CS-A1153

Databases

Course Introduction

June 4, 2020

Instructors:

Prof. Nitin Sawhney

Dr. Lukas Ahrenberg

TAs: Etna Lindy & Ville Vuorenmaa

Aalto University
School of Science

Department of Computer Science

Databases

Acknowledgements

These slides are based in part on presentation materials created by **Kerttu Pollari-Malmi** and **Juha Puustjärvi** in previous years and on the course text book: **A First Course in Database Systems**, Third Edition. Pearson by Jeffery D. Ullman and Jennifer Widom.

Thanks to Etna Lindy & Ville Vuorenmaa for translating prior lecture slides for this course.

Databases

Course Content & Learning Outcomes

Course Content

- The course covers basics of information management including the relational model, design principles of databases, and database theory.
- Basic concepts and methods in database systems. **Relational databases**: relational algebra, UML-design, functional dependencies, normalization, transactions, and SQL queries. One session on No-SQL databases.

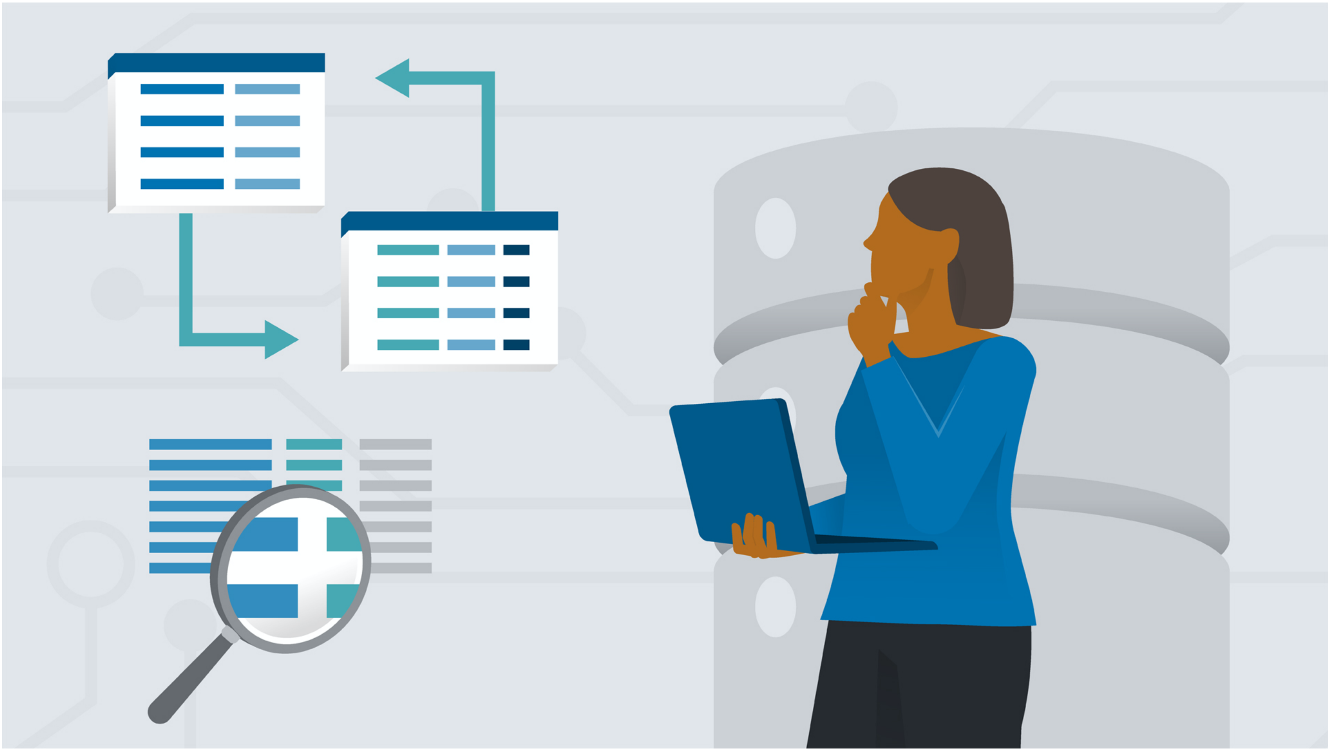
Learning Goals

- Understanding the role of conceptual modeling in managing data
- Learning the commonly used database modeling and querying languages
- Designing simple databases and writing queries
- Conduct a group project to design a database for managing university courses

Aalto University
School of Science

Databases

Why learn about Databases?



Aalto University
School of Science

Databases

Managing, organizing and acting on complex information:

- Airline reservations, car sharing, transit information, checking out library books
- Banking, healthcare, online shopping
- Mobile apps for news, social media, music sharing, fitness and messaging

Just about everything digital we consume and produce in our lives is based on interconnected information systems, often built using databases.

Databases

What are they and why useful?

A **Database** is a collection of related information that exists over a long period of time (persistent but changeable).

Database Management System (DBMS):

- Allow creating new databases and specifying their logical structure (***Schema***) using a *data-definition language*.
- Search for (***Query***) and modify the data using a *data-manipulation language*.
- Support storage of large amounts of data (terabytes or more) for long periods of time (***Persistence***), while enabling **efficient access** for data queries and updates.
- Ensure that the data stays intact and can be recovered (***Durability***) despite failures, errors or unintentional misuse.
- Control access to data from many user requests at once (***Concurrent use***), while ensuring they don't interrupt each other, preventing unexpected interactions (***Isolation***) or partial completion of actions (***Atomicity***).

Databases

What about file systems?

File systems can store large amounts of data over long periods of time.

- No guarantee that the data will be preserved if it is not backed up (particularly in each state of change over time).
- Don't support efficient access to data whose location in a particular file is not known.
- File systems do not support high-level query languages.
- No support for data schemas – limited to hierarchical directories.
- File systems do not support many concurrent users.

Hence, unlike Databases File systems don't support Schemas, Queries, efficient access, Durability, or Concurrent use.

Databases

Current Trends

- Connecting object-oriented programming to relational databases (User writes object-oriented program, and program transforms parts of its commands to database queries.)
- Lightweight databases that can run on PCs and Mobile devices
- Larger systems with a subset of features of relational databases and SQL in order to create efficient databases for certain functions.
- Processing of structured documents (XML, JSON) for web-based information access
- Data Warehouses: Collection and aggregation of legacy databases and heterogeneous data from different sources.
- NoSQL and Object-oriented databases (OODBMS)

Databases

Components of DBMS

- **Database** (data and metadata):
  - Actual information (data) and description of data structure (metadata).
- **Storage Manager**:
  - Takes care of data transmission between RAM and database e.g. located in hard drive
  - File manager
  - Buffer manager
- **Query processor**:
  - Finds as efficient way as possible to perform queries and updates.
- **Transaction manager**:
  - Controls concurrent user operations (concurrency control) and ensures atomicity of updates.

Databases

Components of DBMS

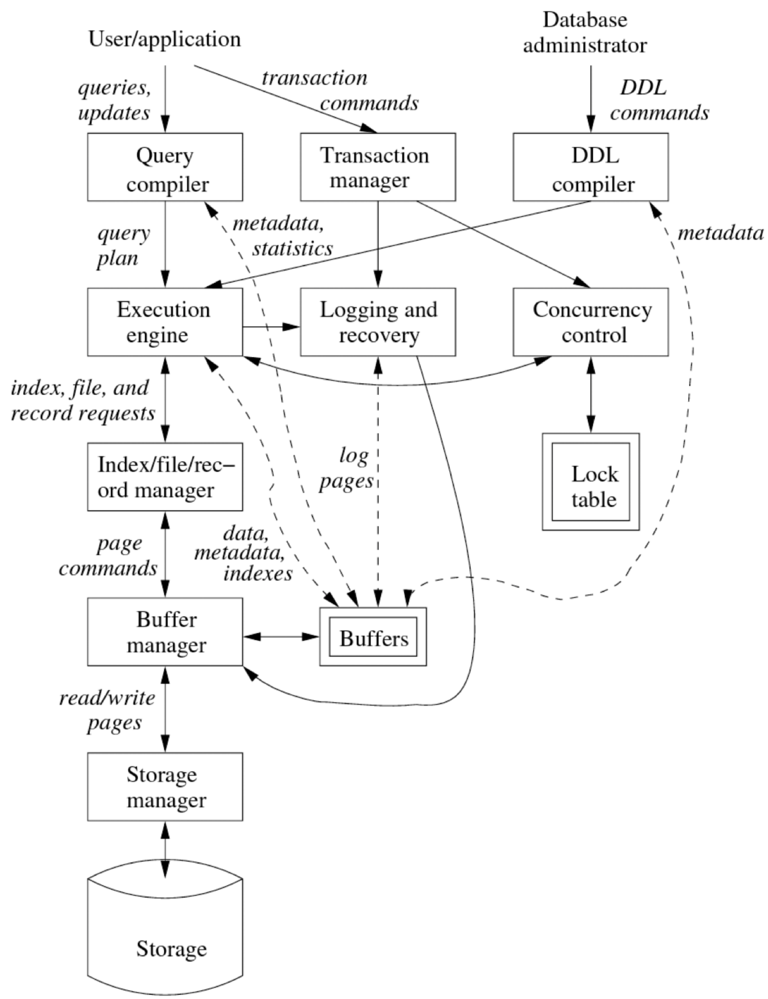


Figure 1: Database management system components

Aalto University
School of Science

Databases

Components of DBMS

> Storage Manager

Storage Manager

- For reasons of efficiency database management systems control the use of external memory (e.g. hard disk) themselves instead of using services provided by the operating system.

Components of Storage Manager:

File manager

- Takes care of the placement of files in the external memory as well as block transfers between external memory and RAM.

Buffer manager

- Takes care of RAM management
- Investigates which disk page of RAM block received from the file manager will be placed.
- If necessary, asks file manager to write pages back to external memory.

Databases

Components of DBMS

> Query Processor

Query Processor

- Translates queries performed in a high-level language (e.g. SQL) to plan and execute the query.
- Often the query can be performed in many different orders. Query processor is responsible for finding the most efficient execution order.
- Query processor also optimises single operations related to the query, e.g. using indexes when possible.

Databases

Data Models

A **data model** is a notation describing data or information.

Descriptions of data models consist of:

- **Structure** of the data
- **Operations** on the data
- **Constraints** on the data

Prominent data models used today:

- **Relational data model**, with object-relational extensions.
- **Semi-structured data model**, including XML and data standards

Databases

Relational Model

- A database consists of two-dimensional tables which are called **relations**.
- Every relation has a group of named **attributes** (columns).
- The name of the relation and set of attributes is called a **schema** for that relation e.g. Movie (title, year, length, genre).
- Each row of the table (**tuple**) has values for different attributes.
- Values of attributes have to be **atomic** (e.g. single numerical value or string, not group or tuple). Values can have types.

THE RELATIONAL MODEL OF DATA

title	*year*	*length*	*genre*
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

Figure 1: An example relation

Aalto University
School of Science

Databases

Semi-structured Model

```
<Movies>
    <Movie title="Gone With the Wind">
        <Year>1939</Year>
        <Length>231</Length>
        <Genre>drama</Genre>
    </Movie>
    <Movie title="Star Wars">
        <Year>1977</Year>
        <Length>124</Length>
        <Genre>sciFi</Genre>
    </Movie>
    <Movie title="Wayne's World">
        <Year>1992</Year>
        <Length>95</Length>
        <Genre>comedy</Genre>
    </Movie>
</Movies>
```

Figure 2: Movie data as XML

Databases

Relational Model

- Relations are sets of tuples, not lists of tuples.
- Attributes can be presented in arbitrary order.
- ***Relation schema*** defines what attributes belongs to relation and what is the type of attributes.
- E.g. Movies (title: string, year: integer, length: integer, genre: string)
- ***Instance*** of relation means a set of tuples for a given relation at a certain time.

THE RELATIONAL MODEL OF DATA

title	*year*	*length*	*genre*
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

Figure 1: An example relation

Databases

Key of Relation

Set of Attributes forming a constraint on the relation

```
Movies(
    title:string,
    year:integer,
    length:integer,
    genre:string,
    studioName:string,
    producerC#:integer
)
MovieStar(
    name:string,
    address:string,
    gender:char,
    birthdate:date
)
StarsIn(
    movieTitle:string,
    movieYear:integer,
    starName:string
)
MovieExec(
    name:string,
    address:string,
    cert#:integer,
    netWorth:integer
)
Studio(
    name:string,
    address:string,
    presC#:integer
)
```

Figure 5: Example database schema about movies

Aalto University
School of Science

Databases

Course Requirements

Course Requirements:

- **Exam** (obligatory)
- **Project** (obligatory): with score of minimum of 30/40, you get +1 bonus to your exam grade, however you have to pass the exam without the bonus to pass the course
- **Exercises** (voluntary, 0–6 extra points for a passed exam)
- **Course feedback** (voluntary, 20 extra points for exercises, with maximum of 260 points, with all extra points taken into account.)
- **Course text book:** Ullman, Widom: A First Course in Database Systems, 3rd edition or New International Edition.