

Databases

Jörg Endrullis

VU University Amsterdam

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**)

Why not just store data in files?

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. . .

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

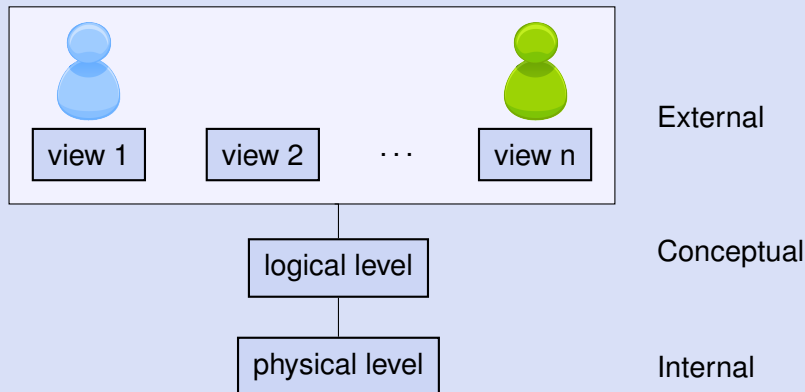
■ **avoidance of duplication**

- 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 . . .

View of Data

ANSI SPARC Architecture: 3 levels



- Different applications might use different views
- Data is stored only once at the physical level
 - good for consistency

ANSI SPARC Architecture: 3 levels

- **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

In this course, we work with **relational databases**. Their view and logical level represented data as **relations/tables**.

Example relational database instance

customer			
<u>id</u>	name	street	city
1928374	Johnson	12 Alma	Palo Alto
3211231	Jones	34 Main	Harisson
0192837	Smith	7 South	Rye

account	
depositor	<u>accountnr</u>
1928374	101343
3211231	217343
0192837	201762

- 
- **row = tuple record:** (3211231, Jones, 34 Main, Harisson)

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

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

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

Queries should:

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

Relational databases usually use SQL as query language ...

Imperative vs. Declarative Languages

Kowalski

Algorithm = Logic + Control

Imperative/procedural languages:

- explicit control
- implicit logic

Declarative/non-procedural languages:

- implicit control
- explicit logic

Examples of declarative languages

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

SQL = Structure 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 CUSTOMER  
WHERE NAME = 'Jones' AND CITY = 'Harisson'
```

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.

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 ...

Relational Model: Schema

Database schema

= structure of the database = relations + constraints

Example schema

- customer(id, name, street, city)
- account(depositor → customer, accountnr)

Database instance

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

Example instance

customer			
<u>id</u>	name	street	city
1928374	Johnson	12 Alma	Palo Alto
0192837	Smith	4 North	Rye

account	
depositor	<u>accountnr</u>
1928374	101343
0192837	215569

Integrity Constraints

Example schema with key constraints

- customer(id, name, street, city)
Primary key constraint on id
- account(depositor → customer, accountnr)
Foreign key constraint on depositor

Various types of constraints:

- data types, constrained data types (domains)
- columns constraints (e.g. unique, nullability, counter, ...)
- check constraints (logical expression for domain integrity)
(e.g. age \geq 18 AND age \leq 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) NOT NULL,  
  points NUMERIC(2) 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 solved_homework AS  
  SELECT id, name, homework FROM solved;
```

How to Design Database Schemes?

How to design database schemes?

- Entity-relationship models (ER models)
- UML class diagrams

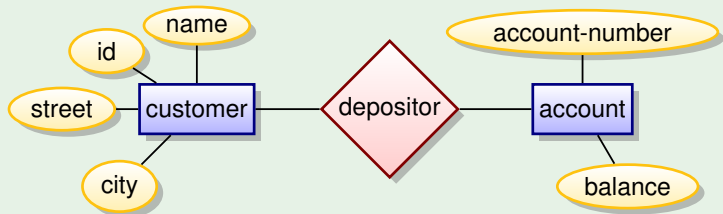
They are

- widely used for database design
- usually converted to the relational model

Entity Relationship Model

Entity relationship model

- entities = objects
 - e.g. customers, accounts, bank branches
- relationship between entities
 - e.g. account 101343 is held by customer Johnson
 - **relationship set descriptor** links customers with accounts



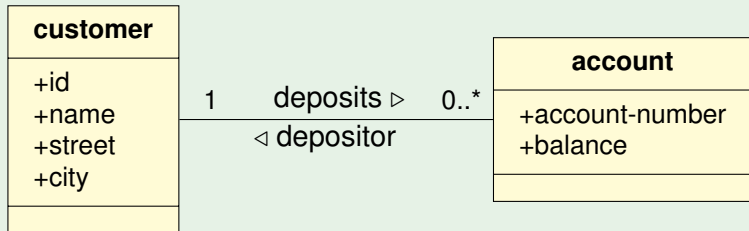
UML Class Diagram

UML class diagrams

- frequently used in database design
- similar to E/R diagrams:

Entities/Relationships \Rightarrow Classes/Associations

Example Schema as UML Class diagram



Motivation for Database Management Systems

Motivation for database management systems

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

A **transaction** is a collection 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

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**