LESSON

Introduction to Databases

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO DATABASES LESSON!

In this lesson, we discuss databases, their history and evolution, the different types of databases, and the Database Management System (DBMS). We also discuss the specialised concepts and terminology necessary to understand databases. At the end there is a task for you to put into practice what you have learned.

INTENDED LEARNING OUTCOMES

Metacognition is a word that describes thinking about our own thinking. Developing your ability to do this - specifically to identify and think about the approaches and processes you use to plan, monitor, implement, and assess your own programming - will create a “virtuous circle” in which you accelerate your own development of expertise.

In this task, you can begin to build your metacognitive capacity to see the bigger picture in your coding journey, and to self-assess your coding proficiency, by starting with a high-level overview of your own learning objectives. As you go through the task, refer back to these objectives to provide a thought scaffold for what you are learning and how the various aspects relate to the bigger picture of Databases.

You will know that you have succeeded at this task if, by the time you complete it, you are able to:

- Describe data management issues and how they are addressed by a DBMS.
- Describe commonly implemented features of commercial DBMS’s.
- Describe different types of DBMS’s.
- Review DBMS end-user tools.

Refer to the self-assessment table provided after the programming task near the end of the document for a checklist to assess your own degree of achievement of each of these learning outcomes. Try to gauge your own learning and address any weak areas before submitting your task, which will become part of your formative assessment portfolio.

SECTION 1: UNDERSTANDING COMPUTER DATABASE MANAGEMENT SYSTEMS

CONCEPT: DATA VS INFORMATION

We use databases to store, manage, and manipulate data. But - what is data? And how does data differ from information? First of all, Imagine having a text file that looks like this:

```
Tripoli34Belgrade36Peshawar42Cordoba17Sydney22Pyongyang25Houston28Xi’an1
9Tehran23
```

This is what data looks like before it becomes information. Data describes raw pieces of input with no context or meaning. Information, on the other hand, is processed and formatted data. It is structured in a logical way that gives it meaning, and can be used to find patterns in the data, or even be able to make predictions based on trends detected in data analysis. In order to do this, though, the data itself needs to be accurate and structured in a way that makes it possible to process logically and correctly. For example, if we were to format the data from the example above, it would be much easier to work with, and would begin to provide more obvious meaning:

```
Tripoli 34
Belgrade 36
Peshawar 42
Cordoba 17
Sydney 22
Pyongyang 25
Houston 28
Xi’an 19
Tehran 23
```

The format above makes it much easier to create information from the data. We can now see that we are looking at cities. If we knew the numbers were maximum temperatures in Celsius over the year 2020, we could process the data to make a table of information:

City	Maximum Temperatures 2020 (Celsius)
Tripoli	34
Belgrade	36

Peshawar	42
Cordoba	17
Sydney	22
Pyongyang	25
Houston	28
Xi’an	19
Tehran	23

Now that the information can be easily understood, we can start to pick up patterns about which cities around the world were hottest on a particular recorded day. We could also expand this table with more information, such as the maximum yearly temperatures of each city each year for the last 50 years. That information could then be used to determine patterns such as whether the peak temperatures are rising over the years, which could be an indication of global climate change, and would be relevant for long term agricultural planning, for example. This could lead to decisions being made about budget and resource distributions in the area. The ultimate aim of logically structuring data to make accessible information is to be able to see patterns more easily, and to make strategic decisions based on extrapolated information acquired from the data.

CONCEPT: THE DATABASE

A computer database is used to efficiently store and manage data. Think of a database as a well organised electronic filing cabinet that has the ability to store, order and manipulate data while also 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, the descriptions of the data characteristics and relationships that link data variables together.

Put simply, a database is a tool that helps us to turn data into information. In the example in the previous section, we used a table to do this, which made the information easy to access and process. Databases store both the processed information and metadata. Some database structures use tables to do this, similar to the example we created above; others use different structures. Metadata is data about the data itself.

Continuing with our example temperature table, metadata would be information like: only a number can be input as a temperature, no element can be left empty but a temperature can be zero, etc.

This lesson addresses Unit Standard 114049: Demonstrate an understanding of Computer Database Management Systems.

Specific Outcomes:

1. **SO1: Describe data management issues and how it is addressed by a DBMS. (Range: Access, shared use, security, integrity, privacy, reliability, risk, performance, integration, data administration.)**
2. **SO2: Describe commonly implemented features of commercial database management systems. (Range: Data access tools, recovery, audit, distributed data management, backup, transaction processing.)**
3. **SO3: Describe different types of DBMS`s. (Range: Hierarchical, Relational, Network, Object.)**
4. **SO4: Review DBMS end-user tools.**

DATA MANAGEMENT ISSUES AND HOW THESE ARE ADDRESSED BY A DBMS (SO1)

The first computerised databases took over the job of paper-based data storage systems. If you think of an old library index card system, this is the sort of approach that was used before data could be digitised.

There were a number of problems with paper-based data storage, most of which are obvious if you consider the limitations of physical storage and manual access. For example:

- Data took up a lot of space to store
- Creating and storing data was slow, as was finding and retrieving data
- It was very difficult and time-consuming to backup a dataset
- Data could be improperly captured or filed and there were few checks and balances, unless human oversight procedures were put into place (once again, time consuming and costly in terms of requiring a human resource)
- Mis-filing could lead to duplicates
- Contradictory data could exist in different places
- Consolidating data was difficult
- Data could be incomplete or otherwise lack integrity or become invalid without triggering any sort of alarm or update procedure

Digital storage of data was the first step towards solving these problems.

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.

1.1 EVOLUTION OF THE DATABASE

Database models have evolved considerably since digital data storage began in the mid-twentieth century. It is worth a brief consideration of this evolution as many of the older database models are still in use as legacy systems in various environments you might encounter professionally. As we follow this development process, keep in mind the issues each new database model evolved to address.

Hierarchical (IMS): late 1960s and 1970s

The IMS database, released in around 1968, was able to store fields (bits of information) based on its data type (Stonebraker, & Hellerstein, 2005). This is known as a record type. Then, each instance of a record type had to follow predetermined data descriptions based on the record type. Next, each subset of a field needed a key - something that would make it uniquely identifiable. Finally, record types needed to be arranged into a tree, where each record type, other than the root one, had a unique parent record type. This made the process of populating the database quite restrictive, and often ended up being a manual process that took a lot of time. It also ran the risk of repeated information, which could lead to inconsistent data. Examples: these could store a book with information on chapters and verses, or a database of songs, recipes, models of phones and anything that can be stored in a nested format. An XML document is another example.

Network (CODASYL): 1970s

CODASYL (Committee on Data Systems Languages) were proponents of the network data model (Stonebraker, & Hellerstein, 2005). Rather than using trees to organise their data, they wanted to use a network of record types -- each with their own key. This meant that an instance of a record could now have multiple parents. This made the database more flexible, but equally more complex.

Relational: 1970s and early 1980s

In 1970, Ted Codd proposed a new model that would require far less maintenance than the IMS databases (Stonebraker, & Hellerstein, 2005). He wanted to be able to store data in tables, and didn't want to need any physical storage. Enter the relational database. Unlike its two predecessors, this database did not need to specify a storage proposal and had the flexibility to store and represent any type of data. Codd was met with some scepticism by CODASYL users, but relational databases have stood the test of time.

Entity-Relationship: 1970s

While the debate of relational versus CODASYL databases raged into the mid-1970s, Peter Chen came up with the Entity-Relationship (E-R) model as an alternative to both (Stonebraker, & Hellerstein, 2005). In this model, the database contains entities, where each entity has attributes. The attributes are the characteristics that describe the entity, and at least one attribute needs to be unique to that entity -- this would be a key. There could also be relationships between entities. Next, unlike relational databases, E-R models didn't require the process of converting to first, second, and then third normal form (a process you will learn about in a later unit. Rather, the database was converted straight to the third normal form automatically.

Extended Relational: 1980s

This was a decade of trying to improve the relational model, hence its nickname: "the R++ era" (Stonebraker, & Hellerstein, 2005). One of these improvements was proposed by Zaniolo, who added set-valued attributes (being able to create your own data types), aggregation (being able to tuple-reference as a data type), and generalisation (being able to have specialisations that inherit attributes from its ancestors). Unfortunately, despite these additions, the implementation didn't seem to be any more efficient than its predecessor, and the ideas were not widely adopted.

Semantic: late 1970s and 1980s

Around the R++ era, another movement was forming. The Post Relational model was being designed in a way that adapted the original relational model to be "semantically impoverished" (Stonebraker, & Hellerstein, 2005). This Semantic Data Model made use of classes -- records that followed the same schema. Like the Extended Relational Models, the SDM had capabilities for aggregation and generalisation. However, the SDM generalised the aggregation so that an attribute in a class could be a set of instances in another class. This allowed for inverse attributes to be defined as well. The generalisation capability was also extended so that classes could generalise other classes to graphs as well as trees. Classes were also able to have class variables. However, the Semantic Data Model was extremely complex and, like the Extended Relational models, didn't really improve processing performance, and so was not widely utilised.

Object-oriented: late 1980s and early 1990s

The 1980s saw a focus on trying to smooth the way that object-oriented coding languages and relational databases could interact with one another (Stonebraker, & Hellerstein, 2005). Historically, there was a mismatch around naming systems, data type systems, and returning data, which caused a lot of unnecessary difficulties. The solution: create a persistent programming language that both the program and the database could understand, while being conservative with memory usage. Unfortunately, despite multiple companies attempting to create this new language, it was not well picked-up by the market. The theories as to why included the cost, the lack of standardisation, the lack of universality, and the time it would take to switch all existing interactions would not be worth the hours.

Object-relational: late 1980s and early 1990s

In the 1980s, Geographic Information Systems (GIS) were becoming widely used, but there was a problem with storing geographical coordinates in a database: storing coordinates requires two pieces of information to be saved and searched (longitude and latitude), but databases up to that point were solely one-dimensional. Moreover, there wasn't a data type available in databases that could store coordinates. Enter the Object-Relational Models (Stonebraker, & Hellerstein, 2005). These databases could be customised to the user's needs -- including the creation of user-defined data types, operators and access methods. This enabled those wanting to store GIS data in a multidimensional indexed system that could facilitate easy storing and searching. The OR Model was welcomed commercially and has been relatively successful. However, its lack of standardisation has still been a hindrance to widespread use.

Semi-structured (XML): late 1990s to the present

XML Models have two main features: schema last and the XML data Model (Stonebraker, & Hellerstein, 2005). Schema last means that the schema is not required in advance of adding data to the database. This means data doesn't have to be consistent with a pre-existing schema to be input. Data instances must be self-describing through the use of attributes, because there is no schema to give it meaning. This is not universally better, but it does make storing semi-structured types of data easier. However, because semi-structured data is rather niche, this is not widely used.

XML Data Model, on the other hand, have four main features that have been inspired by its predecessors: XML records can be hierarchical (Like IMS), they can have links to other records (like CODASYL and SDM), they can have set-based attributes (like SDM), and they can inherit from other records (like SDM). Beyond this, XML models are able to have attributes on a record that can be one of many possible data types. For example, a bank branch can be stored either in the form of the branch name (string), or the branch code (integer). However, as you can imagine, this makes the tree indexes increasingly complicated.

Because of the niche nature of schema last and the complexity of XML Data models, semi-structured models are not widely used.

NoSQL: late 1990s to the present

NoSQL databases are a non-relational database model that evolved to enable the storage and management of data according to a non-tabular (not using tables) model. The evolution of the database type has been driven by the rise of big data and the need to scalably and responsively store and manipulate large, rapidly changing, datasets where records can often differ considerably from one another. We'll look at NoSQL more in the next section.

COMMONLY IMPLEMENTED FEATURES OF COMMERCIAL DATABASE MANAGEMENT SYSTEMS (SO2)

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. The DBMS evolved to manage functions such as **C**reating data, **R**etrieving data, **U**pdating data, and **D**eleting data (basic operations usually performed on data, represented by the acronym CRUD).

Think about it: Considering the examples of challenges posed by pre-digital data storage, what other things do you think a DBMS needs to handle in order to enable reliable, efficient, and effective data management? Research the answer if you're unsure.

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 (Tutoriallink.com) are as follows:

- **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 as data is drawn from a variety of sources or end-users.

- **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 (Tutorial Link, n.d.).

- **Improved data security:** The DBMS makes it easy for companies to enforce data privacy and security policies.

DIFFERENT TYPES OF DBMS'S (SO3)

There are many different types of databases and DBMS's. These databases 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, & Crockett describe some categories of databases as follows:

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.

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.

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.

The degree to which data is structured is another way of classifying databases. Data that exist in their original, or raw, state are known as **unstructured data**. In other words, they are in the format in which they were collected. **Structured data** is the result of formatting unstructured data to facilitate storage, use, and generate information. You apply structure based on the type of processing that you intend to perform on the data. For example, imagine that you have a stack of printed invoices. If you just want to store these invoices so that you are 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 format so that you can perform the desired computations.

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 type of database you may have heard of is the **relational database**. A relational database is structured to recognize 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 lesson (Rob, Coronel, & Crockett, 2008).

NoSQL databases use a new generation database management system. Historically, DBMSes have been relational, and we have used tools such as SQL (Structured Query Language) to interact with them. In a relational database, data is in tables that *relate* to one another in ways that represent real-life relationships between data entities. For example, in a database containing data about the employees at a company, you could have a table of people's *names* and another table of *job titles*. The tables will relate to one another because each person in the first table will have a job title listed in the second table (we'll discuss this in more detail in a subsequent lesson). In contrast, NoSQL (Not Only SQL) is a new generation DBMS that is not based on the traditional database model. NoSQL databases generally have the following characteristics (Coronel, & Morris, 2016):

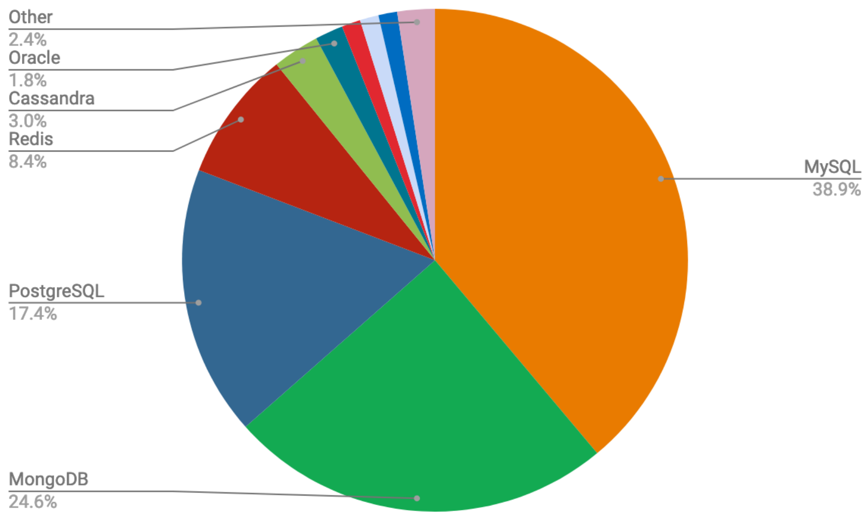
NoSQL databases generally have the following characteristics:

- 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

NoSQL databases are ideal for distributed data storage for large quantities of data, as they are flexible, scalable, and high-performance. They are typically used for Big Data and data generated from real-time web apps. Believe it or not, you are using a NoSQL database every time you search for a product on Amazon, or watch a video on Youtube, or send a message to a friend on Facebook!

DBMS END-USER TOOLS (SO4)

Some popular DBMS today are:



(Sale Grid, 2019)

Relational DBMS

- **MySQL:** This popular open-source RDBMS is named after "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

- **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:** Oracle's was the first commercially usable RDBMS launched in the market and is the leader in terms of worldwide RDBMS software revenue (Rob, Coronel, & Crockett, 2008).

NoSQL DBMS

- **MongoDB, Cassandra, and Redis**: all NoSQL DMBSs. Look them up and see what you can find out about them!

TERMINOLOGY

Specialised vocabulary is required for the description of databases. We look at some of these terms as described by Rob, Coronel, & Crockett 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 1 byte of computer storage.

- **Field:** A field is a character or group of characters that have a specific meaning. It is used to define and store data. In our first table example above, the two fields would be cities and maximum temperatures.
- **Record:** A record is a logically connected set of one or more fields that describes a person, place or thing (a data element). Continuing our example, all data about a particular city would be a record about that city.
- **File:** A file is a collection of related records.

Have a look at the ANIMAL file below:

A_SPECIES	A_SERIAL	A_REGION	A_NAME	A_GENUS	A_DOB	H_NAME
Lion	0002002	Namibia	Leah Laved	Panthera	16 April 2004	John Mack
Tiger	102030421	Serbia	Ted Tix	Panthera	23 March 1996	Lindy Queue
Jellyfish	4365	Atlantic Ocean	Jill Jam	Chrysaora	8 September 2017	Sam Kins

In this file:
A_SPECIES = animal species
A_SERIAL = animal serial number
A_REGION = animal region
A_NAME = animal name
A_GENUS = animal genus
A_DOB = animal date of birth
H_NAME = the name of the human who spotted the animal

Using the proper file terminology, can you identify the file components shown in the table above? Try and jot down your answers to the following questions and then check these on the next page.

- What are the records in this file?
- How many fields make up each record and what are their names?
- What is the file name?

Answers:
The ANIMAL file contains 3 records. Each record is composed of 7 fields: A_SPECIES, A_SERIAL, A_REGION, A_NAME, A_GENUS, A_DOB and
H_NAME. The 3 records are stored in a file named ANIMAL because it contains animal data.

Extra resource

*This lesson merely provides a brief overview of what databases are and how they are used. Database design is a very interesting field that takes a while to learn. If you'd like to know more about database design, we highly recommend reading the book "**Database Design - 2nd Edition**" by Adrienne Watt. The first 8 short and easy-to-read chapters of this book provide more detail regarding what has been covered in this lesson.*

Compulsory Task

Create a text file called **databases.txt** where you will answer the following questions.

- Explain the difference between data and information.
- Define each of the following terms in your own words:
  a) Data
  b) Field
  c) Record
  d) File
- DBMS provides a solution to a variety of issues in storing and managing data. Outline these issues, indicating how they are addressed by DBMS's.
- What are the commonly implemented features of commercial DBMS's?
- Research "sparse data". What is it and when might it be found?
- Explain the different types of DBMS's.
- Research and list three NoSQL DBMS's, reviewing their features.
- Look up ACID properties of databases. Describe these in your own words.
- Given the file below, answer the following questions:
  - How many records does the file contain?
  - 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, Durban, 64700	R13179975.00
21-7Y	Jane Grant	0181-898-9909	218 Clark Blvd, Cape Town, NW3, TRY	R45423415.00

25-9T	George Dorts	0181-227-1245	124 River Dr, Cape Town, N6, 7YU	R78287312.00
29-7P	Holly Parker	33-5-592000506	180 Boulevard du General, Durban, 64700	R20883467.00

Remember to refer to the self-assessment table provided on the next page to assess your own degree of achievement of each of the learning outcomes. Try to gauge your own learning and address any weak areas before submitting your task, which will become part of your formative assessment portfolio.

Self-assessment table

Intended learning outcome (SAQA US 11049)	Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- I can successfully:		
Describe data management issues and how they are addressed by a DBMS (SO1).	Identify the problems DBMS address	Include examples	Outlines ways which database management systems address the issues
Describe commonly implemented features of commercial database management systems (SO2).	List and Identify the purpose of each feature	Identify the way in which each feature contributes to the solution of database management issues	
Describe different types of DBMS`s (SO3).	Describe characteristics of each DBMS type	Give examples of the use of each DBMS type.	
Review DBMS end-user tools (SO4).	Identify the features and limitations of end-user DBMS tools	Outline the interaction between the tools and the database	Ensure the review stays on topic (i.e. is specific to the use of end-user DBMS tools rather than, e.g., DB servers)
Intended learning outcomes (Hyperion extension)	Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- I can successfully:		
Research “sparse data”. What is it and when might it be found?	Explain the concept of sparse data	Contrast sparse data with the alternative in a manner that clarifies both concepts	
Look up ACID properties of databases. Describe these in	Explain the concept	Put across this and other	I can explain concepts I’ve answered questions on in this

your own words.	represented by the acronym ACID in a database	concepts I'm explaining entirely in my own words without copying and pasting or engaging in any form of plagiarism	task without referring to my written notes - i.e., I have learned and internalised these concepts from the process of answering the questions
Review a table of data	Look at a typical database table and identify its components	Remember and apply the appropriate terms for aspects of DBMS	

Rate us

Share your thoughts

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

Think that the content of this lesson, task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES

Coronel, C., & Morris, S. (2016). *Database Systems: Design, Implementation and Management* (12th ed., pp. 1-70). Mason, OH: Cengage Learning.

Coronel, C., Morris, S., & Rob, P. (2011). *Database systems* (9th ed., pp. 4-26). Boston: Cengage Learning.

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

Pokorny, J. (2013). *NoSQL databases: a step to database scalability in a web environment*. International Journal of Web Information Systems.

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/**

Stonebraker, M., & Hellerstein, J. (2005). What goes around comes around. *Readings in database systems*, 4, 1724-1735.

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