Hyperiondev

LEARNER GUIDE

Databases

Aligned to SAQA US 114049, 115373, 114048, 115367

Learner Information

Details	Please Complete this Section
Name & Surname:	_____
Department:	_____
Facilitator Name:	_____
Date Started:	_____
Date of Completion:	_____

Contents

Programme Overview

Welcome to this learning programme that will lead you to greater understanding of **formulating recommendations for a change process**.

As you work your way through the learning programme you will gain competence against the following Unit Standard:

Programme: Databases

Unit Standard(s):
SAQA ID 114049
Demonstrate an understanding of Computer Database Management Systems
NQF Level 5, 7 Credits
SAQA ID 115373
Demonstrate an understanding of sort and search techniques used in computer programming
NQF Level 5, 6 Credits
SAQA ID 114048
Create database access for a computer application using structured query language
NQF Level 5, 9 Credits
SAQA ID 115367
Demonstrate logical problem solving and error detection techniques
NQF Level 5, 8 Credits

This learning programme is **intended for** all persons who need to formulate recommendations for a change process. This Unit Standard is intended for managers in all economic sectors. These managers would typically be second level managers such as heads of department, section heads or divisional heads, who may have more than one team reporting to them.

Programme entry level requirements

It is assumed that people learning towards this Unit Standard are already competent in:

- Communication at NQF Level 4.
- Mathematical Literacy at NQF Level 4.
- Computer Literacy at NQF Level 4.

Programme Outcomes

This learning programme is outcomes-based which means we take the responsibility of learning away from the facilitator and place it in your hands.

Your learning will begin in the workshop where you will identify the skills and knowledge you require in order to meet the specific outcomes and assessment criteria contained in the unit standard.

In this learning programme, we will be covering the following **learning outcomes**:

US: 114049

Demonstrate an understanding of Computer Database Management Systems

- Describe data management issues and how it is addressed by a DBMS.
- Describe commonly implemented features of commercial database management systems
- Describe different type of DBMS`s.
- Review DBMS end-user tools.

US: 115373

Demonstrate an understanding of sort and search techniques used in computer programming

- Demonstrate an understanding of how abstract data types are stored on computers.
- Demonstrate an understanding of sort techniques used to sort data held in data structures.
- Demonstrate an understanding of search techniques.

US: 114048

Create database access for a computer application using structured query language

- Review the requirements for database access for a computer application using SQL
- Design database access for a computer application using SQL.
- Write program code for database access for a computer application using SQL.
- Test programs for a computer application that accesses a database using SQL.
- Document programs for a computer application that accesses a database using SQL

US: 115367

Demonstrate an understanding of search techniques.

- Describe different approaches to problem solving.
- Use logical operators in descriptions of rules and relationships in a problem situation.
- Simplify Boolean expressions with Boolean algebra and Karnaugh maps.
- Describe the basic concepts of error detection.

During the workshop you will complete a number of class activities that will form part of your formative assessment. In this you have the opportunity to practice and explore your new skills in a safe environment. You should take the opportunity to gather as much information as you can to use during your workplace learning and self-study.

The workshop will be followed by summative assessment tasks to be completed through self-study in your workplace. In some cases you may be required to do research and complete the tasks in your own time.

Assessment

It is important to note that the onus is on you, as the learner, to prove your competence. You therefore need to plan your time and ensure that your Portfolio of Evidence is kept up to date and handed in timeously.

A Portfolio of Evidence is a collection of documents of work you have produced to prove your competence. You will compile your portfolio from activities, tools and checklists associated with the unit standard and relevant to the unit standard being assessed.

You will be given the following documents to assist you in creating a portfolio of evidence:

- **Learner Guide**: The Learner Guide is designed to serve as a guide for the duration of your learning programme and as the main source document for transfer of learning. It contains information (knowledge and skills required) and application aids that will assist you in developing the knowledge and skills stipulated in the specific outcomes and assessment criteria. The learner guide also indicates the formative assessment class activities that you need to complete towards your Portfolio of Evidence.
- **Learner Workbook**: The learner Workbook contains all the class activities that you will be completing to show formative learning. These will be assessed as part of your portfolio of evidence as formative assessment. You will be handing in the Learner Workbook as part of your Portfolio of Evidence.
- **Learner Portfolio of Evidence Guide**: The Learner Portfolio of Evidence Guide provides details about the assessment, such as the assessment preparation, plan and specific summative assessment activities that you need to complete in the workplace.

Both formative and summative assessment is used as part of this outcomes-based learning programme:

- **Formative Assessment**: In order to gain credits for this Unit Standard you will need to prove to an assessor that you are competent. The Class Activities throughout your Learner Workbook are designed not only to help you learn new skills, but also to prove that you have mastered competence. You will be required to develop a Portfolio of Evidence to hand in to an assessor so that you can be assessed against the outcomes of this Unit Standard. Where you encounter a Class Activity icon, you must complete the formative assessment activity in the Learner Workbook. Comprehensive guidelines for the development of your

Portfolio of Evidence may be found in the Learner Portfolio of Evidence Guide for the particular learning programme that you are working with.

- **Summative Assessment**: The NQF's objective is to create independent and self-sufficient learners. This means that you will also be required to do independent research and assignments, such as Knowledge Questions, Practical Activity (completed in the workplace), Witness Testimony and Logbook.

  The assessment process is discussed in detail in the Learner Portfolio of Evidence Guide. When you are ready, you will advise your mentor that you are ready for assessment. He or she will then sign off the required sections in the Learner Portfolio of Evidence Guide and you will be able to submit your Portfolio of Evidence for assessment. The summative assessment activities placed in the Learner Portfolio of Evidence Guide for your convenience. If any of your assessment is conducted using observation, role plays or verbal assessment, place a signed copy of the checklists, once completed by your mentor or line manager in your Learner Portfolio of Evidence Guide, as indicated.

The Training Provider will assess your portfolio. If successful, you will receive the credit value of this learning programme. The entire assessment process is explained in the Learner Portfolio of Evidence Guide and you are urged to read this guide as soon as possible as it explains the assessment process in detail and clarifies your rights and responsibilities to ensure that the assessment is fair, valid and reliable.

If you are not successful, you will receive all the guidance needed to resubmit your Portfolio of Evidence within a specific time period, as per the Training Provider requirements.

Learning map (delivery structure)

<table>
<tr><td>Assessment</td><td colspan="2">Formative Assessment 30%</td><td colspan="2">Summative Assessment70%</td></tr>
<tr><td rowspan="3">Learning activities
For 300 hours of notional learning</td><td>Contact Learning Theory input Formative assessment (workbook activities): group activities, simulations</td><td>Prescribed reading, support, coaching</td><td>Learning and application at the workplace</td><td>Summative assessment in PoE: knowledge questions, practical workplace activity, Witness Testimony, logbook</td></tr>
<tr><td>105 hours</td><td>0 hours</td><td colspan="2">195 hours
8 hours</td></tr>
<tr><td colspan="4">Compilation of Portfolio of Evidence</td></tr>
<tr><td colspan="2">Complementary workplace practices</td><td colspan="3"></td></tr>
</table>

Learner Support

Please remember that as the programme is outcomes based – this implies the following:

- You are responsible for your own learning – make sure you manage your study, practical, workplace and portfolio time responsibly.
- Learning activities are learner driven – make sure you use the Learner Guide, Learner Workbook and Learner Portfolio of Evidence Guide in the manner intended, and are familiar with the Portfolio requirements.
- The Facilitator is there to reasonably assist you during contact, practical and workplace time of this programme – make sure that you have his/her contact details.

Computer Database Management Systems

Aligned to SAQA US 114049

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

Aligned to SAQA Unit Standard 114049

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.

SPECIFIC OUTCOME 1 (US 114049)

5. DATA MANAGEMENT ISSUES AND HOW THESE ARE ADDRESSED BY A DBMS

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.

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 skepticism 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.

SPECIFIC OUTCOME 2 (US 114049)

6. COMMONLY IMPLEMENTED FEATURES OF COMMERCIAL DATABASE MANAGEMENT SYSTEMS

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.

SPECIFIC OUTCOME 3 (US 114049)

7. DIFFERENT TYPES OF DBMS'S

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** are 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 organizes 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

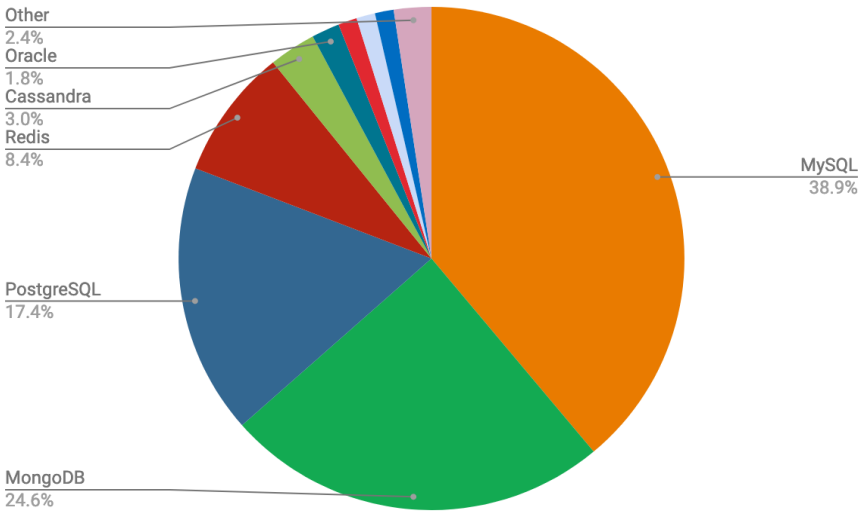
- They support distributed database architectures
- They provide high scalability, high availability and fault tolerance
- They support very large amounts of sparse data
- They are geared toward performance rather than transaction consistency

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!

SPECIFIC OUTCOME 4 (US 114049)

8. DBMS END-USER TOOLS

Some popular DBMS today are:



(Sale Grid, 2019)

Relational DBMS's

- **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's

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

9. 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 joy 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 1

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's provide 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 your own words.	Explain the concept represented by the acronym ACID in a database	Put across this and other concepts I'm explaining entirely in my own words without copying and pasting or engaging in any form of plagiarism	I can explain concepts I've answered questions on in this 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	

Sorting and Searching

Aligned to SAQA US 115373

Introduction

WELCOME TO THE SORTING AND SEARCHING LESSON!

In 2007 the former CEO of Google, Eric Schmidt, asked the then presidential candidate Barack Obama what the most efficient way to sort a million 32-bit integers was (watch it here!). Obama answered that the bubble sort would not be the best option. Was he correct? By the end of this task you might be able to answer that question! In this lesson you will learn about data sources and types, data structures, and how to use these, including ordering and finding data in different types of data structures. At the end there is also a task for you to put into practice what you have learned in this lesson.

INTENDED LEARNING OUTCOMES

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

- Demonstrate an understanding of how abstract data types are stored on computers.
- Demonstrate an understanding of sort techniques used to sort data held in data structures.
- Demonstrate an understanding of search techniques used to search datasets.

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.

CONCEPT: DATA SOURCES

Data scientists work with data that comes from various sources. Sometimes records in a database are entered manually, in a traditional data-capture process; sometimes data is downloaded from the Internet or obtained using a web API, or it may be obtained from an open-source or proprietary data supply

source, as is the case with many databases used for big-data research. Wherever data comes from, it generally ends up as a file on a computer.
Before we begin working with files, let us consider File Systems. Your computer harddrive is organised into a hierarchical structure of files and directories (washington.edu, n.d.):

- **Files:** A file is a resource on your computer that contains and stores information. There are different types of files that serve different purposes - storing audio, video, text, settings and metadata, program files, and so on. Some common file types you will encounter containing data extracted from databases are CSV, XML, JSON and Python files. You will learn about working with some of these files in this task.

- **Directories:** A directory is a cataloguing structure which contains references to other computer files and possibly other directories. Simply put, a directory is a location for storing files on your computer.

Your file system starts with a **root directory**, typically denoted on Unix by a forward slash ('/'), and on Windows by a drive letter (e.g. 'C:/') (washington.edu, n.d.). Now that we are familiar with file systems, we can dive into the different types of files that can be used as sources of data.

SPECIFIC OUTCOME 1 (US 115373)

1. UNDERSTANDING ABSTRACT DATA TYPES

DATA TYPES

Data types define the types and ranges of values that variables can have, and thus what can be stored in these variables. Some examples are given in the table below.

Data type	Explanation	Example

Integer numbers	Integers (`int`) are whole numbers (i.e. numbers that have no decimal points).
	
	This is any number from -2147483648 to 2147483648
		`int number1 = 5;`
		`int number2 = 9865;`
		`int number3 = -5986;`

	This is often the most commonly used data type.	
Floating point numbers	Floating-point numbers (**`float`**) are similar to integers. The critical difference is that floats can contain decimal points. With floats, in Java, it's essential to add an 'f' at the end of your float (should it contain a decimal point) so that Java knows it has floating-point numbers.	`float num1 = 5.5879f;` `float num2 = -87.48f;`

Long numbers	Long numbers (**`long`**) are very similar to integers; the significant difference is that they can hold much *larger* numbers.
	
	A long variable value can be any number from -9223372036854775808 to 9223372036854775808
	
	A key point to remember with long values is that if you compile your program for only 64-bit-based machines, your program won't work on 32 bit based systems.
	
	Typically integers are the preferred data type, but if you need a number more significant than an ints range, you should use the long data type.
		`long num1 = 489;`
		`long num2 = 999999999;`
		`long num3 = -156;`

Double numbers	Doubles (**`double`**) are floats with a much more extensive range.
	
	Doubles can have a total of 15 decimal places.
		`double num1 = 159.126;`
		`double num2 = 1234.156165156156;`
		`double num3 = -156.8778;`

Boolean	Booleans (**`boolean`**, commonly known as true or false values) have only two values, true/false. Booleans are commonly used in **`if`** statements and loops as they are used to tell the program when (and when not) to run a particular portion of code.	`boolean bool1 = true;` `boolean bool2 = false;`

Characters	Characters (**`char`**) are any single character. If you take a look at your keyboard, every key is one character (including spaces, punctuation, and numbers - however, numbers assigned to a **`char`** variable are essentially text and not treated numerically).
	
	It's important to remember that when creating a **`char`** variable in Java, the character assigned to the variable should always be encased inside **single** quotation marks.
		`char char1 = 'a';`
		`char char2 = '5';`
		`char char3 = '!';`
		`char char4 = '☺';`

Strings	Strings (**`string`**) can be considered an upgrade to characters, in that this data type can hold values that are the combination of multiple characters into a single variable. Strings are thus used	`String string1 = "Logan";`
		`String string2 = "Serge is 24";`
		`String string3 = "a";`

	to hold all text longer than a single character. However, strings can also hold only one character. It's important to remember that when creating a `string` variable in Java, the text assigned to the variable should always be placed inside **double** quotation marks.	

Objects	Objects are a crucial component in Java; almost anything you see in Java will be an object.
	Whenever you see the keyword "new" in Java, it's our way of telling the program that we are creating a new object. In creating an object, we use whatever the object name is in the creation statement. In the example, the objects being created are Pokemon, and so a Pokemon constructor for new objects is called.
		`Pokemon furret = new Pokemon("Normal", 2, "Keen eye");`
		
		`Pokemon pikachu = new Pokemon("Electric", 5, "thunderbolt");`

ABSTRACT DATA TYPES (ADTs)

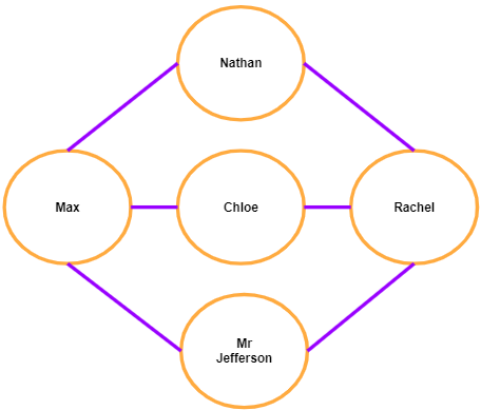
Abstract data types are different from "just plain" data types, in that the term *abstract data type* refers to a way of understanding a conceptual abstraction, a semantic model for storing, accessing, and manipulating data, including the values, operations, and behaviours defined as being allowed for users to perform for each specific abstract data type. Some examples are given in the table below.

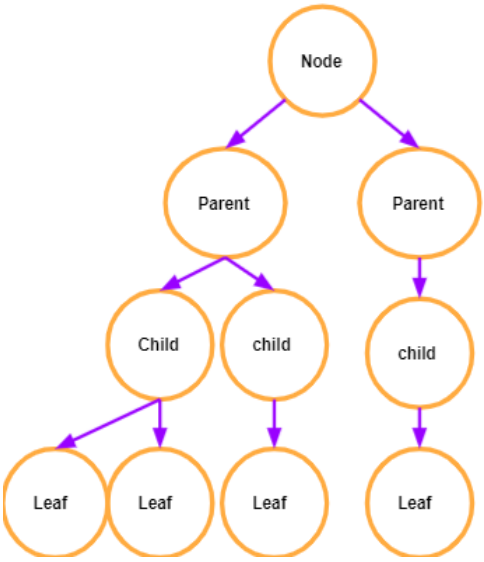
Abstract Data Type	**Explanation**	**Example**

Queues	Queues are part of the java.util package (i.e. to use queues you will need to import an external library).
	
	Queues follow the FIFO principle which stands for “first in first out” - i.e. the first object added to the queue becomes the first extracted from the queue.
	
	This is similar to how an ATM queue works. Imagine there is one ATM and a row of people. The first person to arrive will be the first one to leave as well.
	
	The same principle can be applied to a Java queue- whichever data element you enter first will be the first element to leave the queue.
	
	It’s also important to know that the queue is a type of linked list so you will also need to import the linked list class.
		`import java.util.LinkedList;`
		`import java.util.Queue;`
		`//Creating new Queue`
		`Queue<String> students = new LinkedList<>();`
		
		`//Adding students to queue`
		`students.add("Kylie");`
		`students.add("Julia");`
		`students.add("Glitch");`
		
		`//output of 'students' will be -> [Kylie, Julia, Glitch]`
		
		`//removes the first student added`
		`students.remove();`
		
		`//output of students will be -> [Julia, Glitch]`

Stacks	Similar to queues, stacks are part of the java.util class. As with queues, you will need to import this class to make use of a stack.
	
	Stacks follow the rule of LIFO which stands for last in first out - i.e. the last object added to the queue becomes the first extracted from the queue.
		`import java.util.Stack;`
		`//Create new stack object`
		`Stack<String> myPets = new Stack<>();`
		
		`//Pushing our items to the stack`
		`myPets.push("Dog");`
		`myPets.push("Cat");`
		`myPets.push("Ferret");`
		
		`//Output of MyPets -> [Dog, Cat, Ferret]`
		
		`//Removing the last element`

	As an example of this, think of a stack of paperwork on a desk. The sheet of paper at the bottom was the first thing added to that stack; however, this will be the last thing we see as we work through the stack.
	
	The same principle applies to a stack in Java, when adding (we use the *.push()* method to add to a stack) anything after that value is entered will be removed first (we can use the *.pop* method to remove data from a stack)
		`myPets.pop();`
		
		`//Output of MyPets -> [Dog, Cat]`

Graphs	The graph data structure shows the relationship between multiple elements. For example, on most social media platforms, whenever you follow someone new, you may get new suggestions on who to follow based on the person you recently followed. This is because the person you follow has a relationship with other people outside your social circle. In the example on the side, we have a couple of people's names and their connections with one another. When using graphs, we have the following definitions.	

	Vertex - A vertex in this example is the different people we have listed. **Edge** - An edge is a line that shows the relationship between each person.	
Trees	Trees are very similar to a family tree or a business structure. Trees start off with a 'node'; a node is the starting point of the tree and does not have a previous generation. Trees then have children. Children are the portions of data that grow off from the node. Children have a parent node (in this case, the original node would be the parent); if two children stem from the same node, they are then declared siblings. Once we have children, we can then have parents. Parents are the original source of the data that extends from them. And finally, we have a leaf. A leaf is the last set of data to appear in the tree. It is the child of a parent node but does not have any children of its own.	

1.1 Data structures to store abstract data types

ARRAYS

The array is a data structure, provided by Java, which stores a fixed number of elements of the same type. An array is used to store a collection of data. However, it is more useful to think of an array as a collection of variables of the same type.

Declaring Arrays

To use an array in a program, a variable to reference the array must be declared, and the array's element type must be specified. The array can have any element type. All elements that are in the array will have the same data type. To declare a variable, use the following syntax:

```
elementType[] arrayVariable;
```

The following example declares a variable `numArray` that references an array that stores elements of the type `double`.

```
double[] numArray;
```

Creating Arrays

After declaring an array variable you can create an array using the new operator. The syntax is as follows:

```
elementType[] arrayVariable = new elementType[arraySize];
```

The following example declares an array variable called numArray, creates an array of 5 double elements and assigns the reference to the array to numArray.

```
double[] numArray = new double[5];
```

The array size must be given when space for an array is allocated. The size cannot be changed after an array is created. To obtain the size of an array, use `arrayVariable.length`.

Numeric data types are assigned the default value 0, `char` types are assigned the value `\u0000` and boolean types are assigned the value `false` when an array is created.

Array Indices

To access array elements you use an index. Array indices range from 0 to

`arrayVariable.length - 1`. For example, the array created above (`numArray`), holds five elements with indices from 0 to 4. You can represent each element in the array using the following syntax:

```
arrayVariable[index];
```

For example, `numArray[1]` represents the second element in the array `numArray`.

Initialising an Array

To assign values to the elements we will use the syntax:

```
arrayVariable[index] = value;
```

The example below initialises the array referenced by the variable `numArray`:

```
numArray[0] = 23.6;
numArray[1] = 45.12;
numArray[2] = 8.4;
numArray[3] = 77.7;
numArray[4] = 1.34;
```

You can use shorthand notation in Java, known as the array initialiser, which combines the declaration, creation and initialisation of an array in one statement. The syntax for an array initialiser is as follows:

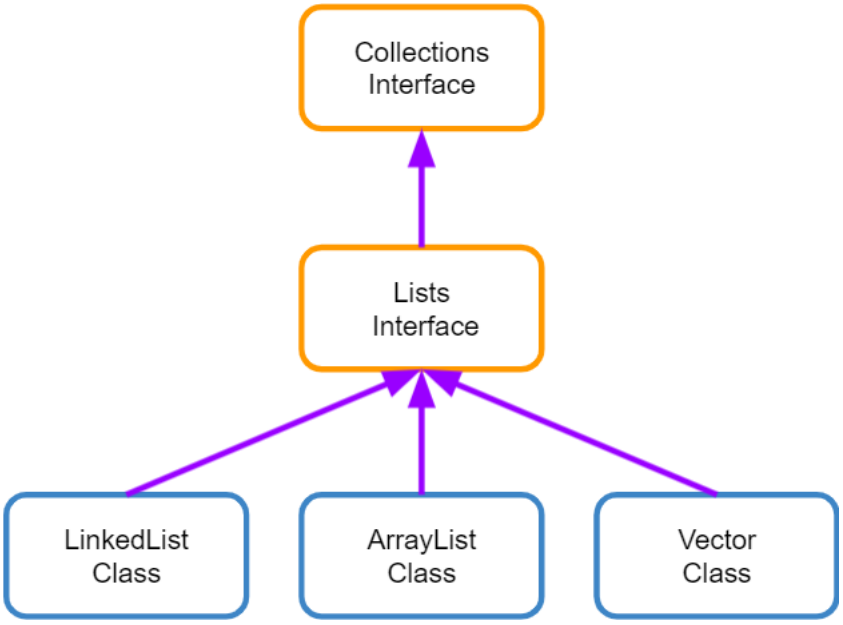
```
elementType[] arrayVariable = {value1, value2, ..., valueN};
```

For example, the following array initialiser statement is equivalent to the example above in which each element was assigned a value individually:

```
double[] numArray = {23.6, 45.12, 8.4, 7.77, 1.34};
```

LISTS

Lists are an interface in Java that is used to implement a large number of different list types. In the image below, you can see the relationship between a list and the classes that implement it.



Unlike many other data structures in Java, a list is not created in the way you may be familiar with.

```
List<String> listOne = new List<String>;
```

This will cause your program to contain a syntax error. The idea of a list is to assist us in implementing the following...

```
//ArrayList
List<String> arrayList = new ArrayList<>();
```

```
//LinkedList
List<String> linkedList = new LinkedList<>();
```

```
//Stacks
List<String> stackList = new Stack<>();
```

```
//Vectors
List<String> vectorList = new Vector<>();
```

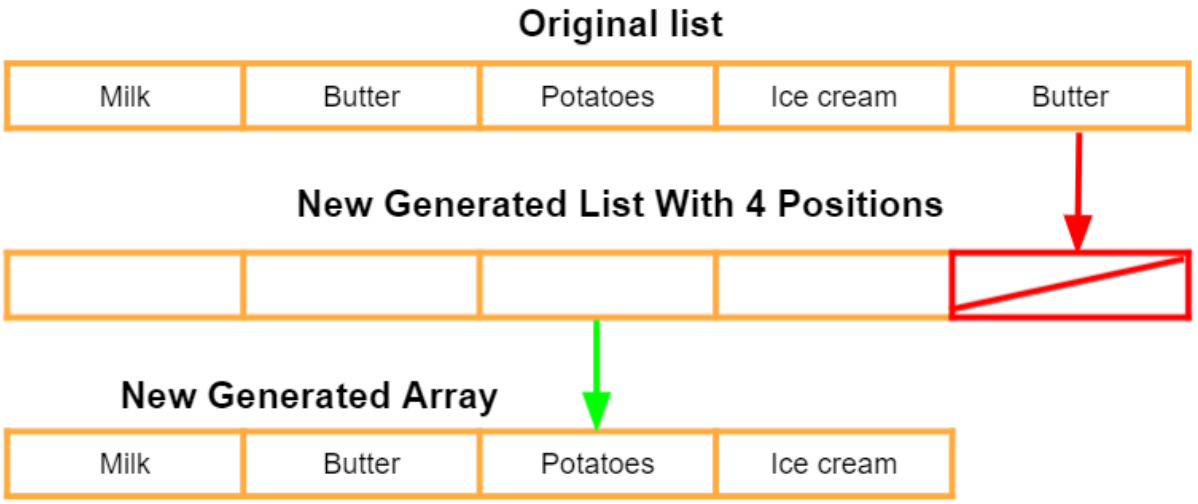
ARRAY LISTS

Array lists are our most commonly used data structure; these ArrayLists have an unlimited number of slots for how much data they can store.

So then, why would we use any other data structure if arrays are so powerful? The reason for us using different data structures (other than features) is because array lists can be viewed as “slower” (more information about speed when you learn about time complexity.)

ArrayLists become slower with the more data we feed it.

When adding/removing data Java takes all the data from our current list and copies it over to an entirely new list with the added/removed data we provided, take a look at this example below.



As you can see above, our original array list has a total of 5 values, but we’ve accidentally added butter twice. So when we remove it, our program needs to create a whole new list with only four values and then copy that data over to an entirely new array list.

While this may not be a massive problem with arrays with a small amount of data, imagine we have an array with over 9000 sets of data, and we need to remove one; this would cause our program to take a long time to complete this process.

Declaring and creating arraylists

Declaring array lists is slightly different compared to creating an array. Because ArrayLists have no size limits (i.e. after creating an array list, you can add or remove data at any time)

With ArrayList, you have to specify what data type your array will be using...

```
List<String> stringList = new ArrayList<>();
List<Integer> intList = new ArrayList<>();
List<Double> doubleList = new ArrayList<>();
```

As you can see above, inside our diamond brackets **<>** we place our data type. Unlike when creating a variable, we used the complete word (rather than int, we would use Integer).

Adding/Removing data from an arraylist

Now that we've created an ArrayList, how do we add values to our list? Well, there are a few ways we can do this.

- **Arraylist default values**
    - Let's say we already have data that we know we want to be placed inside our array; we can use something known as **Arrays.asList** to achieve this.

```
List<String> computerParts = new ArrayList<>(Arrays.asList("test1",
"test2", "test3"));
```

    - As you can see within our brackets at the end of our array creation, we've added the **asList** method. This can take any number of values, and these will automatically be placed inside the array when it's created.
- **Adding data later on**
    - As we know, sometimes we won't know all the data we need right at the start. This is why we may need to add data later on within the program. This is where the **.add** method comes into play.

```
List<String> computerParts = new ArrayList<>();

computerParts.add("i9 processor");
computerParts.add("GTX 3080");
computerParts.add("5TB SSD");
computerParts.add("16GB RAM stick");

System.out.println(computerParts);
/*output
[test1, test2, test3, i9 processor, GTX 3080, 5TB SSD, 16GB RAM stick]
*/
```

    - In the example above, you can see that all the data we added to our array is printed out. You can use this at any part of your program.

- **Removing data**
    - Similarly to how we may want to add data at a later point in our program, we may want to do the same thing with removing data from our array. To achieve this, we'll make use of the **.remove()** method. This takes in one parameter, which is the index of the value you want to remove.
    - Let's say that our last PC already has a 16GB ram stick; we no longer need to purchase this from the store, so we want to remove it from our list.

```
computerParts.remove(3);
System.out.println(computerParts);
/*output
[i9 processor, GTX 3080, 5TB SSD]
*/
```

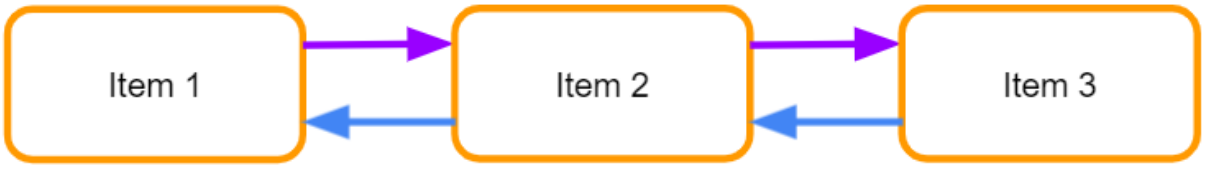
    - We removed the third element because the RAM stick was the 4th item in our list (as we count from 0 in programming).

LINKED LISTS

Linked lists differ from Arraylists in the sense of how they add and remove data. ArrayLists only contain one list, which is copied over to a new list after an element changes.

Linked lists data types are actually created with two lists. This is where performance is boosted compared to an ArrayList. Additionally, linked lists contain many of the same features as an ArrayList but with the combination of a queue (first in, first out).

So let's take a look at this in more detail.



Because of the way LinkedLists are created, the way elements are placed into a list is by taking the one/two nodes (item1, item2, and item3 are nodes) and setting the value between those two nodes. This is where it becomes more efficient than ArrayList because we're not entirely recreating the array. As we know, sometimes we won't know all the data we need right at the start. This is why we may need to add data later on within the program. This is where the **.add** method comes into play.

Declaring and creating LinkedLists

Because we're making use of the lists interface the way we create a linkedList is very similar to other list creations.

```
List<String> linkedList = new LinkedList<>();
```

/* If you want to know how to make a list that uses other data types and not just a string refer back to "Declaring and creating ArrayLists" */

Adding/Removing data from an LinkedList

Now that we've created an ArrayList, how do we add values to our list? Well, there are a few ways we can do this.

- **LinkedList default values**
    - Let's say we already have data that we know we want to be placed inside our array; we can use something known as **Arrays.asList** to achieve this.

```
List<String> testSubjects = new LinkedList<>(Arrays.asList("CN2971",
"KS8G"));
```

    - As you can see within our brackets at the end of our array creation, we've added the **asList** method. This can take any number of values, and these will automatically be placed inside the array when it's created.
- **Adding data later on**
    - As we know, sometimes we won't know all the data we need right at the start. This is why we may need to add data later on within the program. This is where the **.add** method comes into play.

```
List<String> testSubjects = new LinkedList<>(Arrays.asList("CN2971",
"KS8G"));

testSubjects.add("SP");
testSubjects.add("MMKM");
```

```
System.out.println(testSubjects);
/*output
* [CN2971, KS8G, SP, MMKM]
*/
```

- In the example above, you can see that all the data we added to our array is printed out. You can use this at any part of your program.

- **Removing data**
  - Similarly to how we may want to add data at a later point in our program, we may want to do the same thing with removing data from our array. To achieve this, we'll make use of the **.remove()** method. This takes in one parameter, which is the index of the value you want to remove.
  - Let's say that our last subject was released back into the wild, so we want to remove it from our list.

```
testSubjects.remove(3);

System.out.println(testSubjects);
/*output
* [CN2971, KS8G, SP]
*/
```

  - We removed the third element because the MMKM subject was the 4th item in our list (as we count from 0 in programming).

ArrayList vs LinkedList

So now that we've gone over these two arrays, which one should you use?

In many cases, you will almost always use an ArrayList. The benefits in terms of data searching and memory usage compared to a LinkedList are far more superior.

The times we use LinkedList is in the scarce case that it needs to be used in a specific algorithm, and even with that in mind, those events will only appear very rarely.

It's suggested that you play around with both and do your own research for both of these; however, ArrayLists will most likely be the most commonly used data structure you'll be using.

CONCEPT: SORTING AND SEARCHING DATA

Disorganised data does not provide us with information, as discussed in the previous lesson. We have to have methods for arranging data logically - sorting it - and for looking through it to find things (whether we want to just read an element, or insert/delete data at a specific point, for example inserting a Surname in the correct place in a set of data alphabetically sorted data about Employees). In this lesson, we're going to take a close look at the ways we can go about sorting and searching data.

CONCEPT: ALGORITHMS

As you know, an algorithm is a clear, defined way of solving a problem - a set of instructions that can be followed to solve a problem, and that can be repeated to solve similar problems. Every time you write code, you are creating an algorithm — a set of instructions that solves a problem.

There are a number of well-known algorithms that have been developed and extensively tested to solve common problems. In this lesson, you will be learning about algorithms used to sort and to search through a data structure that contains a collection of data (e.g. a List). In other words, in this lesson, you are going to learn about **algorithms** that manipulate the data in those data structures.

There are two main benefits that you should derive from learning about the sorting and searching algorithms in this lesson:

1. You will get to analyse and learn about algorithms that have been developed and tested by experienced software engineers, providing a model for writing good algorithms.
2. As these algorithms have already been implemented by others, you will see how to use them without writing them from scratch, i.e. you don't have to reinvent the wheel. Once you understand how they work, you can easily implement them in any coding language in which you gain proficiency.

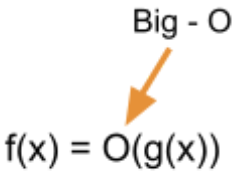
Let's consider algorithms for searching and sorting data.

Specific Outcome 2 (US 115373)

10. SORTING ALGORITHMS

If you want to sort a set of numbers into ascending order, how do you normally go about it? The most common way people sort things is to scan through the set, find the smallest number, and put it first; then find the next smallest number and put it second; and so on, continuing until they have worked through all the numbers. This is an effective way of sorting, but for a program, it can be time-consuming and memory-intensive.

There are a number of algorithms for sorting data, such as the Selection Sort, Insertion Sort, Shell Sort, Bubble Sort, Quick Sort, and Merge Sort. These all have different levels of operational complexity, and complexity to code. An important feature to consider when analysing algorithms and their performance is how efficient they are - how much they can effectively do in how long. With sorting algorithms, we're interested in knowing how quickly a particular algorithm can sort a list of items. We use something called **Big O notation** to evaluate and describe the efficiency of code. Big O notation, also called Landau's Symbol, is used in mathematics, complexity theory, and computer science. The O refers to *order of magnitude* - the rate of growth of a function dependent on its input. An example of Big O notation in use is shown below:



Don't be intimidated by the maths - you don't have to understand the intricacies of it to be able to understand algorithm efficiency using Big O Notation. We'll take a very brief, high-level look at it below, which should be enough to assist you in understanding the Big O complexities of the algorithms we look at in this lesson.

There are seven main mathematical functions that are used to represent the complexity and performance of an algorithm. Each of these mathematical functions is shown in the image below. Each function describes how an algorithm's complexity changes as the size of the input to the algorithm changes.

As you can see from the image, some functions (e.g. O(log n) or O(1)) describe algorithms where the efficiency of the algorithm *stays the same regardless of how big the input to the algorithm becomes* (shaded green to indicate excellent performance in the image). A sorting function with this level of Big O performance would continue to sort things really well and really quickly even when sorting very large datasets. Conversely, other functions (e.g. O(n!) and $O(2^n)$), describe a situation where the bigger the input gets, the worse the complexity and efficiency get. Sorting functions with this level of efficiency might work okay for small datasets, or datasets that are already largely ordered, but become very inefficient and slow as the size and/or degree of disorder in the input data they're handling increases.

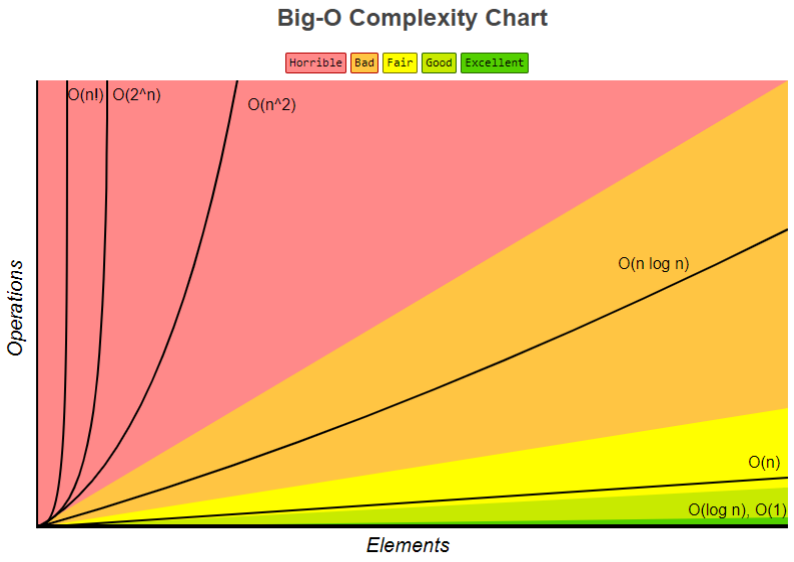


Image source: **http://bigocheatsheet.com/**

Keep the above image in mind and refer back to it as we discuss different sorting and searching algorithms and refer to their complexity and performance using Big O notation.

As previously stated, there are a number of algorithms for sorting data. The table below provides a comparison of the time complexity of some of these.

Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)

Image source: **http://bigocheatsheet.com/**

If you want to have a look at animated visual representations of sorting methods in action, **this is a great resource** that lets you run various sorts at your chosen speed, or step through them one operation at time so that you can trace exactly what is happening. Sitting down with the algorithm for a particular sorting method and stepping through an animated sort is a great way to understand it. You can further deepen your understanding by creating a **trace table** of the examples you're working with.

SORTING: DEEP DIVE

In the next section we will take a detailed look at three of the most popular sorting methods (and equally popular topics for interview questions!): Bubble Sort, Quick Sort, and Merge Sort.

Bubble Sort

Bubble Sort got its name because the sorting algorithm causes the larger values to 'bubble up' to the top of a list. Performing a Bubble Sort involves comparing an item (*a*) with the item next to it (*b*). If *a* is bigger than *b*, they swap places. This process continues until all the items in the list have been compared, and then the process starts again; this happens as many times as one less than the length of the list to ensure that enough passes through the list are made. Let's look at the example below:

8	3	1	4	7	**First iteration:** Five numbers in random order.
3	**8**	1	4	7	3 < 8 so 3 and 8 swap
3	1	**8**	4	7	1 < 8, so 1 and 8 swap
3	1	4	**8**	7	4 < 8, so 4 and 8 swap

3	1	4	7	**8**	7 < 8, so 7 and 8 swap (do you see how 8 has bubbled to the top?)
3	1	4	7	8	**Next iteration:**
1	**3**	4	7	8	1 < 3, so 1 and 3 swap. 4 > 3 so they stay in place and the iteration ends

Bubble Sort is $O(n^2)$ and is written below (Adamchick, 2009):

```
void bubbleSort(int ar[]) {
    for (int i = (ar.length - 1); i >= 0; i--) {
        for (int j = 1; j <= i; j++) {
            if (ar[j-1] > ar[j]) {
                int temp = ar[j-1];
                ar[j-1] = ar[j];
                ar[j] = temp;
            }
        }
    }
}
```

Practise this concept with some number sequences of your own. It can be a little tricky to comprehend initially, but becomes quite simple as you become more familiar with it.

Quick Sort

Quick Sort, as the name suggests, is a fast sorting algorithm. We start with a *pivot* point (usually the first element, but can be any element). We then look at the next element. If it is bigger than the pivot point, it stays on the right, and if it is smaller than the pivot point, it moves to the left of it. We do this for every number in the list going through the entire list once. At this point we have divided the array into values less-than-pivot, pivot, and greater-than-pivot. These sublists each get a new pivot point and the same iteration applies. This 'divide and conquer' process continues until each element has become a pivot point. See the example below:

4	3	2	8	7	5	1	**Pivot point:** 4
(**3**	2	1)	4	(**8**	7	5)	Divided list with new pivot points for each (3 and 8)
(**2**	1)	3	4	(**7**	5)	8	Divided list with new pivot points each (2 and 7)

(**1**)	2	3	4	(**5**)	7	8	Divided list with new pivot points each (1 and 5)
1	2	3	4	5	7	8	Nothing changes with above pivot points so the list is sorted

Quick Sort is O(log n) and is written below (Dalal, 2004):

```
public void quicksort(int[] list, int low, int high) {
    if (low < high) {
        int mid = partition(list,low,high);
        quicksort(list, low, mid-1);
        quicksort(list, mid+1, high);
    }
}
```

This is the partition method that is used to select the pivot point and move the elements around:

```
public int partition(int[] list, int low, int high) {

    // The pivot point is the first item in the subarray
    int pivot = list[low];

    // Loop through the array. Move items up or down the array so that they
    // are in the proper spot with regards to the pivot point.
    do {
        // can we find a number smaller than the pivot point? keep
        // moving the 'high' marker down the array until we find
        // this, or until high=low
        while (low < high && list[high] >= pivot) {
            high--;
        }
        if (low < high) {
            // found a smaller number; swap it into position
            list[low] = list[high];
            // now look for a number larger than the pivot point
            while (low < high && list[low] <= pivot) {
                low++;
            }

            if (low < high) {
                // found one! move it into position
                list[high] = list[low];
            }
```

```
        }
    } while (low < high);

    // move the pivot back into the array and return its index
    list[low] = pivot;
    return low;
}
```

Note how the partition method is actually doing most of the heavy lifting, while the quicksort method just calls the partition method. This is a bit confusing at first, but keep practising to help you get the hang of the concept.

Merge Sort

Like Quick Sort, Merge Sort is also a 'divide and conquer' strategy. In this strategy, we break apart the list to then put it back together in order. To start, we break the list into two and keep dividing until each element is on its own. Next, we start combining them two at a time and sorting as we go. Have a look at the example below. We start by dividing up the elements:

4	3	2	8	7	5	1	**List of numbers**
(4	3	2	8)	(7	5	1)	The list is divided into 2
(4	3)	(2	8)	(7)	(5	1)	Divided lists are divided again
(4)	(3)	(2)	(8)	(7)	(5)	(1)	Divided until each element is on its own

Next, we start to pair the elements back together, sorting as we go.

First merge

(4)	(3)	(2)	(8)	(7)	(5)	(1)	**Individual elements**
(4	3)	(2	8)	(7	5)	(1)	Elements are paired and compared
(3	4)	(2	8)	(5	7)	(1)	Sorted paired lists

Now we combine again. We sort as we combine by comparing the items:

Second merge part 1

(3	4)	(2	8)	
2				1. 3 vs. 2 — 3 > 2 so 2 becomes index 0
2	**3**			2. 3 vs 8 — 3 < 8 so 3 becomes index 1
2	3	**4**		3. 4 vs. 8 — 4 < 8 so 4 becomes index 2
2	3	4	**8**	4. 8 is leftover so becomes index 3

Second merge part 2

(5	7)	(1)	
1			5. 5 vs. 1 — 5 > 1 so 1 becomes index 0
1	**5**		6. 5 vs. 7 — 5 < 7 so 5 becomes index 1
1	5	**7**	7. 7 is leftover so becomes index 2

Finally, we sort and merge the last two lists:

Third merge

(2	3	4	8)	(1	5	7)	
1							1. 2 vs. 1 — 1 < 2 so 1 becomes index 0
1	**2**						2. 2 vs. 5 — 2 < 5 so 2 becomes index 1
1	2	**3**					3. 3 vs. 5 — 3 < 5 so 3 becomes index 2
1	2	3	**4**				4. 4 vs. 5 — 4 < 5 so 4 becomes index 3
1	2	3	4	**5**			5. 8 vs. 5 — 8 > 5 so 5 becomes index 4
1	2	3	4	5	**7**		6. 8 vs. 7 — 8 > 7 so 7 becomes index 5
1	2	3	4	5	7	**8**	7. 8 is leftover so becomes index 6

Merge Sort is O(n log(n)) and is written below (Dalal, 2004):

```
public void mergeSort(int[] list, int low, int high) {
    if (low < high) {
        // find the midpoint of the current array
        int mid = (low + high)/2;
        // sort the first half
        mergeSort(list, low, mid);
        // sort the second half mergeSort(list,mid+1,high);
        // merge the halves
        merge(list, low, high);
    }
}
```

The above method finds the midpoint of the array and recursively divides it until all the elements are the only ones in their own arrays. Once this is done the following method is called recursively to put them back together in order (Dalal, 2004):.

```
public void merge(int[] list, int low, int high) {

    // temporary array stores the ''merge'' array within the method.
     int[] temp = new int[list.length];

     // Set the midpoint and the end points for each of the subarrays
    int mid = (low + high)/2;
    int index1 = 0;
    int index2 = low;
    int index3 = mid + 1;

    // Go through the two subarrays and compare, item by item,
    // placing the items in the proper order in the new array
    while (index2 <= mid && index3 <= high) {
        if (list[index2] < list[index3]) {
        temp[index1] = list[index2];
        index1++; index2++;
        }
        else {
        temp[index1] = list[index3];
        index1++; index3++;
        }
    }

    // if there are any items left over in the first subarray, add them to
    // the new array
    while (index2 <= mid) {
    temp[index1] = list[index2];
    index1++;
```

```
    index2++;
    }

    // if there are any items left over in the second subarray, add them
    // to the new array
    while (index3 <= high) {
    temp[index1] = list[index3];
    index1++;
    index3++;
    }

    // load temp array's contents back into original array
    for (int i=low, j=0; i<=high; i++, j++) {
    list[i] = temp[j];
    }
}
```

As mentioned earlier, Merge Sort is used in the Collections Framework as `Collections.sort()`. For example:

```
import java.util.*;

public class Sort {
    public static void main(String[] args) {

        List<Integer> myList = new ArrayList<>();
        myList.add(10);
        myList.add(2);
        myList.add(23);
        myList.add(14);

        System.out.println("List: " + myList);

        Collections.sort(myList);
        System.out.println("Sorted List: " + myList);
    }
}
```

SPECIFIC OUTCOME 3 (US 115373)

11. SEARCHING ALGORITHMS

There are two main algorithms for searching through elements in code, namely **linear search** and **binary search.** Linear search is closest to how we, as humans, search for something. If we are looking for a particular book on a messy bookshelf, we will look through all of them until we find the one we're looking for, right? This works very well if the group of items are not in order, but it can be quite time-consuming. Binary search, on the other hand, is much quicker but it only works with an ordered collection.

SEARCHING: DEEP DIVE

In the next section we will take a detailed look at three of the most popular sorting methods (and equally popular topics for interview questions!): Bubble Sort, Quick Sort, and Merge Sort. Let us look at these two in more detail. If you want to have a look at visual representations of any of the searching methods above, have a look **here**.

Linear Search

As mentioned, linear search is used when we know that the elements are not in order. We start by knowing what element we want, and we go through the list comparing each element to the known element. The process stops when we either find an element that matches the known element, or we get to the end of the list. Linear search is O(n) and is executed below (Dalal, 2004):

```
public int sequentialSearch(int item, int[] list) {
    // if index is still -1 at the end of this method, the item is not
    // in this array.
    int index = -1;
    // loop through each element in the array. if we find our search
    // term, exit the loop.
    for (int i=0; i<list.length; i++) {
        if (list[i] == item) {
            index = i;
            break;
        }
    }
    return index;
}
```

Output:

```
List: [10, 2, 23, 14]
Sorted List: [2, 10, 14, 23]
```

Binary Search

Binary search works if the list is in order. If we go back to our book example, if we were looking for *To Kill a Mockingbird* and the books were organised in alphabetical order, we could simply go straight to 'T', refine to 'To' and so on until we found the book. This is how binary search works. We start in the middle of the list and determine if our sought-for element is smaller than (on the left of) or bigger than (on the right of) that element. By doing this we instantly eliminate half of the elements in the list to search through! We continue to halve the list until either we find our sought-after element, or until all possibilities have been eliminated, i.e. there are no elements found to match our sought-after element. Let's look at an example:

We are looking for the number 63 in the following list:

3	10	63	80	120	6000	7400	8000

In the list above, the midpoint is between 80 and 120, so the value of the midpoint will be 100 (*(120+80) ÷ 2 = 100*). 63 is less than 100, so we know 63 must be in the left half of the list:

3	10	63	80	~~120~~	~~6000~~	~~7400~~	~~8000~~

Let's half what's left. The midpoint is now 37.5, which is smaller than 63, so our element must be on the right side.

~~3~~	~~10~~	63	80	~~120~~	~~6000~~	~~7400~~	~~8000~~

We're left with our last two elements. The midpoint of 63 and 80 is 71.5, which is greater than 63, so our element must be on the left side.

~~3~~	~~10~~	63	~~80~~	~~120~~	~~6000~~	~~7400~~	~~8000~~

And it is! We've found our element.

Binary search is O(log n) (UCT, 2014) and is written below (Dalal, 2004):

```
public int binarySearch(int item, int[] list) {
    // if index = -1 when the method is finished, we did not find the
    // search term in the array
    int index = -1;

    // set the starting and ending indexes of the array; these will
    // change as we narrow our search
    int low = 0;
    int high = list.length-1;
    int mid;

    // Continue to search for the search term until we find it or
    // until our ''low'' and ''high'' markers cross
    while (high >= low) {
        mid = (high + low)/2; // calculate the midpoint of the current array
        if (item < list[mid]) { // value is in lower half, if at all
            high = mid - 1;
        }
        else if (item > list[mid]) {
            // value is in upper half, if at all
            low = mid + 1;
            }
        else {
                // found it! break out of the loop
            index = mid;
            break;
        }
    }
    return index;
}
```

It is worth noting that the Java Collections Framework uses the binary search algorithm in the form of `Collections.binarySearch()`. This means that you need to sort your list before you search it. See the example below:

```
import java.util.*;

public class BinarySearch{
   public static void main(String[] args) {

      List<String> list = new ArrayList<>();
```

```
        list.add("Java");
        list.add("Programming");
        list.add("Is");
        list.add("Fun");

        // Sort the list into alphabetical order
        Collections.sort(list);
        System.out.println("Sorted List: " + list);

        int index = Collections.binarySearch(list, "Java");
        System.out.println("'Java' in List is at " + index);

        index = Collections.binarySearch(list, "Python");
        System.out.println("'Python' in List is at " + index);
    }
}
```

Output:

```
Sorted List: [Fun, Is, Java, Programming]
'Java' in List is at 2
'Python' in List is at -5
```

Note what the output prints as 'Python's' index: -5. This is because if you were to insert this element into the list, it would have an index value of 4. It sounds strange but it is actually calculated using *(-(insertion_index) - 1)*, which is *(-(-5) -1)*.

Compulsory Task 2

- Create a Java file called **ArrayLists.java**
- Design a class called Album. The class should contain:
    - The data fields *albumName* (String), *numberOfSongs* (int) and *albumArtist* (String).
    - A constructor that constructs a Album object with the specified *albumName*, *numberOfSongs,* and *albumArtist*.
    - The relevant *get* and *set* methods for the data fields.
    - A *toString()* method that returns a string that represents an Album object in the following format:
      (*albumName, albumArtist, numberOfSongs*)

- Create a new ArrayList called *albums1*, add 5 albums to it and print it out *(You may want to look back to the Java Collections Framework task for guidance)*.

- Sort the list according to the *numberOfSongs* and print it out.

- Swap the element at position 1 of *albums1* with the element at position 2 and print it out.

- Create a new ArrayList called *albums2*.

- Using the *addAll* method add 5 albums to the *albums2* List and print it out.

- Copy all of the albums from *albums1* into *albums2*.

- Add the following two elements to *albums2*:
    - (Dark Side of the Moon, Pink Floyd, 9)
    - (Oops!... I Did It Again, Britney Spears, 16)

- Sort the courses in *albums2* alphabetically according to the album name and print it out.

- Search for the album “Dark Side of the Moon” in *albums2* and print out the index of the album in the List.

Compulsory Task 3

1. Using the following ArrayList: [“blue”, “six”, “hello”, “game”, “unorthodox”, “referee”, “ask”, “zebra”, “run”, “flex”]
    - Create a Java file called **BubbleSort.java**
    - Implement the Bubble sort algorithm on the ArrayList and print out the sorted list.

Compulsory Task 4

1. Using the following array: [27, -3, 4, 5, 35, 2, 1, -40, 7, 18, 9, -1, 16, 100]
    - Create a Java file called **Sort&Search.java**
      Which searching algorithm would be appropriate to use on the given list?
    - Implement this searching algorithm to search for the number 9. Add a comment to explain why you think this algorithm was a good choice.
    - Research and implement the Insertion sort on this array.
    - Implement the searching algorithm you haven’t tried yet in this Task on the sorted array to find the number 9. Add a comment to explain where you would use this algorithm in the real world.

Self-assessment table

Intended learning outcome (SAQA US 115373)	Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- **I can successfully:**	
Demonstrate an understanding of how abstract data types are stored on computers (SO1).	Identify different abstract data types used in computer programming. Range: Including but not limited to: Queue, stack, graph, tree data types in a computer.	Identify different data structures used to store abstract data types in a computer. Range: Including but not limited to: Arrays, lists, linked lists.
Demonstrate an	Demonstrate different types of	Demonstrate the working of

understanding of sort techniques used to sort data held in data structures (SO2).	sort techniques Range: Including but not limited to: Selection sort, Insertion sort, Bubble sort (at least 2)		different types of sort techniques Range: Including but not limited to: Selection sort, Insertion sort, Bubble sort (at least 2)	
Demonstrate an understanding of search techniques (SO3).	Demonstrate (identify and apply) different types of search techniques Range: Including but not limited to: Binary search		Explain the working of different types of search techniques Range: Including but not limited to: Binary search	
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**:			
Understand sorting algorithms	Understand how bubble sort works and be able to implement it in code	Understand how merge sort works and be able to implement it in code	Understand how merge sort works and be able to implement it in code	
Understand searching algorithms	Understand how linear search works	Implement linear sort in code	Understand how binary search works	Implement binary search in code
Understand computational complexity	Calculate the computational complexity of sorting algorithms		Calculate the computational complexity of searching algorithms	

Design and Build a Relational Database

Aligned to SAQA US114048

Introduction

WELCOME TO THE DESIGNING AND BUILDING A RELATIONAL DATABASE LESSON!

In this lesson, we introduce the relational database and related concepts, including data independence, data modelling, and types of database languages. We will also discuss how to evaluate and design good table structures with normalisation to control data redundancies, and therefore avoid data anomalies, and look at the concept, role, and importance of indexing. At the end there is a task for you to put into practice what you have learned.

INTENDED LEARNING OUTCOMES

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

- Understand and explain key data concepts
- Understand and explain key relational database concepts
- Understand and explain normalisation
- Normalise data to 3NF

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.

SPECIFIC OUTCOME 1 (US 114048)

1. REQUIREMENTS FOR DATABASE ACCESS FOR A COMPUTER APPLICATION USING SQL

There are a number of concepts you need to understand in order to be able to properly consider the requirements for database access for a computer application using SQL. These are explained below, drawing from the Bella Visage Learner Guide for the *National Certificate: Information Technology (Systems Development)*.

1.1 DATA CONCEPTS

ABSTRACTION

Abstraction is the process of separating ideas from specific instances of those ideas at work. Computational structures are defined by their meanings (semantics), while hiding away the details of how they work. Abstraction tries to factor out details from a common pattern so that programmers can work close to the level of human thought, leaving out details which matter in practice, but are immaterial to the problem being solved. For example, a system can have several abstraction layers whereby different meanings and amounts of detail are exposed to the programmer; low-level abstraction layers expose details of the computer hardware where the program runs, while high-level layers deal with the business logic of the program.

Abstraction captures only those details about an object that are relevant to the current perspective; in both computing and in mathematics, numbers are concepts in programming languages. Numbers can be represented in myriad ways in hardware and software, but, irrespective of how this is done, numerical operations will obey identical rules.

Abstraction can apply to control or to data: **Control abstraction** is the abstraction of actions while **data abstraction** is that of data structures.

- Control abstraction involves the use of subprograms and related concepts' control flows
- Data abstraction allows handling data bits in meaningful ways. For example, it is the basic motivation behind data typing (strings, integers, floats, etc).

The recommendation that programmers use abstractions whenever suitable in order to avoid duplication (usually of code) is known as the **abstraction**

principle. The same term is used to describe the requirement that a programming language provide suitable abstractions.

DATA INDEPENDENCE

One of the biggest advantages databases offer is data independence. It means we can change the conceptual schema at one level without affecting the data at another level. We can change the structure of the database without affecting the data required by users and programs. Data independence is essentially the principle that each higher level of the data architecture is immune to changes to the next, lower level of the architecture. The logical scheme stays unchanged even though the storage space or type of some data may change for reasons of optimization or reorganization. Here, the external schema does not change. Internal schema changes may be required if physical schema are reorganized. Physical data independence is present in most databases and file environments in which aspects such as the hardware storage of data encoding, the exact location of data on the disk, the merging of records, and so on, are hidden from the user.

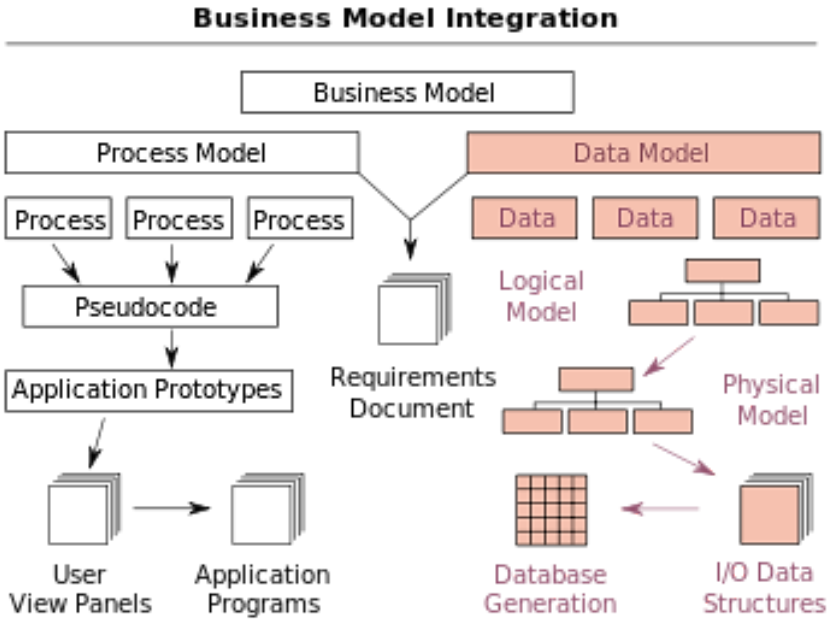
Types of data independence

Data can be considered physically independent or logically independent.

- **Physical data independence** is the ability to modify a physical schema without causing application programs to require rewriting. Modifications at the physical level are occasionally necessary to improve performance. When physical data independence exists, this means we change the physical storage/level without affecting the conceptual or external view of the data. The new changes are absorbed by mapping techniques.
- **Logical data independence** is the ability to modify the logical data schema without causing application programs to require rewriting. Modifications at the logical level are necessary whenever the logical structure of the database is altered (for example, when a new account type such as money-market accounts are added to a banking system). Logical Data independence means that we can e.g. add some new fields to a database requiring adding columns to one or more tables, or remove fields requiring removing some columns from one or more tables, without the user view and associated programs requiring any changes. Logical data independence is more difficult to achieve than physical data independence, as application programs are heavily dependent on the logical structure of the data that they access.

DATA MODELS

The term “data models” is used in two related senses. In the first, a data model is a description of the objects represented by a computer system together with their properties and relationships; these are typically "real world" objects such as products, suppliers, customers, and orders. In the second sense, a data model is a collection of concepts and rules used in defining how data are stored and related,: for example the relational model uses relations and tuples, while the network model uses records, sets, and fields.



The figure above is an example of the interaction between process and data models. Here, the data model is based on data, data relationships, data semantics (the meaning of the data), and data constraints. A data model provides the details of information to be stored, and is of primary use when the final product is the generation of computer software code.

Data models are often used as an aid to communication between business people defining the requirements for a computer system and technical people defining the design in response to those requirements. They are used to show the data needed and created by business processes.

A data model is essentially a navigation tool for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment.

A data model explicitly determines the structure of data. Data models are specified in a data modelling notation, which is often graphical in form.
A data model can be sometimes referred to as a data structure, especially in the context of programming languages. Data models are often complemented by function models, especially in the context of enterprise models.

DATABASE LANGUAGES
Database languages are special-purpose languages, which either define data, manipulate data, or query data (three types).

Data Definition Language (DDL)
A data definition language (DDL) defines data types and the relationships among them.

Data manipulation language (DML)
A data manipulation language (DML) performs tasks such as inserting, updating, updating, or deleting data in a database. Performing read-only queries of data is sometimes also considered a component of DMLs. DMLs are divided into two types, procedural programming and declarative programming. DMLs were initially only used within computer programs, but with the advent of SQL have come to be used interactively by database administrators and end users.

Query languages
A query language, such as **SQL** (Structured Query Language) facilitates searching for information and computing derived information, enabling users to ask questions of the database and get answers in which data specific to the question is presented in such a way as to provide information.

SQL
You will learn much more about SQL in the next unit, entitled **Introduction to SQL**. Here, we are just going to provide a brief overview of SQL in the context of a DML/DDL and a query language. SQL is used to retrieve and manipulate data in a relational database. As a DML, SQL consists of data change statements. which modify stored data but not the schema or database objects. Manipulation of persistent database objects, e.g., tables or stored procedures, via the SQL schema statements rather than the data stored within them, is considered to be part of a separate DDL. In SQL these two categories are similar in their detailed syntax, data types, expressions etc., but distinct in their overall function.

Data manipulation languages have their functional capability organized by the initial word in a statement, which is almost always a verb. In the case of SQL, these verbs are:

- SELECT ... FROM ... WHERE ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...

The purely read-only SELECT query statement is classed with the 'SQL-data' statements and so is considered by the standard to be outside of DML. The INTO form, however, is considered to be DML because it manipulates (i.e. modifies) data. In common practice though, this distinction is not made and SELECT is widely considered to be part of DML.

Most SQL database implementations extend their SQL capabilities by providing imperative functionality, i.e. procedural languages. Examples of these are Oracle's PL/SQL and DB2's SQL_PL.

DMLS tend to have many different flavours and capabilities between database vendors. There have been a number of standards established for SQL by ANSI, but vendors still provide their own extensions to the standard while often also not implementing the entire standard.

1.2 RELATIONAL DATABASE CONCEPTS

WHAT IS A RELATIONAL DATABASE?

As mentioned in the first lesson, a relational database is a database that organises data as a set of formally described tables. Data can then be accessed or reassembled from these tables in many different ways without having to reorganise the database tables.

Each table in a relational database has a unique name and may relate to one or more other tables in the database through common values. These tables contain rows (records) and columns (fields). Each row contains a unique instance of data for the categories defined by the columns. For example, a table that describes a *Customer* can have columns for *Name, Address, Phone Number,* and so on. Rows are sometimes referred to as tuples and columns are sometimes referred to as attributes. A relationship is a link between two tables. Relationships make it possible to find data in one table that pertains to a specific record in another table.

Tables in a relational database often contain a primary key, which is an index for a column or group of columns used as a unique identifier for each row in the table (you will learn more about indexing, and indices, in the next section). For example, a *Customer* table might have a column called *CustomerID* that is unique for every row. This makes it easy to keep track of a record over time and to associate a record with records in other tables.

Tables may also contain foreign keys, which are columns that link to primary key columns working as indices in other tables, thereby creating a relationship. For example, the *Customers* table might have a foreign key column called *SalesRep* that links to *EmployeeID*, which is the primary key in the *Employees* table.

A Relational Database Management System (RDBMS) is the software used for creating, manipulating, and administering a database. Most commercial RDBMSes use the Structured Query Language (SQL). SQL queries are the standard way to access data from a relational database. SQL queries can be used to create, modify, and delete tables as well as select, insert, and delete data from existing tables. You will learn more about SQL in a later lesson.

Specific Outcome 2 (US 114048)

12. REQUIREMENTS FOR DATABASE ACCESS FOR A COMPUTER APPLICATION USING SQL

There are a number of concepts you need to understand in order to be able to properly design database access for a computer application using SQL. These are explained below, drawing from the Bella Visage Learner Guide for the *National Certificate: Information Technology (Systems Development)*.

INDEXING

Simple indices

Given the fundamental importance of indices in databases, it is unfortunate how often the proper design of indices is neglected. It often turns out that the programmer understands detail, but not the broad picture of what indices do.

What is an index?

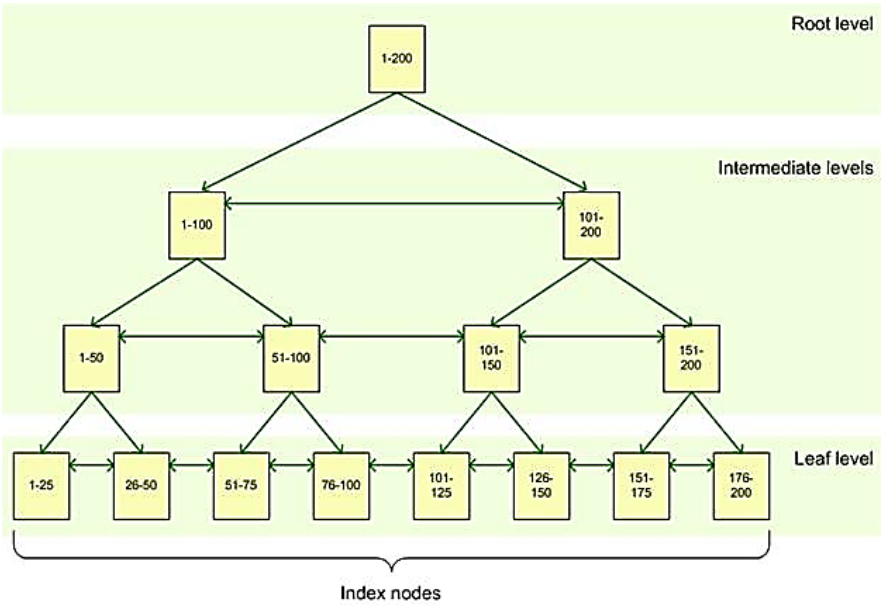
One of the most important routes to high performance in a SQL database is the index. Indices speed up the querying process by providing swift access to rows in the data tables, similarly to the way a book's index helps you find information quickly within that book. Most of this information applies to indices in software like SQL Server, and the underlying structure remains relatively the same across versions thereof, and similar programs.

Index Structures

indices are created on columns in tables or views. The index provides a fast way to look up data based on the values within those columns. For example, if you create an index on the primary key and then search for a row of data based on one of the primary key values, SQL first finds that value in the index, and then uses the index to quickly locate the entire row of data. Without the index, a table scan would have to be performed in order to locate the row, which can have a significant effect on performance.

You can create indices on most columns in a table or a view. You can also create indices on XML columns, but those indices are slightly different from the basic index and are beyond the scope of this lesson.

An index is made up of a set of pages (index nodes) that are organized in a B-tree structure. This structure is hierarchical in nature, with the root node at the top of the hierarchy and the leaf nodes at the bottom, as shown in the figure below.



B-tree structure of a SQL Server index

Index Design

As beneficial as indices can be, they must be designed carefully. Because they can take up significant disk space, you don't want to implement more indices than necessary. In addition, indices are automatically updated when the data rows themselves are updated, which can lead to additional overhead and can affect performance. As a result, index design should take into account a number of considerations.

Database considerations

As mentioned above, indices can enhance performance because they can provide a quick way for the query engine to find data. However, you must also take into account whether and how much you're going to be inserting, updating, and deleting data. When you modify data, the indices must also be modified to reflect the changed data, which can significantly affect performance. You should consider the following guidelines when planning your indexing strategy:

- For tables that are heavily updated, use as few columns as possible in the index, and don't over-index the tables.

- If a table contains a lot of data but data modifications are low, use as many indices as necessary to improve query performance. However, use indices judiciously on small tables because the query engine might take longer to navigate the index than to perform a table scan.

- For clustered indices, try to keep the length of the indexed columns as short as possible. Ideally, try to implement your clustered indices on unique columns that do not permit null values. This is why the primary key is often used for the table's clustered index, although query considerations should also be taken into account when determining which columns should participate in the clustered index.

- The uniqueness of values in a column affects index performance. In general, the more duplicate values you have in a column, the more poorly the index performs. On the other hand, the more unique each value, the better the performance. When possible, implement unique indices.

- For composite indices, take into consideration the order of the columns in the index definition. Columns that will be used in comparison expressions in the WHERE clause (such as WHERE FirstName = 'Charlie') should be listed first. Subsequent columns should be listed based on the uniqueness of their values, with the most unique listed first.

- You can also index computed columns if they meet certain requirements. For example, the expression used to generate the values must be deterministic (which means it always returns the same result for a specified set of inputs).

Planning for Queries

Another consideration when setting up indices is how the database will be queried - in other words, how users will be able to ask questions of the database and get resultant, answering, data returned to them by the query language. As mentioned above, you must take into account the frequency of data modifications. In addition, you should consider the following guidelines:

- Try to insert or modify as many rows as possible in a single statement, rather than using multiple queries.
- Create no clustered indices on columns used frequently in your statement's predicates and join conditions.
- Consider indexing columns used in exact-match queries.

Multi-Level indices

The indexing described so far involves ordered index files. A binary search is applied to the index to locate pointers to disk blocks or records in the file having a specific index field value.

A binary search (recall the ***Sorting and Searching*** unit) requires $\log_2 b_i$ block accesses for an index with b_i blocks. Each step of the binary search reduces the part of the index file we search by a factor of 2. (Each step divides the search space by 2, or halves the search space).

With multilevel indices, the idea is to reduce the part of the index that we continue to search by a larger factor, the blocking factor of the index, where the bfr_i is greater than 2. The blocking factor of the index, bfr_i is called the **fan-out**, or fo of the multilevel index.

Searching a multilevel index requires approximately ($\log_{fo} b_i$) block accesses which is a smaller number than for a binary search if the fan-out is larger than 2. If the fan out is equal to 2, there is no difference in the number of block accesses.

The index file is called the first level of a multilevel index. It is an ordered file with a distinct value for each key value K(i). Therefore we can create a ***primary index*** for the first level. This index to the first level is called the second level of the multilevel index.

The second level is a primary index, therefore we can use block anchors, and the second level has ***one entry for each block*** in the first level index. The blocking factor for all other levels is the same as for the first level, because the

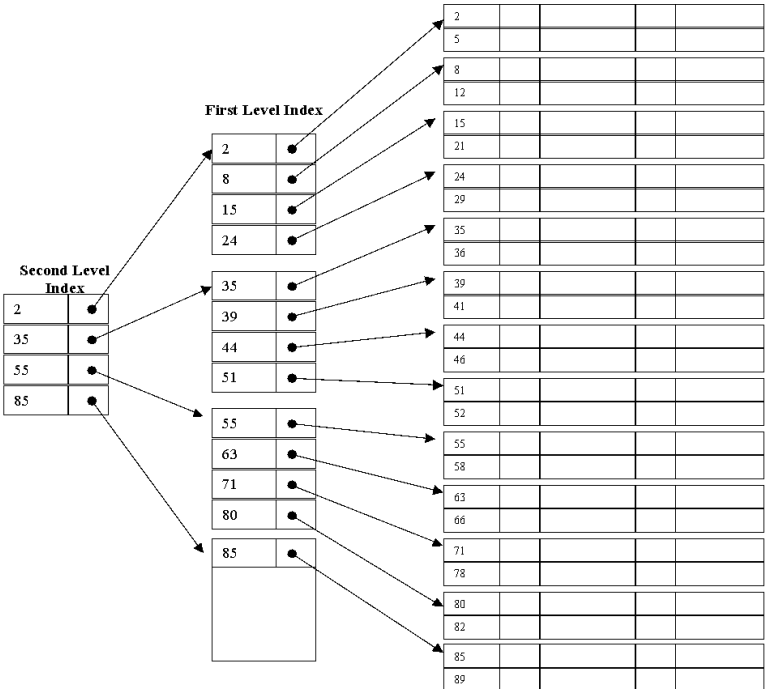
size of each index entry is the same. Each entry has one field value, and one block address.

If the first level has r_1 entries, and the blocking factor (which is also the fan out) for the index is $bfr_i = fo$, then the first level needs $\lceil r_1/fo \rceil$ blocks, which is therefore the number of entries r_2 needed at the second level of the index. If necessary, this process can be repeated at the second level. The third level, which is an index for the second level, has an entry for each second level block, so the number of third level entries is $r_3 = \lceil r_2/fo \rceil$. We only require a second level only if the first level needs more than one block of disk storage, and we require a third level only if the second level requires more than one block as well. This process can be repeated until all the index entries at some level *t* fit in a single block.

The block at the t^{th} level is the top-level index. Each level reduces the number of entries from the previous level by a factor of fo (the index fan out or blocking factor of the index). A multilevel index with r1 first level entries will have approximately t levels, where:

$t = \lceil \log_{fo}(r_1) \rceil$.

The multilevel indices can be used on any type of index, primary, clustering, or secondary, as long as the first level index has distinct values for K(i), and fixed length entries.



Example of a two level Primary Index

Example Question

Assume the dense secondary index from example 2 is converted into a multilevel index. The index blocking factor is bfr_i = 68 index entries per block. This value is also the fan out for the multilevel index. The number of first level blocks, b_1 was calculated as 442.
How many second level blocks?

$b_2 = \lceil b_1/fo \rceil = \lceil 442/68 \rceil = 7$ blocks.

How many third level blocks?

$b_3 = \lceil b_2/fo \rceil = \lceil 7/68 \rceil = 1$ block.

Because all of the index entries at the third level can fit into one block, the third level is the top level of the index, and t = 3. (Remember: t is the number of levels required)

To access a record by searching the multilevel index, we must access one block at each level, plus one block from the data file, so we need t + 1 = 3 + 1 = 4 block accesses. In example 2, using a single level index with a binary search, we required 10 block accesses.

In this example, a dense secondary index was used, meaning there was an index entry for each record in the table. You could also have a multilevel primary index which is non-dense (for example if the first level is a primary index).

In this case, we must access the data block from the file before we can determine whether the record being searched for is in the file. With a dense index, this can be determined by accessing the first index level, without having to access the data file, since there is an entry in the index for each record in the file.

Using multi-level indices, we can reduce the number of block accesses required to search for a record given its indexing field value. Insertions and deletions are still a problem because all the index levels are physically ordered files.

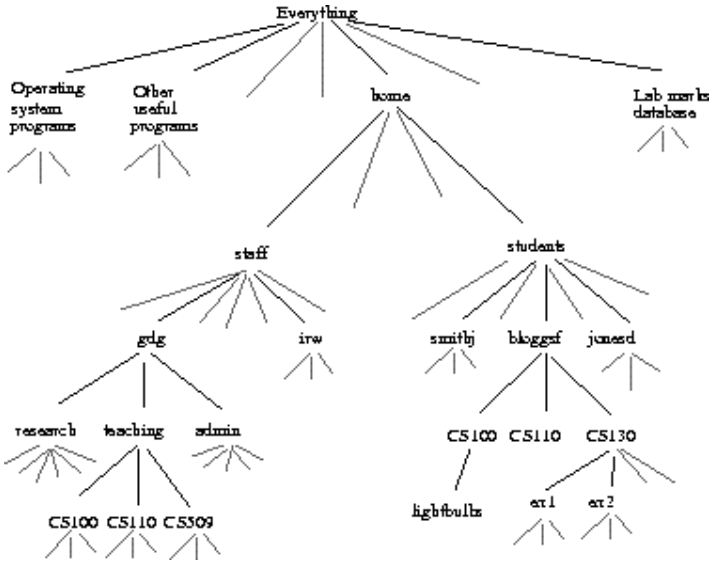
ISAM (Indexed Sequential Access Method) is a file management system developed at IBM that allows records to be accessed either sequentially (in the order they were entered) or randomly (with an index). Each index defines a different ordering of the records. An employee database may have several indices, based on the information being sought. For example, a name index may order employees alphabetically by last name, while a department index may order employees by their department. A key is specified in each index. For an alphabetical index of employee names, the last name field would be the key.

ISAM was developed prior to VSAM (Virtual Storage Access Method) and relational databases.

A tree

A widely used abstract data type (ADT) or data structure implementing this ADT is the tree data structure that simulates a hierarchical tree, with a root value and sub trees of children, represented as a set of linked nodes. You will have encountered this under abstract data types in the ***Sorting and Searching*** unit. As you learned previously, a tree data structure can be defined recursively (locally) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes

(the "children"), with the constraints that no reference is duplicated, and none points to the root.



Alternatively, a tree can be defined abstractly as a whole (globally) as an ordered tree, with a value assigned to each node. Both these perspectives are useful: while a tree can be analysed mathematically as a whole, when actually represented as a data structure it is usually represented and worked with separately by node (rather than as a list of nodes and an adjacency list of edges between nodes, as one may represent a digraph, for instance). For example, looking at a tree as a whole, one can talk about "the parent node" of a given node, but in general as a data structure a given node only contains the list of its children, but does not contain a reference to its parent (if any).

2.2 ADVANCED RELATIONAL DATABASE CONCEPTS

DATA REDUNDANCY

When the same data is stored in different places for no reason, this is called data redundancy. This could cause issues when the same versions of data are not updated consistently. This could cause different versions of the same information to be kept in a database, which could lead to misinformation when trying to retrieve the most recent data. This is known as **data inconsistency**. Redundancy could also negatively impact the security of the data because

having multiple versions of something increases the risk of someone being able to access it without authorisation.

Redundancies can also lead to abnormalities in the data, known as anomalies. **Data anomalies** can occur for when changes to the data are unsuccessful. For example:

- *Update anomalies:* Occur when the same information is recorded in multiple rows. For example, in an Employee table, if the office number changes, then there are multiple updates that need to be made. If these updates are not successfully completed across all rows, then an inconsistency occurs.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		

- *Insertion anomalies:* When there is data we cannot record until we know information for the entire row. For example, we cannot record a new sales office until we also know the salesperson because, in order to create the record, we need to provide a primary key. In our case, this is the EmployeeID.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		
???	???	Atlanta	312-555-1212			

- *Deletion anomalies:* When deletion of a row can cause more than one set of facts to be removed. For example, if John Hunt retires, deleting that row causes us to lose information about the New York office.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
~~1004~~	~~John Hunt~~	~~New York~~	~~212-555-1212~~	~~Dell~~	~~HP~~	~~Apple~~
1005	Martin Hap	Chicago	312-555-1212	Boeing		

NORMALISATION

The aim of normalisation is to correct the tables in a relational database to reduce anomalies by eliminating redundancies. To achieve normalisation, we work through up to six stages called **normal forms**: 1NF (first normal form), 2NF, 3NF, etc. up to 6NF. However, it is rarely necessary to go through all six stages. It is most common to go through the first three (1NF to 3NF). Therefore, in the example below, we will be working through the first three stages. By working through these stages, each version of the tables becomes structurally better than the previous stage.
By the end of normalisation, the final tables should show the following:

- Each table represents a single subject. For example, an Employee table will only contain data that directly pertains to employees.
- No data is unnecessarily stored in more than one table.
- All attributes in a table are dependent on a primary key.

Conversion to First Normal Form (1NF)

Normalisation starts with a simple three-step procedure.

Step 1: Eliminate the repeating groups

To start, the data must be formatted to a table. It is important here that there are no repeating groups of data. Each cell must contain a maximum of one value.

Table name: MUSIC

SONG_NAME	SONG_ARTIST	SONG_ALBUM	SONG_GENRE	REC_LBL	SONG_PRDR
Hello	Adele	25	Soul	XL Recordings	Greg Kurstin
Sunflower	Post Malone	Spiderman: Into the Spider-Verse	Pop	Republic	Carter Lang
Wow.	Post Malone	Hollywood's Bleeding	Hip-Hop	Republic	Louis Bel
One	Ed Sheeran	x	Folk-pop	Asylum	Jake Gosling
Bad Guy	Billie Eilish	When We Fall Asleep, Where Do We Go?	Alternative	Dark Room	Finneas O'Connell
One	Metallica	...And Justice for All	Metal	One on One Studios	Metallica

Hello	Eminem	Relapse	Hip-Hop	Aftermath Shady Interscope	Dr Dre
7 Rings	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown
Thank U, Next					

Take a look at the table above. Note that some songs could reference more than one data entry. For example, a song named “Hello” could refer to one of two songs, each with a different artist. To make sure there is no repeated data, we need to fill in the blanks to make sure there are no null values in the table. Looking at the last row, the empty cells contain the same information as the row above because the two songs come from the same album. Therefore, the filled-in table will look like this:

Table name: 1NF_MUSIC

SONG_NAME	SONG_ARTIST	SONG_ALBUM	SONG_GENRE	REC_LBL	SONG_PRDR
Hello	Adele	25	Soul	XL Recordings	Greg Kurstin
Sunflower	Post Malone	Spiderman: Into the Spider-Verse	Pop	Republic	Carter Lang
Wow.	Post Malone	Hollywood’s Bleeding	Hip-hop	Republic	Louis Bel
One	Ed Sheeran	x	Folk-pop	Asylum	Jake Gosling
Bad Guy	Billie Eilish	When We Fall Asleep, Where Do We Go?	Alternative	Dark Room	Finneas O’Connell
One	Metallica	...And Justice for All	Metal	One on One Studios	Metallica
Hello	Eminem	Relapse	Hip-Hop	Aftermath Shady Interscope	Dr. Dre
7 Rings	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown
Thank U, Next	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown

Step 2: Identify the primary key

Next, we need to identify the primary key. The primary key needs to be able to identify one row of the table, thereby identifying all the remaining rows, or attributes, when needed. For example, SONG_NAME is not a good primary key because “One” and “Hello” each identify 2 different rows. Similarly, SONG_ARTIST would not be a good primary key because “Ariana Grande” and “Post Malone” each identify 2 rows. Therefore, we could use a combination of SONG_NAME and SONG_ARTIST to be able to narrow down our search to a single row. Now, if you know that SONG_NAME = “Hello and SONG_ARTIST = “Adele”, the entries for the attributes can only be “25”, “Soul”, “XL Recordings”, and “Greg Kurstin”

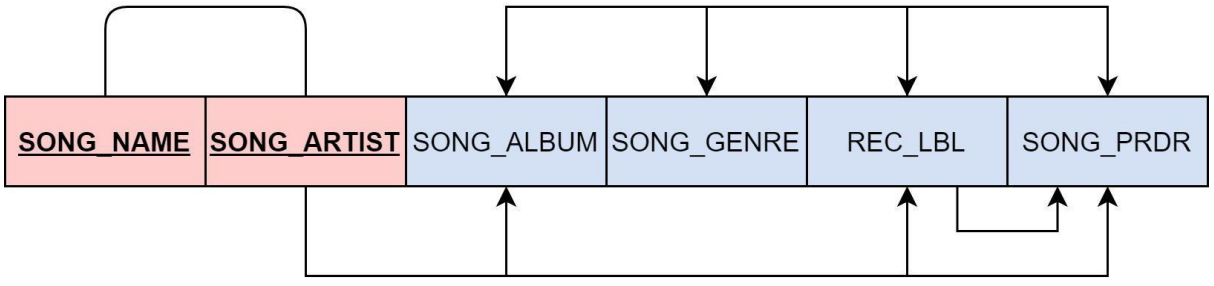
Step 3: Identify all dependencies

The example above, step 1 shows us the following relationship:

SONG_NAME, ARTIST_NAME → SONG_ALBUM, SONG_GENRE, REC_LBL, SONG_PRDR

This means that SONG_ALBUM, SONG_GENRE, REC_LBL and SONG_PRDR are dependent on the combination of SONG_NAME and ARTIST_NAME. However, looking at the table, we could also see that the song artist determines the album name, the record label and the record producer (SONG_ARTIST → SONG_ALBUM, REC_LBL, SONG_PRDR). These are known as partial dependencies because only one of the two primary keys is needed to determine the other attributes.

Finally, knowing the record label means knowing the song’s producer (REC_LBL → SONG_PRDR). This is known as a partial dependency because it is not based on a primary attribute. For a visual representation of this, look at the dependency diagram below:



Note that:

1. The primary key attributes are underlined and shaded in a different colour.

2. The arrows above the attributes indicate all desirable dependencies. Desirable dependencies are those based on the primary key. Note that the entity's attributes are dependent on the combination of SONG_NAME and ARTIST_NAME.

3. The arrows below the diagram indicate partial dependencies and transitive dependencies. These are less desirable than those based on the primary key

We have now put the table in 1NF because:

- All of the key attributes are defined
- There are no repeating groups in the table
- All attributes are dependent on the primary key

Conversation to Second Normal Form (2NF)

The relational database design can be improved by converting the database into a format known as the second normal form. Here, we try to rid the tables of partial dependencies, as they can cause anomalies.

Step 1: Write each key component on a separate line

Write each key component on a separate line, then write the original (composite) key on the last line:

SONG_NAME
SONG_ARTIST
SONG_NAME SONG_ARTIST

You may have noticed from the diagram above that SONG_ARTIST is not a primary key on its own for any attribute, therefore giving it its own new table isn't necessary. Therefore, the keys that will have their own tables are SONG_NAME and SONG_NAME SONG_ARTIST. We will name these two tables ARTIST and GENRE (because the table is used to determine the genre) respectively.

Step 2: Assign corresponding dependent attributes

Then use the dependency diagram above to determine those attributes that are dependent on other attributes by looking at the arrows underneath the diagram. With this, we create three new tables:

ARTIST (SONG_ARTIST, SONG_ALBUM, REC_LBL, SONG_PRDR)
GENRE (SONG_NAME, SONG_ARTIST, SONG_GENRE)

The dependency diagram below shows the result of Steps 1 and 2.

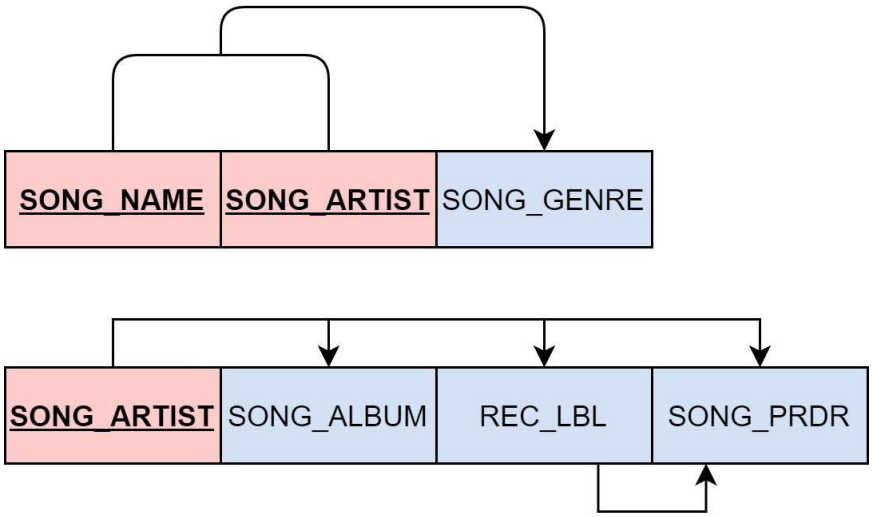


Table name: GENRE, Table name: ARTIST

Note how there are no longer any partial dependencies that could cause anomalies in the tables. However, there is still transitive dependency in the Employee table.

We have now put the table into 2NF because:

- It is in 1NF
- It includes no partial dependencies because no attribute in each table is dependent on only a part of a primary key

Conversation to Third Normal Form (3NF)

We will now eliminate the transitive dependencies.

Step 1: Identify each new determinant

Write the determinant of each transitive dependency as a primary key in its own table. In this example, the determinant will be REC_LBL.

Step 2: Identify the dependent attributes

As mentioned, SONG_PRDR is dependent on REC_LBL (REC_LBL → SONG_PRDR). Therefore, we can name the new table Label.

Step 3: Remove the dependent attributes from transitive dependencies

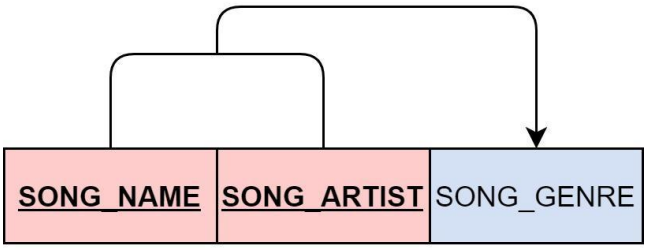
Eliminate all dependent attributes in the transitive relationship from each of the tables that have such a relationship. In this example, we eliminate SONG_PRDR from the ARTIST table shown in the dependency diagram above to leave the ARTIST table dependency definition as SONG_ARTIST → SONG_ALBUM, REC_LBL.

Notice that the REC_LBL remains in the ARTISTtable to serve as the foreign key. You can now draw a new dependency diagram to show all of the tables you have defined in the steps above. Then check the new tables as well as the tables you modified in Step 3 to make sure that each table has a determinant and that the tables don't contain inappropriate dependencies. The new dependency diagram should look as follows:

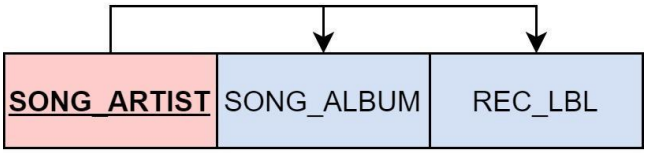
Table name: GENRE

Table name: ARTIST

Table name: LABEL

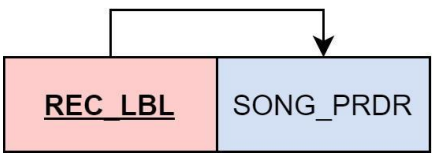


This conversation has eliminated the original ARTIST table's transitive dependency. The tables are now said to be in third normal form (3NF).



We have now put the table into 3NF because:

- It is in 2NF
- It contains no transitive dependencies



Compulsory Task 5

Answer the following questions:

- Define a database and the three database language types covered in this lesson (8)
- Using the INVOICE table given below, draw its dependency diagram and identify all dependencies (including transitive and partial dependencies). You can assume that the table does not contain any repeating groups and that an invoice number references more than one product.
  Hint: This table uses a composite primary key

INV_NUM	PROD_NUM	SALE_DATE	PROD_LABEL	VEND_CODE	VEND_NAME	QUANT_SOLD	PROD_PRICE
211347	AX-P126VBB	15-Jun-2018	Lawnmower	111	Mowers and Bulbs	1	R4099.90
211347	GF-5106D2F	15-Jun-2018	Lightbulb	111	Mowers and Bulbs	8	R24,99
211347	RB-163698G	15-Jun-2018	Blue paint	063	All Things Paint	1	R80,90
211348	AX-P126VBB	15-Jun-2018	Lawnmower	111	Mowers and Bulbs	2	R4099.90
211349	TF-7465023Q	17-Jun-2018	2m ladder	207	Tall Things	1	R699,99

- Draw new dependency diagrams to show the data in 2NF.
- Draw new dependency diagrams to show the data in 3NF.
- Use this table to illustrate one of each kind of anomaly described in the task. In a text file called **anomalies.txt**, how you would change the table and which anomaly that change would create.

Self-assessment table

Intended learning outcome	Assess your own proficiency level for each intended learning outcome. Check each box when the statement

(SAQA US 11048)	becomes true- **I can successfully**:				
Review the requirements for database access for a computer application using SQL (SO1)	Understand and explain Abstraction	Understand and explain Data independence (physical and logical)	Understand and explain Data models	Understand and explain DDLs and DMLs	
Design database access for a computer application using SQL (SO2)	Understand and use simple indices	Understand and use multi-level indices	Understand index-sequential files	Understand tree structures	
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**:				
Relational database concepts	Understand and explain Tables	Understand and explain Relationships between Tables	Understand and explain Data Redundancy	Understand and explain Data anomalies	Understand and explain the Primary and foreign keys and their roles in relating Tables
Normalisation	Define normalisation	Convert to 1NF	Convert to 2NF	Convert to 3NF	

Introduction to SQL

Aligned to SAQA US114048

Introduction

WELCOME TO THE INTRODUCTION TO SQL LESSON!

In this lesson, you will be introduced to the database language SQL, and the concepts of Data Definition vs Data Manipulation language (DDL vs DML) and associated commands. We'll show you how to create and manipulate tables and indexes, query data and make queries more specific with criteria, and create joins. The lesson concludes with a summary of the functionality available to support large objects, and how to create triggers, stored procedures, and functions. At the end there is a task for you to put into practice what you have learned.

INTENDED LEARNING OUTCOMES

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

- Write program code for database access for a computer application using SQL
- Test programs for a computer application that accesses a database using SQL

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.

SPECIFIC OUTCOME 3 (US 114048)

13. PROGRAM CODE FOR DATABASE ACCESS FOR A COMPUTER APPLICATION USING SQL

CONCEPT: STRUCTURED QUERY LANGUAGE (SQL)

SQL (Structured Query Language) was introduced in Unit 3. Here we will look at this query language in more detail.

According to Encyclopaedia Britannica (2020), SQL is a "language designed for eliciting information from databases". Coronel et al. (2011) expand on this definition, adding that SQL's commands allow a user to create tables of data, from which the data can be manipulated and sorted. We can, therefore, simplify by saying that SQL is a language that we use to turn data into information.

Like many coding languages, SQL's vocabulary is quite similar to English, which makes it a fairly easy language to learn.

SQL statements fit into two general categories you saw defined in a previous lesson (Oracle, 2017):

1. **SQL as a data definition language (DDL):** includes commands to create, change and drop database objects (such as tables) as well as define access rights to those database objects. They also allow you to add comments to the data dictionary.

2. **SQL as a data manipulation language (DML):** includes commands to manipulate data in existing objects like insert, update, delete and retrieve data within the database tables.

The table below lists the SQL **data definition commands** (Coronel, et al., 2011):

Command	Description
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema

NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column when no value is given
CHECK	Used to validate data in an attribute
CREATE INDEX	Creates an index for the table
CREATE VIEW	Creates a dynamic subset of rows or columns from one or more tables
ALTER TABLE	Modifies a table (adds, modifies or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

SQL Data Definition Commands (Coronel, et al., 2011, p. 221)

The table below lists the SQL **data manipulation commands** (Coronel, et al., 2011):

Command	Description
INSERT	Inserts rows into a table
SELECT	Select attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition

ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more tables rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
Comparison Operators	
=, <, >, <=, >=, <>	Used in conditional expressions
Logical Operators	
AND, OR, NOT	Used in conditional expressions
Special Operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate Functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given

AVG	Returns the average of all values for a given column

SQL Data Manipulation Commands (Coronel, et al., 2011, pp. 221-222)

CREATING TABLES

To create new tables in SQL you use the CREATE TABLE statement. As its arguments, it expects all the columns we want in the table, as well as their data types. The syntax of the CREATE TABLE statement is shown below:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
   ....
);
```

In the above, the constraints are optional and are used to specify rules for data in a table. Constraints that are commonly used in SQL are:

- NOT NULL: Ensures that a column cannot have a NULL value
- UNIQUE: Ensures that all values in a column are different
- PRIMARY KEY: Uniquely identifies each row in a table
- FOREIGN KEY: Uniquely identifies a row in another table
- CHECK: Ensures that all values in a column satisfy a specific condition
- DEFAULT: Sets a default value for a column when no value is specified
- INDEX: Used to create and retrieve data from the database very quickly

To create a table called Employee, for example, that contains five columns — EmployeeID, LastName, FirstName, Address, and PhoneNumber — you would do the following:

```
CREATE TABLE Employee (
    EmployeeID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    PhoneNumber varchar(255)
);
```

The EmployeeID column is of type *int* and will, therefore, hold an integer value. The LastName, FirstName, Address, and PhoneNumber columns are of type *varchar* and will, therefore, hold characters. The number in brackets indicates the maximum number of characters, which in this case is 255. Note that these are all within brackets with a semicolon at the end. This is what indicates that the line is concluded.

The CREATE TABLE statement above will create an empty Employee table that will look like this:

EmployeeID	LastName	FirstName	Address	PhoneNumber

When creating tables, it's advisable to add a primary key to one of the columns as this will help keep entries unique and will speed up select queries. Primary keys must contain unique values, and cannot contain null values. A table can only contain one primary key, however, the primary key may consist of single or multiple columns (as you may remember from the previous lesson).

You can add a **primary key** as the index when creating the Employee table, as follows:

```
CREATE TABLE Employee (
    EmployeeID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    PhoneNumber varchar(255),
    PRIMARY KEY (EmployeeID)
);
```

To name a primary key constraint, and define a primary key constraint on multiple columns, you use the following SQL syntax:

```
CREATE TABLE Employee (
    EmployeeID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
```

```
    PhoneNumber varchar(255),
    CONSTRAINT PK_Employee PRIMARY KEY (ID,LastName)
);
```

In the example above there is only one primary key named PK_Employee. However, the value of the primary key is made up of two columns: ID and LastName.

Inserting Rows

The table that we have just created is empty and needs to be populated with rows or records. We can add entries to a table using the INSERT INTO command. There are two ways to write the INSERT INTO command:

1. Do not specify the column names where you intend to insert the data. This can be done if you are adding values for all of the columns of the table. However, you should ensure that the order of the values is in the same order as the columns in the table. The syntax will be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

   So, for example, to add an entry to the Employee table you would do the following:

```
INSERT INTO Employee
VALUES (1234, Smith, John, 25 Oak Rd, 0837856767);
```

2. The other way is to specify both the column names and the values to be inserted. The syntax will be as follows:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

   Therefore, to add an entry to the Employee table using this way, you would do the following:

```
INSERT INTO Employee (EmployeeID, LastName, FirstName, Address, PhoneNumber)
VALUES (1234, Smith, John, 25 Oak Rd, 0837856767);
```

SELECT

The SELECT statement is used to fetch data from a database. The data returned is stored in a result table, known as the result-set. The syntax of a SELECT statement is as follows:

```
SELECT column1, column2, ...
FROM table_name;
```

column1, column2, ... are the column names of the table you want to select data from. The following example selects the FirstName and LastName columns from the Employee table:

```
SELECT FirstName, LastName
FROM Employee;
```

If you want to select all the columns in the table, however, you use the following syntax:

```
SELECT * FROM table_name;
```

The asterisk (*) means that we want to fetch all of the columns.

You can also use the ORDER BY command to sort the results in ascending or descending order. The ORDER BY command sorts the records in ascending order by default. You need to use the DESC keyword to sort the records in descending order.

The ORDER BY syntax is as follows:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC/DESC;
```

The example below selects all Employees in the Employee table and sorts them in descending order by the FirstName column:

```
SELECT * FROM Employee
ORDER BY FirstName DESC;
```

WHERE

Similar to an *if-statement*, the WHERE clause allows us to filter data depending on a specific condition. The syntax of the WHERE clause is as follows:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL statement selects all the employees with the first name John, in the Employee table:

```
SELECT * FROM Employee
WHERE FirstName = 'John';
```

Note that SQL requires single quotes around strings, however, you do not need to enclose numeric fields in quotes. You can use logical operators (AND, OR) and comparison operators (=,<,>,<=,>=,<>) to make WHERE conditions as specific as you like.

For example, suppose you have the following table which contains the most sold albums of all time:

Artist	Album	Released	Genre	sales_in_millions
Michael Jackson	Thriller	1982	Pop	70
AC/DC	Back in Black	1980	Rock	50
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soul	44

You can select those of them that are classified as rock and have sold under 50 million copies by simply using the AND operator as follows:

```
SELECT *
FROM albums
WHERE genre = 'rock' AND sales_in_millions <= 50
ORDER BY released
```

WHERE statements also support some commands, that allows you a quick way to check commonly used queries. These are:

- IN: compares the column to multiple possible values and returns true if it matches at least one
- BETWEEN: checks if a value is within an inclusive range
- LIKE: searches for a pattern

For example, if we want to select the pop and soul albums from the table above, we can use:

```
SELECT * FROM albums
WHERE genre IN ('pop', 'soul');
```

Or, if we want to get all the albums released between 1975 and 1985, we can use:

```
SELECT * FROM albums
WHERE released BETWEEN 1975 AND 1985;
```

FUNCTIONS

SQL has many functions that do all sorts of helpful stuff. Some of the most regularly used ones are:

- COUNT(): returns the number of rows
- SUM(): returns the total sum of a numeric column
- AVG(): returns the average of a set of values
- MIN() / MAX(): gets the minimum or maximum value from a column

For example, to get the most recent year in the Album table we can use:

```
SELECT MAX(released)
FROM albums;
```

JOINS

In complex databases, there are often several tables connected to each other in some way. A JOIN clause is used to combine rows from two or more tables, based on a column they both share.

Look at the two tables below:

VideoGame

ID	Name	DeveloperID	Genre
1	Super Mario Bros.	2	Platformer
2	World of Warcraft	1	MMORPG
3	The Legend of Zelda	2	Adventure

GameDeveloper

ID	Name	Country
1	Blizzard	USA
2	Nintendo	Japan

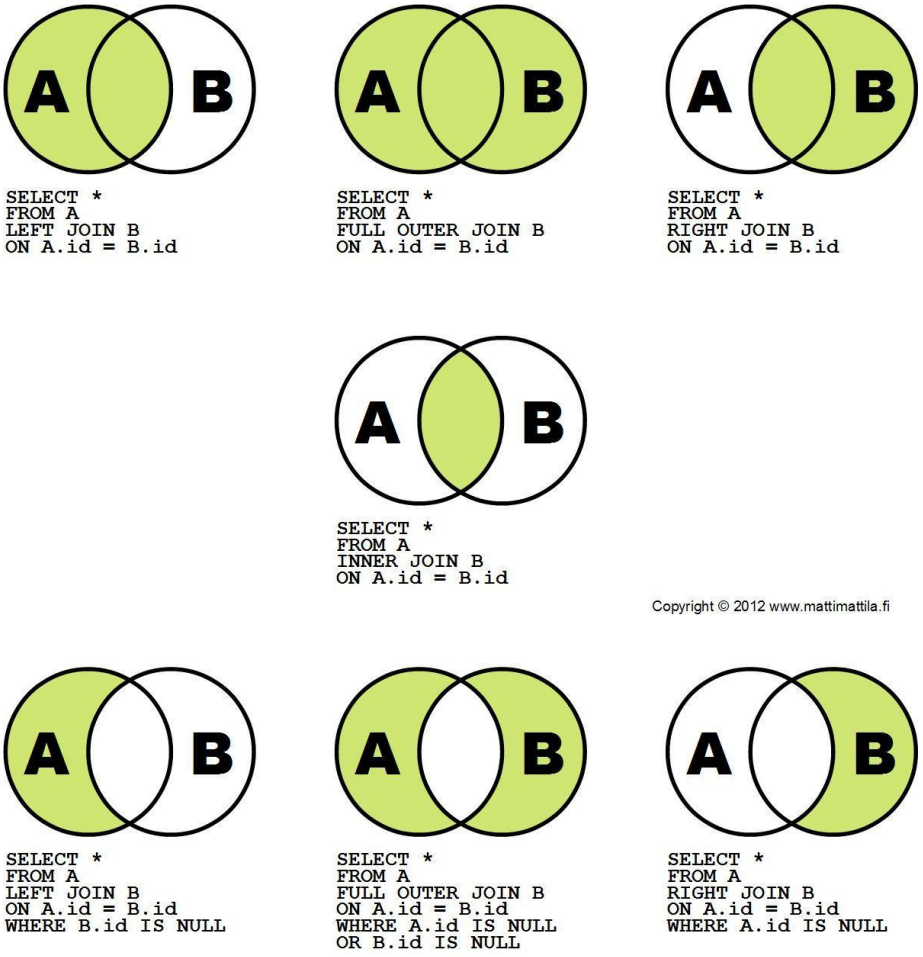
The VideoGame table contains information about various video games, while the Game Developer table contains information about the developers of the games. The VideoGame table has a DeveloperID column that holds the game developer ID and represents the ID of the respective developer from the GameDeveloper table. This logically links the two tables and allows us to use the information stored in both of them at the same time.

If we want to create a query that returns everything we need to know about the games, we can use INNER JOIN to acquire the columns from both tables.

```
SELECT VideoGame.Name, VideoGame.Genre, GameDeveloper.Name,
GameDeveloper.Country
FROM VideoGame
INNER JOIN GameDeveloper
ON VideoGame.DeveloperID = GameDeveloper.ID;
```

The INNER JOIN is the simplest and most common type of JOIN, however, there are many other different types of JOINs in SQL, namely:

- INNER JOIN: Returns records that have matching values in both tables
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
- FULL JOIN: Returns all records when there is a match in either left or right table



Graphical representation of common SQL joins [(Mattila, 2012)]

ALIASES

Notice that in the VideoGame and GameDeveloper tables each has a column called Name. This can become confusing. Aliases are used to give a table or column a temporary name. An alias only exists for the duration of the query and is often used to make column names more readable.

The alias column syntax is:

```
SELECT column_name AS alias_name
FROM table_name;
```

The alias table syntax is:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

The following SQL statement creates an alias, for the Name column from GameDeveloper table:

```
SELECT Name AS Developer
FROM GameDeveloper;
```

This SQL statement shortens the query drastically by setting aliases to the table names:

```
SELECT games.Name, games.Genre, devs.Name AS Developer, devs.Country
FROM VideoGame AS games
INNER JOIN GameDeveloper AS devs
ON games.DeveloperID = devs.ID;
```

UPDATE

The UPDATE statement is used to modify the existing rows in a table.

To use the UPDATE statement you:

- Choose the table where the row you want to change is located.
- Set the new value(s) for the wanted column(s).
- Select which of the rows you want to update using the WHERE statement. If you omit this, all rows in the table will change.

The syntax for the update statement is:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Take a look at the following Customer table:

CustomerID	CustomerName	Address	City
1	Maria Anderson	23 York St	New York
2	Jackson Peters	124 River Rd	Berlin
3	Thomas Hardy	455 Hanover Sq	London
4	Kelly Martins	55 Loop St	Cape Town

The following SQL statement updates the first customer (CustomerID = 1) with a new address and a new city:

```
UPDATE Customer
SET Address = '78 Oak St', City= 'Los Angeles'
WHERE CustomerID = 1;
```

DELETING ROWS

Deleting a row is a simple process. All that you need to do is select the right table and row you want to remove. The DELETE statement is used to delete existing rows in a table.

The DELETE statement syntax is as follows:

```
DELETE FROM table_name
WHERE condition;
```

The following statement deletes the customer Jackson Peters from the Customer table:

```
DELETE FROM Customer
WHERE CustomerName = 'Jackson Peters';
```

You can also delete all rows in a table without deleting the table:

```
DELETE * FROM table_name;
```

DELETING TABLES

The DROP TABLE statement is used to remove every trace of a table in a database. The syntax is as follows:

```
DROP TABLE table_name;
```

For example, if we what to delete the table Customer, we do the following:

```
DROP TABLE Customer;
```

If you want to delete the data inside a table, but not the table itself, you can use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

SUPPORT FOR LARGE OBJECTS

Blobs
Blobs are very similar to the text datatype. They contain all the same lengths and very similar wording.

The key difference between text and blobs is that text data types will convert the data in your table to/from the character set that they have assigned to them. Blobs, however, convert their data from the object inserted into a binary data type.

So what does this mean and why are blobs useful?

Because of how the data in a blob field is converted, we are actually able to insert data that we may once have not been able to. A key example of this would be a picture.

Typically databases are known for only storing text/numerical data, but what about pictures? Every social media platform you go onto will usually have a profile picture, but how would they link this image to your account? This is where the blob field comes into play; it converts your photo into binary and, from there, inserts it into a database field. Once all that has been completed, it will then find the image and show it on the website.

Below is an example of where we could use the blob type.

```
CREATE TABLE users (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  user_name VARCHAR(25),
  profile_picture MEDIUMBLOB
);
```

Blobs Storage size

Blobs can store different size files (remember that it's not just limited to images, it can store almost any file type). Depending on the usage - where we might need the blob - we can decide on what blob type we will use.

Blob Type	File Storage Size	Example
TINYBLOB	>= 255 bytes	A pixel on an image
BLOB	>= 64 kb	Simple text files
MEDIUMBLOB	>= 16MB	Profile picture
LONGBLOB	>= 4GB	A Movie

TRIGGERS

What are triggers in SQL?

Think of triggers as a way of provoking certain actions based on a particular event taking place on a table or view. The timing of the event also plays an important role in the way in which a trigger is executed. Let us imagine for a second that we want to ensure that a certain action takes place when we perform a change in a table. There are typically two options for our triggering time: (a) BEFORE and (b) AFTER, where the 'before' keyword signifies that we wish for the change to take place before our SQL command runs, and 'after'

signifies firing the trigger after a SQL command has run. In addition to these two special triggering-time keywords, we have the INSTEAD OF trigger which can be used only on views and not tables, like BEFORE and AFTER.

The life-cycle of trigger statements

When we start exploring the different types of triggers (as we will in the sections below), you will notice that the life-cycle of a trigger involves two key statements: **(a)** the **Create Trigger** and the **(b) Drop Trigger** statements. It is through these two statements that we can effect any change in our tables or views using triggers. So, let us have a look at the two statements individually and assess the level of importance they each bear.

The CREATE TRIGGER Statement

You will have noticed by now that triggers do not exist in isolation, and that in fact, they exist because of either tables or views. To begin with, the reason we would need to have a trigger is so we may better manage the processes going on in both our tables and views. Pretty much like querying a table (where statements like SELECT would be used), setting up a trigger will require us to firstly create the trigger. To achieve this, we use the **CREATE TRIGGER** statement which can then hold all the business logic/procedures we would like to implement.

Below is an example of the basic syntax for creating a trigger:

```
CREATE TRIGGER trigger_name
ON table_name
AS
{sql_statements}
```

What we have in the example above is a very fundamental CREATE TRIGGER statement which should be fired on a given table. Please note that since the above is just a template for the implementation of the CREATE TRIGGER statement, the **trigger_name** as well as the **table_name** and **sql_statements** have been labelled in accordance with the general structure of the CREATE TRIGGER statement.

The DROP TRIGGER Statement

Let us say you now wish to take down a trigger you had created for a certain table or view. To achieve this, you would have to evoke the DROP TRIGGER statement which will then ensure that the trigger you had created is taken down

effectively, without causing any harm to the data in our table or view. Please see an example of the basic syntax for a DROP TRIGGER statement:

```
DROP TRIGGER [IF EXISTS] trigger_name
ON {DATABASE | ALL SERVER}
```

Notice that when we now drop a trigger, we introduce the IF EXISTS conditional which checks whether the specified trigger being sought to be dropped actually exists, and then will proceed to drop it if the conditional finds the trigger does exist.

The three types of triggers

- **Data manipulation language (DML) triggers**
  The easiest way to understand data manipulation triggers is by thinking of already existing data (tables), on which INSERT, UPDATE, and DELETE may be used. The example below covers a typical structure for a data manipulation language trigger using what is known as **T-SQL** or **Transact-SQL**. Think of T-SQL as a way of enforcing certain checks and balances on existing data, so that the data best reflects the business processes.

```
CREATE TRIGGER trigger_name ON table_name
FOR UPDATE
NOT FOR REPLICATION
AS

BEGIN
  INSERT INTO another_table
  SELECT a_column
  FROM inserted
END
```

- **Data definition language (DDL) triggers**
  While the DML triggers are primarily concerned with enforcing changes on a table and view level, data definition language (**DDL**) triggers operate on the database and server level. This means that through DDL triggers we can effect change on a level a little higher than that which DML triggers target.

Let us have a look at the general structure of a DDL trigger. Please see example below:

```
CREATE TRIGGER trigger_name
ON {DATABASE | ALL SERVER}
AS {sql_statement}
```

You will notice that at this level, as mentioned above, the trigger created is solely concerned with effecting change at the database level, as opposed to the table level. How do we know this from looking at the example above? you may ask. The answer lies in the **ON** keyword, after which we have specified two possibilities, either the database or the server. Following this, we have the **AS** keyword which should then evoke SQL statements that will set in motion the specified changes.

- **Logon triggers**
  Logon triggers are used for effecting change right after a user has logged in onto a system. These types of triggers help establish strong security and ensure that access to databases is limited only to existing users of a system. Let us briefly have a look at the most basic implementation of a logon trigger, illustrated in the example below:

```
CREATE TRIGGER trigger_name
ON ALL SERVER FOR LOGON
AS
BEGIN
  {sql_conditionals}
  /*where sql_conditionals
    is all the sql if-else
    statements to check
    against a given
    user_name*/
END
```

You will notice that the logon trigger in the example sets out to carry out specific actions only when a certain username is accessing the server and database.

STORED PROCEDURES AND FUNCTIONS

A **function** in programming is a block of reusable code that performs a single or related action multiple times. Functions will usually take in data, process it and

return it. Functions can be used over and over again making them a good alternative to having repeating blocks of code within your program (the same definition is true across all programming languages, not just SQL). A function in SQL would be a set of SQL statements that perform a specific task; functions are compiled and executed every time they are called within the server.

```
USE `library_db`;
DROP function IF EXISTS `book_description`;

USE `library_db`;
CREATE FUNCTION library_db.`book_description` (author_nm VARCHAR(50),
book_tl VARCHAR(50))
RETURNS VARCHAR(100) DETERMINISTIC
BEGIN
RETURN CONCAT(author_nm, ' - ', book_tl);
END;
```

This function will combine the author name and book title, and return it as one string. We pass in the word deterministic so that we always get the same answer whenever the inputs have been passed in; this is a form of exception handling to avoid unexpected results.

A **stored procedure** is a set of instructions or commands that are executed in order. In SQL a stored procedure is a prepared block of SQL code (queries or commands) that you can save, and that will be reused over and over again. Stored procedures are usually defined as a script that contains a series of commands that you can schedule to run at certain times. Stored procedures can chain multiple database statements like INSERT, UPDATE, SELECT and DELETE to perform repetitive scripts. Stored procedures are pre-compiled, meaning that once they are compiled the compiled format is saved and executed as needed without being compiled each time.

```
USE `library_db`;
DROP procedure IF EXISTS `add_new_book`;

DELIMITER $$
USE `library_db`$$
CREATE PROCEDURE library_db.`add_new_book` (IN id_val INT, IN title_bk
VARCHAR(50), IN author_name VARCHAR(50), IN quantity INT)
BEGIN
      INSERT INTO books(id, title, author, qty)
    VALUES(id_val, title_bk, author_name, quantity);
```

```
END$$

DELIMITER;
```

This is a Stored Procedure that is used to add a new book to the library database. The IN keyword is an input parameter; by default when creating a function the keyword is always IN but for stored procedures the keyword can be IN, OUT, or INOUT. This is because we sometimes want to pass data into the procedure (IN), or get data out from the procedure (OUT) and in other cases both (INOUT).

The key differences between a stored procedure and a function are:

- A function must have a return value and it can only return a single value. A stored procedure can return nothing (0), a single value, or multiple values.

- Functions can only have input parameters, whereas stored procedures can have input and output parameters.

- We can call a function from a stored procedure, but a stored procedure can not be called from a function.

- A stored procedure allows us to use multiple SQL statements while a function will only allow a SELECT statement to be used within it.

These differences summarize the effect these routines can have on a database. A stored procedure can affect the state of the database and or alter the state of the environment parameters, while a function cannot do either.

MYSQL

MySQL is a popular open-source relational database management system owned by MySQL AB. While it is written in C and C++, it is based on SQL. It also supports many different languages and operating systems. MySQL makes using SQL easy because It has an intuitive interface that allows you to add, remove, modify, search and join tables in a database. You will learn more about MySQL in the next lesson where we will be installing it and starting to create and manipulate databases with it.

Compulsory Task 6

Answer the following questions:

- Go to the **w3schools website's SQL browser IDE**. This is where you can write and test your SQL code using their databases. Once you are happy with it, paste your code in a text file named **Student.txt** and save it in your task folder.
- Write the SQL code to create a table called Student. The table structure is summarised in the table below (Note that STU_NUM is the primary key):

Attribute Name	Data Type
STU_NUM	CHAR(6)
STU_SNAME	VARCHAR(15)
STU_FNAME	VARCHAR(15)
STU_INITIAL	CHAR(1)
STU_STARTDATE	DATE
COURSE_CODE	CHAR(3)
PROJ_NUM	INT(2)

- After you have created the table in question 1, write the SQL code to enter the first two rows of the table as below:

STU_NUM	STU_SNAME	STU_FNAME	STU_INITIAL	STU_STARTDATE	COURSE_CODE	PROJ_NUM
01	Snow	John	E	05-Apr-14	201	6
02	Stark	Arya	C	12-Jul-17	305	11

- Assuming all the data in the Employee table has been entered as shown below, write the SQL code that will list all attributes for a COURSE_CODE of

305.

STU_NUM	STU_SNAME	STU_FNAME	STU_INITIAL	STU_STARTDATE	COURSE_CODE	PROJ_NUM
01	Snow	Jon	E	05-Apr-14	201	6
02	Stark	Arya	C	12-Jul-17	305	11
03	Lannister	Jamie	C	05-Sep-12	101	2
04	Lannister	Cercei	J	05-Sep-12	101	2
05	Greyjoy	Theon	I	9-Dec-15	402	14
06	Tyrell	Margaery	Y	12-Jul-17	305	10
07	Baratheon	Tommen	R	13-Jun-19	201	5

- Write the SQL code to change the course code to 304 for the person whose student number is 07.

- Write the SQL code to delete the row of the person named Jamie Lannister, who started on 5 September 2012, whose course code is 101 and project number is 2. Use logical operators to include all of the information given in this problem.

- Write the SQL code that will change the PROJ_NUM to 14 for all those students who started before 1 January 2016 and whose course code is at least 201.

- Write the SQL code that will delete all of the data inside a table, but not the table itself.

- Write the SQL code that will delete the Student table entirely.

Self-assessment table

<table>
<tr><th>Intended learning outcomes (SAQA US 11048)</th><td colspan="5">Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- I can successfully:</td></tr>
<tr><td rowspan="2">Write program code for database access for a computer application using SQL (SO3)</td><td>Understand and be able to explain the difference between DDL and DML</td><td>Identify and use DDL SQL commands</td><td>Identify and use DML SQL commands</td><td>Create and manipulate tables and indexes</td><td>Query data and make queries more specific with criteria</td></tr>
<tr><td>Understand and be able to use BLOBs in SQL</td><td>Understand and be able to use Triggers in SQL</td><td>Understand and be able to use functions in SQL</td><td colspan="2">Understand and be able to use stored procedures in SQL</td></tr>
</table>

Java Database Programming, Testing & Documentation

Learning assumed to be in place: basic familiarity with Java

Aligned to SAQA US114048

Introduction

WELCOME TO THE INTRODUCTION TO THE JDBC TASK!

In this lesson, we learn how to connect to a MySQL database with JDBC, and cover procedures for testing and documentation. At the end there is a task for you to put into practice what you have learned.

INTENDED LEARNING OUTCOMES

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

- Write program code for database access for a computer application using SQL
- Test programs for a computer application that accesses a database using SQL
- Document programs for a computer application that accesses a database using SQL
- Install and run MySQL servers and clients, and use the JDBC to create java applications that interact with these

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.

14.PROGRAM CODE FOR DATABASE ACCESS FOR A COMPUTER APPLICATION USING SQL

In this lesson, you're going to extend your knowledge of SQL using the MySQL variant, and get further practice writing code, testing your code, and documenting your code using comments.

INTRODUCTION TO JAVA DATABASE CONNECTIVITY

JDBC is the Java API for developing Java database applications. Liang (2011, p. 1286) defines JDBC as providing a uniform interface to Java programmers for accessing and manipulating a wide range of relational databases. "Using the JDBC API, applications written in the Java programming language can execute SQL statements, retrieve results, present data in a user-friendly interface, and propagate changes back to the database" (pp. 1286-1287). This makes data handling an easy and user-friendly process. To use JDBC, you need to download a JDBC driver to link to the database you want to work with. Each database will have its own specific JDBC driver, such as the MySQL JDBC driver to access the MySQL database.

In summary with the JDBC API, you are able to (Ciubotaru, & Muntean, 2013, p. 136):

1. Establish a connection with a database or access any tabular data source
2. Send SQL statements
3. Process the results

SETTING UP YOUR ENVIRONMENT

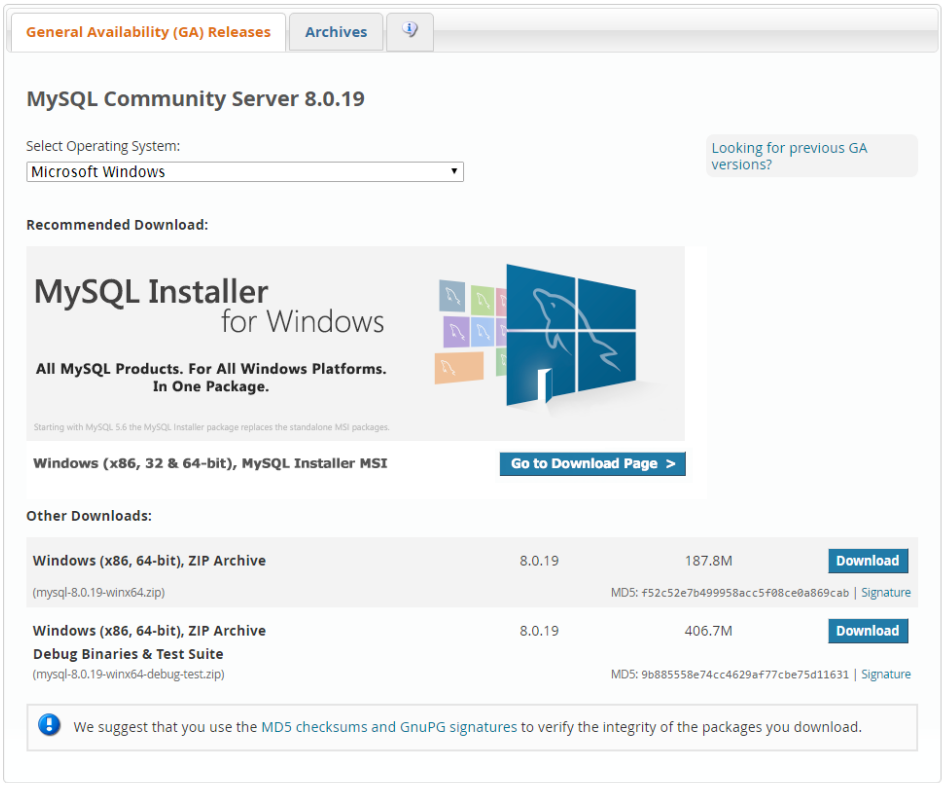
Before you start developing with JDBC you need to set up your environment. This includes downloading and installing MySQL, installing the JDK and a text editor. At this point, you should have already installed the JDK and a text editor such as TextPad, NotePad++ or Sublime. However, if not, you can download the JDK **here** and the text editor Sublime **here**.

We will now explain the steps to download and install MySQL.

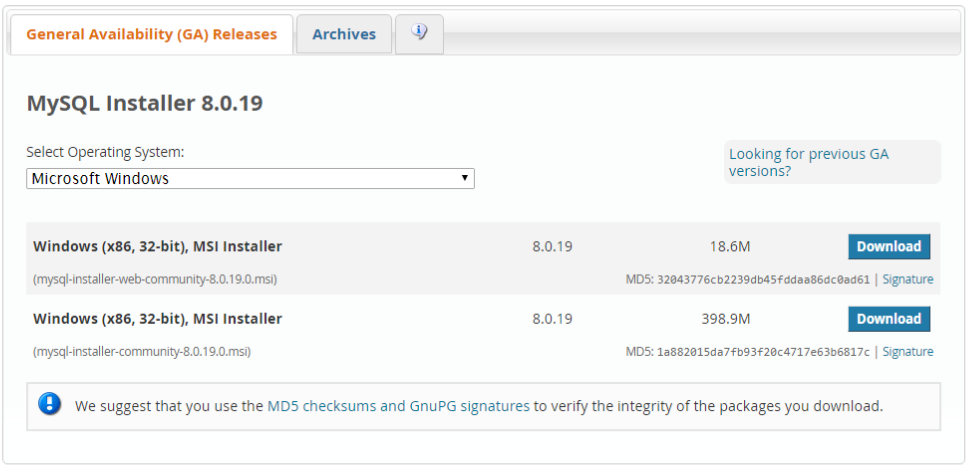
DOWNLOAD AND INSTALL MYSQL

For Windows:

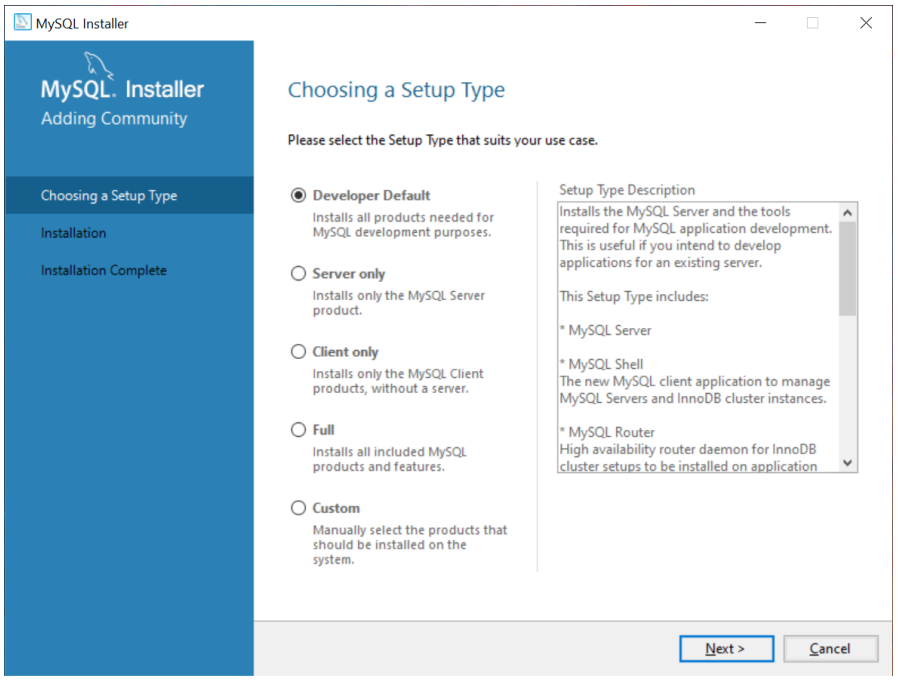
1. Download MySQL Installer from **https://dev.mysql.com/downloads/mysql/** and execute it.
2. Choose the appropriate Setup Type for your system. Typically you will choose Developer Default to install MySQL. If a default option is not provided, complete the following:
    a. Click on the *General Availability (GA) Releases* tab.
    b. Under *Select Operating System*, select *Microsoft Windows* from the dropdown menu.



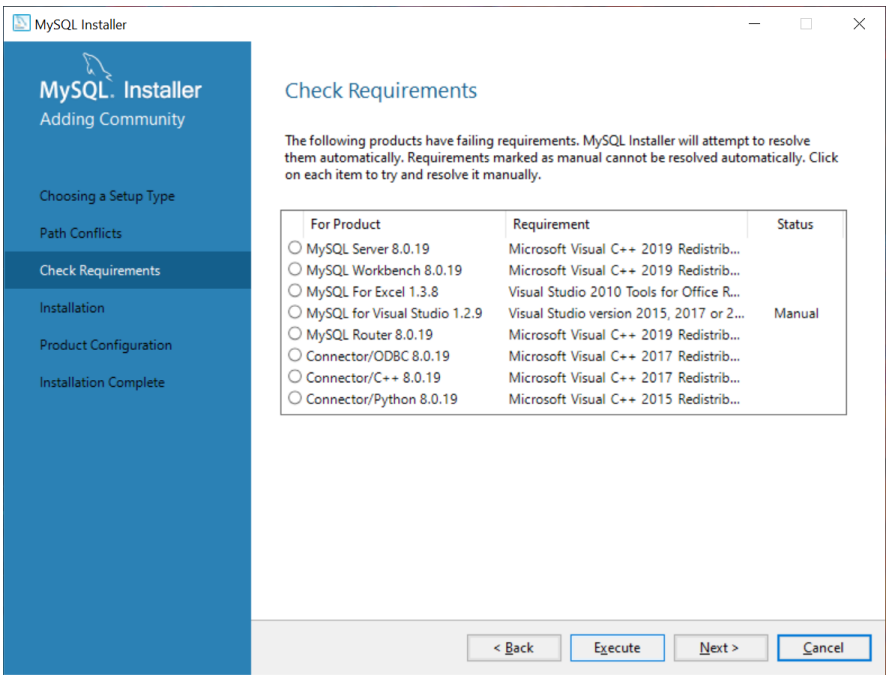
    c. Click on the *Go to Download Page* button and Download the *Windows (x86, 32-bit), MSI Installer* (The bigger one. For the 8.0.19 version it was 398.9 MB).



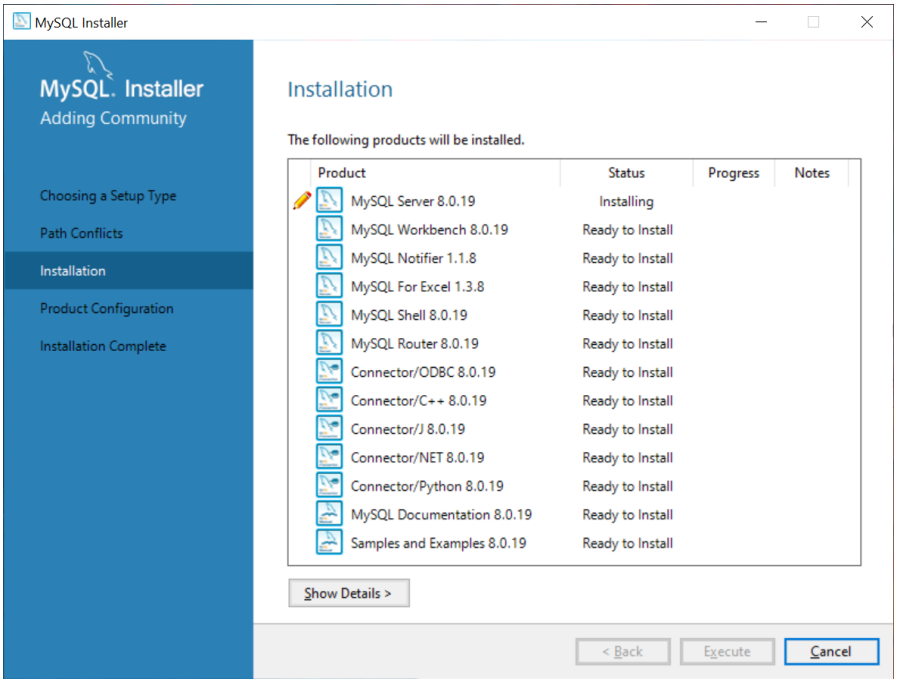
3. It is not necessary to login or sign up. Simply click *No thanks, just start my download* to continue the installation process.
4. Once it is downloaded, open the installer. Allow MySQL to make changes to your device.
5. Select *Develop Default* as your *Setup Type* and click *Next*.



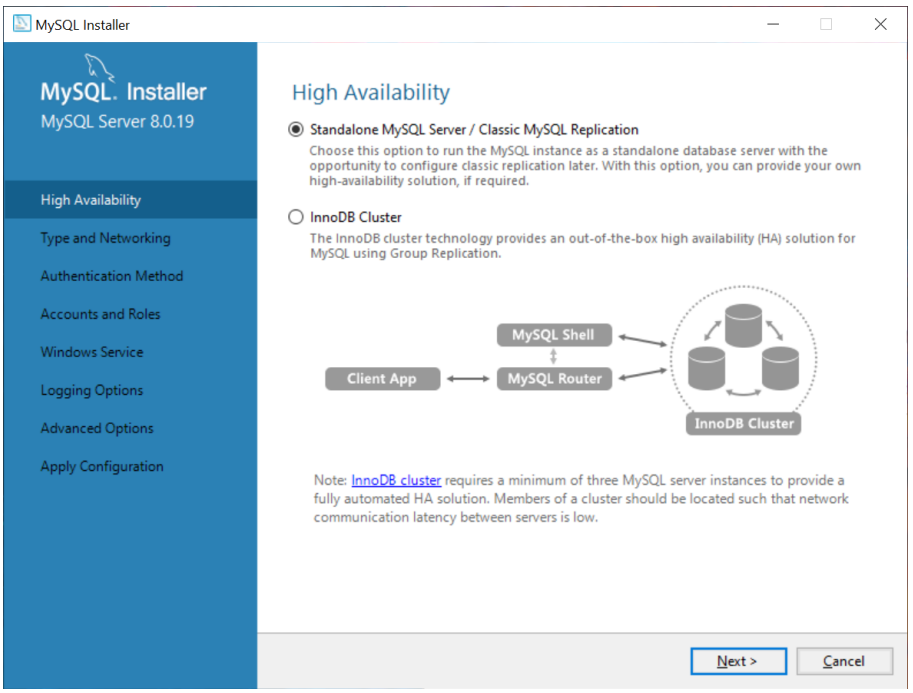
6. When you get to the *Check Requirements* step, simply click *Execute* at the bottom. Allow all the required programs to be installed



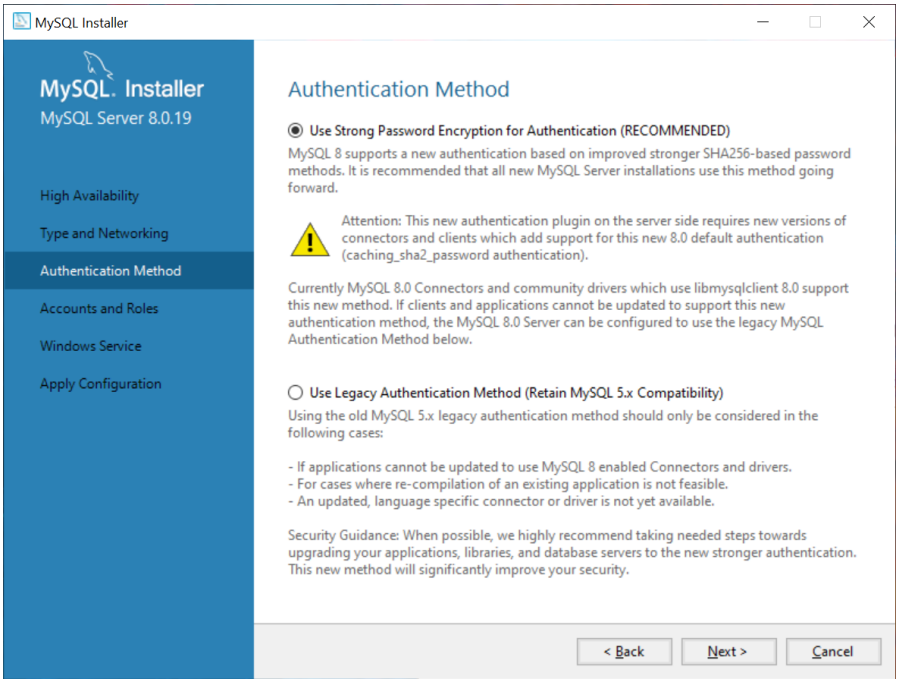
7. Once the downloads are completed, click *Next*
8. When you get to *Installation,* click *Execute* and wait for everything to download. Once that is completed, click *Next*



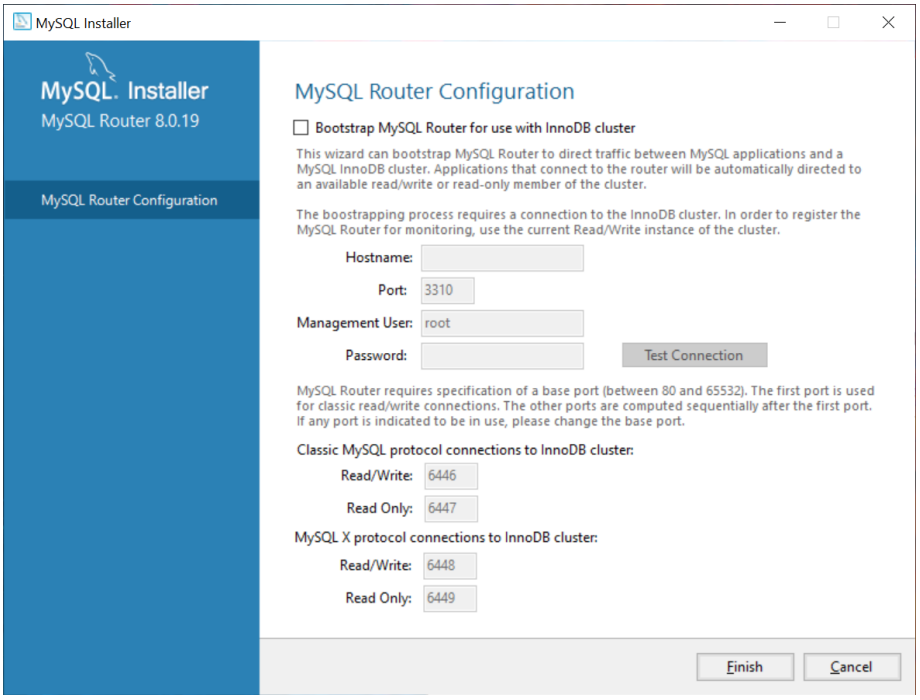
9. On the *Product Configuration* page, click *Next*. When asked about *High Availability*, choose *Standalone MySQL /Classic MySQL Replication* and click *Next*



10. On the *Authentication Method* page, select *Use Strong Password Encryption for Authentication* and click *Next*



11. **When asked to add a password, be sure to make a note of it so you don't forget it!** You can keep *root* as the user.



12. When asked, type in your created password for root and click *Next*
13. Finally, to *Apply Configuration*, click *Execute*. Once that is completed click *Finish*
14. Once you get to *Installation Complete*, click *Finish*. Your MySQL console and Workbench should open automatically.

For Mac:

1. Download the *MySQL DMG Archive* from the **MySQL website**. Your operating system should be automatically detected, however, if it is not:
    a. Select the *General Available (GA) Releases* tab.
    b. In *Select Operating System* choose *macOS*
    c. It is not necessary to login or sign up. Simply click *No thanks, just start my download* to continue the installation process
2. Install MySQL:
    a. Go to *Downloads* and Double-click the ".dmg" file.
    b. Double-click the downloaded MySQL download.
    c. Follow the on-screen instructions to install MySQL.
    d. A superuser called root is created with a temporary password during the installation. **It is extremely important that you take note of the password.** You can copy and save it somewhere or take a screenshot. Just make sure you don't lose it!
    e. MySQL will be installed in "/usr/local/mysql". Take note of this installed directory!

f. Eject the ".dmg" file

3. If you make a mistake you need to:
    a. stop the server by clicking on the 'Apple' Icon → System Preferences → MySQL → Stop
    b. remove the directories "/usr/local/mysql-5.7.{xx}..."
    c. re-run the installation
    d. restart the server by clicking on the 'Apple' Icon → System Preferences → MySQL → Start
    e. You may also need to reboot your machine.
4. Remember to take note of your MySQL installed directory.

For Linux:

1. Check if your distribution includes MySQL in the native repository:
    a. Open a terminal
    b. Run `apt show install mysql-server` (as superuser if necessary)
    c. Unless apt show returns "E: No packages found", skip to "Install MySQL from native repository".
2. Download the DEB package from the MySQL **website**:
    a. It is not necessary to login or sign up. Simply click *No thanks, just start my download* to continue the installation process
3. Install MySQL from download:
    a. Open a terminal in the folder where you saved the DEB package
    b. Run `dpkg -i: "dpkg -i [name of DEB package]"`
    c. At some point during installation, you will be prompted to enter a *root* password. **It is critical you remember this!** It is difficult to reset.
    d. Congratulations! Installation is complete.
4. Install MySQL from native package manager:
    a. From the terminal, run `sudo apt-get install mysql-server`
    b. At some point during installation, you will be prompted to enter a *root* password. It is critical you remember this! It is difficult to reset.
    c. After the server finishes installing, run `sudo apt-get install mysql-client`
    d. Congratulations! Installation is complete.

USING MYSQL

We will now briefly discuss the basics of using MySQL. Some of the examples below are based on Hock-Chuan of Nanyang Technological University's (2020a) guide to MySQL.

Starting the Server

MySQL is a client-server system. This means that the database is run as a server application and users can access the database server locally or remotely by using a client program. To start up the server you need to do the following:

For Windows:

- If you used the MSI installer, MySQL should be running once your computer starts.

For Mac:

- Simply click on the Apple Icon → System Preferences → MySQL → Start MySQL Server

The MySQL server should now be started and able to handle requests from the client.

For Linux:

- From the terminal, run `service mysql start`
- This may prompt you for an admin password, after which it will start up.

SHUTTING DOWN THE SERVER

For Windows:

- Press Ctrl+C. This initiates a normal shut down. Do not use the windows close button to close the server.

  You should get the following message from the MySQL server console:

```
XXXXXX XX:XX:XX [Note] mysqld: Normal shutdown
```

```
......
XXXXXX XX:XX:XX  InnoDB: Starting shutdown...
XXXXXX XX:XX:XX  InnoDB: Shutdown completed; log sequence number 0 44233
......
XXXXXX XX:XX:XX [Note] mysqld: Shutdown complete
```

For Mac:

- Simply click on the 'Apple' Icon → System Preferences → MySQL → Stop MySQL Server

Please ensure you shutdown the MySQL server correctly, otherwise, you might corrupt the database and have problems restarting it.

For Linux:

- From the terminal, run `service mysql stop`
- This may prompt you for an admin password, after which it will shut down.

STARTING A CLIENT

As previously stated, MySQL is a client-server system. Once you start the server, one or more clients can be connected to it. A client can be local, meaning that it is run on the same machine, or remote, meaning it is from another machine on the same network.

We will now start a command-line client with the superuser "root".

For Windows:

- Firstly, you need to find the directory path for the MySQL bin. The path should be C:\Program Files\MySQL\MySQL Server 8.0\bin
- Start a Command Prompt and enter the following:

```
cd C:\Program Files\MySQL\MySQL Server 8.0\bin

// Start a client as "root"
mysql -u root -p
```

```
Enter password:
   // Enter the root's password
   // Nothing will be shown when typing the password
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 8.0.19 MySQL Community Server - GPL

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
// You can now enter SQL commands.
```

For Mac:

- Firstly, make sure you know the directory path for the MySQL bin folder.
- Open a new terminal and change the directory to your MySql bin.

```
cd /usr/local/mysql/bin

// Start a client with "root"
./mysql -u root -p
Enter password:
   // Enter the root's password
   // Nothing will be shown when typing the password

Welcome to the MySQL monitor.  Commands end with ; or \g.
......
mysql>
// You can now enter SQL commands.
```

For Linux:

- From the terminal, run `mysql -u root -p`
- If access is denied even if the password is correct, it may be necessary to run this as a superuser by using `sudo mysql -u root -p`.

CHANGING THE PASSWORD FOR ROOT

The superuser "root" can do anything to the databases. This includes deleting all of them. Therefore, security is imperative when it comes to the “root” superuser. This means that you need to change the root's temporary password immediately after logging in.

Your client should now be started. We will, therefore, continue with the current client session. Simply enter the following:

```
// You need to replace XXXXX with your chosen password
// Strings need to be enclosed by a pair of single-quotes ('').

mysql> alter user 'root'@'localhost' identified by 'XXXXX';
Query OK, 0 rows affected (0.00 sec)

// logout and terminate
mysql> quit
Bye
```

CREATE A NEW USER

The superuser "root" is a privileged user and is meant to be used for database administration. It is not meant to be used for everyday operational purposes. We, therefore, should create another user with fewer privileges. We will call this user "otheruser". To create a new user you need to start a client with the "root" superuser as shown above.

```
// If it is not already started, start a client.
mysql -u root -p     // Windows
./mysql -u root -p   // Mac OS

// We can give otheruser the password swordfish
mysql> create user 'otheruser'@'localhost' identified by 'swordfish';
Query OK (0.01 sec)

// The newly created user has no privileges.
// Let's grant otheruser all privileges to all databases and tables

mysql> grant all on *.* to 'otheruser'@'localhost';
Query OK (0.01 sec)

mysql> quit

// Now otheruser has all the same privileges as root, except being able
to grant privileges
```

CLIENT SESSION TIPS

Before we go any further, here are some tips on using the client:

- Terminate your command with a semicolon (;) like in Java.

- A single command can span many lines. To show continuation after pressing Enter, the new line prompt changes to `->`. Once you are finished with the command add the `;` at the end.
- Use `\c` to cancel (abort) the current command
- If you open a single quote without closing it the new line prompt changes to `'>` instead of `->`
- Use the up and down arrow keys to get previous or next commands from the command history.

CREATING A NEW DATABASE AND A NEW TABLE, INSERTING RECORDS, QUERYING AND UPDATING

A MySQL server contains many databases. In turn, a database contains many tables and a table contains many rows (records) and columns (fields).

We will now create a database called “dogs_db” and a table called “java_programming”. The table shall have three columns: id (of the type int), name (of the type varchar(50)) and weight (of the type float).

```
// Start a client with the new user "otheruser"
// cd to the MySQL's bin directory
mysql -u otheruser -p     // Windows
./mysql -u otheruser -p   // Mac OS X

// Create a new database called 'dogs_db'
mysql> create database if not exists dogs_db;
Query OK, 1 row affected (0.08 sec)

// List all the databases on this server
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| ......             |
| dogs_db            |
| ......             |
+--------------------+
x rows in set (0.07 sec)

// Use "song_db" database as the default database
mysql> use dogs_db;
Database changed

// Remove the table "letters" in the default database if it exists
mysql> drop table if exists letters;
Query OK, 0 rows affected, 1 warning (0.15 sec)


// Create a new table called "java_programming" in "dogs_db",
```

```
// with 3 columns of the specified types
mysql> create table java_programming (id int, name varchar(50), weight float);
Query OK, 0 rows affected (0.15 sec)

// List all the tables in "dogs_db"
mysql> show tables;
+---------------------+
| Tables_in_dogs_db   |
+---------------------+
| java_programming    |
+---------------------+
1 row in set (0.00 sec)

// Describe the "java_programming" table by listing its columns' definition
mysql> describe java_programming;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | YES  |     | NULL    |       |
| name     | varchar(50) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.04 sec)

// Insert a row into "java_programming" table.
// Strings are enclosed between single quotes.
// There are no quotes for int and float values.
mysql> insert into java_programming values (1, 'Fluffy', 12.2);
Query OK, 1 row affected (0.03 sec)

// Insert another row into the "java_programming" table.
mysql> insert into java_programming values (2, 'Rover', 42);
Query OK, 1 row affected (0.03 sec)

// Select all columns and rows (*) from table "java_programming".
mysql> select * from java_programming;
+----+-------------+------+
| id | name        |weight|
+----+-------------+------+
| 1  |   Fluffy    | 12.2 |
| 2  |   Rover     |  42  |
+----+-------------+------+
2 rows in set (0.00 sec)

// Select some columns and rows from table "java_programming",
// that match certain the conditions
mysql> select name, grade from java_programming where weight < 20;
+-------------+-------+
| name        | weight|
+-------------+-------+
| Fluffy      | 12.2  |
+-------------+-------+
1 rows in set (0.00 sec)

// Update the given field of the selected records
mysql> update java_programming set weight = 36.4 where name = 'Rover';
Query OK, 1 row affected (0.05 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from java_programming;
+----+-------------+-------+
| id | name        |weight |
+----+-------------+-------+
| 1  |   Fluffy    |  12.2 |
| 2  |   Rover     |  36.4 |
+----+-------------+-------+
2 rows in set (0.00 sec)

// Delete selected records
mysql> delete from java_programming where id = 2;
Query OK, 1 row affected (0.03 sec)

mysql> select * from java_programming;
+----+-------------+------+
| id | name        |weight|
+----+-------------+------+
| 1  |   Fluffy    | 12.2 |
+----+-------------+------+
1 rows in set (0.00 sec)
```

Instead of entering commands one at a time, you can store a set of SQL commands in a file and run it. To do so:

- Use a text editor to create a new file called "mycommands.sql" that contains the following three SQL statements:

```
insert into java_programming values (3, 'Patch', 6);
insert into java_programming values (4, 'Denzel', 54);
Select * from java_programming;
```

- Save the file under "d:\myProject" for Windows or under "Documents" for Mac OS.
- After you created the file, you can use the following source command to run the SQL script:

```
mysql> source d:\myProject\mycommands.sql   // For Windows ONLY
mysql> source ~/Documents/mycommands.sql    // For Mac OS X ONLY
Query OK, 1 row affected (0.00 sec)         // INSERT command output
Query OK, 1 row affected (0.00 sec)         // INSERT command output
+------+-------------+-------+              // SELECT command output
| id   | name        | weight|
+------+-------------+-------+
|  1   |   Fluffy    | 12.2  |
|  3   |   Patch     |  6    |
|  4   |   Denzel    |  54   |
+------+-------------+-------+
```

```
3 rows in set (0.00 sec)
```

This was a short introduction to MySQL. Please use the **Reference Manual** to learn more.

DEVELOPING DATABASE APPLICATIONS USING JDBC

According to Liang (2011, p. 1287), "The JDBC API consists of classes and interfaces for establishing connections with databases, sending SQL statements to databases, and processing the results of SQL statements." This means that a program written in Java can access any database by using SQL.

In general, a JDBC program comprises of the following steps:

1. Connect to a database server by allocating a Connection object.
2. Allocate a Statement object, under the Connection object created, for holding a SQL command.
3. Using the Statement and Connection objects, write a SQL query and execute it.
4. Process the query result.
5. Finally to free up resources, close the Statement and Connection objects.

COMMON JDBC COMPONENTS

DriverManager: A class that manages the list of database drivers. The DriverManager matches connection requests from the java application with the correct database driver. This is done using communication sub-protocol. The first driver that recognises a specific subprotocol will be used under the JDBC to establish a database connection.

Driver: An interface that handles the communication with the database server. Very rarely will you need to interact directly with Driver objects. You use DriverManager objects instead. DriverManager objects are used to manage Driver objects. These objects also abstract the details associated with working with Driver objects.

Connection: An interface that is concerned with all methods for contacting a database. All communication with a database is done through a connection object.

Statement: An interface that is used to submit the SQL statements to the database.

ResultSet: ResultSet objects hold data retrieved from a database after executing a SQL query using Statement objects. It acts as an iterator to allow you to move through the retrieved data.

SQLException: A class that handles errors that occur in a database application.

INSTALLING MYSQL JDBC DRIVER

You need to install an appropriate JDBC driver to run your Java database programs. MySQL's JDBC driver, "MySQL Connector/J", is available on the MySQL site, but is also automatically downloaded with the MSI installer if you have Windows. The information below is based on Hock-Chuan's explanations and guides (2020b).

To install the JDBC on Windows:

If you used the MSI installer, you should have Connector J in your *Program FIles (x86)* folder. If not, follow the following steps:

- Download the latest MySQL JDBC driver **here**.
    - Select *MySQL Connectors → Connector/J*
    - Select *Platform Independent→ Zip Archive*
    - Select *No thanks, just start my download.*
- UNZIP the download file into a temporary folder.
- From the temporary folder, copy the .jar file to your JDK's Extension Directory at *<JAVA_HOME>\jre\lib\ext* (where <JAVA_HOME> is the JDK installed directory), e.g., "c:\program files\java\jdk1.8.0_{xx}\jre\lib\ext".

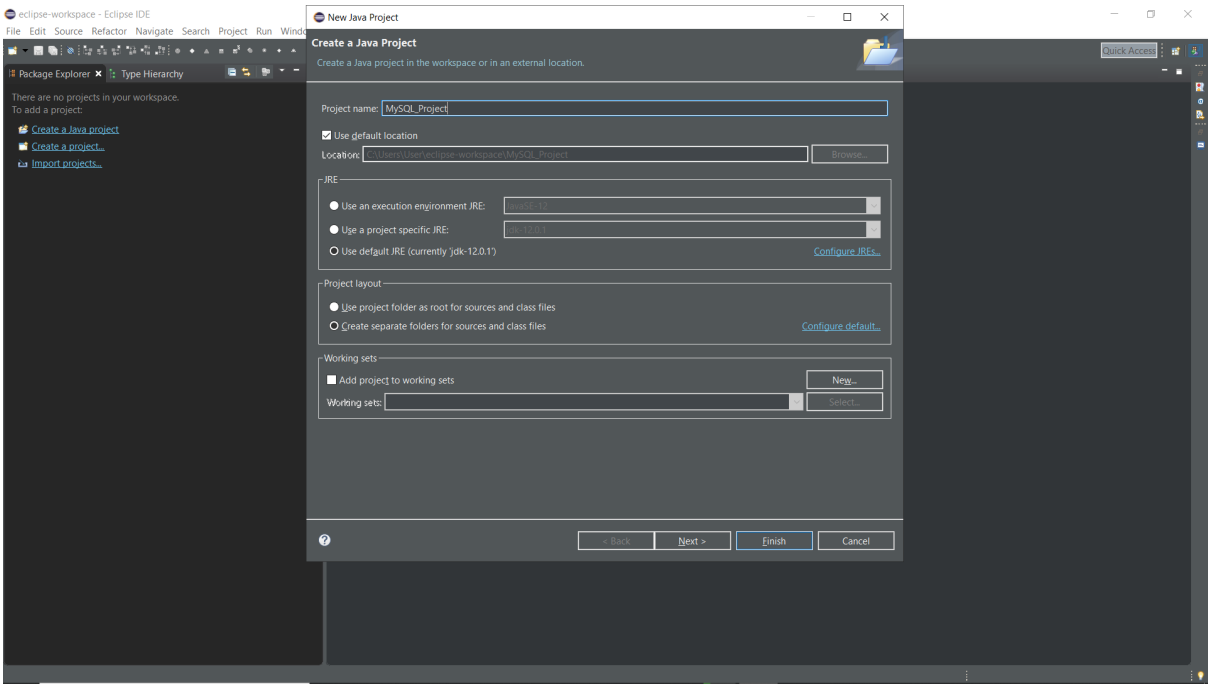
To install the JDBC on Mac OS:

- Download the latest MySQL JDBC driver **here**.
    - Select *MySQL Connectors → Connector/J*
    - Select *Platform Independent → Compressed TAR Archive*
    - Select *No thanks, just start my download*
- Double-click on the downloaded TAR file to expand it.

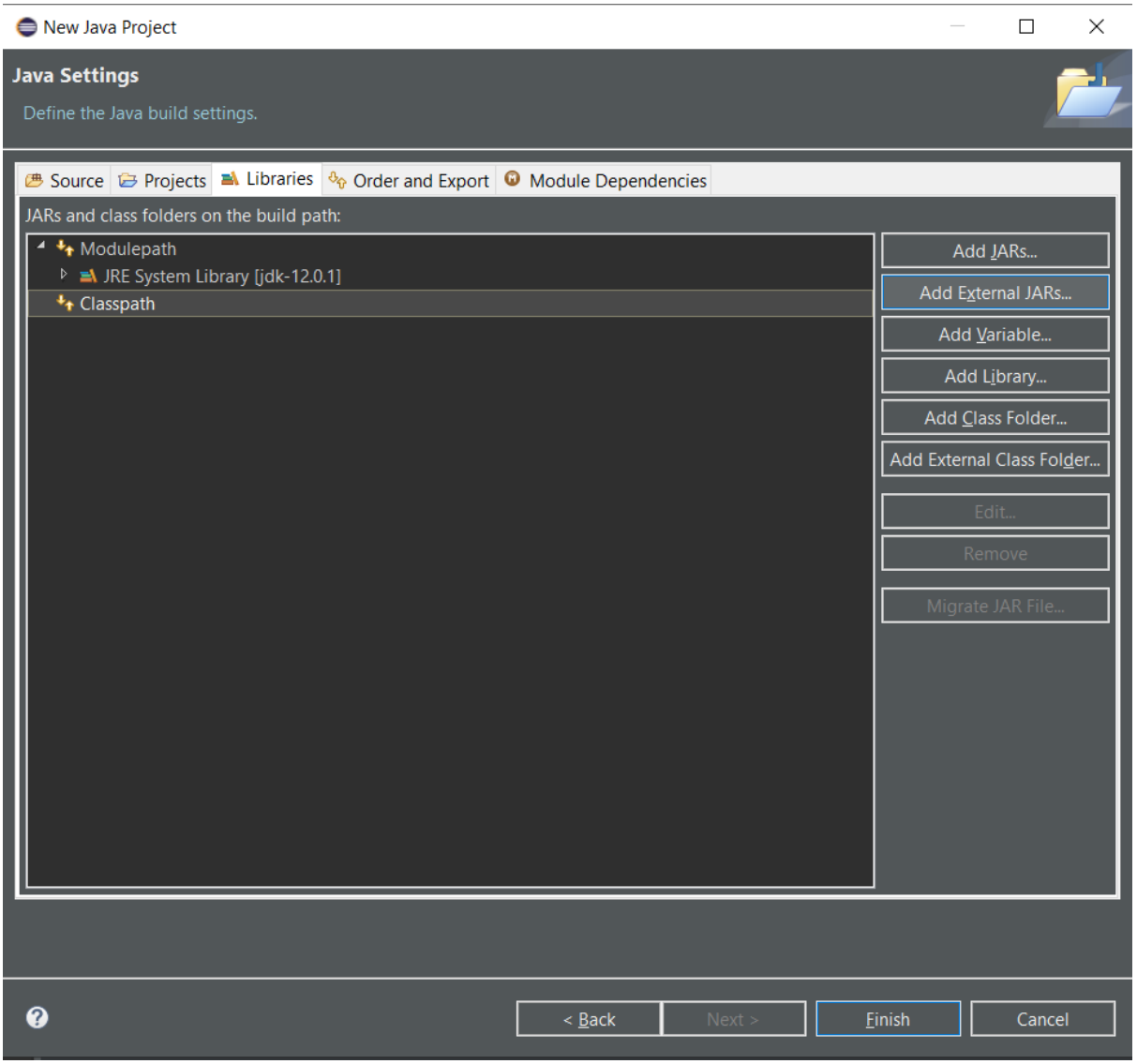
- Copy the .jar file to your JDK's Extension Directory at "/Library/Java/Extensions"

To connect your Java Application to your MySQL server:

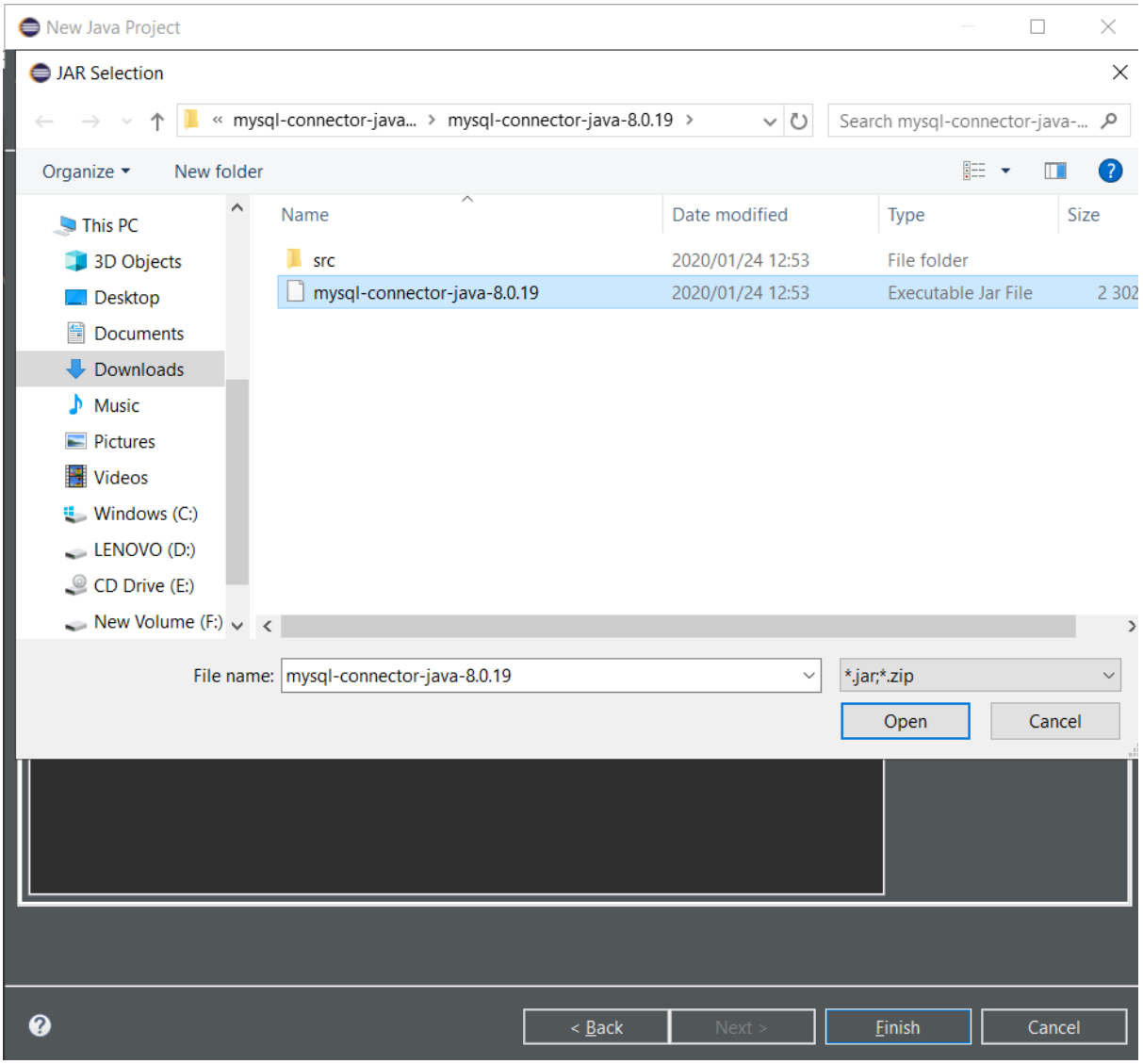
- Create a project in Eclipse



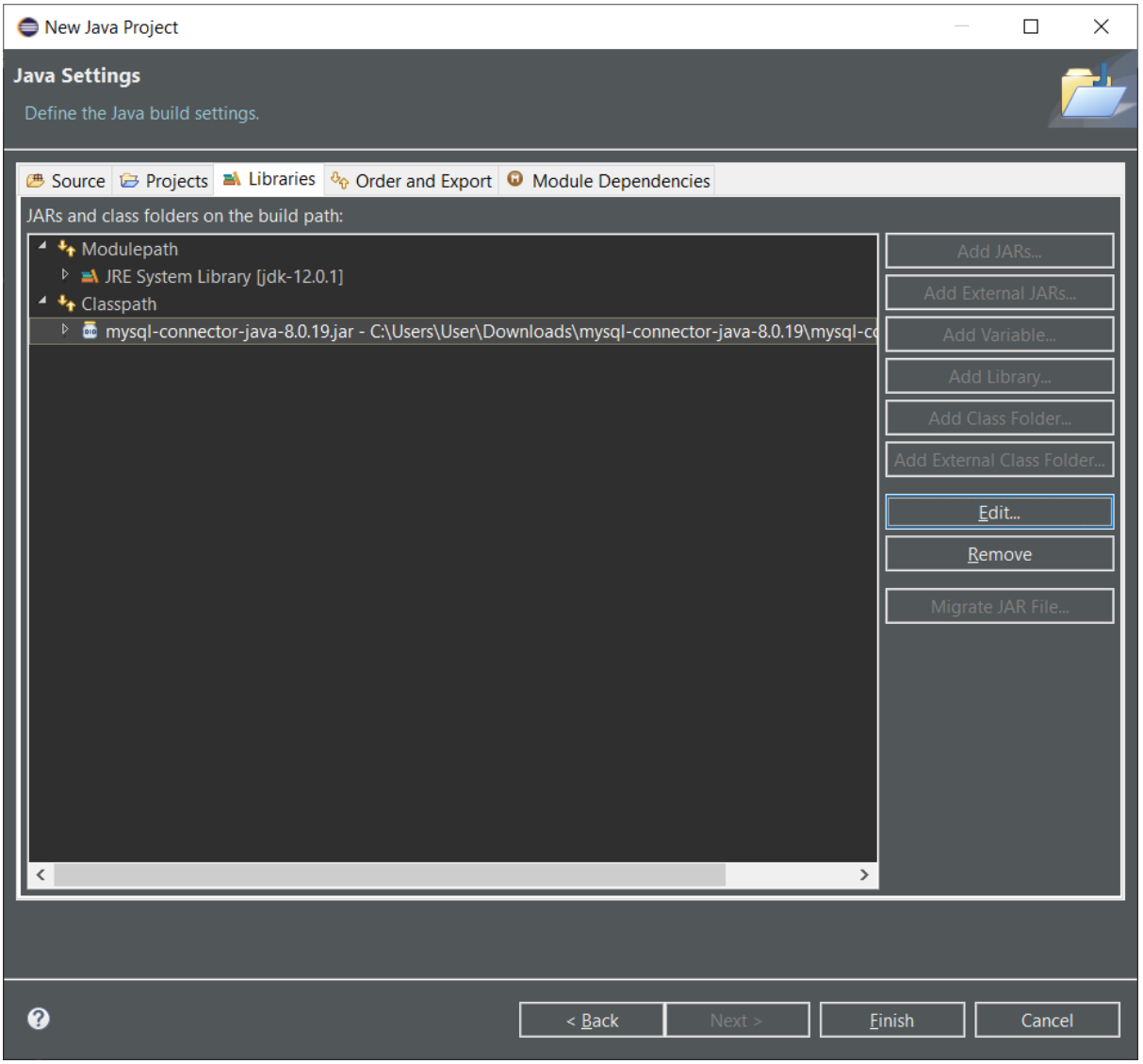
- Instead of clicking the finish option to take the default setup, click the “Next” button, which will allow for some further project setup.
- On the next screen, from the tabs on top, select the “Libraries” tab, and select the “Classpath” option.
- From here, select the “Add External JARS” option.



- Click on it and find the .jar file you've downloaded earlier. At the time of writing, the file was named Mysql-connector-java-8.0.19.jar
- Click the "Open" option.



- After that, click the “Finish” button and your project should be able to interface with your Server already.



SETTING UP A DATABASE

Before we can go any further, we need to set up a database. We will call this database “library_db” and create a table “books” within it with 4 columns: id, title, author, and qty.

id (int)	**title** (varchar(50))	**author** (varchar(50))	**qty** (int)
1001	Java for Beginners	John Holder	5
1002	Java Fundamentals	Sally Williams	5
1003	A Cup of Java	Peter Jones	6

1004	Introduction to Java	Kumar Singh	6
1005	Advanced Java	Kelly Fields	7

- As shown previously, to create a database you need to start the MySQL server and start the MySQL client:

  For Windows:
  - Enter the following into your Command Prompt:

```
cd {path-to-mysql-bin}   // Check your MySQL installed directory
mysql -u otheruser -p
```

  For Mac:
  - To start the server select 'Apple' Icon → System Preferences → MySQL → Start
  - Then enter the following into your terminal:

```
cd /usr/local/mysql/bin
./mysql -u otheruser -p
```

  Note: remember to enter the password for the user "otheruser" and not "root"

- Now, run the following SQL statements to create the database and table. Not that this time you will set up the table to use the ID field as the index, i.e. primary key.

```
create database if not exists library_db;

use library_db;

drop table if exists books;
create table books (
   id int,
   title varchar(50),
   author varchar(50),
   qty int,
   primary key (id));

insert into books values (1001, 'Java for Beginners', 'John Holder', 5);
insert into books values (1002, 'Java Fundamentals', 'Sally Williams', 5);
insert into books values (1003, 'A Cup of Java', 'Peter Jones', 6);
```

```
insert into books values (1004, 'Introduction to Java', 'Kumar Singh', 6);
insert into books values (1005, 'Advanced Java', 'Kelly Fields', 7);

select * from books;
```

JDBC PROGRAMMING

We will now demonstrate JDBC programming using some examples (Hock-Chuan, 2020b). The code below shows how to select, insert, update and delete entries in a database.

Issuing SQL statements:

- Create a new Java source file called **Main.java**. (Remember: the file name must be the same as the class name)
- Save the file in a directory of your choice.
- Enter the following code into your newly created file:

```
import java.sql.*;
public class Main {

    public static void main(String[] args) {
        try {
                // Connect to the library_db database, via the jdbc:mysql:
channel on localhost (this PC)
            // Use username "otheruser", password "swordfish".
            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/library_db?useSSL=false",
                    "otheruser",
                    "swordfish"
                    );
            // Create a direct line to the database for running our queries
            Statement statement = connection.createStatement();
            ResultSet results;
            int rowsAffected;
            // Set up finished, do some stuff:

                 // executeQuery: runs a SELECT statement and returns the
results.
```

```
                results = statement.executeQuery("SELECT title, qty FROM
books");
            // Loop over the results, printing them all.
            while (results.next()) {
                    System.out.println(results.getString("title") + ", "
+results.getInt("qty"));
            }

            // Add a new book:
            rowsAffected = statement.executeUpdate(
                        "INSERT INTO books VALUES (3001, 'Programming 101',
'Jane Doe', 1)"
                );
              System.out.println("Query complete, " + rowsAffected + " rows
added.");
            printAllFromTable(statement);

            // Change a book:
            rowsAffected = statement.executeUpdate(
                    "UPDATE books SET qty=500 WHERE id=1001"
                );
              System.out.println("Query complete, " + rowsAffected + " rows
updated.");
            printAllFromTable(statement);

            // Clear a book:
            rowsAffected = statement.executeUpdate(
                    "DELETE FROM books WHERE id=3001"
                );
              System.out.println("Query complete, " + rowsAffected + " rows
removed.");
            printAllFromTable(statement);

            // Close up our connections
            results.close();
            statement.close();
            connection.close();

        } catch (SQLException e) {
                // We only want to catch a SQLException - anything else is
off-limits for now.
```

```
            e.printStackTrace();
        }
    }

    /**
     * Method printing all values in all rows.
     * Takes a statement to try to avoid spreading DB access too far.
     *
     * @param a statement on an existing connection
     * @throws SQLException
     */
        public static void printAllFromTable(Statement statement) throws
SQLException{

            ResultSet results = statement.executeQuery("SELECT id, title,
author, qty FROM books");
        while (results.next()) {
            System.out.println(
                    results.getInt("id") + ", "
                    + results.getString("title") + ", "
                    + results.getString("author") + ", "
                    + results.getInt("qty")
                );
        }
    }
}
```

A note from our coding mentor

Melanie

You may encounter an error when trying to run your code that looks something like this: "`SQLException: Access denied for user 'otheruser'@'localhost'`*". To solve this, from the MySQL console, execute "*`ALTER USER 'otheruser'@'localhost' IDENTIFIED WITH mysql_native_password BY 'your_password_here';`*" to activate the* `mysql_native_password` *plugin. Then do "*`FLUSH PRIVILEGES;`*" to reset.*

Let's take a look at the program you just wrote. Firstly, notice that there is not much actual programming involved in JDBC programming. You simply need to specify the database-URL, write the SQL query, and process the query result. The rest of the code can be thought of as a standard template for a JDBC program.

Now let's look at the following lines of code:

```
Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/library_db?useSSL=false",
                    "otheruser",
                    "swordfish"
                    );
```

`jdbc:mysql://localhost:3306/library_db?useSSL=false` is the database-URL and takes the following form: `jdbc:mysql://{host}:{port}/{database-name}` Port represents the TCP port number of the MySQL server and database-name is the name of the database, which is in this instance library_db. "otheruser" and "swordfish" are the username and password of an authorised MySQL user.

Next let's examine this statement:

```
Statement statement = connection.createStatement();
```

In this line, we are allocating a Statement object inside the Connection using `connection.createStatement()`.

Select

The next step in our process is to execute a SQL command (in this case Select). To do this we use the method `statement.executeQuery("SELECT ...")`, which returns the query result in a ResultSet object called `results`. The ResultSet models the table which is returned. This table can then be accessed using a row cursor. The cursor is initially positioned before the first row in ResultSet. It is then moved, using `results.next()`, to the first row. `results.getXxx(column Name)` can then be used to retrieve the value of the column for that row. *Xxx* corresponds to the data type of that column, for example, int, float, double and String. At the last row the `results.next()` returns false, which terminates the while-loop. `results.getString(columnName)` can also be used to retrieve all data types.

ResultSet columns within each row should be read in a left-to-right order, and each column should be read only once using the `getXxx()` methods. Issuing `getXxx()` to a cell more than once can cause an error.

A feature of JDK called try-with-resources automatically closes all the opened resources in the try-clause. It therefore closes the Connection and Statement objects automatically.

Insert and Delete

The insert and delete actions are fairly straightforward, as you can see from the code above. It is worth noting that you cannot add an entry if an existing entry has the same primary key; you will first have to delete the existing entry.

Update

Unlike Select, where we use `executeQuery()` to return a ResultSet object, Update, Insert and Delete return an int value, which indicates the number of records affected..

For more information on JDBC, you can find the homepage **here** or the online JDBC online tutorial **here**.

SPECIFIC OUTCOME 4 (US 114048)

15. TESTING AND DEBUGGING

Normally the first step in debugging is to attempt to reproduce the problem. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Also, specific user environment and usage history can make it difficult to reproduce the problem.

After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. For example, a bug in a compiler can make it crash when parsing some large source file. However, after simplification of the test case, only few lines from the original source file can be sufficient to reproduce the same crash. Such simplification can be made manually, using a divide-and-conquer approach. The programmer will try to remove some parts of original test case and check if the problem still exists. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear.

After the test case is sufficiently simplified, a programmer can use a debugger tool to examine program states (values of variables, plus the call stack) and track down the origin of the problem(s). Alternatively, tracing can be used. In simple cases, tracing is just a few print statements, which output the values of variables at certain points of program execution.

- **Print debugging** (or tracing) is the act of watching (live or recorded) trace statements, or print statements, that indicate the flow of execution of a process.
- **Remote debugging** is the process of debugging a program running on a system different from the debugger. To start remote debugging, a debugger connects to a remote system over a network. The debugger can then control the execution of the program on the remote system and retrieve information about its state.
- **Post-mortem debugging** is debugging of the program after it has already crashed. Related techniques often include various tracing techniques, and/or analysis of memory dump (or core dump) of the crashed process. The dump of the process could be obtained automatically by the system (for example, when process has terminated due to an unhandled exception), or by a programmer-inserted instruction, or manually by the interactive user.
- **"Wolf fence" algorithm**: Edward Gauss described this simple but very useful and now famous algorithm in a 1982 article for communications of the ACM as follows: "There's one wolf in Alaska; how do you find it? First build a fence down the middle of the state, wait for the wolf to howl, determine which side of the fence it is on. Repeat process on that side only, until you get to the point where you can see the wolf." This is implemented e.g. in the Git version control system as the command git bisect, which uses the above algorithm to determine which commit introduced a particular bug.
- **Delta Debugging** - technique of automating test case simplification
- **Saff Squeeze** - technique of isolating failure within the test using progressive inlining of parts of the failing test.

SPECIFIC OUTCOME 5 (US114048)

16. DOCUMENTATION

The documents associated with a software project and the system being developed have a number of associated requirements:

1. They should act as a communication medium between members of the development team.
2. They should be a system information repository to be used by maintenance engineers.
3. They should provide information for management to help them plan, budget, and schedule the software development process.
4. Some of the documents should tell users how to use and administer the system.

Product documentation

Product documentation is concerned with describing the delivered software product. Unlike most process documentation, it has a relatively long life. It must evolve in step with the product which it describes. Product documentation includes user documentation which tells users how to use the software product and system documentation which is principally intended for maintenance engineers.

User Documentation

Users of a system are not all the same. The producer of documentation must structure it to cater for different user tasks and different levels of expertise and experience. It is particularly important to distinguish between end-users and system administrators:

1. End-users use the software to assist with some task. This may be flying an aircraft, managing insurance policies, writing a book, etc. They want to know how the software can help them.

2. System administrators are responsible for managing the software used by end-users. This may involve acting as an operator if the system is a large mainframe system, as a network manager in the system involves a network of workstations or as a technical guru who fixes end-users software problems and who liaises between users and the software supplier.

To cater for these different classes of user and different levels of user expertise, there are at least 5 documents (or perhaps chapters in a single document) which should be delivered with the software system (Figure1).

The *functional description* of the system outlines the system requirements and briefly describes the services provided. This document should provide an overview of the system. Users should be able to read this document with an introductory manual and decide if the system is what they need.

The *system installation document* is intended for system administrators. It should provide details of how to install the system in a particular environment. It should contain a description of the files making up the system and the minimal hardware configuration required. The permanent files which must be established, how to start the system and the configuration dependent files which must be changed to tailor the system to a particular host system should also be described. The use of automated installers for PC software has meant that some suppliers see this document as unnecessary. In fact, it is still required to help system managers discover and fix problems with the installation.

The *introductory manual* should present an informal introduction to the system, describing its 'normal' usage. It should describe how to get started and how end-users might make use of the common system facilities. It should be liberally illustrated with examples. Inevitably beginners, whatever their background and experience, will make mistakes. Easily discovered information on how to recover from these mistakes and restart useful work should be an integral part of this document.

A more general *system administrator's guide* should be provided for some types of system such as command and control systems. This should describe the messages generated when the system interacts with other systems and how to react to these messages. If system hardware is involved, it might also explain the operator's task in maintaining that hardware. For example, it might describe how to clear faults in the system console, how to connect new peripherals, etc.

As well as manuals, other, easy-to-use documentation might be provided. A quick reference card listing available system facilities and how to use them is particularly convenient for experienced system users. On-line help systems, which contain brief information about the system, can save the user spending time in consultation of manuals although should not be seen as a replacement for more comprehensive documentation

System Documentation

System documentation includes all of the documents describing the system itself from the requirements specification to the final acceptance test plan. Documents describing the design, implementation and testing of a system are essential if the program is to be understood and maintained. Like user documentation, it is important that system documentation is structured, with overviews leading the reader into more formal and detailed descriptions of each aspect of the system.

For large systems that are developed to a customer's specification, the system documentation should include:

1. The requirements document and an associated rationale.
2. A document describing the system architecture.
3. For each program in the system, a description of the architecture of that program.
4. For each component in the system, a description of its functionality and interfaces.
5. Program source code listings. These should be commented where the

comments should explain complex sections of code and provide a rationale for the coding method used. If meaningful names are used and a good, structured programming style is used, much of the code should be self-documenting without the need for additional comments. This information is now normally maintained electronically rather than on paper with selected information printed on demand from readers.

6. Validation documents describing how each program is validated and how the validation information relates to the requirements.

Compulsory Task 7

Follow these steps:

- Ensure that your environment is set up and you have followed all the steps outlined in this task.
- Using the MySQL client:
    - Insert the following 3 new rows into the java_programming table:

id	name	grade
55	Carl Davis	61
66	Dennis Fredrickson	88
77	Jane Richards	78

    - Select all records with a grade between 60 and 80.
    - Change Carl Davis's grade to 65.
    - Delete Dennis Fredrickson's row.
    - Change the grade of all people with an id greater than 55 to 80.
- After executing each instruction given above, take a screenshot of your console and send it to your mentor. Number your screenshots 1 to 5 in order of execution.

- Modify the Java program UpdateTest.java to set the qty for Introduction to Java to 0.

- Modify the Java program as follows: InsertTest.java to delete all books with id > 8000; and insert: (8001, 'Java ABC', 'Kevin Jones', 3) and (8002, 'Java XYZ', 'Kevin Jones', 5);

- Test and debug your code.

- Include comments to explain complex sections of code and provide a rationale for the coding method used.

Compulsory Task 8

Follow these steps:

- Create a program that can be used by a bookstore clerk. Use the comment functionality to document what you have done making the code easily understandable to anyone reading your program. The program should allow the clerk to:
    - enter new books into the database
    - update book information
    - delete books from the database
    - search the database to find a specific book.

- Create a database called **ebookstore** and a table called **books**. The table should have the following structure (note that the id field is the primary key):

id	Title	Author	Qty
3001	A Tale of Two Cities	Charles Dickens	30
3002	Harry Potter and the Philosopher's Stone	J.K. Rowling	40

3003	The Lion, the Witch and the Wardrobe	C. S. Lewis	25
3004	The Lord of the Rings	J.R.R Tolkien	37
3005	Alice in Wonderland	Lewis Carroll	12

- Populate the table with the above values. You can also add your own values if you wish.

- The program should present the user with the following menu:

```
1. Enter book
2. Update book
3. Delete book
4. Search books
0. Exit
```

  The program should perform the function that the user selects. The implementation of these functions is left up to you.

- Include comments to explain complex sections of code and provide a rationale for the coding method used.

- Test and debug your code.

- Feel free to add more functionality and complexity to the program. This is your chance to show off all the programming concepts you have learnt so far!

Self-assessment table

Intended learning outcome (SAQA US 114048)	Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- **I can successfully**:			
Write program code for database access for a computer	Create and read data	Update data	Delete data	Use criteria to select a limited subset of data searched or on

application using SQL (SO 3)				which specific operations are performed	
Test programs for a computer application that accesses a database using SQL (SO 4)	Understand and be able to apply the testing and debugging process for code		Explain and apply including different types of and approaches to debugging (e.g. print, remote, post-mortem, wolf-fence, delta, Saff squeeze)		
Document programs for a computer application that accesses a database using SQL (SO 5)	Understand and be able to create product documentation	Understand and be able to create user documentation	Understand and be able to create system documentation		
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**:				
Install and run MySQL servers and clients, and use the JDBC to create java applications that interact with these and run MySQL code	Download and install MySQL and the JDK	Run a MySQL server and client sessions	Create MySQL users and databases	Connect a java application to a MySQL server	Within a MySQL database, create new tables,insert records, query, update, and delete

Logical Problem Solving & Error Detection techniques

Aligned to SAQA US 115367

Learning assumed to be in place: basic Mathematics skills at NQF level 3; Information gathering skills at NQF level 5

Introduction

WELCOME TO THE INTRODUCTION TO THE JDBC TASK!

In this lesson, we are going to dive into the topic of logical problem solving using logic gates, truth tables, boolean algebra, and Karnaugh maps. We will also look at error detection and correction methods. At the end, as usual, there is a task for you to put into practice what you have learned.

INTENDED LEARNING OUTCOMES

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

- Describe different approaches to problem solving
- Use logical operators in descriptions of rules and relationships in a problem situation
- Simplify Boolean expressions with Boolean algebra and Karnaugh maps
- Describe the basic concepts of error detection

Refer to the self-assessment table provided after the compulsory 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.

There are a number of concepts you need to understand in order to be able to demonstrate logical problem solving and error detection techniques. These are explained below, drawing from the Bella Visage Learner Guide for the *National Certificate: Information Technology (Systems Development)* (licenced for use by HyperionDev).

SPECIFIC OUTCOME 1 (US 115367)

1. APPROACHES TO LOGICAL PROBLEM SOLVING AND ERROR DETECTION

In this section, you're going to be introduced to different problem solving techniques and identify situations where specific problem solving techniques would be more suitable than others. We're going to consider the top-down problem solving approach with relatable real life problems, and facilitate break down problems pictorially.

Top-down and **bottom-up** problem-solving strategies are both strategies of information processing and knowledge ordering, used in a variety of fields including the humanities and sciences, management and organization, and software development in IT. In practice, they can be seen as a style of thinking and ordering steps.

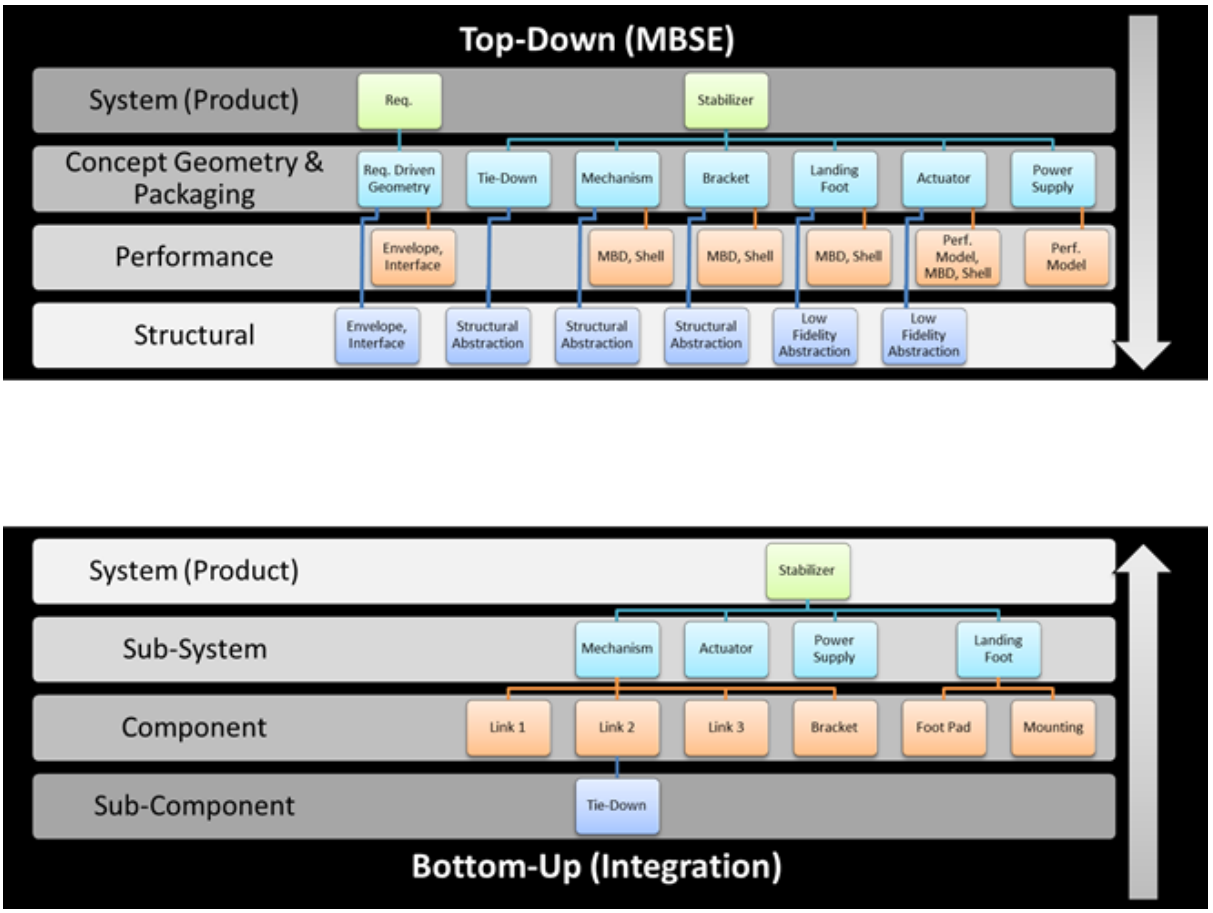
A **top-down approach** (also known as stepwise design and in some cases used as a synonym of decomposition) is essentially the breaking down of a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes", a concept used to make problems easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model. Essentially, the top down approach starts with the big picture, and breaks down from there into smaller segments.

A **bottom-up approach** is the piecing together of systems to give rise to more complex systems, thus making the original systems sub-systems of the emergent system. Bottom-up processing is a type of information processing aligned for things like using incoming data from the environment to form a perception. Information enters the eyes in one direction (input), and is then turned into an image by the brain that can be interpreted and recognized as a perception (output). In a bottom-up approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often

resembles a "seed" model, whereby the beginnings are small but eventually grow in complexity and completeness. However, "organic strategies" may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.
In the software development process, the top-down and bottom-up approaches play a key role.

Top-down approaches emphasize planning and a complete understanding of the system. It is inherent that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system. Top-down approaches are implemented by attaching the stubs in place of the module. This, however, delays testing of the ultimate functional units of a system until significant design is complete.

Bottom-up approaches emphasize coding and early testing, which can begin as soon as the first module has been specified. This approach, however, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system, and that such linking may not be as easy as first thought. Reusability of code is one of the main benefits of the bottom-up approach



Top-Down vs. Bottom-Up approaches

SPECIFIC OUTCOME 2 (US 115367)

17. LOGICAL PROBLEM SOLVING USING LOGIC GATES

In this lesson, you're going to use logical operators in drawing truth tables, and combine different operators to form Boolean expressions by setting up truth tables We will .apply this to examples of problem situations where a specific operator can be used, and consider which of the operators should be used to represent given situations.

TRUTH TABLES

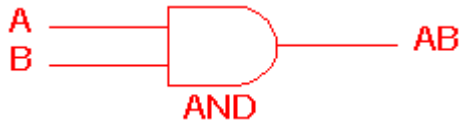
Truth tables are used to help show the function of a logic gate. If you are unsure about truth tables and need guidance on how to go about drawing them for

individual gates or logic circuits then use the truth tablesection link. We will look at Truth tables again in the next section on Boolean algebra.

Logic Gates

Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

AND gate



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

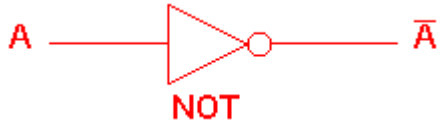
The AND gate is an electronic circuit that gives an output of TRUE, i.e. 1., only if **all** its inputs are TRUE (1). A dot (.) is used to show the AND operation i.e. A**.**B, as per the 3rd column in the truth table above). Bear in mind that this dot is sometimes omitted i.e. AB .

OR gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives an output of TRUE (1) if **one or more** of its inputs are TRUE. A plus (+) is used to show the OR operation, as per the 3rd column in the truth table above).

NOT gate



NOT gate	
A	$\overline{A}$
0	1
1	0

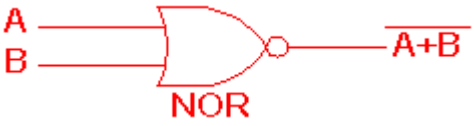
The NOT gate is an electronic circuit that outputs an inverted version of the input. (it is also known as an **inverter**). If the input variable is **A**, the inverted output is expressed as **NOT A**. This is also shown as A' (say "A prime"), or A with a bar over the top, as per the 3rd column in the truth table above.

NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

This is a **NOT-AND** gate which is equal to an **AND gate followed by a NOT gate**. The outputs of all NAND gates are TRUE (1) if any of the inputs are FALSE (0). The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

NOR gate



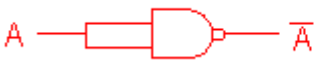
2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

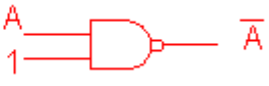
This is a **NOT-OR** gate which is equal to an **OR gate followed by a NOT gate**. The outputs of all NOR gates are FALSE (0)if any of the inputs are TRUE (1).

The symbol is an OR gate with a small circle on the output, where once again the small circle represents inversion.

Making NOT gates out of NAND and NOR gates

The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. This can also be done using NOR logic gates in the same way.



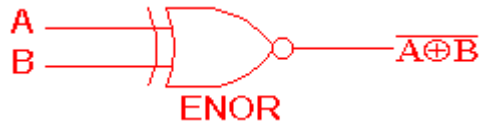


EXOR gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

This **Exclusive-OR** (EXOR or XOR) gate is a circuit which will give an output of TRUE (1) if **either, but not both**, of its two inputs are TRUE (1). An encircled plus sign (⊕) is used to show the EOR operation.

EXNOR gate



2 Input EXNOR gate		
A	B	$\overline{A⊕B}$
0	0	1
0	1	0
1	0	0
1	1	1

This **Exclusive-NOR** (EXNOR or XNOR) gate circuit does the opposite to the EXOR gate. It will give an output of FALSE (0) if **either, but not both**, of its two inputs are TRUE (1). The symbol is an EXOR gate with a small circle on the output. As usual the small circle represents inversion.

The NAND and NOR gates are called *universal functions*, as with either one the AND, OR and NOT functions can be generated.

Note: A function in *sum of products* form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates. A function in product of sums form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

Table 1: Logic gate symbols

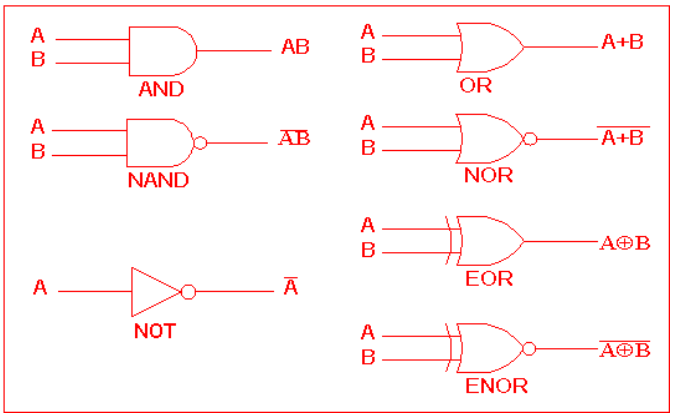


Table 2 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the other gate functions. Also note that a truth table with 'n' inputs has 2^n rows. You can compare the outputs of different gates.

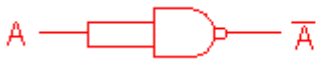
Table 2: Logic gates representation using the Truth table

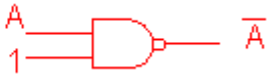
NOT gate	
A	$\overline{A}$
0	1
1	0

INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Example

A **NAND** gate can be used as a NOT gate using either of the following wiring configurations.





(You can check this out using a truth table as well.)

SPECIFIC OUTCOME 3 (US 115367)

18. BOOLEAN EXPRESSIONS AND THEIR SIMPLIFICATION

In this lesson, you're going to consider the rules of Boolean algebra, and use these to simplify basic expressions. You're also going to be introduced to Karnaugh maps to represent Boolean expressions, and practice simplifying these by writing down the simplified expression from the map.

LAWS OF BOOLEAN ALGEBRA

The following is a set of simplified ways to remember the laws of Boolean algebra:

1. Anything ANDed with a 0 is equal to 0 → A.0 = 0
2. Anything ANDed with a 1 is equal to itself →. A. 1 = A
3. Anything ORed with a 0 is equal to itself. → A + 0 = A
4. Anything ORed with a 1 is equal to 1 → A + 1 = 1
5. Anything ANDed with itself is equal to itself → A. A = A
6. Anything ORed with itself is equal to itself → A + A = A
7. Anything ANDed with its own complement equals 0 → $A.\bar{A} = 0$
8. Anything ORed with its own complement equals 1 → $A+\bar{A} = 1$
9. Anything complemented twice is equal to the original → $\bar{\bar{A}} = A$
10. The two variable rule → $A + \bar{A}B = A + B$
11. De Morgan's theorem:

$\overline{A+B} = \bar{A}.\bar{B}$	NOT(A OR B)=(NOT A) AND (NOT B)
$\overline{A.B} = \bar{A}+\bar{B}$	NOT (A AND B) = (NOT A) OR (NOT B)

Source: https://grace.bluegrass.kctcs.edu/~kdunn0001/files/Simplification/4_Simplification_print.html

BOOLEAN EXPRESSIONS/FUNCTIONS

Boolean algebra deals with binary variables and logical operation. A **Boolean Function** is described by an algebraic expression called a **Boolean expression** which consists of binary variables, the constants 0 and 1, and the logical operation symbols. Consider the following example

$F(A, B, C, D) = A + \overline{BC} + ADC$ Equation No. 1

Boolean Function — Boolean Expression

Here the left side of the equation represents the output Y. So we can state equation no. 1:

$Y = A + \overline{BC} + ADC$

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.

It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

$$F(A, B, C) = A + BC$$

The output will be TRUE (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2^n where n is the number of input variables (n=3 for the given equation). Hence there are $2^3 = 8$ possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Methods to simplify the Boolean function

The methods used for simplifying the Boolean function are as follows.

- Karnaugh-map or K-map.
- NAND gate method

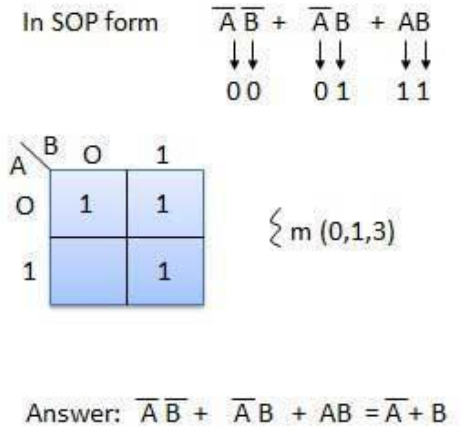
Karnaugh-map or K-map

The Boolean theorems and De-Morgan's theorems are useful in manipulating the logical expression. We can realize the logical expression using gates. The number of logic gates required for the realization of a logical expression should be reduced to the minimum possible value by using the K-map method. This method can be used in two ways, the sum of products form and the product of sums form.

SUM OF PRODUCTS (SOP) FORM

The SOP form is in the form of the sum of three terms AB, AC, BC with each individual term being a product of two variables, e.g. A.B or A.C etc. Therefore, the expression is known as an expression in SOP form. The sum and products in SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions. Remember that 0 is the value given for barred (FALSE) variables, and 1 is the value given for the unbarred (TRUE) variables. The SOP form is indicated by Σ.

An example of SOP form is as follows:

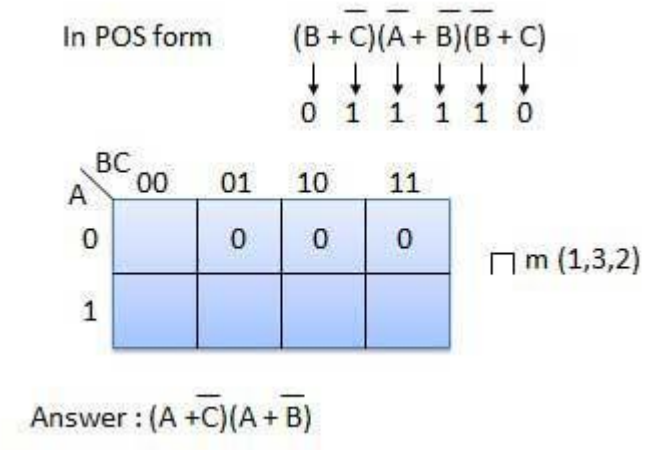


Answer: $\overline{A}\,\overline{B}$ + $\overline{A}$B + AB = $\overline{A}$ + B

PRODUCT OF SUMS (POS) FORM

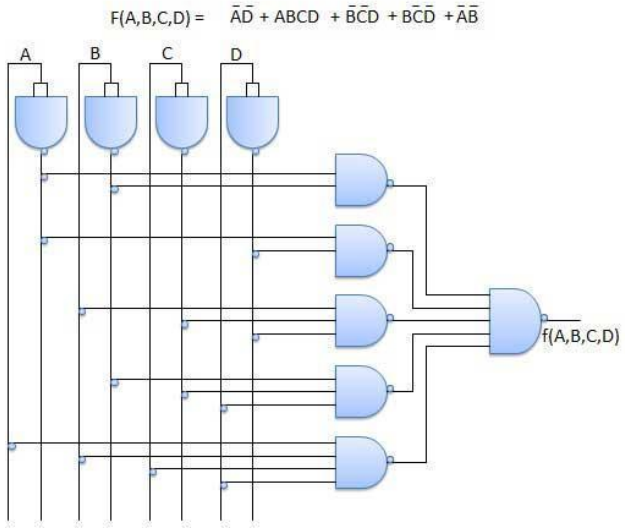
The SOP form is the product of three terms (A+B), (B+C) and (A+C) with each term being the sum of two variables. Such expressions are said to be in the product of sums (POS) form. Remember that 0 is the value given for barred (FALSE) variables, and 1 is the value given for the unbarred (TRUE) variables. The POS form is indicated by Π.

An example of POS form is as follows.



NAND gates Realization

NAND gates can be used to simplify Boolean functions as shown in the example below.

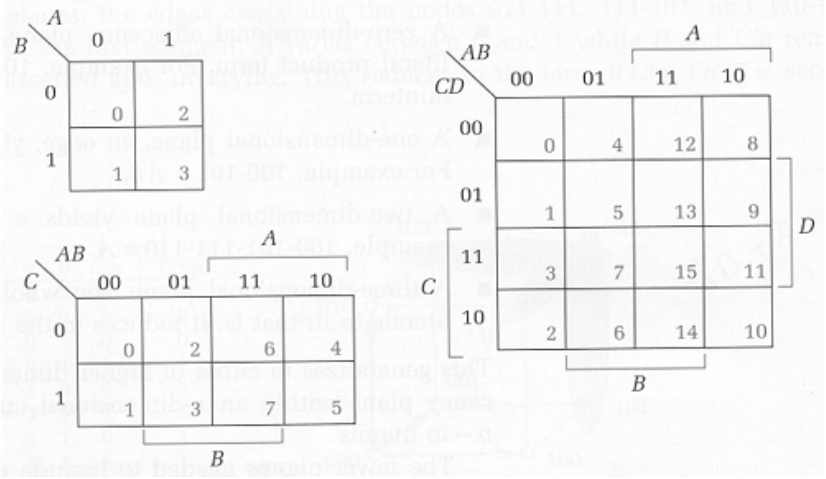


KARNAUGH MAPS

A Karnaugh Map, or K-Map, is a grid-like representation of a truth table. It is really just another way of presenting a truth table, but the mode of presentation gives more insight. A Karnaugh map has zero and one entries at different positions. Each position in a grid corresponds to a truth table entry.

A K-map shows the value of a function for every combination of input values just like a truth table, but a K-map spatially arranges the values so it is easy to

find common terms that can be factored out. The image below shows the form of a 2, 3, and 4 variable K-map.



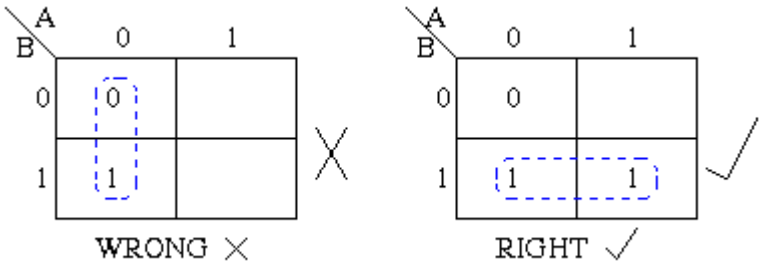
How Can a Karnaugh Map Help?

At first, it might seem that the Karnaugh Map is just another way of presenting the information in a truth table. In one way that's true. However, any time you have the opportunity to use another way of looking at a problem, this may offer advantages. In the case of the Karnaugh Map the advantage is that the Karnaugh Map is designed to present the information in a way that allows easy grouping of terms that can be combined.

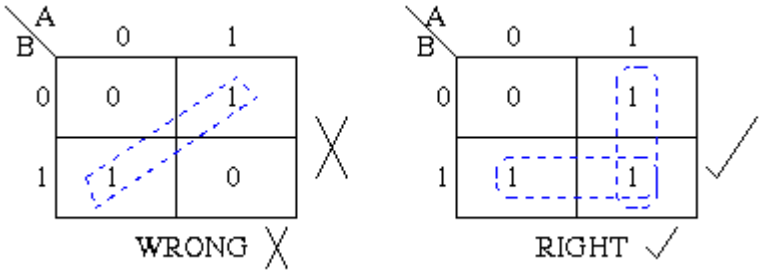
Karnaugh Maps - Rules of Simplification

The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

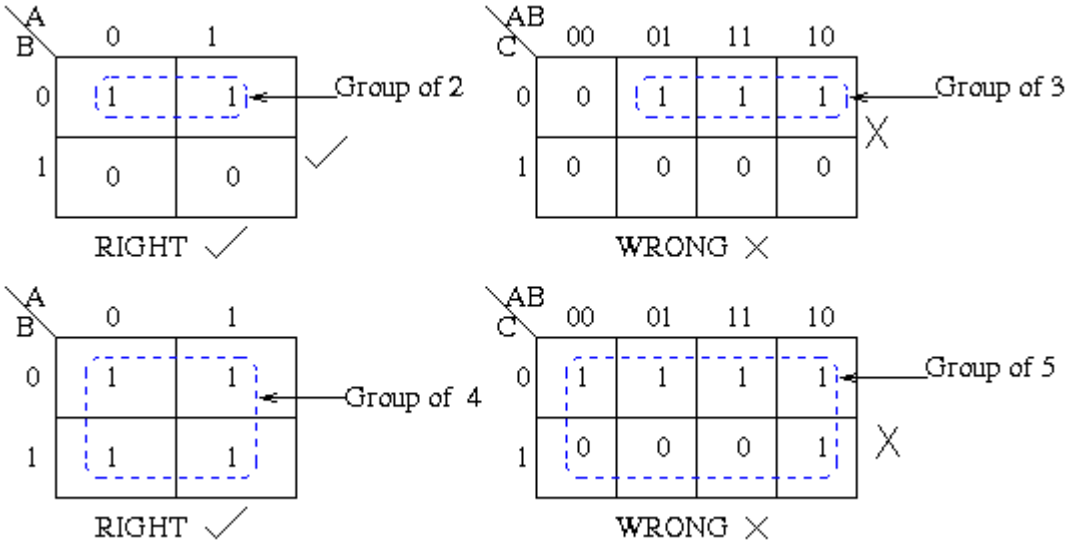
- **Groups may not include any cell containing a zero**



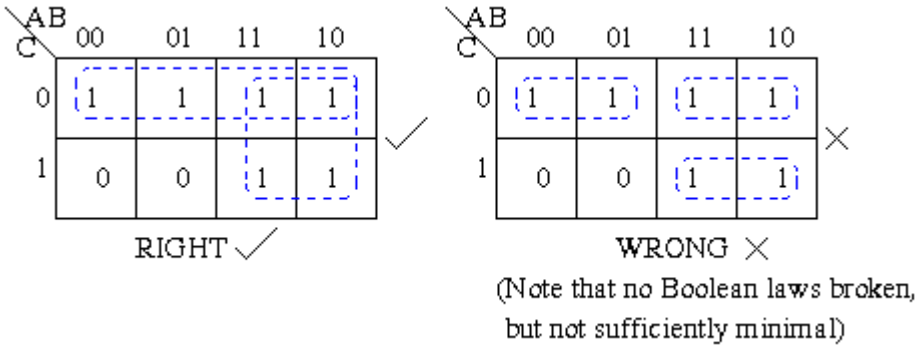
- **Groups may be horizontal or vertical, but not diagonal.**



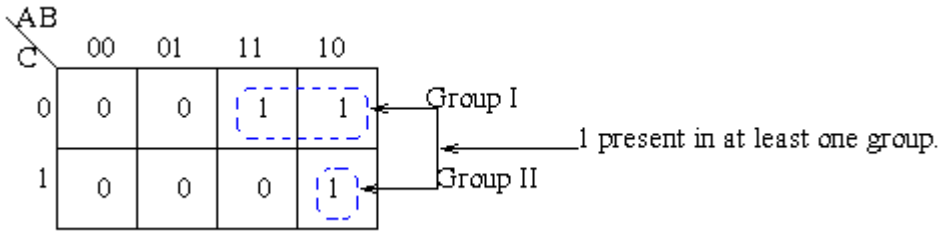
- **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
  That is if n = 1, a group will contain two 1's since $2^1 = 2$.
  If n = 2, a group will contain four 1's since $2^2 = 4$.



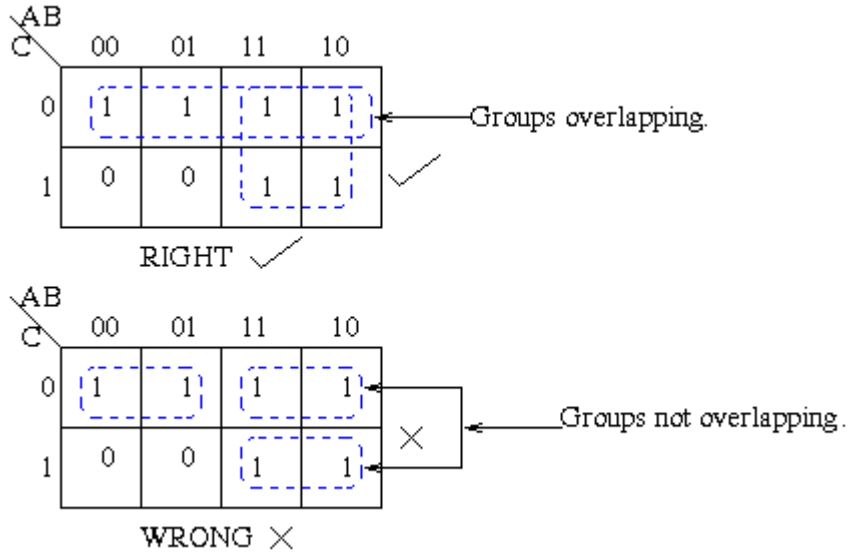
- **Each group should be as large as possible.**



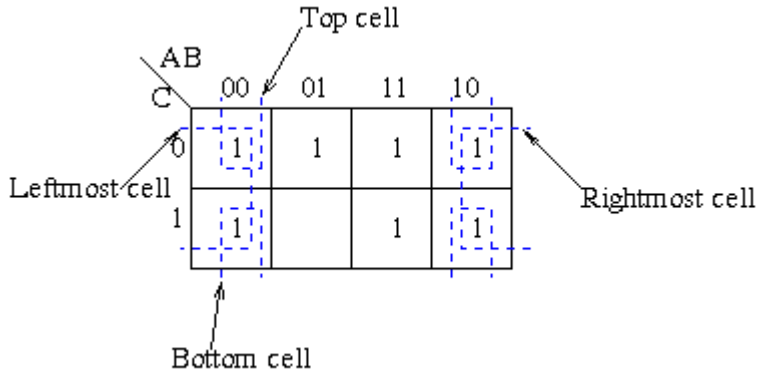
- **Each cell containing a *one* must be in at least one group.**



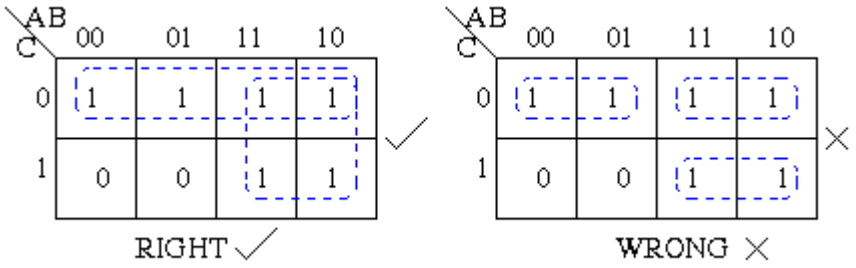
- **Groups may overlap.**



- **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**



K-map rules summary

1. No zeros allowed.
2. No diagonals.
3. Only 2^n (powers-of-2 numbers) of cells in each group .
4. Groups should be as large as possible.
5. Every 1 must be in at least one group.
6. Overlapping is allowed.
7. Wraparound is allowed.
8. Fewest number of groups possible.

SPECIFIC OUTCOME 4 (US 115367)

19. BASIC CONCEPTS OF ERROR DETECTION

In this lesson, you're going to look at how to identify common causes of errors, error isolation techniques, and various testing techniques.

COMMON CAUSES OF ERRORS

In computer programming, a logic error is a bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash). A logic error produces unintended or undesired output or other behavior, although it may not immediately be recognized as such.

Logic errors occur in both compiled and interpreted languages. Unlike a program with a syntax error, a program with a logic error is a valid program in the language, though it does not behave as intended. The only clue to the existence of logic errors is the production of wrong solutions.

Common causes

The mistake could be a simple error in a statement (for example, an incorrect formula), an error in an algorithm, or even the wrong algorithm selected. There are also numerous other causes, such as incorrect type casting, variable scoping, missing code fragments and wrong problem (or requirement) interpretation.

Debugging logic errors

One of the ways to find these type of errors is to output the program's variables to a file or on the screen in order to define the error's location in code. Although this will not work in all cases, for example when calling the wrong subroutine, it is the easiest way to find the problem if the program uses the incorrect results of a bad mathematical calculation.

Truncation errors in numerical integration are of two kinds:

local truncation errors – the error caused by one iteration, and

global truncation errors – the cumulative error caused by many iterations.

ERROR ISOLATION TECHNIQUES

What is Error Correction and Detection?

Error detection and correction has great practical importance in maintaining data (information) integrity across noisy Communication Networks channels and lessthan- reliable storage media.

Error Correction: Send additional information so incorrect data can be corrected and accepted. Error correction is the additional ability to reconstruct the original, error-free data.

There are two basic ways to design the channel code and protocol for an error correcting system :

- **Automatic Repeat-Request (ARQ) :** The transmitter sends the data and also an error detection code, which the receiver uses to check for errors, and request retransmission of erroneous data. In many cases, the request is implicit; the receiver sends an acknowledgement (ACK) of correctly received data, and the transmitter re-sends anything not acknowledged within a reasonable period of time.
- **Forward Error Correction (FEC) :** The transmitter encodes the data with an error-correcting code (ECC) and sends the coded message. The receiver never sends any messages back to the transmitter. The receiver decodes what it receives into the "most likely" data. The codes are designed so that it would take an "unreasonable" amount of noise to trick the receiver into misinterpreting the data.

Error Detection: Send additional information so incorrect data can be detected and rejected. Error detection is the ability to detect the presence of errors caused by noise or other impairments during transmission from the transmitter to the receiver.

VARIOUS TESTING TECHNIQUES

The box approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

White-Box testing

White-box testing (also known as **clear box testing**, **glass box testing**, **transparent box testing** and **structural testing**) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs

to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration, and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

- API testing (application programming interface) – testing of the application using public and private APIs
- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies
- Mutation testing methods
- Static testing methods

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory and specification-based testing.

Compulsory Task 9

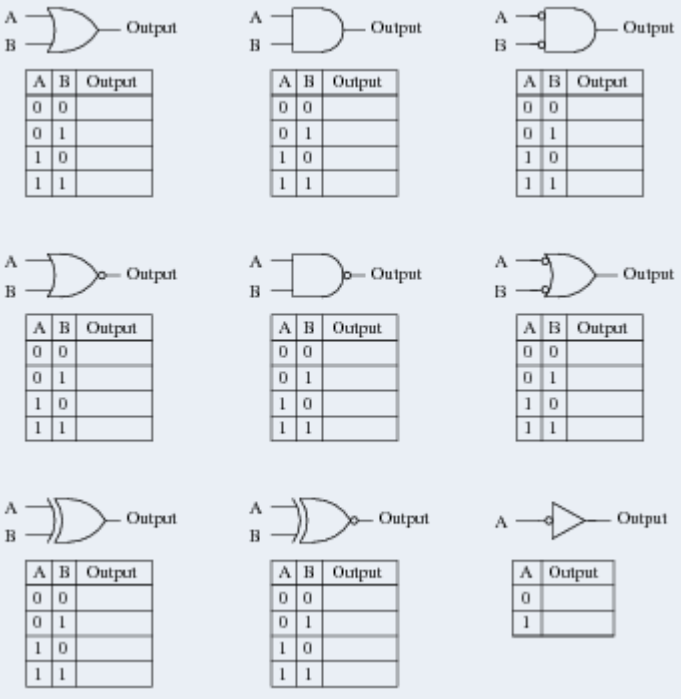
- Describe two different approaches to problem solving.

(2 x 2 = 4)

- Answer the following and say what the theorem that governs this relationship is called.
    - NOT(A OR B)=
    - NOT (A AND B) =
    - The theorem is:

(1+1+1 = 3)

- Identify each of these logic gates by name, and complete their respective truth tables



(9+9 = 18)

Self-assessment table

Intended learning outcome (SAQA US 115367)	Assess your own proficiency level for each intended learning outcome. Check each box when the statement becomes true- **I can successfully**:	
Describe different	Explain and apply the	Explain and apply the

approaches to problem solving (SO 1)	top- down approach		bottom-up approach	
Use logical operators in descriptions of rules and relationships in a problem situation (SO 2)	Understand and be able to create and solve simple truth tables		Understand and be able to use logic gates, and relate these to their truth tables	
Simplify Boolean expressions with Boolean algebra and Karnaugh maps (SO 3)	Use and understand the laws of Boolean algebra	Simplify Boolean expressions	Create and solve Karnaugh maps and associated truth tables	
Describe the basic concepts of error detection (SO 4)	Understand common causes of logic errors	Know how to debug logic errors	Understand use black-box testing	Understand and use white-box testing

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 (Unit 1)

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 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**

REFERENCES (Unit 2)

Adamchik, V. (2009). Sorting. Retrieved 25 February 2020, from **https://www.cs.cmu.edu/~adamchik/15-121/lectures/Sorting%20Algorithms/sorting.html**

Dalal, A. (2004). Searching and Sorting Algorithms. Retrieved 25 February 2020, from **http://www.cs.carleton.edu/faculty/adalal/teaching/f04/117/notes/searchSort.pdf**

University of Cape Town. (2014). Sorting, searching and algorithm analysis — Object-Oriented Programming in Python 1 documentation. Retrieved 25 February 2020, from **https://python-textbok.readthedocs.io/en/1.0/Sorting_and_Searching_Algorithms.html**

REFERENCES (Unit 3)

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

Coronel, C., Morris, S., & Rob, P. (2011). *Database systems* (9th ed., pp. 174-218). Boston: Cengage Learning.

Bella Visage Incorporated. (2021). 48872 National Certificate: Information Technology (Systems Development). Learner Guide.

REFERENCES (Unit 4)

Coronel, C., Morris, S., & Rob, P. (2011). *Database Systems* (9th ed., pp. 220-297). Boston: Cengage Learning.

Encyclopaedia Britannica. (2020). SQL | computer language. Retrieved 17 February 2020, from https://www.britannica.com/technology/SQL

IBM Corporation. (2010). Database SQL Programming, Triggers. (pp. 217-224). Accessed: 19 October 2021. https://www.ibm.com/docs/el/ssw_ibm_i_71/sqlp/rbafy.pdf

Mattila, M 2012, Visual representation of common SQL joins, flickr.com, accessed 17 February 2020,**https://www.flickr.com/photos/mattimattila/8190148857**.

Oracle. (2017). Types of SQL Statements. Retrieved 17 February 2020, from **https://docs.oracle.com/database/121/SQLRF/statements_1001.htm#SQLRF30042**

SQLServerTutorial. (2021). SQL Server Triggers. Accessed: 20 October 2021. https://www.sqlservertutorial.net/sql-server-triggers/

REFERENCES (Unit 5)

Ciubotaru, B., & Muntean, G. (2013). *Advanced network programming – Principles and techniques: Network application programming with Java* (p. 136). London: Springer Science & Business Media.

Hock-Chuan, C. (2020a). Java Tutorial - An Introduction to Java Database Programming (JDBC) by Examples with MySQL. Retrieved 19 February 2020, from https://www3.ntu.edu.sg/HOME/EHCHUA/PROGRAMMING/java/JDBC_Basic.html

Hock-Chuan, C. (2020b). An Introduction to Java Database Programming (JDBC) by Examples. Retrieved 20 February 2020, from https://www3.ntu.edu.sg/HOME/EHCHUA/PROGRAMMING/java/JDBC_Basic2.html

Liang, Y. (2011). *Introduction to Java programming: Comprehensive version* (8th ed., pp. 1273-1308). New Jersey: Prentice Hall.

REFERENCES (Unit 6)

Ciubotaru, B., & Muntean, G. (2013). *Advanced network programming – Principles and techniques: Network application programming with Java* (p. 136). London: Springer Science & Business Media.

Hock-Chuan, C. (2020a). Java Tutorial - An Introduction to Java Database Programming (JDBC) by Examples with MySQL. Retrieved 19 February 2020, from https://www3.ntu.edu.sg/HOME/EHCHUA/PROGRAMMING/java/JDBC_Basic.html

Hock-Chuan, C. (2020b). An Introduction to Java Database Programming (JDBC) by Examples. Retrieved 20 February 2020, from https://www3.ntu.edu.sg/HOME/EHCHUA/PROGRAMMING/java/JDBC_Basic2.html

Liang, Y. (2011). *Introduction to Java programming: Comprehensive version* (8th ed., pp. 1273-1308). New Jersey: Prentice Hall.

Author unknown. *Electronics*Tutorials. Retrieved 12 October 2021, from

https://www.electronics-tutorials.ws/boolean/bool_6.html