HyperionDev

Databases

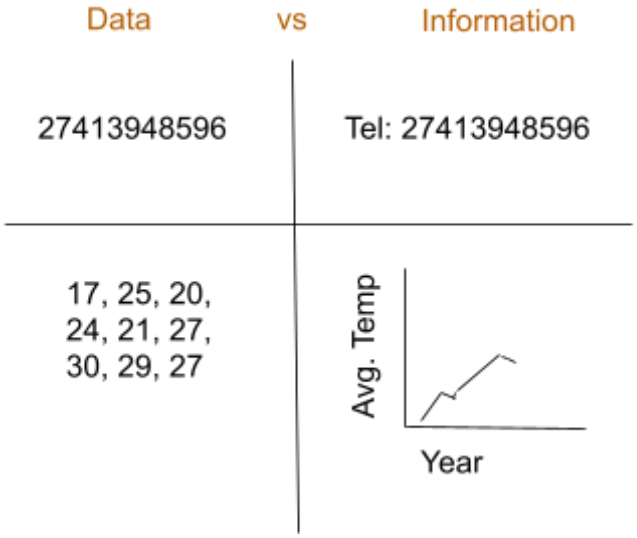
Task

Visit our website

Introduction

In this task, we discuss the different types of databases and database management systems.

In order to properly understand databases, you must first understand the difference between data and information. Simply put, data are raw facts (note that the word data is plural – we say “data are”, not “data is”). Raw data has not yet been processed to reveal its meaning. Information, on the other hand, is the result of processing raw data to reveal its meaning. Data processing refers to a whole spectrum of activities: it can be as simple as organising data to identify patterns in it, or as complex as using statistical modelling to draw inferences from your dataset.



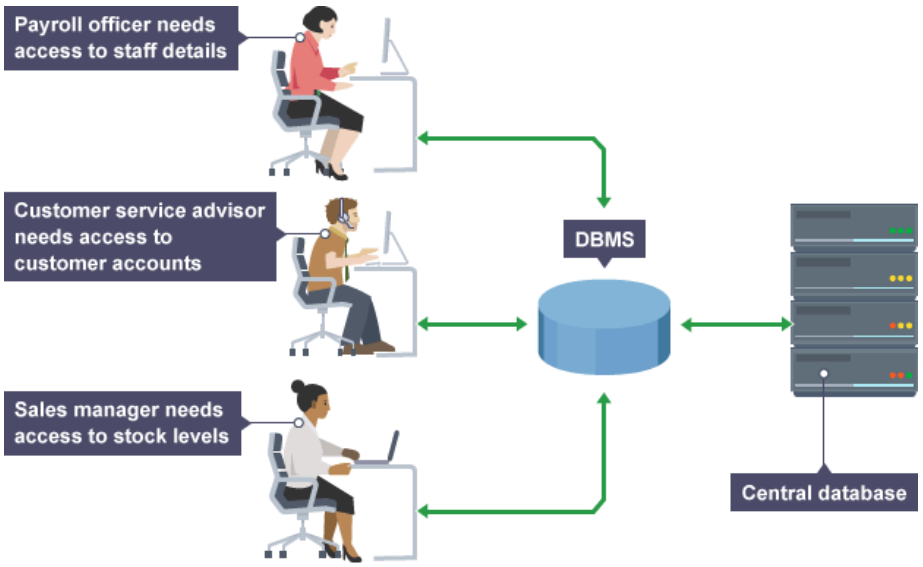
From the above, it should be clear that decision-making relies on information, not just raw data. However, useful information requires accurate data, which must be generated properly and stored in a format that is easy to access and process. Just like any other basic resource, the data environment should be carefully managed. We will learn more about this in this task.

Database management systems

A computer database is used to efficiently store and manage data. Think of a database as a well-organised electronic filing cabinet that can store, order, and manipulate data while providing easy access to the user. Databases are shared, integrated computer structures that store a collection of end-user data, the raw facts of interest, metadata, descriptions of the data characteristics, and relationships that link data variables together.

Databases are powered by software, known as a **database management system (DBMS)**, which helps manage the contents of the cabinet. A DBMS is a collection of

programs, which works to manage the database structure and control access to the data stored in the database.



The DBMS manages the interaction between the end user and the database (bbc.co.uk)

The DBMS serves as an intermediary between the user and the database. It receives all application requests and translates them into the complex operations required to fulfil those requests.

Much of the database's internal complexity is hidden from the application programs and end-users by the DBMS. There are some very important advantages to having a DBMS between the end-user application and the database. Firstly, the DBMS allows the data in the database to be shared among multiple applications or users. Secondly, the DBMS integrates many different users' views of the data into a single data repository.

The DBMS helps make data management much more efficient and effective. Some of the advantages of a DBMS (Tutorial Link, n.d.) include:

- **Better data sharing:** by managing the database and controlling access to data within it, the DBMS makes it possible for end-users to have more efficient access to data that is better managed.
- **Improved data integration:** by facilitating a variety of end-users to access data in a well-managed manner, the DBMS helps provide a clearer and more integrated view of the organisation's operations to the end-users.
- **Minimised data inconsistency:** data inconsistency occurs when different versions of the same data appear in different places. A properly designed database greatly reduces the probability of data inconsistency through data normalisation, which reduces redundancy and increases data integrity.

- **Improved data access**: a query is a specific request for data manipulation (e.g. to read or update the data) sent to the DBMS. The DBMS makes it possible to produce quick answers to spur-of-the-moment queries.
- **Improved decision-making:** better quality information (on which decisions are made) is generated due to better-managed data and improved data access.
- **Increased end-user productivity:** the availability of data and the ability to transform data into usable information encourages end-users to make quicker and more informed decisions.

Types of databases

There are many different types of databases. These can be classified according to the number of users supported, where the data are located, the type of data stored, the intended data usage, and the degree to which the data are structured. Rob, Coronel, and Crockett (2008) describe various categories of databases as follows:

Users

A database can be classified as either **single user** or **multi user**. A database that only supports one user at a time is known as a single-user database. With a single-user database, if user A is using the database, users B and C must wait until user A is done. A desktop database is a single-user database that runs on a personal computer. A multi-user database, on the other hand, supports multiple users at the same time. A workgroup database is a multi-user database that supports a relatively small number of users (usually less than 50) or a specific department within an organisation. When a multi-user database supports many users (more than 50) or is used by the entire organisation, across many departments, it is known as an enterprise database.

Location

A database can also be classified based on location. A **centralised database** supports data located at a single site, while a **distributed database** supports data distributed across several different sites.

Purpose

A popular way of classifying databases is based on how they will be used and based on the time sensitivity of the information gathered from them. An example of this is an **operational database**, which is designed to primarily support a company's day-to-day

operations. Operational databases are also known as online transaction processing (OLTP), transactional, or production databases.

Structure

The degree to which data is structured is another way of classifying databases. Data, in its original, or raw, state is known as **unstructured data**. In other words, they are in the format in which they were collected. **Structured** data results from formatting unstructured data to facilitate storage and generate information. You apply structure based on the type of processing that you intend to perform on the data. For instance, imagine that you have a stack of printed invoices. If you just want to store these invoices so that you're able to retrieve or display them later, you can scan and save them in a graphical format. However, if you want to derive information from them, such as monthly totals or average sales, having the invoices in a graphical format will not be useful. You could, instead, store the invoice data in a structured spreadsheet or relational table format so that you can perform the desired computations.

Examples

Analytical databases focus on storing historical data and business metrics used exclusively for tactical or strategic decision-making. They typically comprise of two components: a data warehouse and an online analytical processing (OLAP) front end. Analytical databases allow the end-user to perform advanced data analysis of business data using sophisticated tools. A data warehouse, on the other hand, focuses on storing data used to generate the information required to make tactical or strategic decisions.

A very common type of database you may have heard of is the **relational database.** A relational database is structured to recognise relations between stored items of information. To put it more simply, a relational database organises data into tables and links them based on defined relationships. These relationships then enable you to retrieve and combine data from one or more tables with a single query. We will learn more about relational databases in the next task.

A **NoSQL** database is a new-generation database management system that is not based on the traditional relational database model. Believe it or not, you are using a NoSQL database every time you search for a product on Amazon, watch a video on YouTube, or send a message to a friend on Facebook.

NoSQL databases generally have the following characteristics:

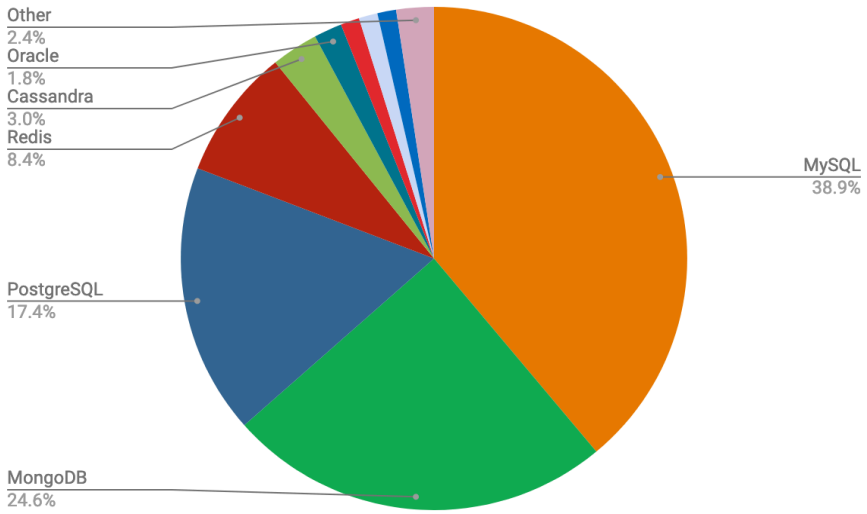
- They are not based on the relational model.
- They support distributed database architectures.
- They provide high scalability, high availability, and fault tolerance.

- They support very large amounts of sparse data.
- They are geared toward performance rather than transaction consistency.

You can find out more about NoSQL databases **here** if you are interested in reading further.

Popular DBMSs

Some popular DBMSs today are:



Pie Chart (ScaleGrid, 2019)

- **MySQL**: this popular open-source relational database management system (RDBMS) is named after "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for structured query language.
- **MongoDB**: this is a NoSQL database. It is defined as “an open-source database used by companies of all sizes, across all industries, and for a wide variety of applications.” It is “an agile database that uses a flexible document data model so schemas can change quickly as applications evolve, [and] provides functionality developers expect from traditional databases, such as secondary indexes, a full query language, and strict consistency” (MongoDB, 2017).
- **PostgreSQL**: this is defined as, “a powerful, open-source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance” (PostgreSQL, n.d.).
- **Oracle RDBMS:** this was the first commercially usable RDBMS launched in the market and is the leader in terms of worldwide RDBMS software revenue (Rob, Coronel, and Crockett, 2008).

SQL database basic file terminology

Specialised vocabulary is required for the description of databases. We look at some of these terms as described by Rob, Coronel, and Crockett (2008) below:

- **Data**: data are raw facts, such as a telephone number, customer name, or birth date. Unless they have been organised in some logical manner, data have little meaning. A single character is the smallest piece of data that can be recognised by a computer. It requires only one byte of computer storage.
- **Field**: a field is a character or group of characters that has a specific meaning. It is used to define and store data.
- **Record**: a record is a logically connected set of one or more fields that describes a person, place, or thing.
- **File**: a file is a collection of related records.

For example, below is a `CUSTOMER` file:

C_NAME	C_PHONE	C_ADDRESS	C_POSTCODE	A_NAME	A_PHONE
Alfred Smith	082 345 2341	207 Willow St, Port Elizabeth	6390	Leah Hahn	084 259 2073
Kathy Dunne	083 567 9012	556 Bad St, Cape Town	7100	Alex Alby	085 785 3938
Paul Farris	076 782 1232	2148 High St, Benoni	1522	Leah Hahn	084 259 2073

Table adapted from Rob, Coronel, and Crockett (2008)

The `CUSTOMER` file contains three **records**. Each record is composed of six **fields**, four containing customer data denoted as "`C`", and two containing the data of agents working with these customers, denoted as "`A`". The **fields** are: `C_NAME`, `C_PHONE`, `C_ADDRESS`, `C_POSTCODE`, `A_NAME`, and `A_PHONE`. The three records are stored in a file that is named `CUSTOMER`, as it contains customer **data**.

Working with SQL

Let's get started by installing and using a DBMS - SQLite. This management system works entirely from command-line and isn't wrapped in any syntactic complexities.

Let's start by installing SQLite:

- **Windows:**
  - **Download the precompiled binaries as a zip file.**
  - Copy the .exe file to your current directory (where you create files).

- **Linux:**

```
> sudo apt install sqlite3
```

- **Mac:**
  - MacOS comes bundled with SQLite. There is no need to install it.

Now that you have SQLite installed, let's start working with a database. While SQLite *can* accept SQL commands directly from command-line, this just isn't a very user-friendly solution. Instead, we will ask SQLite to run code from ***.sql** files that we will make. To do this, we must first open up SQLite, and provide it with a database file name (this can be any file name, as long as it ends with **.db**, it will be recognized as a database). To do this:

```
> sqlite3 my_database.db
```

This opens up the SQLite management programme, and all database commands are done on **my_database.db**. Once you are in SQLite, you can run your script with:

```
sqlite> .read my_script.sql
```

The file **my_script.sql** will contain the SQL statements that you would like to run.

Different types of statements

SQL provides a powerful language with which to manipulate and view data. The four main types of SQL queries on a table in a database are:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

SQL is meant to be human-readable (almost similar to English), so these statements do exactly as expected: `SELECT` will show specific data, `INSERT` will add data to a table, `UPDATE` will change existing values in a table and `DELETE` will delete rows in a table.

In Data Science, knowing how to query data appropriately is of utmost importance. This is where we will start - so we will get to know the `SELECT` statement more in-depth.

The select statement

The SELECT statement follows a general format:

```
SELECT column_1, column_2
FROM table_name
WHERE condition
```

How does this translate to English? Well, it's basically asking to select rows `column_1` and `column_2` from a table called `table_name`. These rows should match the specified condition.

There are some tricks that the more lazy SQL programmers can use. Using wildcards can save a lot of time. For example, the * wildcard means 'everything'. For example:

```
SELECT *
FROM table_name
WHERE condition
```

does the exact same thing as the previous `SELECT` statement, but selects all columns.

Your first database

Included in this task folder is a file called **create_table.sql** and one called **view_table.sql**. **create_table.sql** is a script that uses a `CREATE TABLE` statement (not yet covered, so don't worry about it) and an `INSERT` statement (also not yet covered) to create a table called Student in a database and populate it with values.

view_table.sql is a script that simply runs a `SELECT` statement to view everything in the table.

To set up your database, run the following commands:

```
> sqlite3 School.db
```

```
sqlite> .read create_table.sql
sqlite> .read view_table.sql
```

The Student table is as follows:

- `student_id`: a 13 character string for the primary key of the student
- `first_name`: the first name of the student
- last_name: the last name of the student
- `email`: the email address of the student
- `stu_subject`: the subject that the student is taking. This can either be “Data Science”, “Software Engineering”, “Web Dev”, or “Databases”.
- `mark`: the mark achieved by the student in their subject.

Extra resource

If you’d like to know more about database design, we highly recommend reading the book ***Database Design – 2nd Edition*** by Adrienne Watt.

Practical Task 1

Create a word document called MyDatabaseTheory and answer the following questions:

1. Define each of the following terms:
    a) Data
    b) Field
    c) Record
    d) File

2. Given the file below, answer the following questions:

    e) How many records does the file contain?
    f) How many fields are there per record?

PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5U	Holly Parker	33-5-592000506	180 Boulevard du General, Paris, 64700	$13179975.00
21-7Y	Jane Grant	0181-898-9909	218 Clark Blvd, London, NW3, TRY	$45423415.00
25-9T	George Dorts	0181-227-1245	124 River Dr, London, N6, 7YU	$78287312.00
29-7P	Holly Parker	33-5-592000506	180 Boulevard du General, Paris, 64700	$20883467.00

Convert your document to pdf and submit it to your task folder.

Practical Task 2

Use SQLite to set up a database file called **School.db**. Run the **create_table.sql** script to set up a table in this file.

Create the following script files to run the accompanying queries:

- **failed.sql** - List the *first name*, *surname* and *email addresses* of all failing students (a fail is a mark of 44 or less)
- **distinction.sql** - List the *first name* and *surname* of all students with a mark of 80 or above
- **near_distinction.sql** - List the *email addresses* of all students with a mark from 70 to 79 (inclusive).
- **javascript_junkies.sql** - List *all attributes* of all students doing Web Dev.

Share your thoughts

HyperionDev strives to provide internationally excellent course content that helps you achieve your learning outcomes.

Do you think we've done a good job or do you think the content of this task, or this course as a whole, can be improved?

Share your thoughts anonymously using this **form**.

Reference list

MongoDB. (2017). MongoDB Overview. Retrieved May 14, 2019, from MongoDB Datasheet: **https://s3.amazonaws.com/info-mongodb-com/MongoDB_Datasheet.pdf**

PostgreSQL. (n.d.). PostgreSQL: The World's Most Advanced Open Source Relational Database. Retrieved from postgresql.org: **https://www.postgresql.org/**

Redis. (n.d.). Redis. Retrieved May 14, 2019, from redis.io: **https://redis.io/**

Rob, P., Coronel, C., & Crockett, K. (2008). Database Systems: Design, Implementation and Management. London: Cengage Learning EMEA, 2008.

Sale Grid. (2019, March 4). 2019 Database Trends - SQL vs. No SQL, Top Databases, Single vs. Multiple Database Use. Retrieved May 14, 2019, from scalegrid.io: **https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/**

Tutorial Link. (n.d.). Advantage and Disadvantages of DBMS. Retrieved from tutorialink.com: **https://tutorialink.com/dbms/advantage-and-disadvantages-of-dbms.dbms**