The data is modeled as relations (which is similar to concept of set in math).

  Relation 1: Person (person_id, first_name, last_name)

  Relation 2: Phone Number (type, country, area, raw_number, **person_id**)

- Relations are also called **tables** since they are implemented as tables in relational DBMS (RDBMS)
- The table defines the data “format” or “template” and the data instances are added as “rows”

Relational model

Person

person_id	first_name	last_name
1	Jon	Snow
2	Ned	Stark

Phone Number

type	country	area	raw_number	person_id
0	AM	EVN	123456	1
1	AM	EVN	112233	1
1	US	CA	333444	2

- person_id is the primary key of Person
- person_id is a foreign key in Phone Number, since it refers to a Person

Data models

ER model

- Conceptual
- Visual
- Useful for thinking and understanding the data in your problem domain
- Unimplementable

Relational model

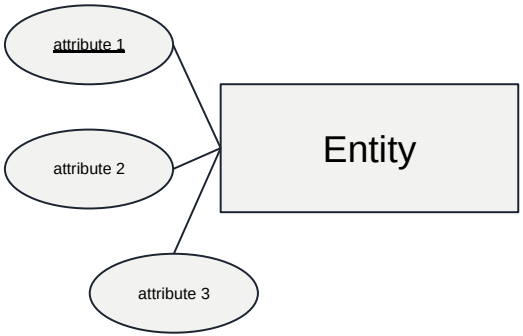
- Practical
- Implementable in SQL
- Real-world

We will work with both ER and Relational models.

ER Modeling

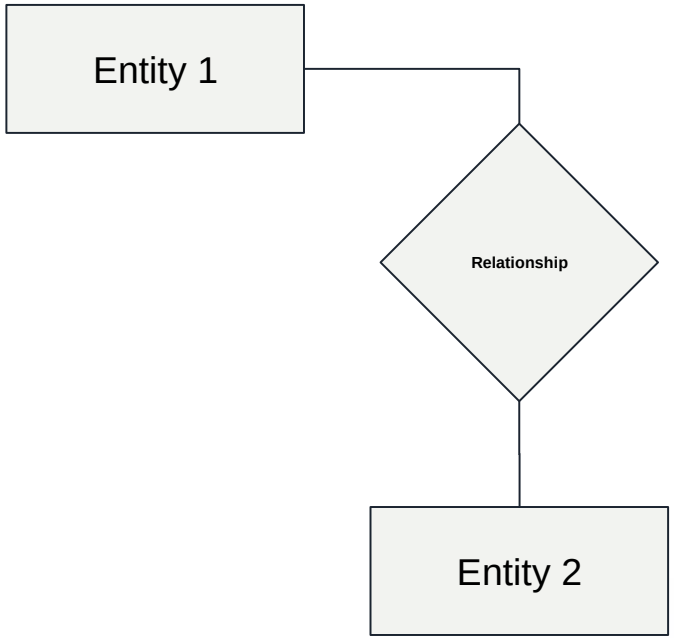
- Entities
- Attributes
- Relationships
- Cardinalities

ER model elements: Entity



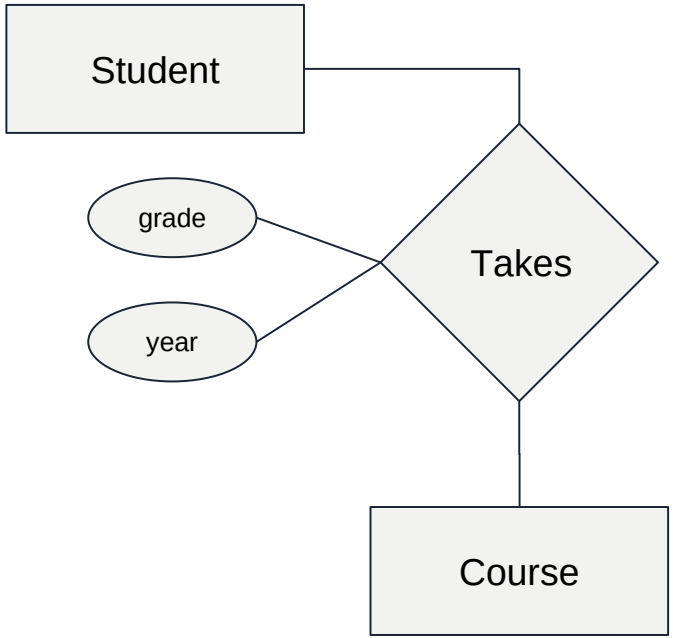
- Entity: A combination of data pieces that comprise a concept in the application
- Entities are nouns
- Attribute: Parts of an entity
- Unique key: One or more attributes that uniquely identify an entity
  - Passport No, Student ID,...
  - Can be **composite**: First name + last name ?
- An entity can have several **candidate** keys. It is possible to uniquely identify an entity with several keys but only one of them must be chosen.

ER model elements: Relationship



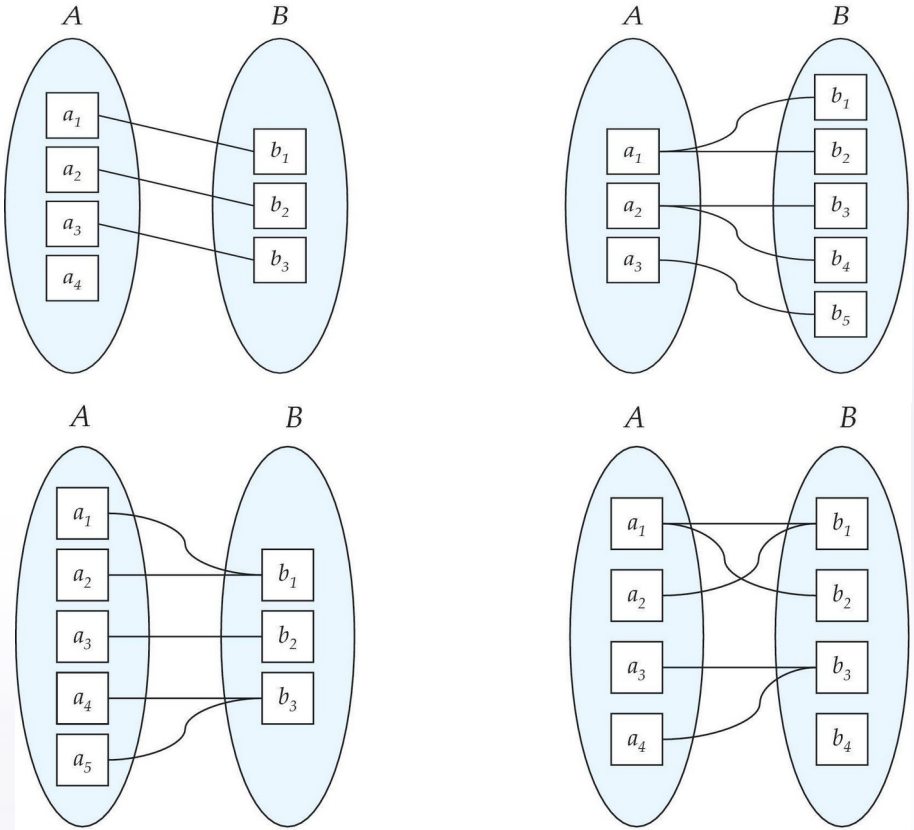
- Relationship: The relationship between two or more entities
  - Person **has** phone number
  - Student **takes** course
  - Instructor **teaches** course
- Relationships are verbs

ER model elements: Relationship



- Relationship can also have attributes
  - Student **takes** course and finishes the course with a **grade**
  - “Grade” is the attribute of the relationship “takes” between “Student” and “Course”
- Attribute of relationships describe additional data

ER model elements: Cardinality

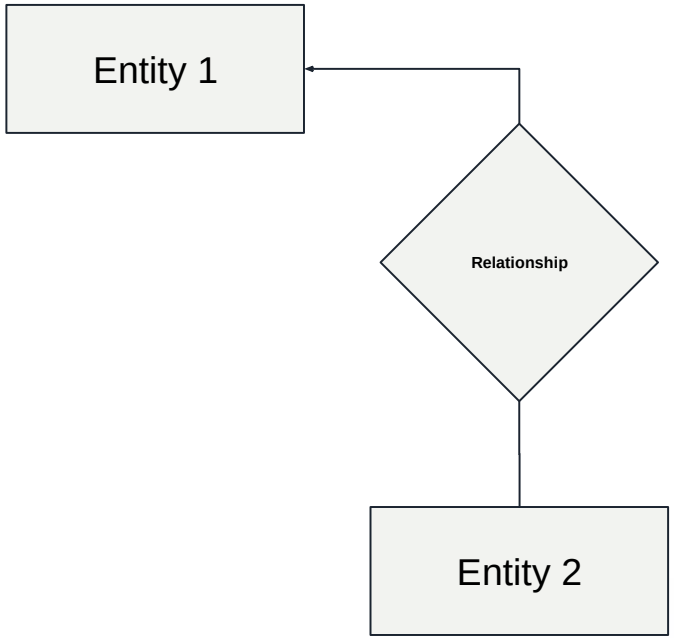


The relationship between two entities can be:

- One to one
- One to many
- Many to one
- Many to many

sourcemind

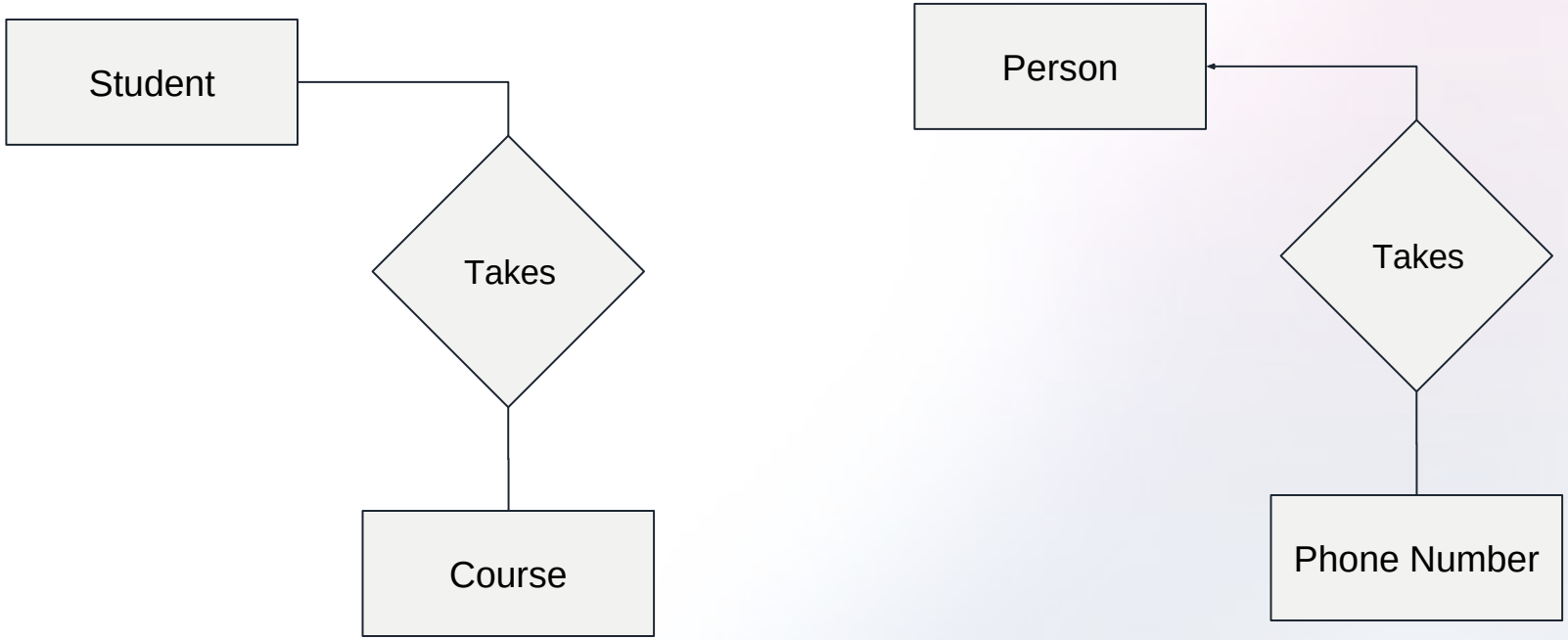
ER model elements: Cardinality



- We use simple line (–) to denote “many”
- We use directed line (→) to denote “one”

- Entity 1 has relationship with many Entity 2

ER model elements: Cardinality



Homework 1: Football match