Databases

Jörg Endrullis

VU University Amsterdam

2015

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**)

Databases

Why not just store data in files?

- no query language
- 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

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

Relational Model

Schema: structure of the database = relations + constraints

Example Schema

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

customer			
id	name	street	city
192837465	Johnson	12 Alma	Palo Alto
019283746	Smith	4 North	Rye

account	
depositor	accountnr
19283465	101343
019283746	215569

Various types of constraints:

- data types, constrained data types (domains), nullability
- columns constraints (e.g. unique, counter, time stamp,. . .)
- check constraints (logical expression for domain integrity) (e.g. `age >= 18 AND age <= 150`)

Relational Model

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

Example Relational Database Instance

customer			
id	name	street	city
192837465	Johnson	12 Alma	Palo Alto
019283746	Smith	4 North	Rye
192837465	Johnson	3 Alma	Palo Alto
321123123	Jones	34 Main	Harisson
019283746	Smith	7 South	Rye

account	
depositor	accountnr
19283465	101343
019283746	215569
192837465	201541
321123123	217343
019283746	201762

- tuple record (row)

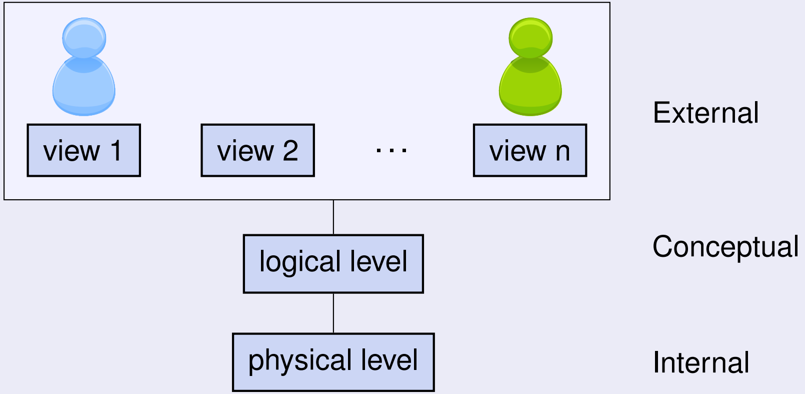
In the **pure relational model**, a table is a **set** of tuples:

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

View of Data

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

ANSI SPARC Architecture: 3 levels



ANSI SPARC Architecture: 3 levels

- **Physical level:** how a record (e.g. information about some product) is stored
  - disk pages, index structures, byte layout, record order
- **Logical level:** also called 'conceptual schema'
  - describes data stored in the database, and
  - relations among the data

SQL DDL (Data Definition Language)

```
CREATE TABLE SOLVED (STUDENT VARCHAR(40),
                     HOMEWORK NUMERIC(2),
                     POINTS NUMERIC(2));

CREATE VIEW SOLVED_HOMEWORK AS
    SELECT STUDENT, HOMEWORK FROM SOLVED;
```

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

Data Independence

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

- applications access the views, not the logical database

Physical data independence: 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
  - …

Declarative Query Language

Queries should:

- describe **what** information is sought
- **not** prescribe any particular method **how** to compute/retrieve the desired information

Kowalski

Algorithm = Logic + Control

Imperative/procedural languages:

- explicit control
- implicit logic

Declarative/non-procedural languages:

- implicit control
- explicit logic
- e.g. logic programming (Prolog), functional programming (Haskell), markup languages (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 POINTS
FROM SOLVED
WHERE STUDENT = 'Ann Smith'
  AND HOMEWORK = 3
```

- 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
- (usually) users do not need to think about efficiency

Motivation for Database Management Systems

Motivation for database management systems

- well-defined **data models** & **data integrity constraints**
  - entity-relationship models (E/R)
  - UML class diagrams
  - relational model
    - e.g. SQL table and constraint definitions
  - meta language for describing
    - data
    - data relationships
    - data semantics
    - data constraints

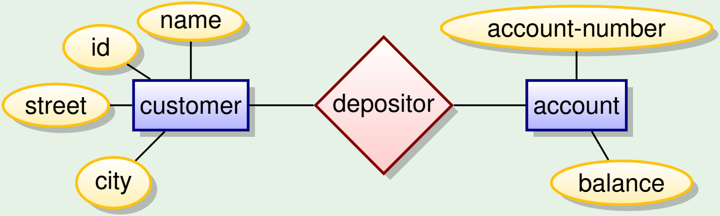
Other models:

- object-oriented models (e.g. ODL)
- semi-structured data models (DTD, XML Schema)
- . . .

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** associates customers with accounts



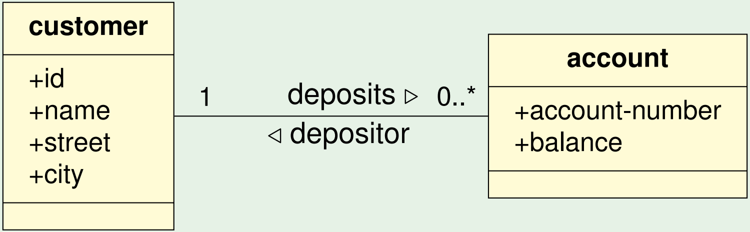
- widely used for database design
- usually converted to the relational model

UML Class Diagram

UML class diagrams

- frequently used in database design
- similar to E/R diagrams
  (**Entities/Relationships** $\Longrightarrow$ **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

Symmary

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 by much larger than main memory)
  - indexing, scalable algorithms
- **security**