Introducción a las Bases de Datos

CI-0127 Bases de Datos, Universidad de Costa Rica

Sivana Hamer

Importante: Este documento recopila contenidos de diversos de sitios web especializados, académicos y documentos compartidos por universidades. Toda la información es utilizada con fines estrictamente académicos. Para más información, puede ver las referencias.

1 Data

Data are known or assumed facts. Nowadays, we generate and use data during our day to day. For example, while I am messaging someone with my phone lots of data is generated and consumed. Somewhere in my phone, there is a phone address that is accessed. Furthermore, if I'm using a messaging application there are servers with IP addresses that must be retrieved to send the message. The message itself also contains data, the message, who sent it, from where and when. Depending on the application other data related to the message may also be possible to recollect such as reactions (thumbs up), replies or who has seen it.

Having access to data has real world value. Companies build their entire business models by the information they store such as social networks. Furthermore, there are institutions that are in charge of creating and maintaining databases such as the national register of citizens or banks. These companies have an interleaved role within our society, having rights to decide who can vote, what they can buy or what products will they be aware of through advertisement. Some companies even lose their reputation by not managing data correctly. Thus, data is quite the commodity.

Throughout our history, we have always generated, used and preserved data. Due to the usefulness of computers we have migrated data stored in other mediums to a digital format. Specifically, data is stored in *databases* (DB) that are collections of data.

2 File databases

The first computer databases are assumed to be files. Databases can be stored in files such as a CSV. For example, if we define a database for students enrollment to courses. One file can store the students enrolled in the University with their ID, name and date of birth. In another file we can have all the courses that they have taken by storing a student, course and semester. We could even have another file with all the information about the courses. Though these databases are still common today, there are several disadvantages:

- **Data integrity:** Changes might cause errors in the integrity of the data. How can we ensure that the student name is the same for the enrollment data? How can we ensure that the semester is valid? What happens if a course changes their name?
- **Data access:** As the data has a certain value for the institution, they might want to retrieve certain information. How can we find all the years of birth of students who have taken a course?
- **Data concurrency:** If there are several people trying to write to the file, we have to deal with concurrent changes. Do we allow concurrent changes? How do we choose which value to store? How do we reduce data anomalies?

- **Data durability:** Our file has to be stored in a computer somewhere. How do we manage the data if there is a crash during changes? How do we protect the computer from breaking?
- **Data security:** As our files are stored in a computer, someone without access might try to gather data. How can we ensure that the data is protected?
- **Data descriptiveness:** Let us assume that someone new is managing the file database, without training. How descriptive is the data? How descriptive are the data conventions? How descriptive are the data relationships?

3 Database Management Systems

Due to the previous disadvantages, Database Management Systems are widely used. A *Database Management System* (DBMS) is the software that manages a database. The goal of DBMS is to conveniently and efficiently store and retrieve data. These systems help:

- **Defining:** Stores the *meta-data* of the data in a *database catalog* or *dictionary*.
- **Constructing:** In charge of the process of storing the data.
- **Manipulating:** Allows functions to Create, Read, Update and Delete (CRUD) data.
- **Sharing:** Allows concurrent use of the database preserving integrity defined by certain rules.
- **Protecting:** Protects both from security threats and software malfunctions.
- **Maintaining:** Allows the database to evolve.

4 Database Systems

A *database system* is a database (collection of data) + DBMS (software that manages the data). Fig. 1 shows an example of a database system.

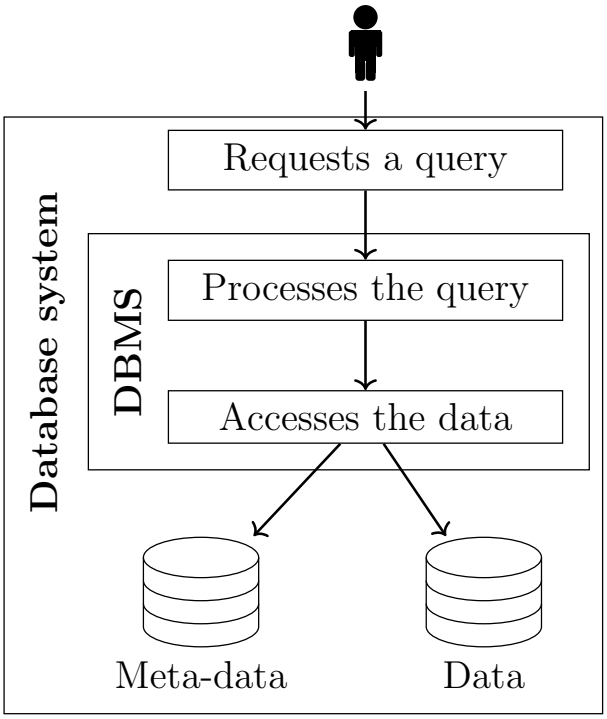

Figure 1: Example of a database system

- A *query* is any interaction with the database. For example, retrieving all the students or adding a new enrollment.
- Someone interacts with the database requesting a query. For example, gathering all the students currently enrolled in a course or adding a new student. People can include *database administrators* (DBA) who are in charge of administering the database system, *programmers* that develop apps that access the database system, and *final users* who use an application that accesses the database system.
- This query is processed by the *query processor* that efficiently gathers the data.
- The data items related to the query are accessed and gathered from physical storage with a *storage manager*.
- The *meta-data* is data about the data (data types, structures and constraints). For example, allowing that year to contain only integers. As the meta-data is a collection of data about data (quite recursive) it is also a database (as seen in the Figure).
- The data represents the *stored data*. For example, a student entry with ID "123456789", name "Ana Arias" and year of birth "1991".
- A database *schema* describes the database design as in the the meta-data of the database.
- The *database state, snapshot or instance* is the actual current data in the database. For example, it would be all the students in the database at a moment in time.

5 Data abstraction

Database systems started out by combining both the physical storage of the data and the models to describe the data. However, this leads to maintenance issues leading to data model changes requiring updates for the DBMS physical storage. Furthermore, users needed to understand the physical storage of the data to interact with the database system.

Therefore, *data abstraction* has been defined to hide the finer grained details of the implementation based on the needs of the user, with database systems separating the abstraction into three different *abstraction levels*. There is an *independence* between levels as their relationship is *mapped* between each other. The levels from lowest the lowest degree of abstraction to highest are:

- **Physical level:** Describes *how* the data is stored. Specifically, in this level the block of bytes that are stored are abstracted from the user.
- **Conceptual or logical level:** Defines *what* data is stored conceptually and *what* is their relationship through *data models* (Section 6). Though there is a relationship between the data and how it is physically stored, there is a *physical level of independence* as the details for the mapping are abstracted from the user.
- **External or view level:** Provides an interaction with the database for different users. There are different *views* defined for each user, limiting access to certain parts of the database. There is a *logical level of independence* as we can change what data we are accessing without having to change the logical level of the DBMS.

Though the data can be abstracted into higher or lower levels, the data only exists in the lowest abstraction level (physical level). A summary of all the levels with their relationships is shown in Fig. 2.

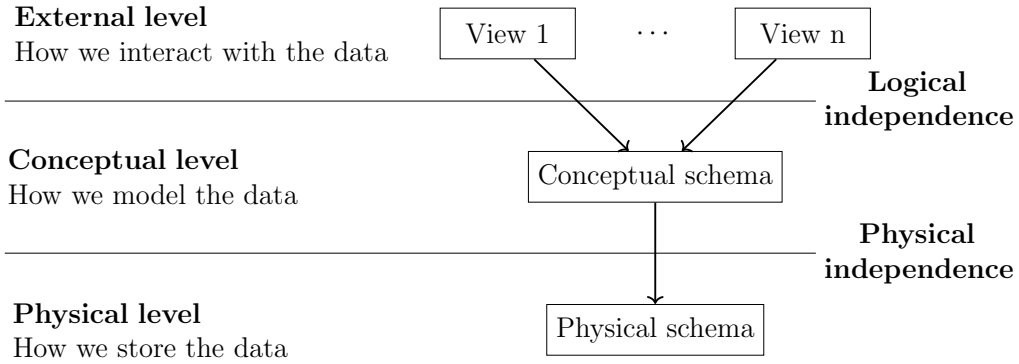

Figure 2: Levels of abstraction

6 Data models

A *data model* is a collection of concepts describing the database structure (data types, relationships and constraints) with a based set of operations to interact with the database. Data models can be categorized as:

- **High-level or conceptual:** Details how the users perceive the data. For example, the entity-relationship model.
- **Representationtal or implementation:** Describes an intermidiate model between how users perceive the data and how it is stored. For example, the relational, network, hierarchical and object-data models.
- **Low-level or physical:** Describes how files are stored in the computer.
- **Self-describing:** Combines bothe description of the data (schema) with the data values (state). For example, NoSQL or key-value.

Depending on the selected data model and/or the DBMS used, the database must be *designed*.

7 Database languages

To interact with the database we require specific languages. Though these languages can be categorized into different types, in practice they both form part of the same language. An example of a database language is SQL.

Data definition languages

Data definition language (DDL) allows us to define the database schema (i.e., meta-data). Specifically, DDL allows us to define the *conceptual* schema. DDL commands for SQL include CREATE TABLE, CREATE INDEX, ALTER TABLE, ALTER INDEX, DROP TABLE and DROP INDEX. An example of the command to define a student TABLE is the following:

```
CREATE TABLE STUDENT (
    ID CHAR(6) NOT NULL,
    NAME VARCHAR(30) NOT NULL,
    BYEAR DATE
);
```

Other schema defining variations of DDL include *Storage Definition Language* (SDL) to specify the internal schema and *View Definition Language* (VDL) to define the views with their mappings. There

are no SDL systems for most relational DBMS and most DBMS use DDL to define the views. In SQL, one can define a view with CREATE VIEW or DROP VIEW. An example for a view in SQL that selects all the names of the students born in 1991 is the following:

```
CREATE VIEW STUDENTNAME (NAM)
AS SELECT NAME
FROM STUDENT
WHERE BYEAR = 1991;
```

Data manipulation language

Data manipulation language (DML) allows us to interact by retrieving, inserting, deleting and modifying the database. DML commands for SQL include SELECT, UPDATE, INSERT and DELETE. An example in that selects all the names of the students born in 1991 sis the following:

```
SELECT NAME
FROM STUDENT
WHERE BYEAR = 1991;
```

There are two types of DML:

- **Low-level or procedural:** Defines the data needed with *how* to get the data. This type is related to relational algebra.
- **High-level, declarative or non-procedural:** Defines the data needed without describing how to get the data. For example, SQL is of this type. This type is related to relational calculus.

Data control language

Data control language (DCL) defines who and what can access the database. In SQL, some DCL commands are GRANT and REVOKE.

8 DBMS architectures

Database systems can have different architectures such as centralized, client-server and n-tiered. The current most relevant architectures are tiered, using the database as the backend. The architecture can have multiple tiers:

- **Two-tier:** The application is in the client machine while the database resides in a server machine. The interaction between tiers is achieved with an application programming interface (API) that allows clients to call the DBMS. The standard API for DBMS is called Open Database Connectivity (ODBC).
- **Three-tiers:** The client machine acts as the front-end with an intermediate application server that stores the business logic. The application server thus interacts with the database server to retrieve the queries. Examples of these systems include web browsers or mobile applications.
- **n-tier:** One can define a variable amount of n tiers. Usually, the intermediate application server can be divided into different responsibilities to better distribute load.

The visual structure of two and three tier architecture is shown in Fig. 3.

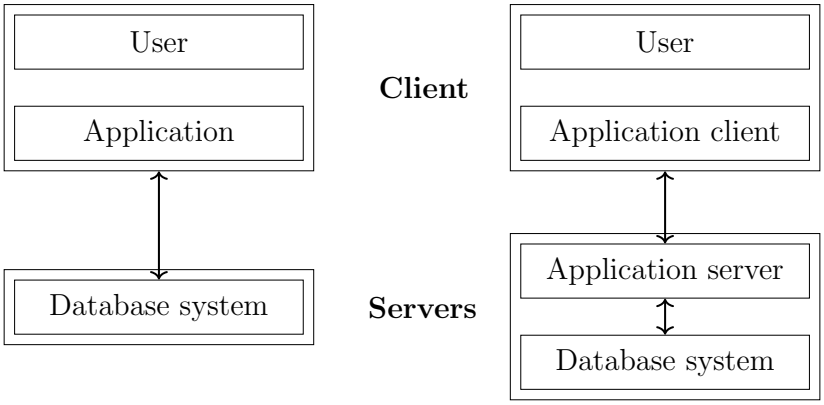

Figure 3: Two (left) and three (right) tier DBMS architecture

9 History

1950s-early 1960s. Data was moved to computer data storage such as punch cards and tapes. There were several limitations with how the data could be read (sequentially) with some main memory being smaller than the data size.

late 1960s-early 1970s. Computers sequence limitations were lifted (no longer sequential). Relational data model began to be explored as they were defined in the 1970s by Edgar Codd.

late 1970s-1980s. DBMS started to use relational models as it became as efficient as other databases (hierarchical and network) with System R and easier to use than other databases. Initial research started for object oriented databases and distributed systems.

1990s. Database vendors added parallel capabilities and object oriented databases. The advent of the internet changed databases to be more query focused and process large amounts of data.

2000s. Advent of semi-structured data such as XML and JSON. Furthermore, several open-source database systems were developed. Due to the large amount of data, data mining became of interest and companies started to pilot NoSQL systems for data-intensive applications.

2010s. NoSQL became adopted for some systems (social networks) and had capabilities improved for users who interacted with the database. Data has been outsourced into cloud services. Concerns with users' privacy and security of the data have become hot topics.

References

[1] R. Elmasri and S. Navathe, *Fundamentals of database systems*, 7th ed. Pearson, 2016, chapters 1 and 2.

[2] A. Crotty and L. Ma. Lecture #1. [Online]. Available: https://15445.courses.cs.cmu.edu/fall2021/schedule.html

[3] C. Faloutsos and A. Pavlo. Lecture #1. [Online]. Available: https://15415.courses.cs.cmu.edu/fall2016/

[4] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY: McGraw-Hill, 2020, chapter 1.

