

# Databases – Introduction

Jörg Endrullis

VU University Amsterdam

## Course Goals & Structure

# Course Goals

## Overall Goal

Thorough understanding of database concepts

- from a user perspective  
(not how databases work internally)

## Learning Goals

- Developing data models
- Reasoning about good/bad design  
(functional dependencies)
- Understanding and writing non-trivial SQL statements
- Basic knowledge of database programming

## Motivation

Storing data is important everywhere in industry!

# Course Structure

- **Introduction & Overview**
- **Relational Model**
- **Data Modelling**
  - entity relationship diagrams (ER)
  - unified modelling language (UML)
- **From Conceptual to Relational Model**
- **Advanced SQL**
  - writing nested queries with joins
- **Functional Dependencies**
  - reasoning about good/bad design
  - normalising a database schema
- **Transactions**
  - analysing transaction schedules
- **Database APIs**

## Database Management Systems

# Databases

A **database (DB)** is a collection of data with

- a certain logical structure
- a specific semantics
- a specific group of users

A **database management system (DBMS)** allows to

- **create, modify** and manipulate a database
- **query** (retrieve) the data using a query language
- support **persistent** storage of **large amounts of data**
- enable **durability** and **recovery** from failure
- **control access** to the data by many users in **parallel**
  - without unexpected interactions among users (**isolation**)
  - actions on the data should never be partial (**atomicity**)

# Motivation for Database Management Systems

## Why not just store data in files?

- **no query language**
- **weak logical structure** (limited to directories)
- **no efficient access**
  - searching through a large file can take hours
- no or **limited protection** from data loss
- **no access control** for parallel manipulation of data

So we need database management systems. . .

## ANSI SPARC Architecture



# Motivation for Database Management Systems

## Motivation for database management systems

- **data independence**

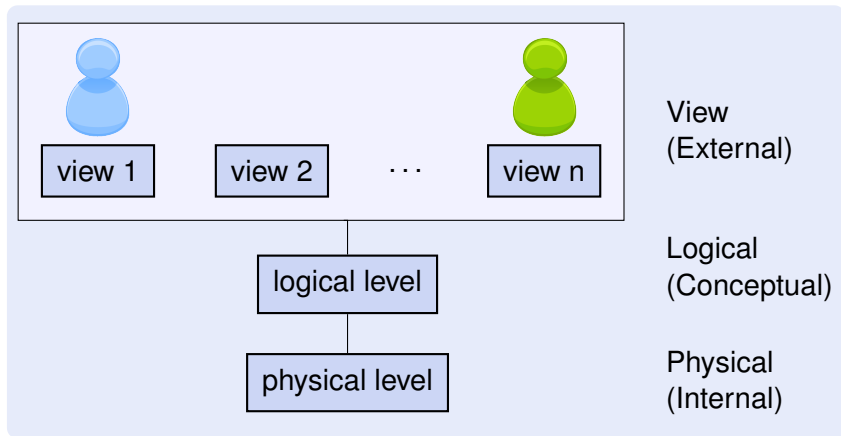
- logical view on the data independent of physical storage
- user interacts with a simple view on the data
- behind the scenes (invisible for the user) are complex storage structures that allow rapid access and manipulation

- **avoidance of duplication**

- different views on the same database
  - for different users or different applications
  - hiding parts of the data for privacy or security

This is achieved by the ANSI SPARC Architecture . . .

# ANSI SPARC Architecture: 3 levels



- Different applications might use different views
- Data stored once at physical level (good for consistency)

# ANSI SPARC Architecture: 3 levels

## ANSI SPARC Architecture

- **View level:**

- application programs hide details of data types
- hide information (e.g. exam grade) for privacy or security

- **Logical level:** also called 'conceptual schema'

- describes data stored in the database, and
- relations among the data

- **Physical level:**

- how the data is stored
- disk pages, index structures, byte layout, record order

This ensures logical and physical data independence. . .

# Data Independence

## Logical data independence

**Logical data independence** is the ability to modify the logical schema without breaking existing applications

- applications access the views, not the logical database

## Physical data independence

**Physical data independence** is the ability to modify the physical schema without changing the logical schema

- e.g. a change in workload might cause the need for
  - different indexing structures
  - different database engine
  - distributing the database on multiple machines
  - ...

## Relational Model


# Relational Model

In this course, we work with **relational databases**.  
View and logical level represent data as **relations/tables**.

## Example relational database instance

Customers			
<u>id</u>	name	street	city
191	George	1 Main	London
302	Elvis	12 East	Amsterdam
239	Lisa	5 North	New York

Accounts	
depositor	<u>accountnr</u>
191	101
302	217
239	205



■ **row = tuple record:** (302, Elvis, 12 East, Amsterdam)

In the **pure relational model**, a table is a **set** of tuples:

- has no duplicate tuples (rows)
- no order on the tuples

# Relational Model: Schema

## Database schema

= structure of the database, that is, relations + constraints

## Example schema

- Customers(id, name, street, city)
- Accounts(depositor → Customers(id), accountnr)

## Database instance

= actual content ('state') of the database at some moment

## Example instance

Customers			
<u>id</u>	name	street	city
191	George	1 Main	London
239	Lisa	5 North	New York

Accounts	
depositor	<u>accountnr</u>
191	101
239	205

## Structured Query Language



# Motivation for Database Management Systems

## Motivation for database management systems

- high-level **declarative query languages**
  - query tells what you want, independent of storage structure
  - efficient data access (automatic query optimisation)

Declarative query languages:

- describe **what** information is sought
- **not** prescribe **how** to retrieve the desired information

# Imperative vs. Declarative Languages

Algorithm = Logic + Control

(Kowalski)

**Imperative** languages:

- explicit control
- implicit logic

**Declarative** languages:

- implicit control
- explicit logic

Examples of declarative languages

- logic programming (e.g. Prolog),
- functional programming (e.g. Haskell),
- markup languages (e.g. HTML), ...

Relational databases usually use SQL as query language ...

# SQL = Structured Query Language

SQL is a declarative data manipulation language. The user describes conditions the requested data is required to fulfil.

## SQL Query

```
select id
from   Customers
where  name = 'Elvis' and city = 'Amsterdam'
```

More concise than imperative languages:

- less expensive program development
- easier maintenance

Database system will optimise the query:

- decides how to execute the query as fast as possible
- users (usually) do not need to think about efficiency

## Data Models & Integrity Constraints

# Motivation for Database Management Systems

## Motivation for database management systems

- well-defined **data models** & **data integrity constraints**
  - relational model
  - meta language for describing
    - data
    - data relationships
    - data constraints

SQL can be used for table and constraint definitions ...

# Integrity Constraints

## Example schema with key constraints

- Customers(id, name, street, city)  
**Primary key constraint** on id
- Accounts(depositor → Customers(id), accountnr)  
**Foreign key constraint** on depositor

## Various types of constraints:

- **data types**, constrained data types (domains)  
(e.g. numeric(2,0), varchar(40), ...)
- **columns constraints**  
(e.g. unique, nullability, counter, ...)
- **check constraints**: logical expression for domain integrity  
(e.g. age >= 18 and age <= 150)

# SQL DDL (Data Definition Language)

## Creating a table with constraints

```
create table Solved (  
  id          int          auto_increment,  
  name        varchar(40)  not null,  
  homework    numeric(2,0) not null,  
  points      numeric(2,0) not null check (points <= 10),  
  primary key (id)  
);
```

Note the data types and constraints!

Solved			
<u>id</u>	name	homework	points

## Creating a view

```
create view SolvedHomework as  
  select id, name, homework  
  from   Solved;
```

## Concurrent Access & Transactions



# Concurrent Access & Transactions

## Motivation for database management systems

- multiple users, **concurrent access**
  - transactions with ACID properties

A **transaction** is a sequence of operations that performs a single logical function in a database application.

## Database management system ensures **ACID properties**

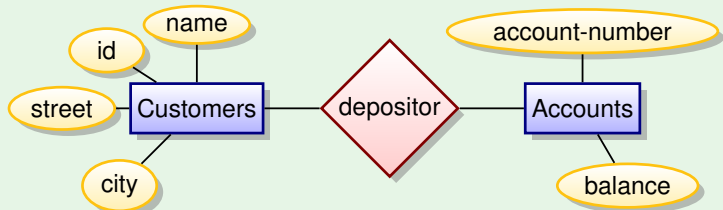
- **Atomicity:** transaction executes fully (commit) or not at all (abort)
- **Consistency:** database remains in a consistent state where all integrity constraints hold
- **Isolation:** multiple users can modify the database at the same time but will not see each others partial actions
- **Durability:** once a transaction is committed successfully, the modified data is persistent, regardless of disk crashes

## Designing Database Schemes

# Entity Relationship (ER) Model

## Entity relationship (ER) model

- entities = objects
  - e.g. customers, accounts, bank branches
- relationship between entities
  - e.g. account 217 is held by customer Elvis
  - **relationship set descriptor** links customers with accounts

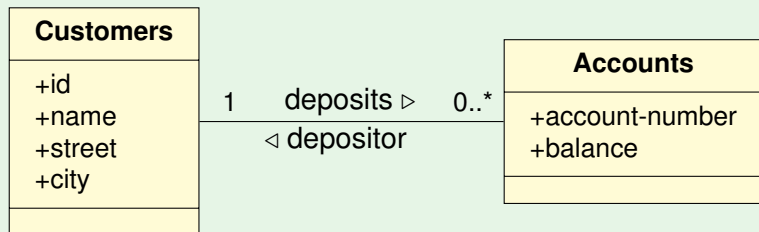


# UML Class Diagram

## UML class diagrams

- similar to ER diagrams:  
**entities/relationships**  $\Rightarrow$  **classes/associations**

## Example schema as UML class diagram



Conceptual design is usually converted to the relational model.

## Summary

## Why Database Management Systems?

- **data independence**
  - logical view on the data independent of physical storage
- **avoidance of duplication**
  - different views on the same database
- high-level **declarative query languages** (what, not how)
  - efficient data access, automatic query optimisation
- **data models & data integrity (consistency)**
- multiple users, **concurrent access**
  - transactions with ACID properties
- **persistent storage, safety and high availability**
  - safety against failure (backup/restore)
- **scalability** (data could be much larger than main memory)
  - indexing, scalable algorithms
- **security** (user permission management)