Student workbook

Database cluster

ICTDBS403 Create basic databases

ICTPRG402 Apply query language

ICTSAD501 Model data objects

­

**TAFE NSW would like to pay our respect and acknowledge Aboriginal and Torres Strait Islander Peoples as the Traditional Custodians of the Land, Rivers and Sea. We acknowledge and pay our respect to the Elders, both past and present of all Nations.**

Version: 20200803

Date created: 20 October 2019

Date modified: 3 August 2020

For queries contact: Technology and Business Services SkillsPoint

© TAFE NSW 2020  
RTO Provider Number 90003 | CRICOS Provider Code: 00591E

This resource can be found in the TAFE NSW Learning Bank.

The content in this document is copyright © TAFE NSW 2020 and should not be reproduced without the permission of TAFE NSW. Information contained in this document is correct at time of printing: 03 August 2020. For current information please refer to our website or your teacher as appropriate.

Contents

[Icon legends 7](#_Toc47344109)

[Getting started 8](#_Toc47344110)

[What will I learn by completing this workbook? 8](#_Toc47344111)

[Topic 1: Analysing requirements 9](#_Toc47344112)

[Overview 10](#_Toc47344113)

[Understanding business operations 10](#_Toc47344114)

[The business domain 11](#_Toc47344115)

[Communicating with your client 11](#_Toc47344116)

[Videos: Communication skills 11](#_Toc47344117)

[Planning tasks 11](#_Toc47344118)

[Video: Getting started with Trello 12](#_Toc47344119)

[Business analysis 13](#_Toc47344120)

[System boundaries 13](#_Toc47344121)

[Video: Systems theory of organisations 13](#_Toc47344122)

[Videos: System context and boundaries 14](#_Toc47344123)

[Videos: Creating context diagrams 15](#_Toc47344124)

[Activity 1.1: Little Athletics scenario: Create a context diagram 16](#_Toc47344125)

[Requirements 16](#_Toc47344126)

[Videos: Developing effective requirements 17](#_Toc47344127)

[Data analysis 17](#_Toc47344128)

[Types of data analysis 18](#_Toc47344129)

[Data analysis techniques 18](#_Toc47344130)

[Methods of gathering information 19](#_Toc47344131)

[Documenting requirements 21](#_Toc47344132)

[Video: Writing functional requirements 21](#_Toc47344133)

[Video: Writing requirements documents 22](#_Toc47344134)

[Activity 1.2: Little Athletics scenario: Document the requirements 22](#_Toc47344135)

[Topic 2: Understanding database environments 24](#_Toc47344136)

[Overview 25](#_Toc47344137)

[Client/Server environments 25](#_Toc47344138)

[Videos: The internet and the world wide web 26](#_Toc47344139)

[Principles of open platforms 27](#_Toc47344140)

[Video: Open source versus closed source software 28](#_Toc47344141)

[Video: What is an API? 29](#_Toc47344142)

[Data sources 29](#_Toc47344143)

[Videos: Data sources 32](#_Toc47344144)

[Server security concepts 33](#_Toc47344145)

[Video: Security and data protection in a Google data centre 34](#_Toc47344146)

[Data mining 35](#_Toc47344147)

[Videos: Data mining 35](#_Toc47344148)

[Accessibility 36](#_Toc47344149)

[Video: Web accessibility 37](#_Toc47344150)

[Topic 3: Data modelling 39](#_Toc47344151)

[Overview 40](#_Toc47344152)

[What is data modelling? 41](#_Toc47344153)

[Videos: Understanding data 41](#_Toc47344154)

[Videos: Data modelling 41](#_Toc47344155)

[Data modelling terminology 42](#_Toc47344156)

[Relational database keys 46](#_Toc47344157)

[Video: DBMS keys 49](#_Toc47344158)

[Video: Clustered vs. non-clustered index structures 49](#_Toc47344159)

[Business rules 51](#_Toc47344160)

[Video: Business rules 52](#_Toc47344161)

[Naming database objects 53](#_Toc47344162)

[Database identifiers 53](#_Toc47344163)

[Naming conventions 53](#_Toc47344164)

[Data modelling techniques 55](#_Toc47344165)

[Data flow diagrams 55](#_Toc47344166)

[Entity Relationship Diagrams (ERDs) 56](#_Toc47344167)

[Video: Data Flow Diagrams 59](#_Toc47344168)

[Video: ERDs 60](#_Toc47344169)

[Activity 3.1: Create an ERD 60](#_Toc47344170)

[Normalisation 61](#_Toc47344171)

[Functional dependence and determination 61](#_Toc47344172)

[Normalisation rules 62](#_Toc47344173)

[Normalisation processes 64](#_Toc47344174)

[Video: Database normalisation—introduction 65](#_Toc47344175)

[Data types 67](#_Toc47344176)

[Video: Table columns and data types 70](#_Toc47344177)

[Data type hierarchies 71](#_Toc47344178)

[Structured data types 74](#_Toc47344179)

[Reference types 75](#_Toc47344180)

[User-defined functions 75](#_Toc47344181)

[Additional resources 76](#_Toc47344182)

[Video: Logical database design and ERDs 76](#_Toc47344183)

[Activity 3.2: Model a database 77](#_Toc47344184)

[Video: SQL Power Architect 82](#_Toc47344185)

[Video: Connecting bridge tables 88](#_Toc47344186)

[Video: Data dictionaries 89](#_Toc47344187)

[Courier company governance database—Data dictionary 91](#_Toc47344188)

[Activity 3.3: Little Athletics scenario: Create the ERD 95](#_Toc47344189)

[Validating the data model 96](#_Toc47344190)

[Topic 4: Creating a database 97](#_Toc47344191)

[Overview 98](#_Toc47344192)

[Query-related terminologies 100](#_Toc47344193)

[Video: Connecting to SQL server using SSMS 104](#_Toc47344194)

[Creating a database and tables 104](#_Toc47344195)

[Create database syntax 104](#_Toc47344196)

[Video: Creating, altering and dropping a database 105](#_Toc47344197)

[Video: Apply forward engineering in SQL Power Architect 105](#_Toc47344198)

[Activity 4.1: Create a database using DDL 106](#_Toc47344199)

[Create table syntax 108](#_Toc47344200)

[Video: Creating and working with tables 109](#_Toc47344201)

[Activity 4.2: Create tables using DDL 110](#_Toc47344202)

[Create the constraints 114](#_Toc47344203)

[Activity 4.3: Add constraints using DDL 115](#_Toc47344204)

[Testing and debugging the database 117](#_Toc47344205)

[Test data 118](#_Toc47344206)

[Types of testing 118](#_Toc47344207)

[Video: Test data generation in SQL 119](#_Toc47344208)

[Video: Creating large tables with random data 120](#_Toc47344209)

[Activity 4.4: Create a test plan 121](#_Toc47344210)

[Debugging and problem solving 121](#_Toc47344211)

[Additional resources 121](#_Toc47344212)

[Populating the tables with data 122](#_Toc47344213)

[INSERT statement 122](#_Toc47344214)

[Import data from a file 122](#_Toc47344215)

[Video: Importing data 122](#_Toc47344216)

[Import the results of a query 123](#_Toc47344217)

[Activity 4.5: Populate tables 123](#_Toc47344218)

[Activity 4.6: Little Athletics scenario: Create the database 125](#_Toc47344219)

[Topic 5: Using SQL 126](#_Toc47344220)

[Overview 127](#_Toc47344221)

[Conventions for formatting SQL 127](#_Toc47344222)

[Selecting and sorting values 128](#_Toc47344223)

[Displaying all columns 128](#_Toc47344224)

[Choosing the columns to be displayed 129](#_Toc47344225)

[Sorting results 130](#_Toc47344226)

[Selecting specific values 132](#_Toc47344227)

[Comparing values 132](#_Toc47344228)

[Video: CAST() and CONVERT() functions 132](#_Toc47344229)

[Filtering results 133](#_Toc47344230)

[Extracting a specific value by position 140](#_Toc47344231)

[Video: RANK, DENSE\_RANK and ROW\_NUMBER functions 142](#_Toc47344232)

[Selecting from multiple tables 143](#_Toc47344233)

[Qualifying column names 145](#_Toc47344234)

[Using aliases 146](#_Toc47344235)

[Using joins 151](#_Toc47344236)

[Expressions using calculations 161](#_Toc47344237)

[Operations on numeric values 161](#_Toc47344238)

[Video: Order of operations 164](#_Toc47344239)

[Operations on date and time values 165](#_Toc47344240)

[Operations on text values 169](#_Toc47344241)

[Using aggregate functions 172](#_Toc47344242)

[Video: GROUP BY 176](#_Toc47344243)

[Activity 5.1: Little Athletics scenario: Create and populate the database 177](#_Toc47344244)

[Activity 5.2: Single table queries 177](#_Toc47344245)

[Activity 5.3: Retrieving information from multiple tables 178](#_Toc47344246)

[Activity 5.4: Retrieving information from multiple tables 179](#_Toc47344247)

[Activity 5.5: Working with dates 180](#_Toc47344248)

[Activity 5.6: Alias tables and other types of joins 181](#_Toc47344249)

[Summary 183](#_Toc47344250)

[Appendix 1: Little Athletics scenario 184](#_Toc47344251)

[Appendix 2: Links 186](#_Toc47344252)

[Websites 186](#_Toc47344253)

[LinkedIn Learning videos 187](#_Toc47344254)

[Image attributions 189](#_Toc47344255)

# Icon legends

|  |  |
| --- | --- |
| Icon | Description |
|  | **Practice activity**  Learning activities help you to gain a clear understanding of the content in this resource. It is important for you to complete these activities, as they will enhance your learning. The activities will prepare you for assessments. |
|  | **Collaboration**  You will have opportunities to collaborate with others during your study. This could involve group activities such as mini projects or discussions that will enable you to explore and expand your understanding of the content. |
|  | **Self-check**  An activity that allows you to check your learning progress. The self-check activity gives you the opportunity to identify areas of learning where you could improve. If you identify these, you could review the relevant content or activities. |
|  | **Resources (required and suggested)**  Additional resources throughout this workbook such as chapters from textbooks, online articles, videos etc. These are supplementary resources, which will enhance your learning experience and may help you complete the unit. |
|  | **Assessment task**  At different stages throughout the workbook, after you have completed the readings and activities, you may be prompted to complete one or more of your assessment tasks. |
|  | **Video**  Videos will give you a deeper insight into the content covered in this workbook. If you are working from a printed version, you will need to look these up using the URL provided. |

# Getting started

## What will I learn by completing this workbook?

Most information systems rely on a database for the storage of data that is to be used by the system. The database is a critical component of the information system and as a developer, your knowledge and understanding of how databases are used in information systems are vital. Relational Database Management Systems (RDBMS) are the most commonly used systems to support the management of data used by information systems.

By completing this cluster you will learn about:

* analysing client and business requirements
* RDBMS and database environments
* designing, modelling and creating databases using different methods
* writing queries using structured query language (SQL).

Each topic includes opportunities to check your progress and understanding as well as activities that will help you to complete the formal assessments.

There are five topics to complete within this workbook. They are:

1. Analysing requirements
2. Understanding database environments
3. Data modelling
4. Creating a database
5. Using SQL.

To complete this workbook, you will require:

* a personal computer with access to the internet
* headphones for listening to online tutorial material and videos
* data modelling software such as SQL Power Architect, Lucid Chart or Vertabelo
* an RDBMS (a database server) installed on your computer, preferably SQL Server.
* a development tool such as SQL Server Management Studio for writing and executing SQL code on the database server.
* a portable drive for backing up your learning material and any activities you complete for this topic.

Topic 1: Analysing requirements

­

# Overview

Organisations such as hospitals, government departments, universities, banks, insurance companies and just about every other business that operates in today's society, rely on data to perform their activities.

RDBMS are used by most organisations to process the data they collect and turn it into information. They will use this information to make business or operational decisions to benefit the organisation.

It is important that the analyst understands how to define what information that the database is supposed to produce for the organisation.

In this topic you will learn about the following:

* Defining the Information requirements of an organisation to be addressed by a data model.
* Data analysis techniques used to develop an understanding of the operations of the organisation.
* Methods used to gather the information and data to be analysed.
* Defining the scope and boundaries of the database system.
* Developing a written requirement report describing the functional requirements of the database.

# Understanding business operations

RDBMS are used extensively throughout the modern world to assist organisations in executing their operations. RDBMS allow organisations to collect, store, process and analyse data. They then turn it into information, which the organisation can use to define the decisions required for successful business operations.

It is important therefore, that the database is designed to meet the organisation’s specific operations and more precisely the business rules of the particular organisation. In other words, the accuracy of an information system will be greatly determined by how closely its database reflects the operations of the organisation.

Data analysis techniques are used to develop an understanding of the business operations and business rules of an organisation in order to define the structure (or schema) of the database.

## The business domain

The business domain refers to the organisation’s core business, or their competitive boundaries. It will inform your requirements and acceptance criteria for the system. To understand the business domain, you need to find out things like who their customers are and what their product is. This information will lead you to an overview of the business domain.

# Communicating with your client

To determine a client’s needs and provide an appropriate solution, you’ll need to be able to communicate with them, both verbally and in writing. This includes using effective listening and questioning techniques, presenting relevant information, obtaining their feedback and writing reports.

 Watch

## Videos: Communication skills

The following video collection and courses from LinkedIn Learning include information about communication strategies that you can use when communicating with clients and co-workers. You can watch the whole course, or just the parts that you need:

* [Effective listening and questioning techniques](https://www.linkedin.com/learning-login/share?forceAccount=true&redirect=https%3A%2F%2Fwww.linkedin.com%2Flearning%2Fcollections%2F6585328392747446272%3Ftrk%3Dshare_collection_url&account=57684225&auth=true) (LinkedIn Learning 07:32 mins)
* [Communication Foundations](https://www.linkedin.com/learning/communication-foundations-2/welcome?u=57684225&auth=true) (LinkedIn Learning 01:24 hrs)

# Planning tasks

At work, you have a responsibility to deliver the work expected of you, to the required standard and within the required timeframe; this is what your employers employ you to do. In some circumstances, an employee has no control over what they do and when they do it – employees working on a production line in a factory, for example, have their work schedule dictated to them by the pace of the production line they are working on.

In a different context, such as ICT support, employees may have varying degrees of flexibility in terms of how they decide which tasks to perform and when. In some circumstances, it won’t matter the precise order in which tasks are completed, so long as they’re completed to the required standard and deadline. This is one of the factors that can be motivating in a job role – the freedom and ability to decide when to complete individual tasks.

In other cases, the task order *will* matter and it’s important to know how to sequence those tasks appropriately. This is where it’s important to be able to access the organisation’s policies and procedures so that you have this understanding and knowledge.

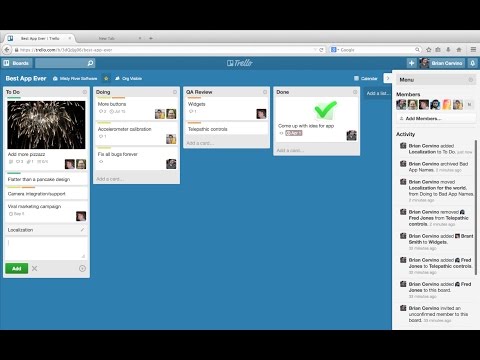
Having established an understanding of what’s to be done in the work role, you should assess and prioritise your workload. This will help you plan your tasks to ensure that the most important or urgent priorities are tackled first and that time-sensitive activities are completed within the required timeframe. A useful place to start is with a to-do list. This is simply a list of tasks that are to be completed within a set time – this may be a day, week or month, depending on the context of your role.

 Watch

## Video: Getting started with Trello

Trello is a free online planning tool that allows you to create to do lists, set up reminders and create notes. Watch the following video to learn how to use this tool:

[Getting Started With Trello](https://www.youtube.com/watch?v=xky48zyL9iA) (YouTube 04:55 mins)

[](https://www.youtube.com/embed/xky48zyL9iA?feature=oembed)

<https://www.youtube.com/watch?v=xky48zyL9iA>

# Business analysis

Part of determining the needs of the organisation includes investigating, analysing and documenting information about the business and its issues. This generally includes determining the:

* system boundaries
* scope of the problem
* objectives
* outcomes or deliverables.

## System boundaries

The purpose of setting system boundaries is to limit the area of investigation. Boundaries need to be set otherwise the scope of the project can grow. Only those systems that are part of the problem or impact the solution need to be investigated. Some examples of questions to ask to identify systems for investigation are as follows:

* Which people or systems does the system interact with?
* What real-world objects and people does the system capture information about?
* What business processes does the system support?

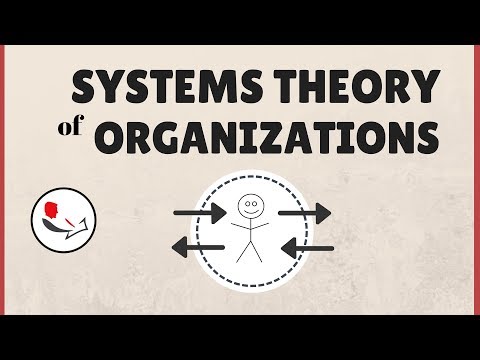
In the earlier stages of a project, you may create a context diagram to show the system boundaries. During the information gathering stage, you’ll expand on this diagram with more detailed information about the other systems that your proposed system will interact with.

 Watch

## Video: Systems theory of organisations

Watch this video to get an introduction to systems theory of organisations

[Systems theory of organizations](https://www.youtube.com/watch?v=1L1c-EKOY-w&feature=youtu.be) (YouTube 10:53 mins).

[](https://www.youtube.com/watch?v=1L1c-EKOY-w)

https://www.youtube.com/watch?v=1L1c-EKOY-w&feature=youtu.be

## Videos: System context and boundaries

Watch the following two videos to learn more about system context and system boundaries.

[Systems Boundary & Environment](https://www.youtube.com/watch?v=8FgsrzfIt0A&feature=youtu.be) (YouTube 06:14 mins).

[](https://www.youtube.com/watch?v=8FgsrzfIt0A)

<https://www.youtube.com/watch?v=8FgsrzfIt0A&feature=youtu.be>

[Business Analysis Training: What is System Context and System Boundary in Requirements Engineering](https://www.youtube.com/watch?v=ejfAB0eUfoc&feature=youtu.be) (YouTube 05:51).

[](https://www.youtube.com/watch?v=ejfAB0eUfoc)

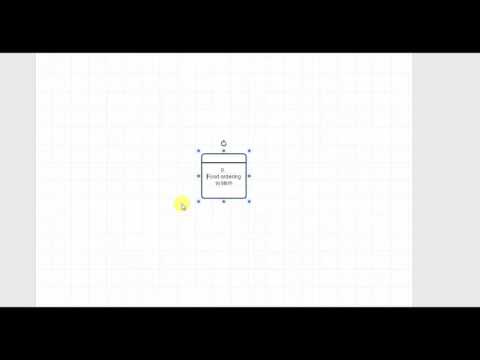
https://www.youtube.com/watch?v=ejfAB0eUfoc&feature=youtu.be

 Watch

## Videos: Creating context diagrams

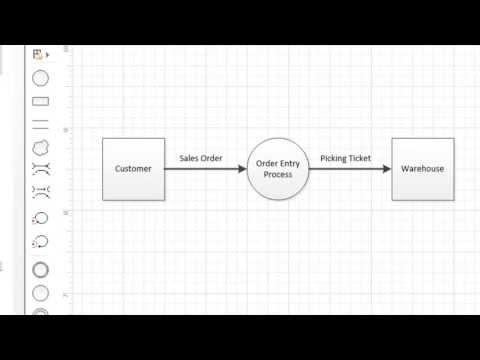
Watch the following two videos to learn how to create context diagrams.

[Drawing the context diagram](https://www.youtube.com/watch?v=yOBMhvKRM2E&feature=youtu.be) (YouTube 10:51 mins).

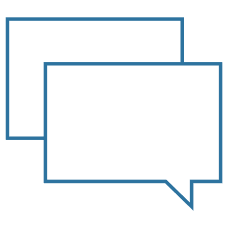
[](https://www.youtube.com/watch?v=yOBMhvKRM2E)

https://www.youtube.com/watch?v=yOBMhvKRM2E&feature=youtu.be

[Systems Documentation - Part III Creating a Context Diagram](https://www.youtube.com/watch?v=IklEoggJda8&feature=youtu.be) (YouTube 08:20 mins).

[](https://www.youtube.com/watch?v=IklEoggJda8)

https://www.youtube.com/watch?v=IklEoggJda8&feature=youtu.be

 Collaboration

## Activity 1.1: Little Athletics scenario: Create a context diagram

Refer to the Little Athletics scenario in [Appendix 1](#_Appendix_1:_Little) for this activity.

Do this activity with one or two other people.

Create a context diagram to represent the processes and some of the functions that the new system will carry. Discuss and compare your diagram with the class.

# Requirements

In the course of analysing the business, you'll also be gathering requirements from stakeholders, so you need to understand what requirements are.

Client needs are the minimum requirements that the client must have to run their business.

Client expectations, on the other hand, are the difference between what the client wants (and usually thinks they’re getting) and what the client needs.

There may also be added-value requirements, which may be things like financial benefits, increased productivity and improved customer satisfaction.

**Business requirements** are high level statements that the client understands, for example: ‘enable travelling sales people to access the CRM data’.

**Functional requirements** are detailed statements that specify exactly what needs to be delivered and are used by business analysts, developers, project managers, testers etc. They describe what the problem is without getting into how a solution would work. They specify specific behaviour or functions, for example, ‘The system will be able to register new users using the following fields: First Name (20 characters), Last name (20 characters), Phone (10 characters), Office (list)’. Functional requirements are often specified in the form of a use case.

**Non-functional requirements** are detailed statements specifying a constraint on the system’s behaviour. They specify criteria that judge the operation of a system such as performance, scalability, capacity, availability, reliability, recoverability, maintainability, security etc. For example, ‘Report generation times must not exceed three seconds’.

**System requirements** are statements specifying what technology (hardware/ middleware/ software) is needed to achieve the non-functional and functional requirements. For example, ‘the CRM application will be coded in PHP/ JavaScript/ HTML5 and will run on a LAMP stack (Linux Apache MySQL PHP) hosted within a small Amazon EC2 instance in the Sydney datacentre’.

 Watch

## Videos: Developing effective requirements

The following course from LinkedIn Learning discusses how to develop effective requirements for software design. You cant watch the whole course or just the parts you need:

* [Software Design: Developing Effective Requirements](https://www.linkedin.com/learning/software-design-developing-effective-requirements?u=57684225&auth=true) (LinkedIn Learning 1:45 hrs).

# Data analysis

Data analysis is the collection and organisation of data to discover information from the data. The information is used to develop a model of a database that will serve to implement and enforce the business rules of an organisation and support the execution of their business.

Data analysis provides answers to questions, the solutions to problems and the detection of important information.

Some reasons for data analysis are:

* inferring unknown information
* developing models to aid in forecasting and prediction of future behaviour
* identifying patterns within the data and hence any anomalies within the patterns
* verifying or disproving assumptions or hypotheses
* detecting faults within processes.

However, it must be stressed that the primary reasons for data analysis in the realms of database design are to establish the nature of the data and how the data is to be used.

## Types of data analysis

There are many different data analysis techniques. The decision to use one technique over another will be determined by the need for the analysis, which may be:

* either confirmatory or exploratory
* either qualitative or quantitative
* either descriptive or inferential
* either prescriptive or predictive.

Data analysis, in most instances, is an iterative process and not necessarily sequential. The quality of the data and the method used are the two factors that will contribute the most to the success of the data analysis.

## Data analysis techniques

Data analysis techniques can be either quantitative or qualitative.

Quantitative analysis:

* provides data that can be counted or expressed numerically
* can be used with statistical analysis to determine things like averages, medians, standard deviations etc., or can be used in mathematical computations
* can be represented visually in tables and graphs
* is either:
  + discrete, represented numerically as a whole number (integer) (e.g. most families have two cars)
  + continuous, represented numerically as a decimal value (e.g. a company’s profit margin is set at 22.5%)
* is limited in:
  + the types of predictions it can be used to make
  + solving a problem (however it can help identify a problem)
  + prioritising the development of the solutions (however it can help rank problems)
  + supporting decision making without including qualitative data analysis.

Qualitative analysis:

* uses descriptive words and language rather than numbers to explain, describe and characterise the subject being investigated
* provides more insightful meaning to quantitative/numerical findings
* can help define known problems
* can provide ideas for further investigation and possible intervention strategies to solve problems
* proves more useful when combined with quantitative analysis to provide a deeper understanding of the data.

## Methods of gathering information

There are several methods that can be used to gather information and data, which can be analysed qualitatively to aid in the design of a relational database.

Table Methods of gathering information

|  |  |
| --- | --- |
| Method | Overview |
| **Focus groups** | Focus groups are an effective method for qualitative data collection. They involve bringing together a group of people with specific knowledge about the information being sought.  In an interactive group setting, the participants are asked specific questions about a particular topic by an experienced group moderator. The participants are permitted to interact with other group members in an effort to identify, clarify and define their attitudes and opinions about the topic. |
| **Interviews** | Interviews are conducted to elicit specific information. They take a question-and-answer format, which can use either open or closed questions. Closed questions will have a specific answer (such as yes or no) while open questions will allow the respondent to elaborate on their answer.  Interviews are often the first method of information gathering to be used on a new project. However, they are resource-intensive and expensive to implement. A one-on-one interview can result in a more meaningful engagement than when working with a larger group. It is most useful to interview stakeholders that have the most expertise and the greatest influence on the processes that are being studied. |
| **Observation** | Observation involves studying users in their working environment to identify process flow, inefficient steps, pain points and improvement opportunities. Observation provides real-time, detailed insights into the information being gathered and avoids the analyst being told misinformation.  Observation is a good way to confirm the accuracy of information gathered by other methods. It can be passive, in that the observer does not interact with the person(s) being observed, or active, in that the observer can interrupt and ask questions of the process during the observation. Observation is time-consuming and should be targeted to clarify ambiguities. |
| **Document analysis** | This method is commonly used to initiate the gathering of requirements. It involves gathering information from existing documents and other related sources of relevant information. It could include statistics-based records, invoices, sales figures, customer lists, product lists, performance reports, minutes from meetings, etc.  Document analysis involves minimal interaction with people; it is often necessary to have assumptions confirmed by an expert. Combined with other information-gathering methods, it can produce an accurate view of the desired information because much of the information gathering will already have been included in the existing organisational documentation. |

# Documenting requirements

Requirements will need to be documented in a Requirements report and approved before work can begin, to ensure you have included everything needed. This report should be approved by your client, however a client may mean an external customer, or it could include your supervisor, depending on the type of work you are doing.

 Watch

## Video: Writing functional requirements

Watch this video to learn more about writing functional requirements.

[Writing Requirements: Write Functional Requirements - Traditional, Agile, Outsourcing](https://www.youtube.com/watch?v=T1GKQtG5b2A&feature=youtu.be) (YouTube 11:40 mins).

[](https://www.youtube.com/watch?v=T1GKQtG5b2A)

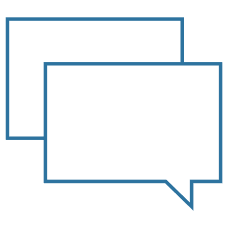
https://www.youtube.com/watch?v=T1GKQtG5b2A&feature=youtu.be

## Video: Writing requirements documents

Watch the following videos from LinkedIn Learning to learn how to structure a requirements document and understand what technical writing skills you need.

[Structuring formal requirements documents](https://www.linkedin.com/learning/project-management-foundations-requirements-2015/structuring-formal-requirements-documents?u=57684225&auth=true) (LinkedIn Learning 04:56 mins)

[Reviewing essential technical writing skills](https://www.linkedin.com/learning/project-management-foundations-requirements-2015/reviewing-essential-technical-writing-skills?u=57684225&auth=true) (LinkedIn Learning 03:33 mins)

 Collaboration

## Activity 1.2: Little Athletics scenario: Document the requirements

Refer to the Little Athletics scenario in [Appendix 1](#_Appendix_1:_Little) for this activity.

Do this activity with one or two other people.

* Write at least three requirements for the system.
* Check your requirements by swapping with another group:
  + Are they clear and unambiguous?
  + Do they include functional and non-functional?
  + Are they within the scope?
  + Do they conflict with other requirements?
* Rewrite the requirements if needed.
* Verify your requirements with another group to ensure that they understand them.

Topic 2: Understanding database environments

­

# Overview

In this topic you will develop an understanding of the environments in which RDBMS operate.

You will learn about the following:

* The relationship between a database server, a web server, the internet and the clients that consume the information provided by the database server.
* How the internet works and the basic principles of open platforms.
* Different sources of data that can be used by organisations to conduct their business.
* The theoretical concepts that underpin the security of data.
* The importance of validation and best practices for communicating with and providing access to audiences with special needs.

We are on the brink of a revolution in the use of data and never before has humanity accumulated the volume of data that we are currently consuming. You will be introduced to data mining, which has risen to become the new frontier for opportunity in today’s world.

When you’ve completed this topic you will have developed an understanding of RDBMS environments and the factors that contribute to the operation of such systems.

# Client/Server environments

The client/server model is an application structure that separates the work to be done between the provider of a service, the servers and the consumers or requestors of the service (the clients). The communication between the two, server and client, will usually be conducted over a network (the internet) on different hardware.

Both the server and the client are computers running either server software or client software, respectively. The internet is dependent on the client-server model and relies on it for the everyday activities of web servers and clients in what we commonly refer to as the world wide web.

In the web server/client relationship, the web server will often host a web server software application such as Microsoft’s Internet Information Server (IIS) or Apache Web Server. The web client will, in most cases, host browser software such as Microsoft Edge, Internet Explorer, Google Chrome, Mozilla Firefox or Safari to name but a few available browsers. The two, server and client, could literally be oceans apart but connected through the internet.

When someone enters an internet address (URL) into a web browser, they’re requesting information from a web server at that address. The client browser sends the request (which includes the client’s return address) to the internet, where its destination is assessed and the request forwarded through the internetworked servers of the internet until it reaches the destination web server.

The web server then generates the response (usually in the form of HTML, CSS, etc.) and sends the response back through the internet to the client’s return address, where the client browser interprets the response and ‘marks-up’ the HTML, CSS, JavaScript, etc. to display the web page to the user.

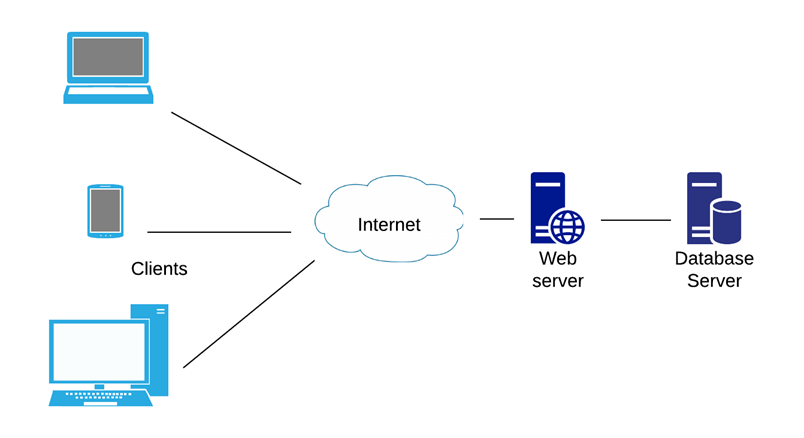


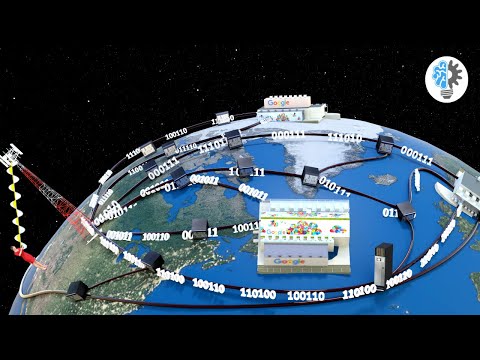
Figure The client/server environment of the internet

 Watch

## Videos: The internet and the world wide web

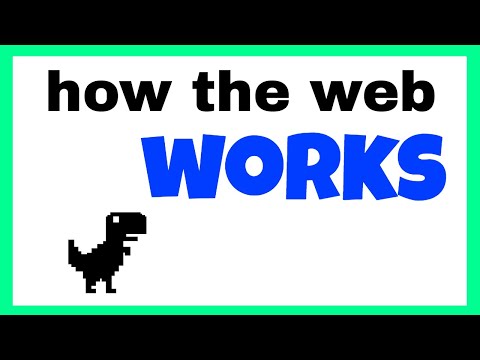
The YouTube videos you watch originate on a data server that may be thousands of kilometres away. Watch these videos to learn how the client/server relationship works on the internet and how data can travel thousands of kilometres to reach its destination so accurately and so quickly.

[How does the internet work?](https://youtu.be/x3c1ih2NJEg) (YouTube 08:58 mins).

[](https://www.youtube.com/watch?v=x3c1ih2NJEg)

https://youtu.be/x3c1ih2NJEg

[How does the world wide web work?](https://youtu.be/XjpAnPM5bo0) (YouTube 07:03 mins).

[](https://www.youtube.com/watch?v=XjpAnPM5bo0)

https://youtu.be/XjpAnPM5bo0

# Principles of open platforms

A platform is something that can be worked on and built onto. Open platforms therefore serve as a staging point for further development by others.

Open platforms in computing describe software that is developed on open standards, which are fully documented and published for anyone to access and use. The open standards allow for the software to be used in ways that may not have been foreseen (or intended) by the original programmers.

This openness is most often achieved by:

* the publishing of Open Platform Application Programming Interfaces (API), allowing third parties to develop additional functionality that can be integrated into the original software
* providing open scope, allowing the software to be used in ways that were never planned
* providing open source, allowing the source code to be studied, modified and distributed; there are varying degrees to which a platform will be open source
* allowing for open usage, including open development, open provision and open operation
* allowing for open adaptability, which means the software platform’s existing functionality can be modified by modifying the source code itself.

Some of the more popular open platform browsers include Firefox, Chromium, Waterfox, Basilisk, Pale Moon, Brave Browser, and Dooble. Chromium is the platform on which Microsoft Edge was built.

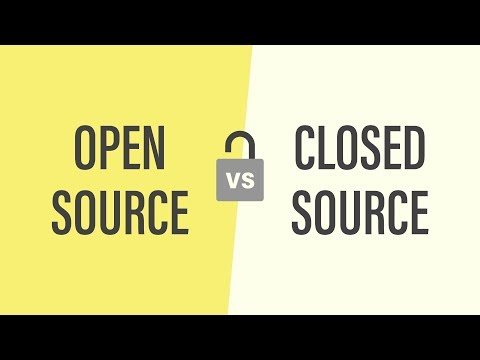
Some of the more popular open platform databases include PostgreSQL, MySQL, MongoDB, MariaDB and SQLite.

 Watch

## Video: Open source versus closed source software

In this video, you’ll learn more about the differences between open-source and closed-source software.

[Open source versus closed source software](https://youtu.be/2q91vTvc7YE) (YouTube 02:28 mins).

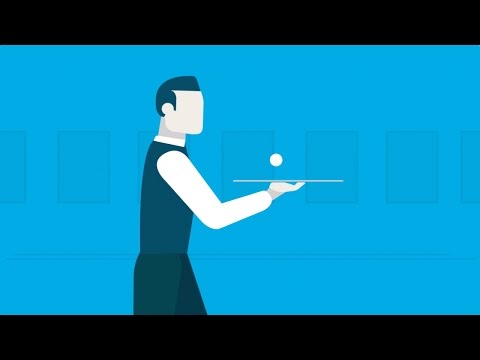
[](https://www.youtube.com/watch?v=2q91vTvc7YE)

https://youtu.be/2q91vTvc7YE

## Video: What is an API?

Watch this video to learn more about APIs.

[What is an API?](https://youtu.be/s7wmiS2mSXY) (YouTube 03:24 mins).

[](https://www.youtube.com/watch?v=s7wmiS2mSXY)

https://youtu.be/s7wmiS2mSXY

# Data sources

There are several possible sources of data for an information technology system. These include the following.

Table

|  |  |
| --- | --- |
| Data source | Overview |
| **Flat files** | Typically a flat file data source will comprise one or more files of data stored as character data (or plain text). The individual attributes of data are usually separated from each other using a comma (comma-separated values or CSV) or a tab character. Flat files are difficult to work with because they can result in huge amounts of redundant data and consequently data anomalies can occur.  Also, systems that rely on flat files as their data source lack the functionality and data management support provided by RDBMS and require the programmer to have a more intimate knowledge of the physical locations of the data required for processing by the system. |
| **Relational databases** | RDBMS provide functionality that simplifies much of the hard work required to define, collect, store, securely access and manipulate the data required by information systems. A relational database is a shared and integrated structure that stores both user data and metadata. The metadata is used to integrate and manage the user data, and describe the characteristics of the user data, and how it is related. Relational databases are the most common data source used by information systems. |
| **NoSQL databases** | NoSQL databases derive their name from ‘not only’ SQL, which means that in certain circumstances SQL can also be used with a NoSQL database. They are non-relational, which means that they don’t use tables in the same way that relational databases do. NoSQL databases are suited to big data applications and real-time web apps. There are several types of NoSQL databases, which include document databases, column databases and Key Value Store. NoSQL databases do not have a pre-defined schema (or data model) like relational databases, so they are able to handle unstructured data. |
| **Cloud** | The term ‘cloud’ or ‘cloud computing’ has evolved out of the common use of the image of a cloud representing the internet in diagrams that depict computing functions interacting with the internet. Cloud data sources are usually databases that run on a cloud platform and access to the database is provided as a service to the data-consuming applications. The database platforms offered include RDBMS, such as Microsoft Azure SQL Databases, and NoSQL platforms such as MongoDb. |
| **XML files** | XML is the acronym for eXtensible Markup Language. XML is a prolific data source in use today because it provides the ability to store both the structure of the data (i.e. the schema or metadata) and the data itself in human-readable plain text. This facilitates the easy exchange of data between organisations that want to share their data but which operate disparate platforms. Unlike HTML, which has a limited set of pre-defined tags, XML is ‘extensible’ because it allows users to define their own tags and tag attributes, so allowing them to define the structure of the data. |
| **Hierarchical** | Hierarchical data sources are most often implemented as hierarchical databases and have an upside-down tree structure in which every table has a single parent. Only one-to-many relationships are supported in hierarchical databases making them somewhat limited in their capabilities. Hierarchical databases were developed in the early days of data storage when information processing systems moved from paper-based systems to computerised systems. |
| **Network** | Network databases were developed to allow more complex relationships than their hierarchical predecessor provided. The network model was developed to allow each record to have more than one parent, however it was still limited to implementing only one-to-many relationships. Network databases are not used very much today because of inherent limitations, such as an inability to run ad-hoc queries on the data and difficulty in restructuring the data. They have consequently been replaced by relational databases. |

 Watch

## Videos: Data sources

Watch the following videos to learn more about each of the data sources discussed above.

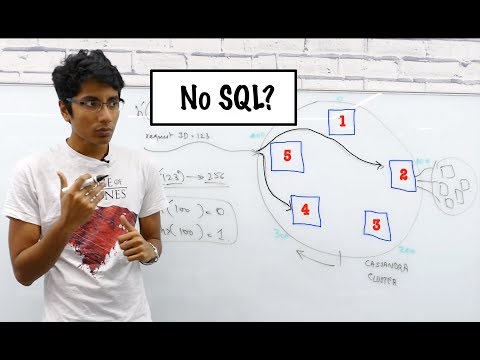
[Understanding flat file databases](https://www.linkedin.com/learning/database-foundations-core-concepts/understanding-flat-file-databases?u=57684225&auth=true) (LinkedIn Learning 02:19 mins).

[An introduction to NoSQL databases](https://youtu.be/uD3p_rZPBUQ) (YouTube 15:40).

[](https://www.youtube.com/watch?v=uD3p_rZPBUQ)

https://youtu.be/uD3p\_rZPBUQ

[What is NoSQL and how are NoSQL databases different?](https://youtu.be/xQnIN9bW0og) (YouTube 27:00 mins).

[](https://www.youtube.com/watch?v=xQnIN9bW0og)

https://youtu.be/xQnIN9bW0og

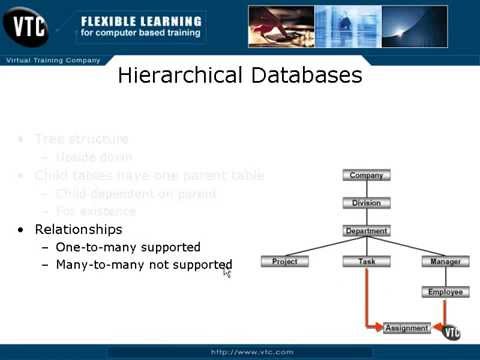
[What is cloud computing?](https://www.linkedin.com/learning/introduction-to-cloud-computing-for-it-pros/what-is-cloud-computing?u=57684225&auth=true) (LinkedIn Learning 03:24 mins).

[Overview of storage as a service](https://www.linkedin.com/learning/introduction-to-cloud-computing-for-it-pros/overview-of-storage-as-a-service?u=57684225&auth=true) (LinkedIn Learning 02:58 mins).

[Introduction to XML](https://www.linkedin.com/learning/microsoft-sql-server-2016-essential-training/introduction-to-xml?u=57684225&auth=true) (LinkedIn Learning 01:29 mins).

[Understanding hierarchical databases](https://www.linkedin.com/learning/database-foundations-core-concepts/understanding-hierarchical-databases?u=57684225&auth=true) (LinkedIn Learning 01:30 mins).

[Hierarchical databases](https://youtu.be/Rk5S3VWhuZw) (YouTube 01:33 mins).

[](https://www.youtube.com/watch?v=Rk5S3VWhuZw)

https://youtu.be/Rk5S3VWhuZw

# Server security concepts

Database server security can be defined as those actions and procedures that are undertaken to safeguard the availability, integrity and confidentiality of the most important asset of an information system, data. Security of data requires an all-of-organisation approach, which is comprehensive and secures all the processes and systems surrounding the data. Both digital and physical systems must be secured, including the hardware, software applications, network and network devices, both internal and external users of the data, company procedures and the data itself.

Availability is an important goal of security and requires that the data should be available to authorised people for authorised purposes. Availability depends on protecting the system against service disruption from any possible cause.

The integrity of the data in an information system requires that the data is free of errors or anomalies. Much of the data’s integrity will be dependent, initially, on how well the database schema has been modelled and then on the integral functionality of the RDBMS that hosts the data. However, the ongoing adherence to the organisation’s security policies by users must be vigilantly maintained to ensure the continued integrity of the data.

It’s not reasonable, for example, that an employee accesses the data from within the organisation using a USB flash drive that may have previously been corrupted with malware. Or for an employee to access systems from outside of the organisation using an unsecured or unencrypted network connection. The integrity of the data must be secured against all possible avenues of corruption.

Confidentiality requires the application of authentication and authorisation policies and procedures to ensure that the people accessing the data are authenticated, i.e. they are who they say they are, and that they’re authorised to access the data for authorised purposes.

Confidentiality aims to prevent the disclosure of data that would violate the privacy rights of individuals and of the organisation itself. These privacy rights include safeguarding against the disclosure of personal information as well as organisational business intelligence and intellectual property rights.

Organisations are obligated by law to meet statutory requirements regarding the confidentiality and safeguarding of personal as well as organisational data.

 Watch

## Video: Security and data protection in a Google data centre

Watch this video tour of a Google data centre to learn about the security and data protection that is in place at their data centres.

[Security and data protection in a Google data centre](https://youtu.be/cLory3qLoY8) (YouTube 07:00 mins).

[](https://www.youtube.com/watch?v=cLory3qLoY8)

https://youtu.be/cLory3qLoY8

## Data mining

With the explosive growth of the internet and its related technologies there has also been an unprecedented growth in the collection of data. Every time you use the world wide web, or tap-on to public transport, or pay for your groceries with your credit card, or make a phone call, or interact with a digital information system in any way, data about your interaction with that system is being stored on a server somewhere for you and everyone else in the world.

Add to that the accumulation of industrial data gathered by sensors in cars, roads and public transport systems, aircraft, manufacturing machines and robots, as well as at cash registers in retail outlets, and you begin to appreciate the scale of this data-gathering revolution. This is an unprecedented accumulation of masses of data.

Data mining is the term given to the analysis of this data in the hope that previously undiscovered and unknown information can be unearthed. This new and previously unknown information can then be used in the decision-making processes of the organisations that hold the data.

Traditionally, data has been used in a reactive manner. That is, a problem or opportunity is first defined and then the data is processed and analysed in order to find a solution to the problem or exploit the opportunity. Data mining, on the other hand, takes a more proactive approach in that data mining tools search masses of data (data warehouses) looking for anomalies, inconsistencies, patterns and possible relationships that were previously unknown.

Once discovered, problems or opportunities that have been hidden in the data become apparent and can be used in the policy and decision-making processes of the organisation. Data mining is used extensively in a whole-of-government approach, where government departments share their data to uncover welfare fraud or to analyse the efficiency of government services to the community. The insurance industry also uses data mining extensively to detect fraudulent claims and to accurately assess insurance risks.

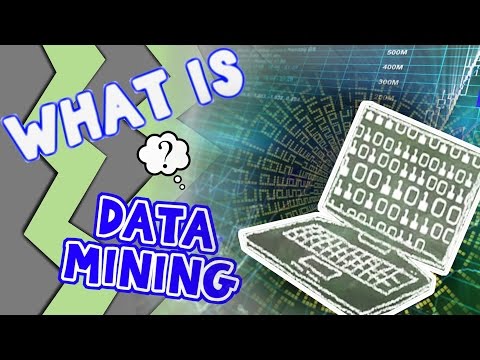
 Watch

## Videos: Data mining

Watch the following videos from YouTube and LinkedIn Learning to learn more about data mining and how it is used.

[Data mining](https://www.linkedin.com/learning/big-data-in-the-age-of-ai/data-mining?u=57684225&auth=true) (LinkedIn Learning 04:39 mins).

[What is Data Mining?](https://youtu.be/R-sGvh6tI04) (YouTube 03:22 mins).

[](https://www.youtube.com/watch?v=R-sGvh6tI04)

https://youtu.be/R-sGvh6tI04

# Accessibility

Accessibility is an important consideration when designing every aspect of communication of a computing system. This can include communications that happen during the analysis phase to identify required information from all stakeholders, including those with special needs, by designing feedback forms and questionnaires or conducting focus groups in an accessible way. Accessibility must also be considered at the user-interface end of the computing spectrum, where it is important to identify the audiences with special needs.

There are also factors of inclusion that affect accessibility and should take into account:

* accessibility for people with disabilities
* access to and quality of hardware, software, and internet connectivity
* computer literacy and skills
* economic situation
* education
* geographic location
* culture
* age, including older and younger people
* language.

Best practice accessibility strategies include the following:

* Provide perceivable information and user interface:
  + Provide text alternatives
  + Provide captions and other alternatives for multimedia
  + Ensure that content can be presented in different ways
  + Ensure that content is easier to see and hear.
* Provide an operable user interface and navigation:
  + Ensure that functionality is available from a keyboard
  + Ensure that users have enough time to read and use the content
  + Ensure that content does not cause seizures and physical reactions
  + Provide mechanisms for users to easily navigate, find content, and determine where they are
  + Ensure the users can use different input modalities beyond the keyboard.
* Understandable information and user interface:
  + Provide text that is readable and understandable
  + Ensure that content appears and operates in predictable ways
  + Ensure that users are helped to avoid and correct mistakes
* Robust content and reliable interpretation:
  + Ensure that content is compatible with current and future user tools.

Read more about web accessibility on the Web Accessibility Initiative’s (WAI) page [Introduction to Web Accessibility.](https://www.w3.org/WAI/fundamentals/accessibility-intro/)

 Watch

## Video: Web accessibility

Watch this video to get an introduction to web accessibility and web W3C web standards.

[Introduction to Web Accessibility and W3C Standards](https://youtu.be/20SHvU2PKsM) (YouTube 04:07 mins).

[](https://www.youtube.com/watch?v=20SHvU2PKsM)

https://youtu.be/20SHvU2PKsM

Topic 3: Data modelling

­

# Overview

This topic will develop your understanding and skills in data modelling. Data modelling is critical to your development as a programmer or analyst because you may often be required to develop the data model for the system being built. This is especially true if you work in a small to medium-sized enterprise.

You will learn about the following:

* The terminology used in data modelling, how to interpret the business rules and information requirements of an organisation so that you can build a data model.
* The different types of model that are most commonly used, such as the Entity Relationship Diagram (ERD), and the various symbols used in ERDs and their meaning.
* An understanding of how effective table structures can be achieved by applying the rules, principles and processes of normalisation.
* The sequence of steps you should take to develop your model, what entities and attributes are and how to analyse the different types of relationships between the entities in your model.
* The functions and features of different data types that are used in databases and how to develop a data dictionary for your data model.
* You will use modelling software to develop and implement your data model and learn about the different types of keys that are used in relational databases.
* The use of timestamps, type hierarchies and structured data types, and the purpose of user-defined functions.

In developing your model you will apply appropriate naming conventions and understand the importance and use of database identifiers and how they impact the usability of the database.

It is important that you carefully read and understand the material in this topic and that you use modelling software (such as SQL Power Architect) to develop the model for the Courier Company Governance System that is described in this topic.

Complete each of the practice activities in this topic paying particular attention to the video tutorials and the completion of the data modelling practice activity for the Little Athletics organisation.

# What is data modelling?

Data modelling is the practice used to transition from real-world processes and objects to their implementation as a database on a computer system. Databases must be designed to meet the very specific operations of an organisation, however the database designers, application programmers and end-users may all have a very different understanding of the data. This can result in designs that don’t reflect the actual operations of an organisation and can fail to meet the information requirements of the users as well as fail to safeguard the integrity requirements of the data.

Database designers, therefore, strive to obtain a clear and precise description and understanding of the data and its use by the organisation. Data modelling condenses the complexities of the organisation’s processes, business rules and data down to a level of abstraction that: identifies the entities; and models the relationships between those entities. The data model is a logical model of the database, since it depicts the things for which data is being stored and how those things are related to each other. The data model can be used to confirm with the users that the database will meet their needs and to communicate to the database developers the logical structure of the database that is to be implemented.

 Watch

## Videos: Understanding data

Watch the videos in the following chapter from the LinkedIn Learning course Learning Data Analytics to learn how to identify data and the different types of data:

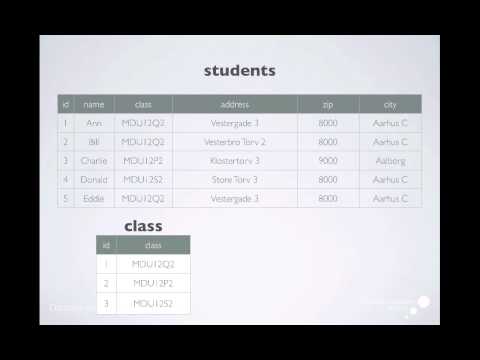
* [2. Fundamentals of Data Understanding](https://www.linkedin.com/learning/learning-data-analytics-2/learning-to-identify-data?u=57684225&auth=true) (LinkedIn Learning 14:19 mins).

## Videos: Data modelling

Watch the following videos to learn more about data modelling. The LinkedIn Learning video gives a short overview, while the YouTube video goes into more depth about normalisation and creating an entity relationship diagram.

* [Introduction to data modelling](https://www.linkedin.com/learning/filemaker-relational-database-design/introduction-to-data-modeling?u=57684225&auth=true) (LinkedIn Learning 02:12 mins).

[Data modelling – an introduction](https://youtu.be/tR_rOJPiEXc) (YouTube 55:21 mins).

[](https://www.youtube.com/watch?v=tR_rOJPiEXc)

https://youtu.be/tR\_rOJPiEXc

## Data modelling terminology

Table Terminology explanations

|  |  |
| --- | --- |
| Terminology | Explanation |
| **Entity** | During analysis, an entity is something (a person, place, thing, event, or transaction, etc.) for which you want to store data in the database. When the database is implemented (created) the entities become tables. Entities can be either concrete or abstracted. For example, in a banking system, a customer (concrete) is something you want to store data about. A transaction, like a funds transfer (abstracted), is also something you would want to store data about. The term entity is used in data modelling because it encompasses both the concrete and the abstracted things that you might want to store data about. As a rule of thumb. you can define something like an entity for your data model if there is more than one of it and it has attributes that you want to store in your database. |
| **Attribute** | During the analysis stage, an attribute is a property or characteristic of an entity. When the database is implemented (created) the attributes become columns in a table. For example, two attributes of a customer would be their first name and last name. Two attributes of a funds transfer would be the transfer date and the transfer amount. Non-prime attributes are attributes that are not part of the primary key of a table. |
| **Relationship** | A relationship describes how entities are connected with each other. For example, a customer could conduct fund transfers, so a relationship exists between customer and fund transfer and could be described as follows: a customer can make many fund transfers and a fund transfer is for only one customer. In data modelling this type of relationship is defined as one-to-many (1:M). In the relational data model there are three types of relationship: one-to-one (1:1), one-to-many (1:M) and many-to-many (M:N) (M:N is the standard notation, however, (M:M) is also acceptable and quite commonly used). |
| **One-to-one relationship** | New employees must provide the Human Resources (HR) department with details of just one bank account into which their weekly pay is deposited and each account must belong to only one employee. So, a bank account belongs to one employee and an employee’s pay goes into only one bank account. So, the relationship ‘EMPLOYEE’s pay goes into BANK ACCOUNT’ is considered 1:1. |
| **One-to-many relationship** | New employees are assigned a job role (or job title) by the HR department. Each employee is required to perform only one job role, however each job role may have many employees that are assigned that same job role. So, the relationship ‘EMPLOYEE is assigned a JOB ROLE’ is considered 1:M. |
| **Many-to-many relationship** | An Employee can be assigned to multiple projects and each project could have several employees assigned to it. So, the relationship ‘EMPLOYEE is assigned to PROJECT’ is considered M:N. |
| **Connectivity** | When considering the type of relationship that exists between entities, i.e. 1:1, 1:M, or M:N, you are interested in how many of one entity is related to the other entity and vice versa. This is referred to as the connectivity and expressed as one of the three types of relationship: 1:1, 1:M, or M:N. |
| **Cardinality** | This is an indication of the minimum and maximum number of instances of one entity related to one instance of the other entity. It can be expressed as one of the following:   * (0,1)—min of 0 and max of 1 * (1,1)—min of 1 and max of 1 * (0,M)—min of 0 and max of M * (1,M)—min of 1 and max of M * (0,6)—min of 0 and max of 6 (or any other value defined by the business rules) * (1,6)—min of 1 and max of 6 (or any other value defined by the business rules). |
| **Optionality** | Optionality is closely associated to cardinality and describes whether an entity’s relationship participation is mandatory or optional. Optionality is usually indicated in both the Chen and the Crow’s Foot model by a small ‘O’ adjacent to the optional entity (you will learn about the Chen and Crow’s Foot models later on). |
| **Primary key** | Generally speaking, a primary key is an attribute (or set of attributes) that contain a value unique to the entity to which it belongs. A primary key uniquely identifies any given row in a table. Each instance of the entity must have the primary key attribute and its value cannot be null. Each entity must have a primary key. Each entity can only have one primary key, although that key can be comprised of multiple attributes. |
| **Foreign key** | A foreign key is an attribute that references the primary key of a related entity. The value of the foreign key must either match one of the values of the primary key of the related entity or be null. |
| **Alternate key** | An alternate key is an attribute (or set of attributes) that could have been used as the primary key but was not. An alternate key is like a primary key in that its value must be unique and it cannot be null. Alternate keys are applied to a table in the database in the form of a unique index defined on the column(s) that make up the alternate key. |
| **Composite entity** | As previously discussed, there are three types of relationships, one of which is the many-to-many (M:N) relationship. However in the relational data model you cannot implement a M:N relationship without first converting it to two 1:M relationships. This conversion is achieved by introducing a composite entity (also known as a bridging entity or an associative entity), which is used to implement a M:N relationship. |
| **Composite primary key** | A composite primary key is a primary key comprising more than one attribute. Composite primary keys will often be used in composite entities. |
| **Redundancy** | There are two types of redundancy, controlled and uncontrolled. Uncontrolled redundancy is the unnecessary and uncontrolled duplication of data, that is, when the same data is stored unnecessarily in different places. This could take the form of duplication of:   * values within a multivalued column * columns of data within a table * rows (records) of data * tables of data.   Avoiding redundancy is the primary goal of data integrity. Redundancy leads to data anomalies, which can affect the integrity of the information extracted from the database because of the inconsistencies that may exist.  Controlled redundancy is the necessary and controlled duplication of data that occurs in the values stored in Foreign Key attributes. When you implement a 1:M relationship, the foreign key always goes in the many table. So, the foreign key may contain multiple values that reference the same primary key on the one side of the relationship. This is controlled redundancy. |
| **Strong or identifying relationships** | A relationship is said to be a strong or identifying relationship when the foreign key that implements the relationship is also part of the primary key of the table. Remember that the foreign key always goes in the table on the ‘many’ side of the relationship. Strong relationships are denoted by a solid relationship line joining the two tables. A strong identifying relationship means that the entity on the many sides of the relationship is existence dependent (it’s a weak entity). i.e. it cannot exist without a related record on the ‘one’ side of the relationship. |
| **Weak or non-identifying relationships** | A relationship is said to be weak or non-identifying when the foreign key that implements the relationship is not part of the primary key of the table. Weak relationships are denoted by a dashed relationship line joining the two tables. A weak non-identifying relationship means that the entity on the many side of the relationship is not existence dependent (it’s a strong entity). i.e. it can exist without a related record on the ‘one’ side of the relationship. |

# Relational database keys

A key in a database is a way of identifying or accessing a row or record in a table. The following table describes the functions and features of the different types of database keys.

Table Definition of key types

|  |  |  |
| --- | --- | --- |
| Key Type | Definition | Function and features |
| **Super key** | Is a table column, or a combination of columns (within the same table), that uniquely identifies each row in a table. | Defined at design time as the initial step in selecting a primary key for a table. Not implemented in the database. |
| **Candidate key** | Is a minimal super key. That is, it can’t be further reduced to produce another super key. | Defined at design time as the second step in identifying possible primary keys for a table. Some entities may have more than one unique attribute that could be the primary key. These attributes are candidate keys until a choice is made. |
| **Primary key** | Is a candidate key that has been chosen by the database designer to uniquely identify all other column values in any given row of a table. | Uniquely identifies each row of a table and can’t contain null values. Each table must have a primary key to ensure entity integrity and to enhance data access performance. |
| **Composite key** | Is a key made up of more than one column and is used as the primary key in composite entities (or bridging entities) or in existence-dependant entities where the relationship with the parent entity is a strong identifying relationship. | Uniquely identifies each row of a table and can’t contain null values. Is comprised of more than one column. |
| **Alternate key** | Is a candidate key that was *not* chosen by the database designer to be the primary key but which can have a unique index (non-clustered) applied to it. | Some entities have more than one attribute that uniquely identifies them. For example, a COMPANY entity may have an *Australian Business Number (ABN)* and a *Company Name* that is both unique. One would be chosen as the primary key, the other would be chosen as an alternate key. |
| **Secondary key** | Is a column, or combination of columns, which is *likely* to return a match when used to search a table. | Is used for retrieval purposes only and is not guaranteed to be unique and might be null. |
| **Foreign key** | A column or combination of columns in one table whose values either reference the values in the primary key of another table or are null (enforcing referential integrity). | Foreign keys are used to implement relationships between tables. The foreign key will always be in the table that is on the many side of a one-to-many relationship. |
| **Surrogate key** | Is a primary key whose values are ‘automatically’ generated by the RDBMS each time a new record is inserted to the table. | Surrogate keys are used where the entity does not have a *natural* primary key. i.e. an attribute that is unique and which all instances of the entity possess. |
| **Clustered index** | An index in which the rows of the data pages of the index are physically stored in order*.* Each table can have only one clustered index because there can only be one set of ‘ordered’ data. The order is based on the key values, so the structure of the index is integral to the data pages. | Used to improve data searching and retrieval performance. Clustered indexes are organised in a similar manner to a phone book. In a phone book the index is defined on the surname and the surnames are in order through the book (note: in RDBMS surnames should not be used as the primary key). By default, primary keys have a clustered index created on them by the RDBMS (though this can be user determined). |
| **Non-clustered index** | An index whose structure is physically separated from the data pages and in which the data pages may be out of order*.* The key of a non-clustered index uses ‘row-pointers’ to locate the desired data page. | Used to improve data searching and retrieval. Non-clustered indexes are organised in a similar manner to a textbook. The index at the rear of the book is a structure that is physically separated from the data pages, however each key of the index has a pointer (page number) that indicates the page on which the data resides. The index key itself is ordered but the data in the pages are out of order. |

Indexes can have both a positive and negative impact on database performance. When an index is defined on a table, the database must re-build the index each time records are either inserted or deleted, or if the indexed columns are updated. This results in a performance hit.

Indexes are beneficial on large tables (with lots of rows) where the data is relatively static (few inserts, updates or deletes) and which are searched often. Indexing should optimise the read performance of the table with benefits that outweigh the write performance of the table.

 Watch

## Video: DBMS keys

Watch this video to learn more about the different types of DBMS keys.

[Concept of Keys in DBMS - Super, Primary, Candidate, Foreign Key, etc.](https://youtu.be/p3yJZH8_bsc) (YouTube 09:15 mins).

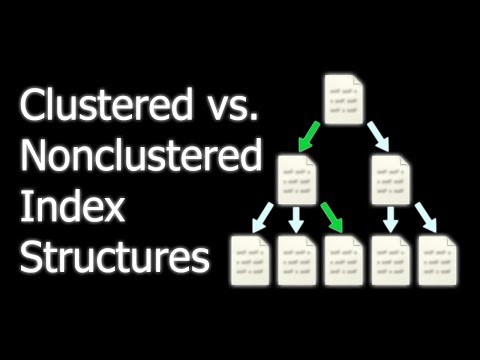
[](https://www.youtube.com/watch?v=p3yJZH8_bsc)

https://youtu.be/p3yJZH8\_bsc

## Video: Clustered vs. non-clustered index structures

Watch this video to learn about the similarities and differences of clustered and non-clustered indexes.

[Clustered vs. non-clustered index structures in SQL server](https://youtu.be/ITcOiLSfVJQ) (YouTube 08:03 mins).

[](https://www.youtube.com/watch?v=ITcOiLSfVJQ)

https://youtu.be/ITcOiLSfVJQ

### Timestamps in keys

In most systems, the TIMESTAMP data type is used primarily for concurrency control. Concurrency control is the process of monitoring and controlling the simultaneous operations performed on a database to ensure that the operations do not conflict with each other. Concurrent operations are those operations that are overlapping.

For read operations, multiple, concurrent reads can be performed without any problem. However, for write operations, if a record is being written-to (updated) by more than one user, then there’s a concurrency control requirement and a timestamp can be used to maintain control of the various ‘versions’ of the row being edited.

The timestamp itself is a unique identifier, which is system generated, using the system clock to generate its value. However, it is not really a time that can be associated with a clock, but rather a unique binary number that indicates the row version, or the order in which the transactions (or operations) were performed on the row. Each time the row is modified, the timestamp value is modified.

For this reason, a timestamp is not a good column data type to use as part of a key. Any foreign keys that reference a primary key with a timestamp column would also have to be updated each time a row in the parent table is modified. The primary key’s index would also have to be updated when a timestamped primary key table has one of its rows modified.

Timestamp values, as opposed to timestamp data types, could be written and used as static values if a unique value is required as part of a key (primary or composite) and, for some reason, an autonumber field is not available or suitable. However bear in mind that this method is also not without its faults. If two users are inserting data at the same time (a slim but possible chance) then the same timestamp may still be generated, so additional unique data would be required to use this as a key value.

# Business rules

When database designers conduct an analysis of an organisation in an attempt to develop a data model, they will first try to identify the entities for which data is required, the attributes of the entities and the nature of the relationships that exist between the entities. Their analysis will help them understand the nature of the organisation’s data and how and when that data is used. However, this is insufficient to provide the analyst with a thorough understanding of how the organisation conducts its business (or operations). Database designers must also understand the business rules of the organisation.

The term business rules relate not only to business organisations, but also to educational institutions, government departments, charity organisations, hospitals or any type of organisation that requires the use of a database.

A business rule is a statement that describes an organisational policy, procedure, or principle. Business rules must be clear, brief, exact, and unambiguous. The business rules must be documented, they must be current and they must reflect the actual operations of the organisation.

The business rules also help define the scope of the database design by defining exactly which operations of the organisation’s business should be implemented in the database. The database model will reflect exactly how the organisation conducts its business and hence will reflect precisely the business rules of the organisation.

Consider the following example of the business rules for a Courier Company Governance System.

**Business rules —courier company governance**

The local government authority (LGA) of a large city requires a database to manage information about the city’s courier companies, the drivers they employ and the vehicles each company owns. The authority is responsible for the governance of the courier driver’s vehicle accreditations and company vehicle ownership.

There are several courier companies operating within the jurisdiction of the LGA. The LGA needs to know the Australian Business Number (ABN), company name and head office phone number for each courier company. No two companies have the same name.

Several types of vehicle are owned by the courier companies. The LGA needs to know the fuel type, fuel capacity, and the vehicle's range for each type of vehicle. In addition, drivers must be accredited to drive any of the types of vehicle and this must be recorded in the system. Driver accreditations expire after six months and the drivers must be re-accredited by completing vehicle-specific training before they’re permitted to drive again.

Each driver works for only one courier company. Smaller companies may have only one driver (owner operated) but the larger companies employ many drivers.

The information to be stored for each driver includes their full name, phone number, driver’s licence number and the types of vehicles they’re accredited to drive. Most drivers are accredited to drive more than one type of vehicle.

The LGA must also be able to produce reports listing the following details of each vehicle owned by the courier companies:

• Vehicle registration number

• Vehicle type

• Owning courier company (each vehicle is owned by only one company)

• Date of manufacture

• Date of last maintenance.

 Watch

## Video: Business rules

Watch this video from LinkedIn Learning to understand the importance of following business rules:

* [Understanding business rules](https://www.linkedin.com/learning/learning-data-analytics-2/understanding-business-rules?u=57684225&auth=true) (LinkedIn Learning 01:29 mins)

# Naming database objects

## Database identifiers

Identifiers are names given to database objects. Database objects include servers, databases, tables, columns, constraints (primary and foreign key constraints), indexes, triggers, stored procedures, functions (both user-defined and system functions), rules, views and user-defined types.

Identifiers form part of the metadata used by the RDBMS to manage the data.

Most database objects require an identifier. However, identifiers are optional for some objects (such as constraints) since the RDBMS will generate an identifier if one is not specified.

There are generally two types of identifiers, regular and delimited.

**Regular identifiers** in RDBMS are similar to identifiers used in programming languages such as C, C++, C#, Java, etc. in that they must follow the naming rules specified by the particular environment.

**Delimited identifiers** are identifiers that do not comply with the environment’s naming rules and must be delimited using delimiters such as double quotes “ “, square brackets [ ], or back-ticks ` ` (to name but a few common delimiters) by surrounding the identifier with the delimiting character.

Identifiers will, more often than not, be used in system and error messages to indicate the name of the object of interest or the name of the object that is causing an error. Consequently, developers should use identifier names that are meaningful to both the database environment and the business environment for which the database is used, to improve the usability of the database.

## Naming conventions

Conventions are an informal standard applied to some aspect of common interaction between people. In data modelling and database design, database developers and programmers have agreed on certain naming conventions, in the hope that adhering to the conventions will make communication and understanding easier. The purpose of naming conventions is to:

* imply additional useful information by the regular use of a naming convention and the name used
* include information about how different database objects may be connected to each other
* ensure uniqueness of names to avoid name clashes
* allow distinction between user-defined names of objects and system-defined identifiers and keywords
* assist in error checking and debugging by providing names that imply the ‘source’ of the error.

More often than not, the data structures and data that support an information system will out-live the applications that access that data. The relational data model has been around for a long time because it does what it does exceptionally well. It is not uncommon for information systems, over time, to have their data-accessing applications re-written and replaced, while the data structures and data remain intact and are re-used with a new front-end.

So, the naming conventions used should account for the longevity of the database and consider that the names actually represent a contract between the database environment and the accessing software applications. Therefore they are not easily changed once they’ve been decided on.

### Table and entity names in the singular

Tables should always be named using the singular name, not the plural.

Each table represents an entity. For example, in a database with a CUSTOMER table, by convention, the CUSTOMER table represents an entity set and that it is a collection of customers. That is, each record in the table represents a single CUSTOMER.

### Column names in the singular

Similarly, columns represent a single value. In fact, one of the cardinal sins of database modelling is a ‘multi-valued’ attribute. Each column should store the data value of a single attribute of the entity.

Examples of multi-valued attributes include the full name or address. Full name can be decomposed to the attributes first\_name, middle\_initial and last\_name, while address could be decomposed to street\_number, street\_name, suburb, post\_code, state. Decomposing multi-valued attributes has an important effect on how efficiently you can search the table.

The following are general naming conventions that should be adhered to:

* Use schema names for a table’s prefix, e.g. *Schemaname.table\_name*
* Primary key suffixed with \_id, e.g. *Customer\_id*
* Foreign key suffixed with \_ref, e.g. *Customer\_ref*
* No spaces or dashes ‘-‘ in database object names (identifiers), i.e. Schemas, tables, columns, constraints, etc.
* Constraint object names:
  + primary Key – tableName\_pk
  + foreign Key – child\_tableName\_parent\_tableName\_fk
* SQL keywords and other system identifiers should all be UPPERCASE.

# Data modelling techniques

Builders use blueprints or plans, which have been prepared by architects and engineers, to model the structures that they’re going to build. Fortunately for us, there is no ambiguity about the information that is communicated to the builder through the plans, simply because the designers and the builder agree on the meaning of the abstracted symbols that are used on the plans.

Similarly, data models use abstracted symbols to represent the various aspects of the structure of the database to be built. There are many data modelling techniques used to design databases, the most common being Data Flow Diagrams (DFD) and Entity Relationship Diagrams (ERD).

## Data flow diagrams

Data flow diagrams are used during the analysis stage of system development. Data flow diagrams contribute to the development of the logical representation of the data model by providing information about the availability of data at any particular point during the processing of the data.

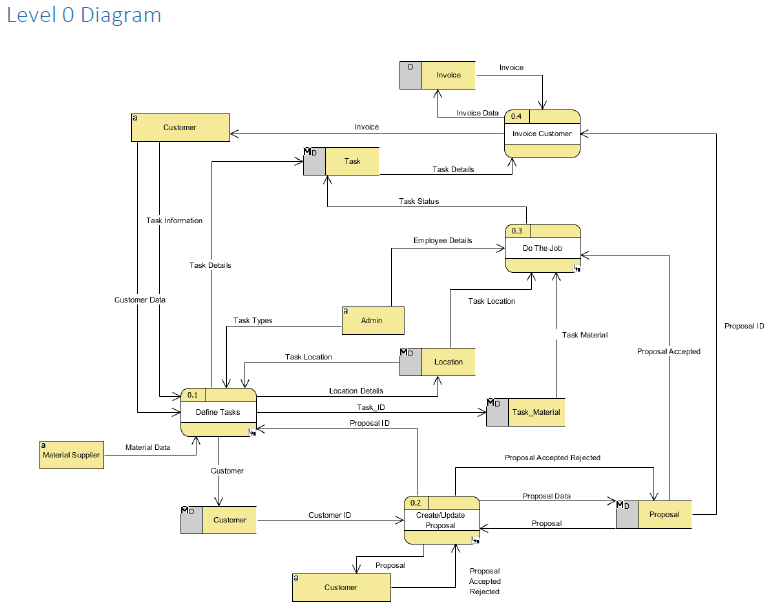


Figure Example of a data flow diagram

## Entity Relationship Diagrams (ERDs)

ERDs are used extensively to model databases. The two types of ERDs that are in use today are the Chen model and the Crow’s Foot model.

Since an entity relationship diagram is a data model, the terms ERD and data model, or model, can be used interchangeably.

### Chen ERD

In the early days of the relational data paradigm, the Chen ERD, developed by Peter Chen in 1976, was used to model the logical structure of relational databases.

The Chen model depicts entities as rectangles, relationships as diamonds and attributes as ovals.

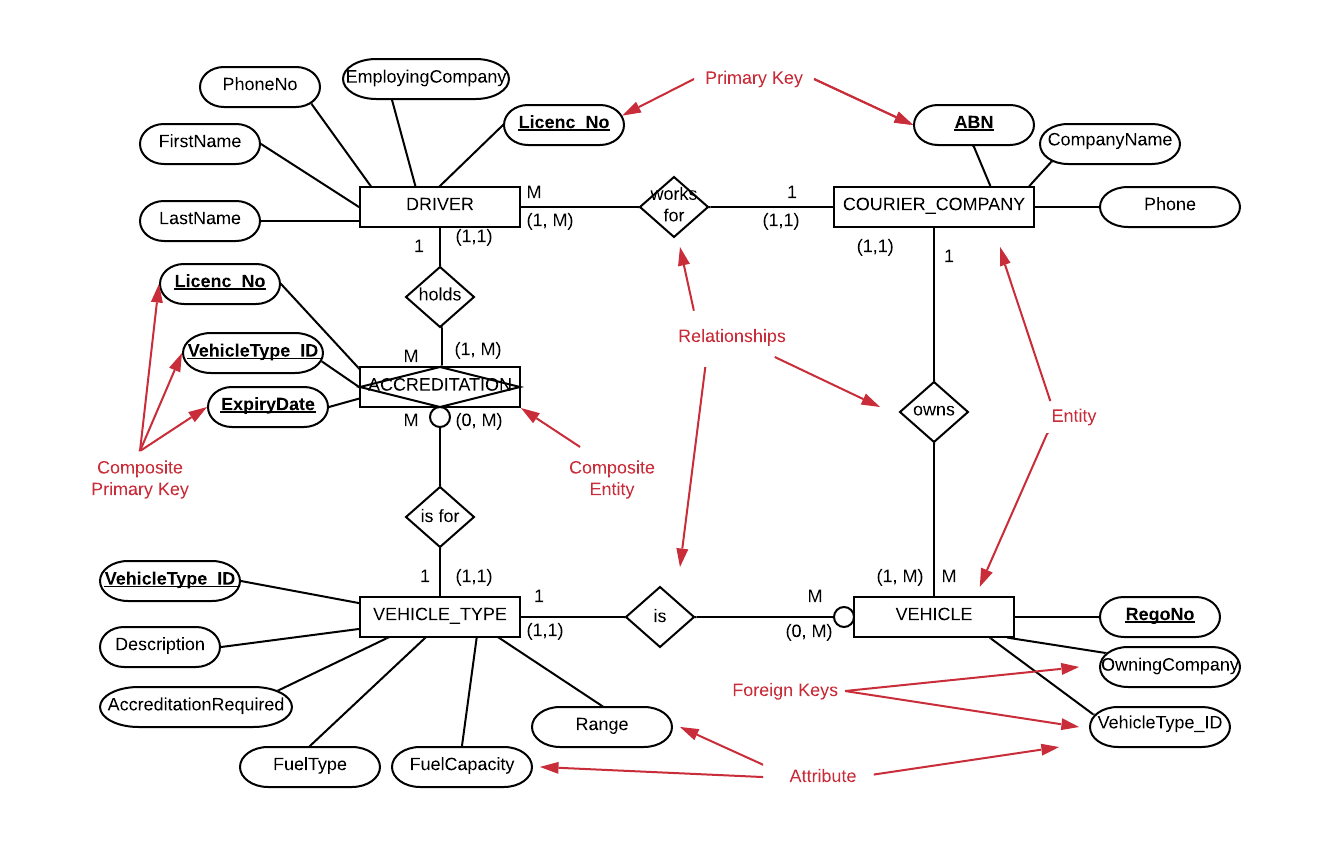


Figure Example of a Chen ERD that includes the attributes

Though the Chen model stipulates the use of ovals to represent attributes, it is quite common to omit the attributes from the model and focus the design process on identifying the entities, the relationships that exist between the entities and the connectivity of the relationships (i.e. the types of relationship).

More recently, the Crow’s Foot ERD has been used to portray more specific detail about where the attributes of the model reside, i.e. which entities the attributes belong to. This is a common practice which aims to reduce the complexity of the model by excluding the attributes from the diagram, only to be added at a later date when the model is further developed, usually using a Crow’s Foot diagram.

The following example demonstrates the use of a Chen ERD without the attributes included in the model.

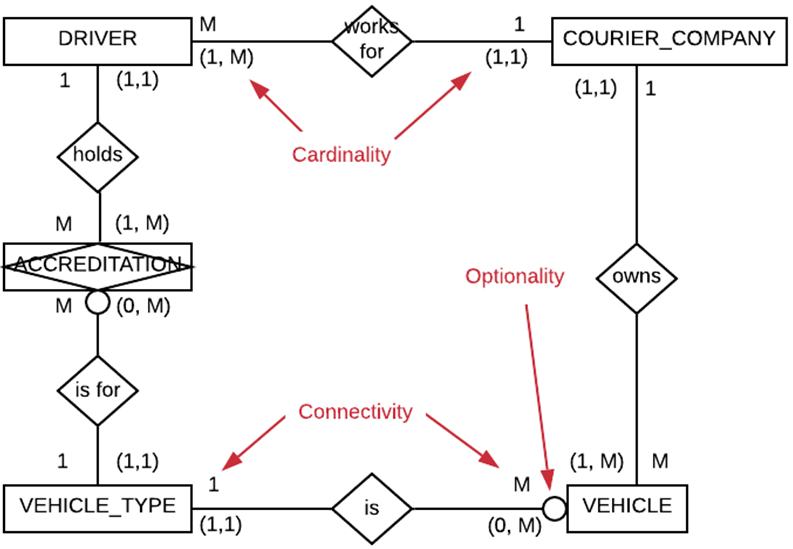


Figure Example of a Chen ERD without the attributes

### Crow’s Foot ERD

The Crow’s Foot ERD is a model that was first introduced by Gordon Everest in the article ‘[Basic Data Structure Models Explained With a Common Example](https://www.researchgate.net/publication/291448084_BASIC_DATA_STRUCTURE_MODELS_EXPLAINED_WITH_A_COMMON_EXAMPLE)’ in 1976. The Crow’s Foot ERD represents entities as boxes containing the attributes, indicating which attributes are the Primary Keys and which are the Foreign Keys. Relationships are indicated by a connecting line that has, at the many ends, what looks like a crow’s foot.

Connecting line diagram ending with three prongs in the represented shape of a crows foot.

Figure Crow's foot example

The Crow’s Foot ERD is the more contemporary model and is used more widely to model relational databases. Though the Chen model is often used to develop the ‘first draft’ of the model, where the emphasis is on identifying the entities and their relationships, the concern for the location of the attributes can be dealt with in the second draft, which is usually where a Crow’s Foot model is used.

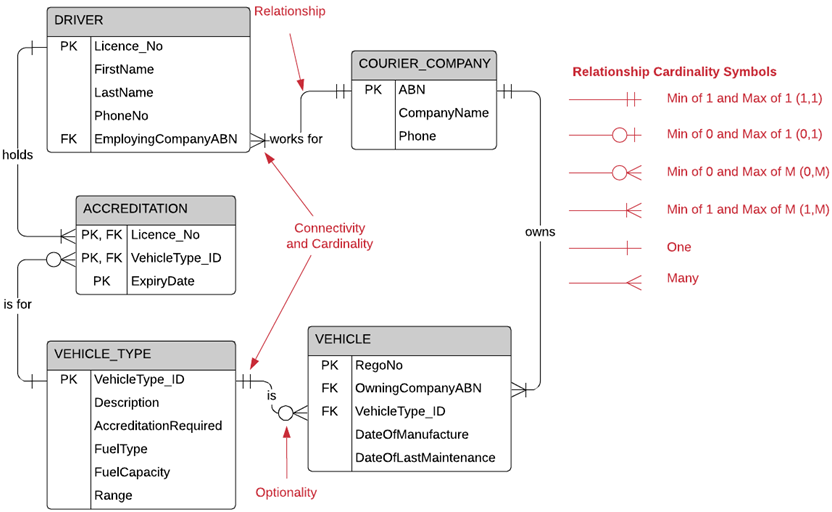


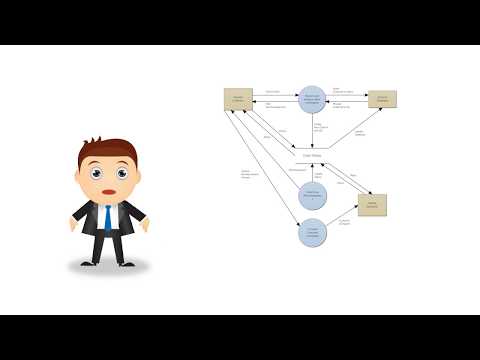
Figure Example of a Crow’s Foot ERD indicating the meaning of the cardinality symbols used

 Watch

## Video: Data Flow Diagrams

Watch this video to learn more about DFDs and symbols.

[Data Flow Diagrams - What is DFD? Data Flow Diagram Symbols and More](https://www.youtube.com/watch?v=6VGTvgaJllM) (YouTube 04:13).

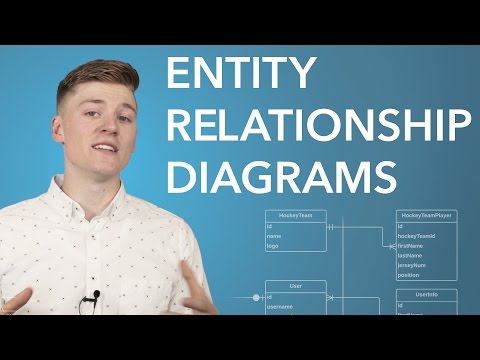
[](https://www.youtube.com/embed/6VGTvgaJllM?feature=oembed)

https://www.youtube.com/watch?v=6VGTvgaJllM

## Video: ERDs

Watch these videos to learn more about ERDs and how to create them using Lucidchart.

[Entity Relationship Diagram (ERD) Tutorial - Part 1](https://youtu.be/QpdhBUYk7Kk) (YouTube 06:57 mins)

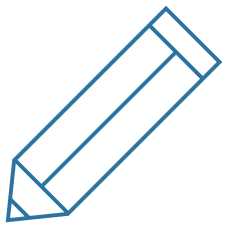
[](https://www.youtube.com/watch?v=QpdhBUYk7Kk)

<https://youtu.be/QpdhBUYk7Kk>

[Entity Relationship Diagram (ERD) Tutorial - Part 2](https://youtu.be/-CuY5ADwn24) (YouTube 13:50 mins)

[](https://www.youtube.com/watch?v=-CuY5ADwn24)

https://youtu.be/-CuY5ADwn24

 Practice activity

## Activity 3.1: Create an ERD

Using the [LucidChart](https://www.lucidchart.com/pages/) diagramming software on the web, reproduce the Crow’s Foot ERD in Figure 6 Example of a Crow’s Foot ERD indicating the meaning of the cardinality symbols used. Exclude the items indicated in red.

# Normalisation

Normalisation is the process of developing good table structures in order to control data redundancies and avoid the creation of anomalies in the data. Normalised databases are much less complicated and much easier to use than de-normalised data structures. They also more accurately reflect the business rules and operations of the organisation.

The aim of normalisation is to ensure that each table in the database is designed to conform to the following principles:

* Each table is representative of one ‘thing’. For example, the DRIVER table should contain only data specific to drivers. The VEHICLE table should contain only data specific to vehicles.
* Data should not be duplicated unnecessarily. That is, each data item should be stored in only one location. This ensures that if the data must be updated (or deleted) it can confidently be located in one location only.
* The determination (finding) of each non-prime attribute of a table is achieved through the use of the primary key only. This refers to using the entire primary key (where a composite primary key is in use) and no other attributes.
* That there are no insert, update, or delete anomalies in all tables in the database that will compromise the consistency and integrity of the data.

## Functional dependence and determination

The rules of normalisation rely on the concepts of functional dependence and determination. The functional dependence ‘A determines B’ is represented as A → B.

The following definitions can be used to describe these concepts.

Table Functional dependence concepts

|  |  |
| --- | --- |
| Concept | Description |
| **Functional dependence** | Attribute *A* determines Attribute *B*. i.e. If Attribute *A* will always return a single value for Attribute *B* whenever the value of Attribute *A* is used to search the table, then you can say that Attribute *A* determines Attribute *B* or A → B.  For example in the DRIVER table in Figure 6, *Licence\_No* → *LastName* i.e. *Licence\_No* can be used to determine *LastName*.  Attribute *A* is known as the *determinant* attribute and Attribute *B* is known as the *dependent* attribute. In this example *Licence\_No* is the *determinant* attribute and *LastName* is the *dependent* attribute. |
| **Full functional dependence (composite keys)** | If all attributes of the composite key *AB* determine attribute *C,* then attribute *C* is said to be fully functionally dependent on *AB.* i.e. *C* can’t be determined by **only part** of the primary key *AB* or AB → C. |

The idea is, that if you used one attribute value to search a table for another attribute value, ***one and only one*** value will be returned.

## Normalisation rules

### First Normal Form (1NF)

A table is in First Normal Form (1NF) if:

* All attributes that make up the primary key are defined
* Each field contains a single value, that is, at the intersection of each row and column there is only one value, not a set of values
* There are no repeating groups, that is, columns representing the same domain of values are not repeated. For example, in an ORDER table you don’t want columns PRODUCT1, PRODUCT2, PRODUCT3, etc. as these are repeating groups
* Each attribute of the table can be determined (found) using the primary key.

### Second Normal Form (2NF)

A table is in Second Normal Form (2NF) if:

* if it meets the requirements of First Normal Form (1NF)

and

* No attribute can be determined (found) by only part of the primary key. This is referred to as a partial dependency.

It is possible for a table to be in 2NF and still contain non-prime attributes that determine other non-prime attributes. This is referred to as a transitive dependency.

### Third Normal Form (3NF)

A table is in Third Normal Form (3NF) if:

* If it meets the requirements of Second Normal Form (2NF)

and

* No attribute can be determined (found) by any other non-prime attribute.

For most business-related transactional databases, 3NF is sufficient. However, there may be occasions where higher levels of normalisation will be required. These are Boyce-Codd Normal Form and Fourth Normal Form (4NF).

### Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form is a special case of 3NF. It applies only if the table has more than one candidate key, i.e. more than one column that could have been selected to be the Primary Key. In the COURIER\_COMPANY table of the Courier Company governance system in Figure 6 Example of a Crow’s Foot ERD indicating the meaning of the cardinality symbols used, there are two candidate keys, the ABN, which is unique to each company, and the company name, which is also unique to each company.

A table is in Boyce-Codd Normal Form (BCNF) if:

* Every determinant in the table is a candidate key.

That is, every attribute that could be used to determine the singular value of any other attribute must be a candidate key.

### Fourth Normal Form (4NF)

A table is in Fourth Normal Form (4NF) if:

* It is in 3NF

and

* it has no multivalued dependencies.

## Normalisation processes

### Achieving 1NF

**Step 1**: Eliminate all repeating groups.

This can be achieved by determining that each entity will represent a single distinct ‘thing’ for which you want to store data and that each entity will become a table. Be sure you don’t confuse entities with attributes.

**Step 2**: Identify the primary keys.

Determine which attribute(s) uniquely identify each instance of the entity.

**Step 3**: Identify all the dependencies.

Identify the attributes that are dependent on other attributes. Paying particular attention to the attributes that are dependent on only part of the primary key (partial dependencies), and those attributes that are dependent on other non-prime attributes (transitive dependencies).

### Achieving 2NF

**Step 1**: Remove any partial dependencies by making a new table.

Place each determinant in the new table as that table’s primary key.

**Step 2**: Relocate each dependant attribute to the new table.

Place each attribute that had been dependent on only part of the primary key into the new table.

### Achieving 3NF

**Step 1**: Remove any transitive dependencies by placing them in a new table.

Move the determinant non-prime attribute into the new table, making it the primary key of the new table.

**Step 2**: Move the corresponding dependent attributes into the new table.

### Achieving BCNF

**Step 1**: Ensure that any transitive dependencies that still exist involve a determinant that is a candidate key. If not, repeat step one of Achieving 3NF above.

### Achieving 4NF

**Step 1**: Ensure that all attributes are dependent on the primary key and independent of each other.

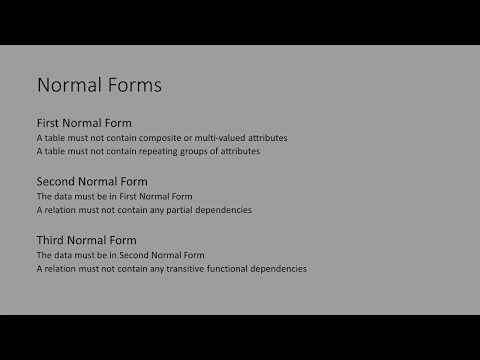
**Step 2**: Ensure that tables do not contain any rows that have two or more multivalued facts about the entity that the table represents.

 Watch

## Video: Database normalisation—introduction

Watch this series of videos on database normalisation to further understand the concepts and the process of normalising a database.

[Database normalisation—introduction](https://youtu.be/y03oYWDLu0Q) (YouTube 03:43 mins)

[](https://www.youtube.com/watch?v=y03oYWDLu0Q)

https://youtu.be/y03oYWDLu0Q

[Database normalisation—first normal form](https://youtu.be/jgUeOjImOOw) (YouTube 07:26 mins)

[](https://www.youtube.com/watch?v=jgUeOjImOOw)

https://youtu.be/jgUeOjImOOw

[Database normalisation—second normal form](https://youtu.be/9L10Q1nAfyg) (YouTube 09:20 mins)

[](https://www.youtube.com/watch?v=9L10Q1nAfyg)

https://youtu.be/9L10Q1nAfyg

[Database normalisation—third normal form](https://youtu.be/_K7fcFQowy8) (YouTube 08:29 mins)



https://youtu.be/\_K7fcFQowy8

# Data types

Computers are fundamentally a collection of digital switches that can be either on = 1 or off = 0. Hence the use of the binary numbering system to represent values in computer systems. Computer memory stores binary values (as collections of bits, or binary digits, 0 or 1). Storing whole numbers can be done using their binary equivalent, but when you want to store characters other than numbers you assign the character a numeric value, which can be represented as a binary value.

For example, the decimal value assigned to the uppercase letter A is 65, which can be represented as the binary value 01000001 (this one byte; 8 bits to a byte). In the real world, numbers can be infinitely large or the difference between two numbers can be infinitely small. Computers however, have a finite amount of memory for storing values. Both the upper limit of the largest number that can be represented and the lower limit of the smallest number that can be represented are finite.

Also, it is sometimes enough to store a reasonable approximation of a value. For example, storing your weight as 85.6 kg is good enough. You don’t need to store your weight to any higher precision than that for most practical purposes. So why do you have to define the data types of the things you want to store in your computer system?

First, the system must know how much memory to set aside for the storage of data. For example in Microsoft SQL Server, the data type int (integer) can store whole numbers in the range -2,147,483,648 through to +2,147,483, 647 and uses 4 bytes of memory.

Secondly, the system must know which operations are permitted on the data. For example, it doesn’t make sense for us to subtract 1 from the word ‘elephant’. i.e. elephant − 1 = error. So the data type chosen for any particular attribute should reflect the possible operations that you anticipate performing on that attribute.

Thirdly, data types determine the range of possible values that can be stored, as mentioned earlier for the int data type. So your choice of data type for any particular attribute (column) should be determined by the range of possible values you want to store.

Finally, data types determine the precision of the values that can be stored.

So, your choice of data type for any particular attribute will be based on those factors.

The following table describes some of the features and usage of SQL Data Types. Note that not all data types are listed here and not all platforms support the data types listed here. Your data type usage will be specific to the platform that you’re developing for.

Table Features and functions of data types

|  |  |  |
| --- | --- | --- |
| Category | Data type | Features |
| **Numeric** | BIT | Ranges from 0 to 1 |
| TINYINT | Ranges from 0 to 255 |
| SMALLINT | Ranges from -32,768 to 32,767 |
| INT | Ranges from -2,147,483,648 to 2,147,483,648 |
| BIGINT | Ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| DECIMAL | Ranges from -10^38 +1 to 10^38 -1 |
| NUMERIC | Ranges from -10^38 +1 to 10^38 -1 |
| FLOAT | Ranges from -1.79E + 308 to 1.79E + 308 |
| REAL | Ranges from -3.40E + 38 to 3.40E + 38 |
| MONEY | Ranges from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| SMALLMONEY | Ranges from - 214,748.3648 to 214,748.3647 |
| **Date/Time** | DATE | Date in the format YYYY-MM-DD 0001-01-01 through 9999-12-31 |
| TIME | Stores date in the format YYYY-MM-DD |
| DATETIME | Stores date and time information in the format YYYY-MM-DD HH:MI:SS |
| TIMESTAMP | Stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC) |
| YEAR | Year stored in 2 digit or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069 in 2-digit format |
| **Character** | CHAR(*size*) | Fixed length with maximum length of 8,000 characters |
| VARCHAR(*size*) | Variable-length character data with maximum length of 8,000 characters |
| VARCHAR(MAX) | Variable-length character data with provided max characters (up to 2GB) |
| TEXT | Variable length character data with maximum length of 2,147,483,647 |
| **Unicode Character** | NCHAR(*size*) | Fixed-length Unicode characters with maximum length of 4,000 characters |
| NVARCHAR(*size*) | Variable length character data with a maximum length of 4,000 characters |
| NVARCHAR(MAX) | Variable-length Unicode character data with a maximum length up to 2GB |
| NTEXT | Variable-length Unicode character data with a maximum string length of 2^30 - 1 (1,073,741,823) bytes. |
| **Binary** | BINARY(*size*) | Fixed-length binary data with maximum length of 8,000 bytes |
| VARBINARY(*size*) | Variable-length binary data with maximum length of 8,000 bytes |
| VARBINARY(MAX) | Variable-length binary data with provided max bytes (2^31-1 bytes) |
| IMAGE | Variable-length binary data with maximum size of 2GB |
| **Miscellaneous** | CLOB | Up to 2GB of character large objects |
| BLOB | Binary Large Objects, usually variable-length binary data from 0 through to 2GB (2^31-1) |
| XML | For storing XML data |
| JSON | For storing JSON data |

**Note:** Where size is not specified, most database platforms will default to a size of 1. Where the data type is defined as variable, such as VARCHAR(size) and NVARCHAR(size), any values entered that exceed the specified size will be truncated. i.e. if you define a column as VARCHAR(10) and you insert the value ‘*this is more than 10 characters’* the data will be truncated (chopped off) and only the first 10 characters will be inserted, ‘*this is mo’* (inclusive of the spaces).

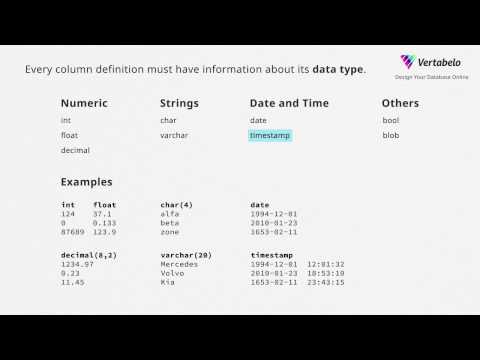
So, variable-length does NOT mean that the column will grow to accommodate larger values. In fact, variable-length indicates that the column will shrink if you enter fewer characters than the specified size, removing any trailing spaces which could cause problems if they end up in the database.

 Watch

## Video: Table columns and data types

View this video on table columns and data types to understand how you choose the data types that should be used for each column.

[Tutorial 2. Table Columns and Data Types](https://www.youtube.com/watch?v=Zpi2GLJgJzI&feature=youtu.be) (YouTube 03:09 mins).

[](https://www.youtube.com/watch?v=Zpi2GLJgJzI)

https://youtu.be/Zpi2GLJgJzI

## Data type hierarchies

The relational data model has survived for as long as it has because of its flexibility and its profound utility. When it was conceived, it was developed to also support the implementation of relationships between entities that are hierarchical. For example, hierarchical data structures that are commonly stored in relational databases include:

* organisational hierarchies
* filing systems
* project tasks that are divided into sub-tasks
* taxonomies of animals, such as mammals
* the navigational links between the pages of a web site.

In cases where such hierarchical relationships exist, the relationships can be either one-to-one or one-to-many. In either case the relationship is an identifying relationship in which the child entities are existence dependent on the parent entity. That is to say, the child entity can’t exist without the existence of the parent.

Consider the following business rules describing the hierarchical structure of employees within an organisation:

Everyone who works for the organisation is considered an employee. All employees have their pay paid into one bank account. This includes the Chief Executive Officer (CEO) who is paid a salary, while other departmental staff are paid by the hour, according to their job type.

The system must also record the CEO’s period of tenure (start date and end date). The organisation has several departments including, but not limited to, sales, human resources, operations and security, with each department having only one manager. All department managers report to the Chief Executive Officer (CEO).

The manager of each department is also paid a salary and is responsible for the management of only one department. Each department staff member (excluding the CEO) works in only one department, although each department could have several staff members working for it.

Figure Organisation chart

In this scenario, the **super-type** entity is the EMPLOYEE since everyone is an employee. The three **sub-types** are the chief executive officer (CEO), department manager and staff. In this case the EMPLOYEE entity is also the **root-type** because each of the sub-types is an EMPLOYEE.

Figure The super type/sub type relationship

The relationships between the super-type (EMPLOYEE) and each of the sub-types (CEO, department manager and staff) is one-to-one. Each of the sub-types has distinct attributes that don’t apply to all instances of the super-type (EMPLOYEE). As an example, the CEO has a salary as well as a tenure start and end date.

The department managers have a salary and a department for which they are responsible. The department staff have a job-type and an hourly rate based on that job-type. So, the general rule for implementing one-to-one relationships is as follows:

*Where two entities are related in a one-to-one relationship, the entities should be combined into a single entity, except where both:*

* *the relationship is a super-type/sub-type relationship*
* *the sub-types have additional attributes that the super-type does not have.*

Consider the business rule in this scenario, ‘*all employees have their pay paid into one bank account*’; it can be reasonably assumed that each bank account belongs to only one employee. So the relationship between employee and their bank account is also one-to-one. However, it is NOT a super-type/sub-type relationship because employees are not bank accounts and vice-versa.

Consequently the employee’s bank account details (BSB number, account number and account name) can be combined into the EMPLOYEE entity.

Here’s the relational data model for the implementation of this scenario:

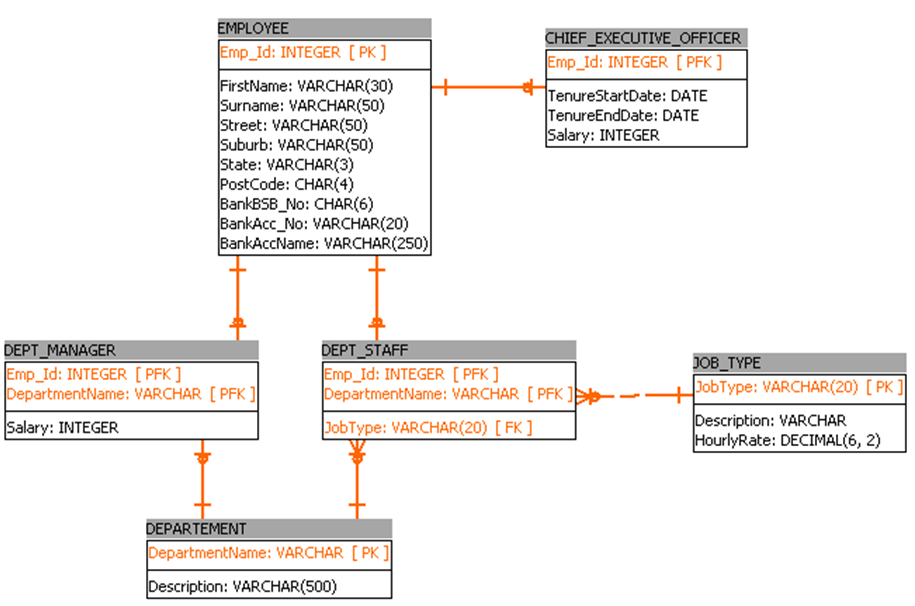


Figure The Crow’s Foot ERD implementing the organisation-employee hierarchy

## Structured data types

Where columns in a database are known to have the same domain of possible values, you can create a structured data type for those columns, which combines the underlying native datatypes of each column into a new, single User-Defined Type (UDT).

A UDT can be based on a single predefined system data type or it can be a structured type that is comprised of a list of attribute definitions using multiple predefined system types. For example, in the EMPLOYEE table in Figure 9 The Crow’s Foot ERD implementing the organisation-employee hierarchy, you could combine the three columns for the bank account details into a single user-defined type.

The following example could be implemented on IBM’s DB2 platform, however it should be noted that not all platforms allow multi-attribute user-defined types.

CREATE TYPE **BANKACCDETAIL\_udt** AS (BankBSB\_No CHAR(6), BankAcc\_No VARCHAR(20), BankAccName VARCHAR(250))

Having defined the type, it can then be used as the type for a column when creating the table as follows:

CREATE TABLE employee (Emp\_Id INTEGER PRIMARY KEY, FirstName VARCHAR(30), Surname VARCHAR(50), Street VARCHAR(50), Suburb VARCHAR(50), PostCode CHAR(4), BankAccDetail **BANKACCDETAIL\_udt** )

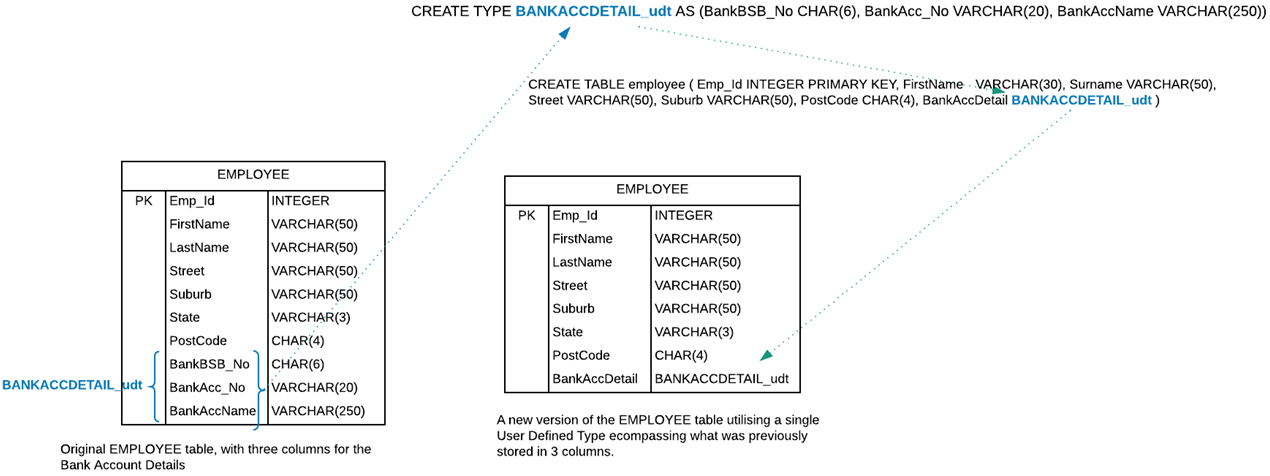


Figure Example of the use of a user-defined data type

Structured types or UDTs are always based on the underlying predefined system-supplied data types, which makes their relationships hierarchical. The underlying system-supplied data types are the root-types, with the next UDT in the hierarchy being the super-type and subsequent inheriting UDT being the sub-types.

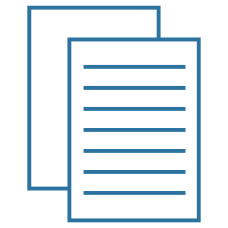
UDTs can be used to define the type for a column, as described above, the type for an entire table (or view), or to define an attribute of another structured type. Once set up, user-defined types are as easy to use as the DBMS predefined types. However they are a lot harder to set up than the example above, the details of which are beyond the scope of this course.

## Reference types

In SQL a reference type is a scalar type that acts as a pointer to a row of a base table that is a typed table. To put it more simply, a reference type is a pointer to a UDT’s value.

## User-**defined functions**

In the context of the use of structured data types and UDTs in SQL, user-defined functions serve to implement ‘methods’ in the structured data type. So you can use a user-defined function as part of the definition of a structured data type. This will process some (or all) of the values contained in the basic attributes of the structured data type to derive a new value. The derived value is the return value of the user-defined function, often referred to as a derived attribute.

 Resources

## Additional resources

Review the following readings for a more detailed understanding of user-defined types, reference types, user-defined functions and type hierarchies:

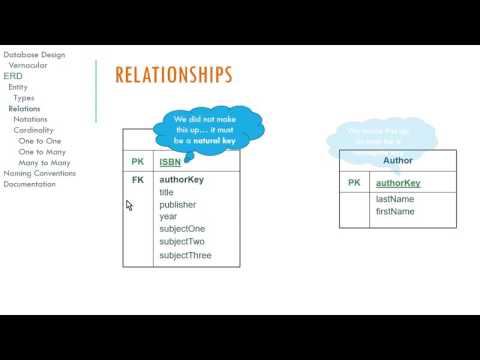
* [Chapter 27 - User-defined types](https://crate.io/docs/sql-99/en/latest/chapters/27.html)
* [Chapter 12 - Reference types](https://crate.io/docs/sql-99/en/latest/chapters/12.html).

 Watch

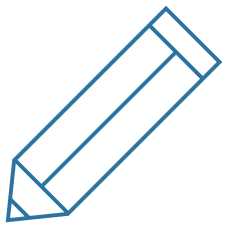
## Video: Logical database design and ERDs

Watch this video to learn more about the process of logical database design and ERDs.

[Logical database design and E-R diagrams](https://youtu.be/ZBgXb66Ckz0) (YouTube 32:22 mins).

[](https://www.youtube.com/watch?v=ZBgXb66Ckz0)

https://youtu.be/ZBgXb66Ckz0

 Practice activity

## Activity 3.2: Model a database

In this activity, you will be developing the design of the data model.

### Step 1: Define the business rules

Review the [courier company governance business rules](#busrules) defined on page 52 to determine their impact on the data model for the database.

### Step 2: Identify the entities required for the model

In this step you will analyse the business rules and identify the entities (the things you want to store data for). In this step, ignore the attributes of the entities.

Use the following rules as a guide to identifying the entities:

1. An entity will be something of which there is usually more than one instance. For example there are many courier companies.
2. An entity will have attributes (data) that you will need to use. For example, for the courier company you must know its ABN, company name, and phone number.

Before continuing, what four entities have you identified?

Note: An important note about **types** of things. The world is full of types of things, such as types of cars, aeroplanes, students, animals, books, etc. Types are a fundamental mechanism for categorising the world as you know it. In relational database modelling, you will often need to store data about **types** of things in contrast to storing data about **instances** of things (though you will usually want to do this as well). This is because the data that relates to one type will be shared by all instances of that type. So to avoid uncontrolled redundancy, you store a single record of the shared data in the **type** table and reference that record from a foreign key in the **instance** table. For example, the Vehicle Type entity will have the attributes *Description*, *Accreditation Required*, *Fuel Type*, *Fuel Capacity* and *Range*. The values for these attributes will be the same for all instances of that *Vehicle Type*.

### Step 3: First draft ERD

The entities you should have identified in previous step are:

* Courier company
* Vehicle
* Vehicle type
* Driver.

Begin the development of a first draft model using the Chen notation and determine:

* which entities are related to each other
* the connectivity and cardinality of the relationships.

From the following business rules you can determine that driver and courier company are related:

* Each driver works for only one courier company.
* Smaller companies may have only one driver (owner operated) but the larger companies employ many drivers.

When you analyse this relationship you can determine that ‘*a driver works for only one courier company*’ and ‘*a courier company may employ many drivers*’. Furthermore, you can determine that ‘*a driver works for a minimum of one and a maximum of one courier company*’ and that ‘*a courier company employs a minimum of one and a maximum of many drivers*’.

So the relationship is mandatory and your model will reflect this as follows:

One courier company can have many drivers working for them. A one to many relationship.

Figure The 1:M relationship between driver and courier company.

From the following business rules you can determine that a courier company and vehicle are related:

* Several types of vehicles are owned by courier companies.
* Each vehicle is owned by only one company.

When you analyse this relationship you can determine that ‘*a courier company can own many vehicles’* and ‘*a vehicle is owned by one courier company’*. Furthermore, you can determine that ‘*a courier company can own a minimum of one and a maximum of many vehicles’* and that ‘*a vehicle is owned by a minimum of one and a maximum of one courier company’*. So the relationship is mandatory and your model will reflect this as follows:

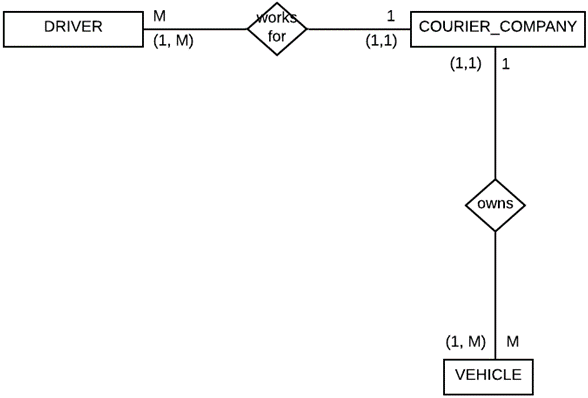


Figure The vehicle entity has now been added to the model

From the following business rule you can determine that vehicle and vehicle type are related:

* Several types of vehicles are owned by the courier companies and for each type of vehicle the fuel type, fuel capacity, and the vehicle’s range are required.

When you analyse this relationship you can determine that ‘*a vehicle is a minimum of one and a maximum of one vehicle types*’ and *‘a vehicle type can have a minimum of zero and a maximum of many vehicles that are of that type’*.

In this case the association of vehicle to vehicle type is optional, because you might not have an instance of a particular vehicle type owned by any of the courier companies. Your model will reflect this as follows:

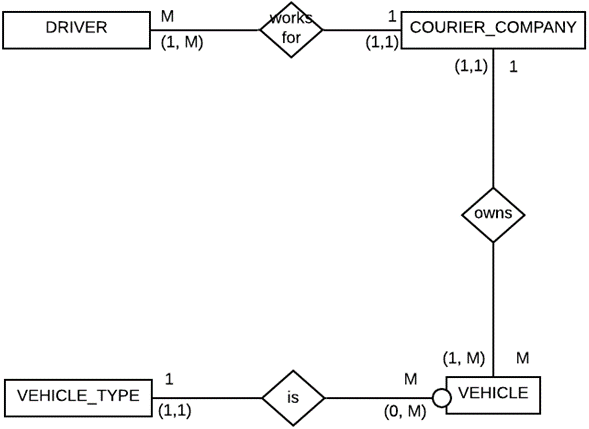


Figure The vehicle type entity has now been added to the model

From the following business rules you can determine that driver and vehicle type are related:

* Drivers must be accredited to drive any of the types of vehicle and this must be recorded in the system.
* The information to be stored for each driver includes their full name, phone number, driver’s licence number and the types of vehicles they’re accredited to drive.
* Most drivers are accredited to drive more than one type of vehicle.

When you analyse this relationship you can determine that ‘*a* *driver may be accredited to drive many vehicle types*’ and ‘*a vehicle type can have many drivers that are accredited to drive that type*’.

So you have established that a many-to-many (M:N) relationship exists between vehicle type and driver.

Initially, you will model the relationship between driver and vehicle type as M:N as follows:

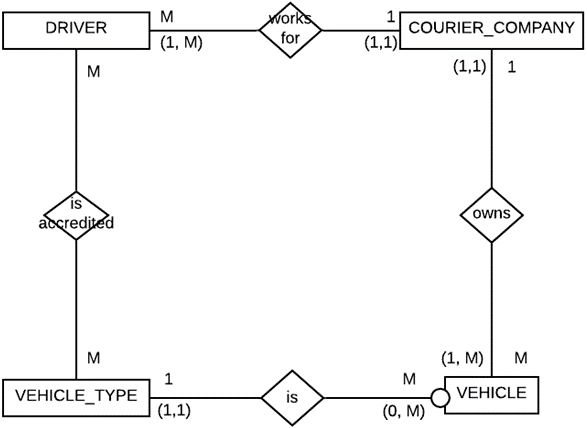


Figure Driver and vehicle type identified as being in an M:N relationship

However to implement a M:N relationship, you will have to convert the relationship to two one-to-many (1:M) relationships by utilising a composite entity. In this case, the **is accredited** relationship will become a separate entity, named ‘*accreditation*’. This will become the **many** entity in both relationships either side of it.

This method is used to implement all many-to-many relationship. Now, let’s implement that in your model:

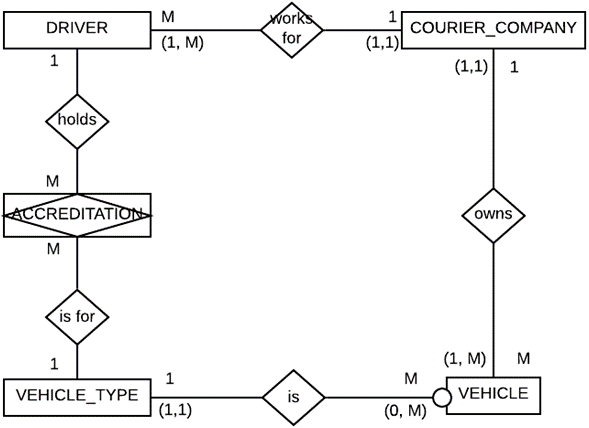


Figure Implementation of the composite entity (ACCREDITATION) converting the M:N relationship to two 1:M relationships

You can now assess the cardinality of the relationship between driver, accreditation and vehicle type as follows; ‘*a driver can have a minimum of* ***one*** *and a maximum of* ***many*** *accreditations’* and *‘an accreditation is for a minimum of* ***one*** *and a maximum of* ***one*** *driver’.*

On the other side you have *‘an accreditation is for a minimum of* ***one*** *and a maximum of* ***one*** *vehicle types’* and ‘*a vehicle type can have a minimum of* ***zero*** *and a maximum of* ***many*** *accredited drivers’.* So accreditation is optional to vehicle type. This means that there may not be an accredited driver in the database for a particular vehicle type.

That completes the first draft of the Chen ERD for the courier company governance database. Your model now looks as follows:

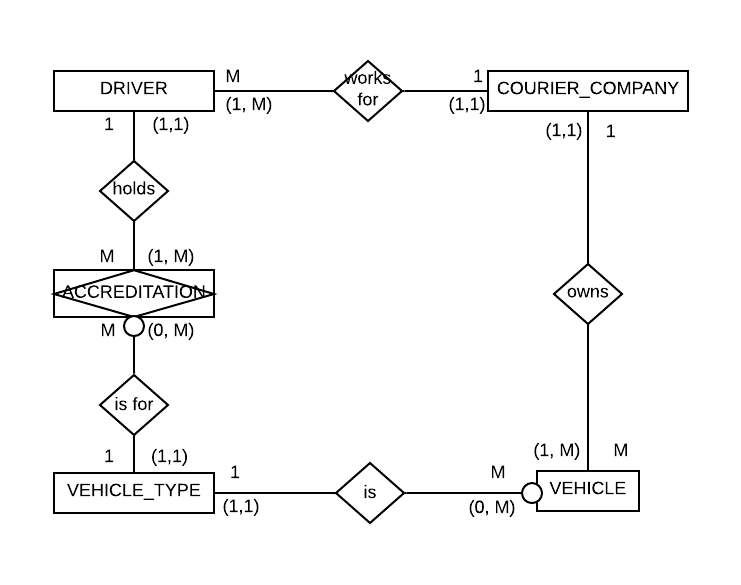


Figure The cardinality and optionality of the relationships either side of ACCREDITATION entity complete

### Step 4: Second draft ERD

Begin the development of the second draft of the model using a Crow’s Foot ERD.

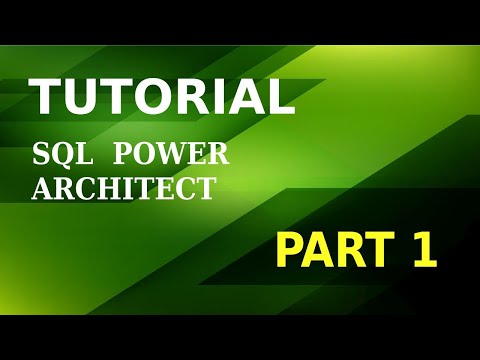
There are many CASE tools that you can use to develop the Crow’s Foot ERD. [Vertabelo.com](https://www.vertabelo.com/) hosts a very good data modelling application that you can try. For this step however, you will use a tool named [SQL Power Architect](http://www.bestofbi.com/downloads/architect/1.0.8/SQL-Power-Architect-Setup-Windows-jdbc-1.0.8.jar).

 Watch

## Video: SQL Power Architect

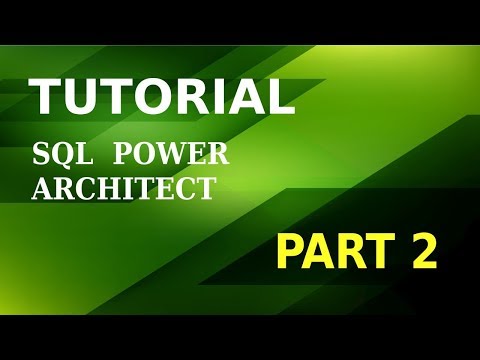
There are several tutorials on YouTube that describe the use of SQL Power Architect. Here are a few to help you get started.

* Tutorial on data modelling in SQL Power Architect:
  + [Part 1—Installation](https://www.youtube.com/watch?v=O9JB1pUmzts) (YouTube 05:16 mins)

[](https://www.youtube.com/embed/O9JB1pUmzts?feature=oembed)

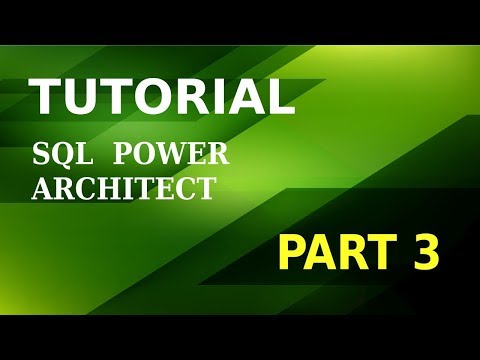
https://www.youtube.com/watch?v=O9JB1pUmzts

* + [Part 2—Creating a physical model](https://www.youtube.com/watch?v=JMY5BxcpCu8) (YouTube 09:00 mins)

[](https://www.youtube.com/embed/JMY5BxcpCu8?feature=oembed)

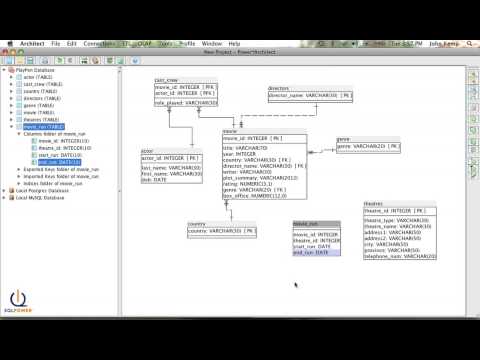
https://www.youtube.com/watch?v=JMY5BxcpCu8

* + [Part 3—Next steps in creating a physical model](https://www.youtube.com/watch?v=6bQAEdXhCJM) (YouTube 08:46 mins)

[](https://www.youtube.com/embed/6bQAEdXhCJM?feature=oembed)

https://www.youtube.com/watch?v=6bQAEdXhCJM

* [Data modelling using SQL Power Architect (2 of 3)](https://youtu.be/hHBSMGr6yDg) (YouTube 06:25 mins).

[](https://www.youtube.com/embed/hHBSMGr6yDg?feature=oembed)

https://youtu.be/hHBSMGr6yDg

Add each of the five entities to the Crow’s Foot Model. At this point in your development of the database, you have begun to define the physical model, so you will begin to refer to the entities as **TABLES** and the attributes as **COLUMNS**.

As you define each of your tables you will add columns to the tables and decide on the data types to be used for the columns. With each of the tables added your model now looks as follows:

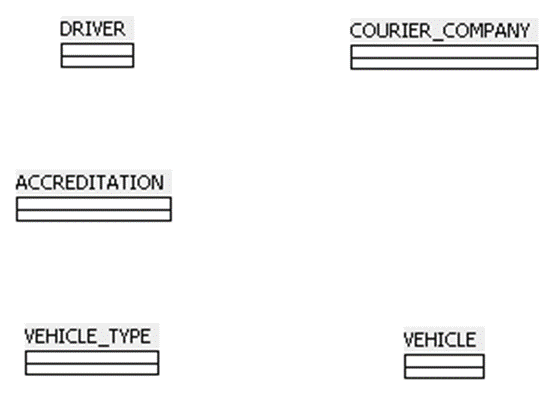


Figure Each of the tables added to the Crow’s Foot model (no columns or data types have been defined yet)

### Step 5: Define the primary keys

The composite entities will implement many-to-many relationships. Since all drivers in your system must have a driver’s licence and the driver’s licence is a natural attribute of Driver and the driver’s licence number is unique to each driver, it makes an excellent primary key for the Driver table.

Newer driver’s licence numbers are eight numeric characters long, while older licence numbers are six alpha-numeric characters long. To accommodate this difference you will use a data type of NVARCHAR(8) for the driver’s licence.

Primary key is licence_no and date type is NVARCHAR(8).

Figure The DRIVER table with primary key column and its data type defined

Similarly for the courier company table, the ABN is a unique, natural attribute that all courier companies must have so you will use it for the primary key. An ABN is 11 numeric characters long, the first two digits designated by the Australian Tax Office (ATO) as checksum digits. So, you may want to apply the ATO’s ABN generating/checking algorithm to the values stored in your database. You’ll use the BIGINT data type.



Figure The COURIER\_COMPANY table with primary key column and its data type defined

For the vehicle table you will use the vehicle registration number (number plate), since all vehicles must be registered and must have a registration number and that number is unique to each vehicle. Vehicle registration numbers are alpha-numeric and can be either five or six alphanumeric characters long. You will use NVARCHAR(6) for the data type of the vehicle table’s primary key.

Primary key is registration_no and data type is NVARCHAR(6).

Figure The VEHICLE table with Primary Key column and its data type defined

For the vehicle type table you will use a surrogate primary key. A surrogate primary key is a primary key that is not a natural attribute of the entity, but rather is usually generated by the system (an auto-generated primary key). Surrogate primary keys are usually integers.

Primary key is VEHICLETYPE_ID and data type is integer.

Figure The VEHICLE\_TYPE table with surrogate primary key column defined as an Integer

### Step 6: Implement the relationships between the related entities

Since you have defined the primary keys you can now implement the relationships between the related entities by defining the foreign keys on the many side of all 1:M relationships. Implement any 1:1 relationships by joining on primary key to primary key.

You will begin with the relationship between the DRIVER table and the COURIER\_COMPANY table.

The DRIVER table is the many table in the relationship, so it will have the foreign key in it. The relationship will be a weak (or non-identifying) relationship because the foreign key that implements the relationship is not part of the DRIVER table’s primary key. Your model now looks like the following:

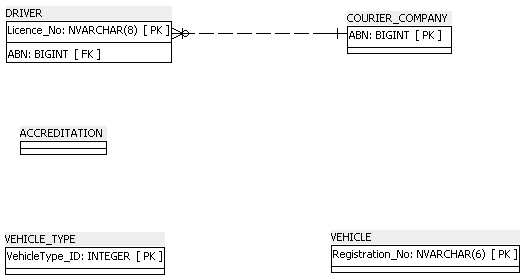


Figure The relationship between DRIVER and COURIER\_COMPANY implemented

Next you will implement the 1:M relationship between COURIER\_COMPANY and VEHICLE. Again the relationship will be a weak (non-identifying) relationship.

In SQL Power Architect when implementing a relationship, select the ‘new non-identifying’ relationship button (on the right side of the screen) and click the ‘one’ table first and the ‘many’ table second.

Your model now looks like the following.

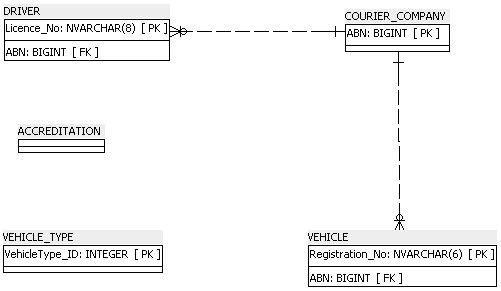


Figure The relationship between COURIER\_COMPANY and VEHICLE implemented

Next, you will implement the 1:M relationship between VEHICLE and VEHICLE\_TYPE. Again, this will be a weak relationship. Your model now looks like the following:



Figure The relationship between VEHICLE and VEHICLE\_TYPE implemented

### Step 7: Implement the relationships to the composite entities

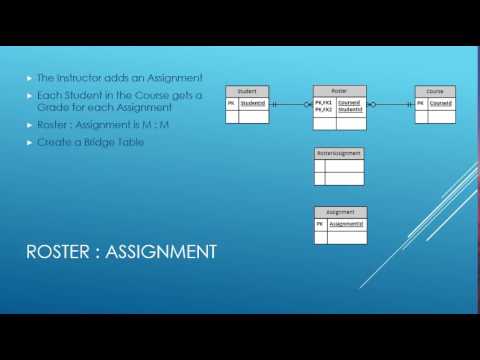
Finally it’s time to implement the two 1:M relationships to the bridging table ACCREDITATION (the composite entity). You will first implement the relationship between ACCREDITATION and VEHICLE\_TYPE.

 Watch

## Video: Connecting bridge tables

Watch this video to learn how to connect a bridge table to another bridge table.

[ERD connecting a bridge table to another bridge table](https://youtu.be/uKd01Ouw_b8) (YouTube 04:40 mins).

[](https://www.youtube.com/watch?v=uKd01Ouw_b8)

https://youtu.be/uKd01Ouw\_b8

In determining the primary key of the bridging table you initially assign the two foreign keys of the tables being bridged as the bridging table’s primary key. This creates a composite primary key. This means that the relationships will be strong or identifying relationships. Your model now looks as follows:

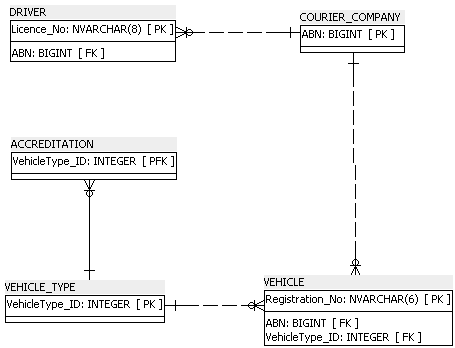


Figure The relationship between ACCREDITATION and VEHICLE\_TYPE implemented

Next you can implement the relationship between DRIVER and ACCREDITATION and it too will be a strong, identifying relationship with the driver’s Licence\_No foreign key being part of the ACCREDITATION table’s primary key. Your model now looks as follows:

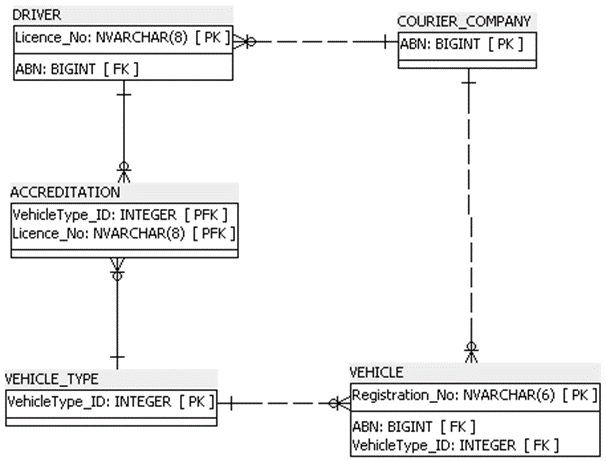


Figure The relationship between DRIVER and ACCREDITATION implemented

### Step 8: Assign attributes

Identify all the other attributes of the entities and their data types and assign them as columns to their respective table in the Crow’s Foot ERD**.**

At this point you can develop a data dictionary to assist in identifying and assigning the additional columns (attributes) required in the database. Knowing which tables your model requires, you can now proceed to identify the columns required by each of the tables and assign the columns to their respective tables.

A data dictionary will assist this process by allowing you to articulate the names of the tables, the names of the columns of each table, the data types required for each column, the format of the data, the range of possible values, whether the column’s data is required or not (nullable), whether each column is a primary key, foreign key or alternate key, and the names of tables that are referenced by the foreign keys.

 Watch

## Video: Data dictionaries

Watch the following video to learn more about data dictionaries:

[5 Minute Metadata - What is a data dictionary?](https://www.youtube.com/watch?v=aOVN0v-HWcQ) (YouTube 04:58 mins)

[](https://www.youtube.com/embed/aOVN0v-HWcQ?feature=oembed)

https://www.youtube.com/watch?v=aOVN0v-HWcQ

Before continuing, identify the following:

* Additional columns (attributes)
* The table each column should be assigned to
* The data type for each column.

The following table depicts the completed data dictionary for the courier company governance database.

## Courier company governance database—Data dictionary

Table The data dictionary for the courier company governance database

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table name | Attribute name | Contents | Data type | Format | Range | Required | FK or PK AK | FK Referenced table |
| DRIVER | Licence\_No | Driver's Licence Number | NVARCHAR(8) | 9999XX or 99999999 | 9999(A-Z)(A-Z) | Y | PK |  |
|  | FirstName | Driver's first name | NVARCHAR(30) | Xxxxxx |  | Y |  |  |
|  | LastName | Driver's last name | NVARCHAR(30) | Xxxxxx |  | Y |  |  |
|  | Phone | Driver's contact phone number | NCHAR(10) | 9999999999 |  |  |  |  |
|  | EmployingCompanyABN | Driver's employer's ABN | BIGINT | 99 999 999 999 |  | Y | FK | COURIER\_ COMPANY |
| COURIER\_COMPANY | ABN | Courier Company's Australian Business Number | BIGINT | 99 999 999 999 |  | Y |  |  |
|  | CompanyName | Courier Company's Name | NVARCHAR(50) | Xxxxxx | 100-999 | Y | AK |  |
|  | CompanyName | Courier Company's Name | NVARCHAR(50) | Xxxxxx | 100-999 | Y | AK |  |
| VEHICLE\_TYPE | VehicleType\_ID | Vehicles Type Identification Number | INTEGER | 999999999 |  | Y | PK |  |
|  | FuelType | Vehicle's Fuel Type | NVARCHAR(30) | Xxxxxx |  | Y |  |  |
|  | FuelCapacity | Vehicle's Fuel Capacity | SMALLINT | 999 |  | Y |  |  |
|  | Range | Vehicle's Range on a full tank of fuel | SMALLINT | 9999 |  | Y |  |  |
|  | Registration\_No | Verhicles RMS Registration Number | NVARCHAR(6) | XXX 999 |  | Y | PK |  |
| VEHICLE | OwningCompanyABN | ABN of the Vehicle's Owning Courier Compy. | BIGINT | 99 999 999 999 |  | Y | FK | COURIER\_ COMPANY |
|  | VehicleType\_ID | Vehicles Type Identification Number | INTEGER | 999999999 |  | Y | FK | VEHICLE TYPE |
|  | DateOfManufacture | Vehicles Build Date | DATE | 99/99/9999 |  | Y |  |  |
|  | DateOfLastMaintenance | Vehicles last maintenance | DATE | 99/99/9999 |  |  |  |  |
|  | VehicleType\_ID | Vehicle's Type Identification Number | INTEGER | 999999999 |  | Y | PK, FK | VEHICLE TYPE |
| ACCREDITATION | Licence\_No | Driver's Licence Number | NVARHCAR(8) | XXX 999 |  | Y | PK, FK | DRIVER |
|  | ExpiryDate | Accreditation Expiry Date | DATE | 99/99/9999 |  | Y | PK |  |
|  | VehicleType\_ID | Vehicles Type Identification Number | INTEGER | 999999999 |  | Y | PK |  |

PK = Primary Key

FK = Foreign Key

AK =Alternate Key

NCHAR = Fixed-length Unicode Character Data NVARCHAR = Variable-length Unicode Character Data

After adding the additional columns to your model, it should now look as follows:

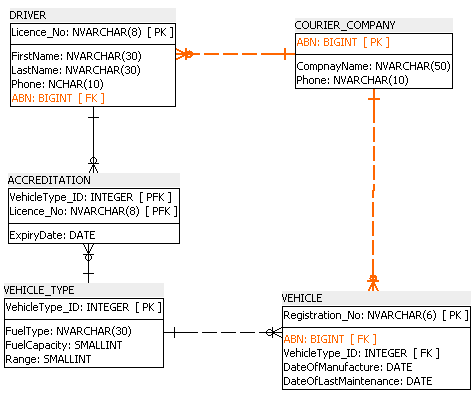


Figure The additional non-key columns (attributes) added to their respective tables (entities)

### Step 9: Reconcile the differences between data

At this stage you can compare the ERD with the data dictionary to ensure that you have implemented the model correctly. You should check that:

* the model is normalised to Fourth Normal Form (4NF)
* primary keys are implemented and are normalised to Fourth Normal Form (4NF)
* foreign keys are implemented and establish the correct relationships between tables
* all other required attributes and their data types are included as specified by the data dictionary.

There are still a few issues that you can deal with to improve your model. The first is the names of the foreign key columns. Specifically ABN in the DRIVER table is not very meaningful, and would be better named ‘EmployingCompanyABN’.

Similarly, ABN in the vehicle table is not very meaningful and would be better named ‘OwningCompanyABN’. Implement the changes to the foreign key columns in DRIVER and VEHICLE.

Your model should now look as follows:

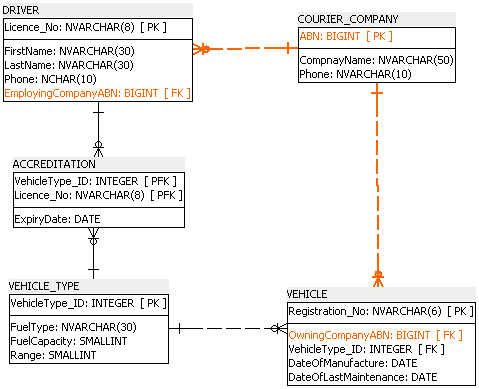


Figure The foreign key columns in DRIVER and VEHICLE given more meaningful names

The second issue that you should consider is the nature of the CompanyName attribute in the COURIER\_COMPANY table. The business rules indicate that ‘*no two companies have the same name*.’ Which effectively means that values in CompanyName are unique.

You can use this to your advantage to improve the performance of the database. Since you are most likely going to use the CompanyName to search the COURIER\_COMPANY table, and that table will rarely be modified, you should define the CompanyName column as an alternate key. You can do this by defining a unique (non-clustered) index on the CompanyName column.

To do this in SQL Power Architect:

1. Right mouse click on the COURIER\_COMPANY table and select ‘New Index…’.
2. Click to select the ‘Unique’ checkbox.
3. Then click to select the CompanyName column below.
4. Click OK.

See the following image.

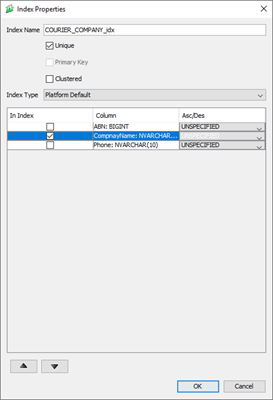


Figure Defining a Unique non-clustered index on the CompanyName column

Your model should now look as follows:

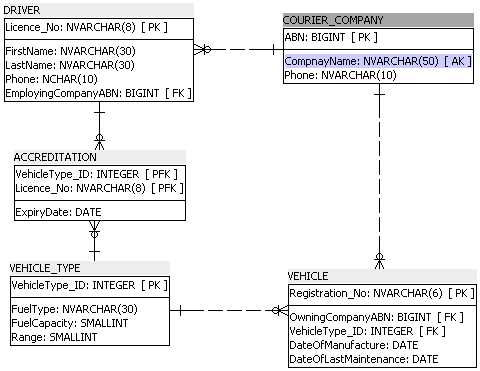


Figure CompanyName column defined as an alternate key by applying a unique non-clustered index to it

Finally, there is one more issue that you must consider in your model. That is the primary key of the ACCREDITATION table, which is the bridging table. According to the business rules, DRIVERS must be ACCREDITED to drive any particular vehicle type.

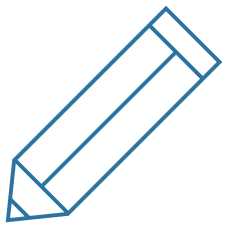
You are recording this fact in the ACCREDITATION table by combining the driver’s licence with the vehicle type as the primary key of the ACCREDITATION table.

However, accreditations expire and the drivers must be re-accredited every six months. This means that you would want to record the fact that a driver is accredited to drive a particular vehicle type more than once.

The difference between these two facts is the expiry date. To overcome this limitation you can add the ExpiryDate to the primary key of the ACCREDITATION table. Implement this change. Your model is now complete and should look as follows:



Figure ExpiryDate added to the primary key of the ACCREDITATION table

 Practice activity

## Activity 3.3: Little Athletics scenario: Create the ERD

Refer to the Little Athletics scenario in [Appendix 1](#_Appendix_1:_Little) for this activity.

Produce a Crow’s Foot Entity Relationship Diagram (ERD) and a data dictionary for the Little Athletics scenario. Discuss your diagrams with your class.

# Validating the data model

It’s important to validate the data model with the client before implementing it. Validation is the process of confirming that the data model meets the specified requirements and needs of the stakeholders. At this stage you can resolve any issues with the design, or the client may have further suggestions. You will then need to obtain final approval from the client before starting on building the database.

Validation can be internal or external:

* Internal validation assumes that the stakeholder’s goals were correctly expressed in the definition of the requirements. If the data model meets the requirements specification, then it is said to be validated internally.
* External validation occurs by asking the stakeholders if the system meets their needs. In most cases external validation is performed by user acceptance testing. Once all of the users or stakeholders have indicated that the system meets their needs, external validation has been achieved.

Topic 4: Creating a database

­

# Overview

Now you’ve arrived at the point where it’s time to actually implement the database that you’ve so painstakingly modelled.

In this topic you will learn about:

* various relational database platforms available and the development tools that you can use to write the code to create and manipulate the database
* the most dominant and persistent language used in information technology for querying databases, SQL, including its terminology and the data definition language (DDL) elements of SQL that you will use to create a relational database
* how to write the SQL code to create the database, its tables, the constraints in the database and how to insert data into the tables.

The activities throughout this resource will assist you in your learning. These activities do not form a part of your final assessment however they will contribute to your understanding of the topic area.

This topic takes a more practical approach to your learning and it is important that you write all the code included in this topic to create the Courier Company Governance System database. It is most important that you practise writing SQL code if you want to be a programmer. Do not just copy and paste the code.

Most importantly, take the time to implement (build) the Little Athletics database in Activity 4.6: Little Athletics scenario: Create the database. This will give you an opportunity to practise what you have been learning and to assess the accuracy of the Little Athletics database data model that you have designed.

Database platforms and tools

There is a myriad of RDBMS available for the development and hosting of relational databases. Most, if not all, of these platforms provide their own environments or tools for building SQL queries.

The following table lists some of the more popular relational database environments and their associated tools for building queries.

Table Overview of environments/tools

|  |  |
| --- | --- |
| Platform | Environment/tool |
| Microsoft SQL Server | SQL Server Management Studio (SSMS) |
| Microsoft Access | Microsoft Access - Query Builder or Query Editor |
| MySQL | phpMyAdmin or MySQL Workbench |
| Oracle Database | Oracle SQL Developer |
| IBM Db2 | IBM Data Studio |
| PostgreSQL | pgAdmin |
| SQLite | SQLiteStudio |

Most RDBMS platforms attempt to comply with the International Standards Organisation (ISO) standard (ISO/IEC 9075-2:2016(en) for Structured Query Language (SQL)).

However, not all platforms adhere strictly to the prescribed standards and there will usually be minor differences between one implementation of SQL and the next. Microsoft’s SQL Server, for example, uses a ‘Microsoft version’ of SQL which they’ve named Transact-SQL (or T-SQL for short).

One of T-SQL’s peculiarities is that Microsoft has made the use of the semi-colon at the end of statements optional. Not such a big deal, since it doesn’t take too much effort to always include the semi-colon at the end of statements in any SQL that you write if you think you might run that code on a platform other than Microsoft’s SQL Server.

The point is, that the differences between one implementation of SQL and the next are usually minor and code that runs successfully on one platform, but not on another will usually require only minor changes to make it compatible with the new platform. That said, you will attempt to implement the courier governance database using standard SQL, which is platform-independent, however, you cannot guarantee that the code will execute successfully on every platform without minor changes.

The most appropriate environment for SQL Server development is SQL Server Management Studio (SSMS). In this workbook you will use Microsoft SQL Server and SQL Server Management Studio (SSMS) 2017. In this version of the development environment Microsoft have maintained the use of the database-diagramming functionality, which has been dropped from subsequent versions of SSMS.

# Query-related terminologies

The following table lists some common terminologies relating to queries, which are useful to know.

Table Query-related terminologies

|  |  |
| --- | --- |
| Terminology | Meaning |
| Ad hoc query | A question requiring information from the database that is formulated on the spur-of-the-moment. |
| Aggregate function | An SQL function used to group values from multiple rows into a single (aggregated) row with a value of some significant meaning, such as the average AVG() of the values of the grouped rows. Aggregate functions include average AVG, maximum MAX(), minimum MIN(), COUNT() and SUM(). |
| Alias | A name given to a column or table in an SQL query that is not its original name. |
| AND | An SQL logical operator used to indicate that both operands either side of it must be true. |
| ANSI | American National Standards Institute – The body governing the standardisation of SQL. |
| AVG() | An aggregate function of SQL that derives the average of a column or domain of values within a column. |
| BETWEEN | An SQL logical operator used to determine whether a value is ‘between’ a range of values, inclusive of those values. |
| COMMIT | An SQL command that commits data changes to permanent storage in the database. |
| CONSTRAINT | A limitation on data enforced by the RDBMS. There are various forms of constraints including Primary Key, Foreign Key, Check, NOT NULL, Unique, Default and Index. |
| COUNT | An aggregate function of SQL used to return the number of records for a given column containing values (not nulls). |
| CREATE … | An SQL command used for creating databases and tables. |
| Data Definition Language (DDL) | The elements of SQL that are used to define the Database structure. |
| Data Manipulation Language (DML) | The elements of SQL that are used to SELECT, INSERT, UPDATE and DELETE data. |
| DELETE | An SQL command that deletes rows from a table according to specified criteria. |
| Derived attribute | An attribute that does not physically exist in a table but is derived by processing the values of other attributes (columns) and system values. e.g. Age can be derived from CurrentDate − DateOfBirth. |
| DISTINCT | An SQL clause used to remove duplicated values from a query result set. |
| DROP | An SQL command that deletes database objects. |
| EXISTS | A Boolean operator that returns True if a subquery returns any rows. |
| GROUP BY | An SQL clause used in conjunction with aggregate functions to group values into ‘summary’ rows for aggregation. |
| HAVING | An SQL clause used to filter aggregated columns. |
| IN | An SQL logical operator to determine whether a value exists within a list of other values. |
| INNER JOIN | An SQL query operation that retrieves records from two tables, where the values of the Primary Key of one table equal the values of the Foreign Key of the other table. |
| Inner query | A query nested inside another query, often referred to as a subquery. |
| INSERT | An SQL command used to insert a new row of data into a table. |
| IS NULL | An SQL logical operator used to check if a field contains a value. |
| LEFT OUTER JOIN | An SQL query operation that retrieves all the records from the table defined on the left side of the join operation and only the matching records from the table on the right side of the operation. |
| LIKE | An SQL logical operator used to find records, which can use a pattern match and wildcard characters to check if a pattern of characters exists within a longer string of characters. |
| MAX() | An SQL aggregate function that returns the maximum value of a given column. |
| MIN() | An SQL aggregate function that returns the minimum value of a given column. |
| NOT | An SQL logical operator used to reverse the value of a given predicate. |
| NULL | A special SQL marker used to indicate the state of an attribute as ‘not containing a value’. |
| OR | An SQL logical operator used to indicate that only one of the operands either side of it must be true. |
| ORDER BY | An SQL clause used for specifying the order of a SELECT query’s result set as either ascending or descending. |
| OUTER JOIN | An SQL join operation that retrieves all records from both tables being joined, regardless of a match or not. |
| Predicate | An expression used in the search condition of WHERE clauses, HAVING clauses and JOIN conditions of FROM clauses, which evaluates to TRUE, FALSE, or UNKNOWN. |
| Query | An SQL statement requesting data or performing a task on the database. |
| Result set | The data row returned by a query. |
| Record | Synonymous with a single row of a table. |
| RIGHT OUTER JOIN | An SQL query operation that retrieves all the records from the table defined on the RIGHT side of the join operation and only the matching records from the table on the left side of the operation. |
| SELECT | An SQL command that retrieves the data from a specified list of columns. |
| SUM() | An SQL aggregate function that sums the values of a given column. |
| UNION | An SQL clause that combines the results of two queries into a single result set. |
| UPDATE | An SQL command used to change the values of existing data in one or more rows of a table. |

 Watch

## Video: Connecting to SQL server using SSMS

Watch this video to learn how to connect to a SQL server database server using SQL Server Management Studio (SSMS).

[Connecting to SQL server using SSMS—Part 1](https://youtu.be/ZNObiptSMSI) (YouTube 09:40 mins).

[](https://www.youtube.com/watch?v=ZNObiptSMSI)

https://youtu.be/ZNObiptSMSI

# Creating a database and tables

There are three ways that you can create a new database in most development environments.

1. Use a case tool, such as SQL Power Architect, to generate (forward engineer) the SQL to create the database, tables, constraints, indexes, etc. from a predefined model.
2. Use the graphical user interface (GUI) elements of a development environment such as SSMS to select and ‘click’ the various menu options that allow you to create the database, tables, constraints, and indexes, etc.
3. Use SQL in a script to create the database, tables, constraints, indexes etc. and execute the script file on a database server.

## Create database syntax

The following describes the syntax for the most basic CREATE DATABASE statement. Elements enclosed in square brackets are optional to the statement.

CREATE DATABASE database\_name

[ CONTAINMENT = { NONE | PARTIAL } ]

[ ON

[ PRIMARY ] <filespec> [ ,...n ]

[ , <filegroup> [ ,...n ] ]

[ LOG ON <filespec> [ ,...n ] ]

]

[ COLLATE collation\_name ]

[ WITH <option> [,...n ] ]

[;]

 Watch

## Video: Creating, altering and dropping a database

Watch the video tutorial creating, altering and dropping a database—part two for details on how to use GUI elements and how to write SQL to create a database using SSMS.

[Creating, altering and dropping a database](https://youtu.be/TuxuHHacIWU) (YouTube 15:16 mins)

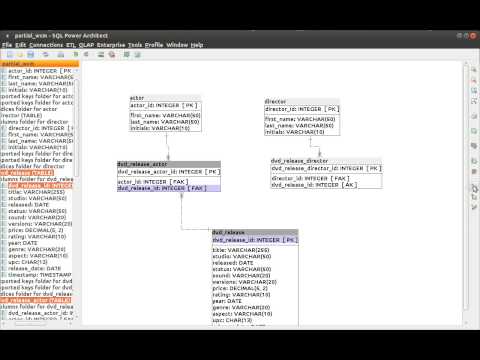
[](https://www.youtube.com/watch?v=TuxuHHacIWU)

https://youtu.be/TuxuHHacIWU

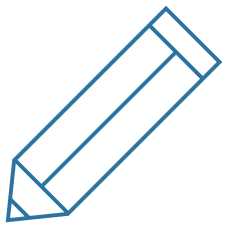
## Video: Apply forward engineering in SQL Power Architect

Watch this video to learn some hints on how to generate the SQL from a data model.

[Apply forward engineering in SQL Power Architect](https://youtu.be/OU9oFIayX2c) (YouTube 01:46 mins).

[](https://www.youtube.com/watch?v=OU9oFIayX2c)

https://youtu.be/OU9oFIayX2c

 Practice activity

## Activity 4.1: Create a database using DDL

For this activity you will use SSMS to write the SQL script/code to create the Courier Company Governance System database.

Throughout this and the following activities you will write SQL code in SSMS to create a database, its tables and its data. Since you are learning how to write SQL to create a database, you’re likely to make errors in your script, which you will have to correct, then re-execute your script to re-create the database.

If you have executed your script at least once, it is likely that the database and some of the tables already exist on the server, so any subsequent execution of your script will cause an error similar to the following:

Msg 1801, Level 16, State 3, Line 9

Database 'CourierCompanyGovernance' already exists. Choose a different database name.

To circumvent this problem, you will include code to delete any database objects that may already exist, before you re-create the database each time you execute your script.

Let’s begin. In SSMS click the ‘New Query’ button to open a new query editor tab. This is where you’ll write your SQL to create the database.

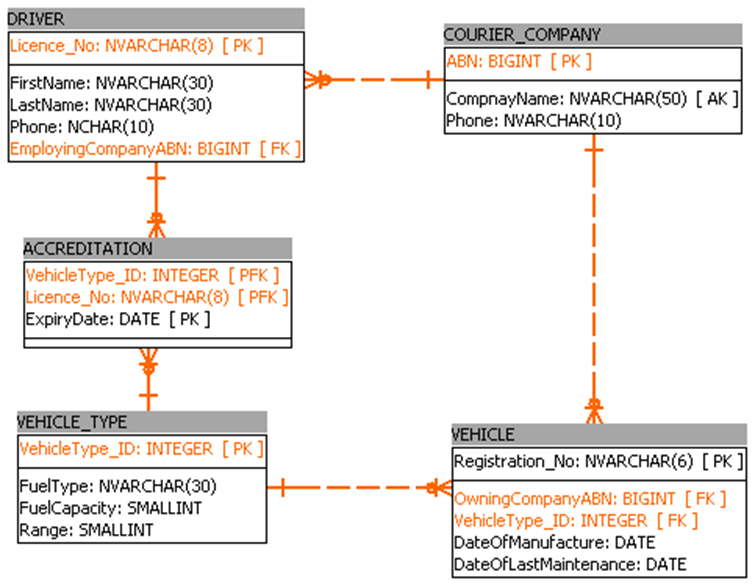


Figure The Courier Company Governance System ERD (Data Model)

You will use Figure 32 above as a guide to writing the SQL code to create the database.

In the SSMS editor type the following code, including the comments but ignoring the line numbers to the left. Replace the comment text for author and date.

1 -- Two dashes denote that what follows is a single line comment.

2 -- Comments are ignored by the query parser.

3 /\* A forward slash followed by an asterisk denotes the beginning of a

4 block comment. This is the traditional C style comment. The end of a block

5 comment is denoted by an asterisk followed by a forward slash.

6 \*/

7 -- ===========================================================================

8 -- This script creates the Courier Company Governance Database

9 -- Author: Your name

10 -- Date: today’s date

11 -- Version: 1.0

12 -- ===========================================================================

13 USE master;

14 GO

15 IF EXISTS ( SELECT name

16 FROM master.dbo.sysdatabases

17 WHERE name = N'CourierCompanyGovernance'

18 )

19 DROP DATABASE CourierCompanyGovernance;

20 GO

21 CREATE DATABASE CourierCompanyGovernance;

22 GO  
23 USE CourierCompanyGovernance;  
24 GO

Save the script file with an appropriate name (*CreateCourierCompanyGovernanceDB.sql*) in a location where you’ll be able to retrieve it later.

An explanation of the code is as follows:

* Lines one to six are comments about comments (coloured green). SQL keywords are coloured blue.
* Lines 7 to 12 are comments documenting the purpose of the script, it’s author, the date that it was created and the version.
* Line 13 connects the query editor to the MASTER database.
* The MASTER database contains all the metadata for the user databases on the server. Line 14 is a GO statement that tells the server to execute all statements above it that have not yet been executed so that their action is committed to the server.
* Lines 15 to 22 check the sysdatabases table of the MASTER database for a database named **‘CourierCompanyGovernance’** and if it EXISTS it is DROPped (deleted) and re-created.
* Line 21 actually creates the database.
* Line 23 instructs the server to connect the text editor to (use) the **CourierCompanyGovernance** database so that all the following statements are executed on that database.

You must transcribe the script accurately. The majority of errors made by people are transcription errors.

Double-check what you have typed (character by character) before you execute your script. If you believe it is correct you can execute the script by clicking the ‘Execute’ button. The courier company governance database now exists on the server.

In SSMS, select the ‘Databases’ node in the ‘Object Explorer’ and refresh the node. Expand the ‘Databases’ node in the ‘Object Explorer’. You should now see the ‘***CourierCompanyGovernance***’ database listed.

However, it does not contain any tables, or any data. You can now add more code to the script to create the tables.

## Create table syntax

The following describes some of the options of the syntax for the basic CREATE TABLE statement. Elements enclosed in square brackets are optional to the statement.

-- Simple CREATE TABLE Syntax (common if not using options)

CREATE TABLE

{ database\_name.schema\_name.table\_name. | schema\_name.table\_name | table\_name }

( { <column\_definition> } [ ,...n ] )

[ ; ]

<column\_definition> ::=

column\_name <data\_type>

<column\_constraint> ::=

[ CONSTRAINT constraint\_name ]

{ { PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

[

WITH FILLFACTOR = fillfactor

| WITH ( < index\_option > [ , ...n ] )

]

[ ON { partition\_scheme\_name ( partition\_column\_name )

| filegroup | "default" } ]

| [ FOREIGN KEY ]

REFERENCES [ schema\_name . ] referenced\_table\_name [ ( ref\_column ) ]

[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ NOT FOR REPLICATION ]

| CHECK [ NOT FOR REPLICATION ] ( logical\_expression )

}

While creating each of the tables ensure that you:

* define the primary key and foreign key constraints
* define any unique indexes to implement the alternate keys.

 Watch

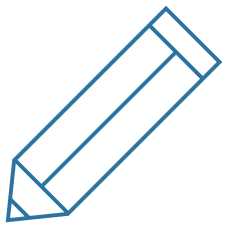
## Video: Creating and working with tables

Watch the video tutorial creating and working with tables—part three for details on how to create tables and define primary and foreign key constraints on the tables.

[Creating and working with tables](https://youtu.be/JLeaM8pK8dE) (YouTube 20:07 mins).

[](https://www.youtube.com/watch?v=JLeaM8pK8dE)

https://youtu.be/JLeaM8pK8dE

 Practice activity

## Activity 4.2: Create tables using DDL

The first table that you will create is the DRIVER table. The CREATE TABLE statement allows you to define the name of the table, the names of each of the columns in the table, the data types and nullability of the columns, the primary key for the table and any indexes that might exist in the table.

You could also define any other constraints that are in the table, such as a foreign key or check constraints. However, you will add these constraints later using the ALTER TABLE statement, after all the tables have been created.

Add the following CREATE TABLE statement to the end of your script to create the DRIVER table:

CREATE TABLE DRIVER (

Licence\_No NVARCHAR(8),

FirstName NVARCHAR(30) NOT NULL,

LastName NVARCHAR(30) NOT NULL,

Phone NCHAR(10) NOT NULL,

EmployingCompanyABN BIGINT NOT NULL,

CONSTRAINT DRIVER\_pk PRIMARY KEY (Licence\_No)

);

GO

Save the script file. Your script should now look as follows.

1 -- Two dashes denote that what follows is a single line comment.

2 -- Comments are ignored by the query parser.

3 /\* A forward slash followed by an asterisk denotes the beginning of a

4 block comment. This is the traditional C style comment. The end of a block

5 comment is denoted by an asterisk followed by a forward slash

6 \*/

7 -- ===========================================================================

8 -- This script creates the Courier Company Governance Database

9 -- Author: Your name

10 -- Date: today’s date

11 -- Version: 1.0

12 -- ===========================================================================

13 USE master;

14 GO

15 IF EXISTS ( SELECT name

16 FROM master.dbo.sysdatabases

17 WHERE name = N'CourierCompanyGovernance'

18 )

19 DROP DATABASE CourierCompanyGovernance;

20 GO

21 CREATE DATABASE CourierCompanyGovernance;

22 GO

23 USE CourierCompanyGovernance;

24 GO

25 CREATE TABLE DRIVER (

26 Licence\_No NVARCHAR(8),

27 FirstName NVARCHAR(30) NOT NULL,

28 LastName NVARCHAR(30) NOT NULL,

29 Phone NCHAR(10) NOT NULL,

30 EmployingCompanyABN BIGINT NOT NULL,

31 CONSTRAINT DRIVER\_pk PRIMARY KEY (Licence\_No)

32 );

33 GO

* Lines 25 to 32 create the DRIVER table, and its constituent columns. The CONSTRAINT defining the primary key (line 31) imposes limitations (constraints) on the Licence\_No column, which require it to be unique and not null.

The metadata name for the primary key object is DRIVER\_pk and is used by the server to identify the primary key object for this table, however, it is not the primary key. The primary key of the table is the Licence\_No column.

Now, you can continue and create the COURIER\_COMPANY table. Add the following CREATE TABLE statement to the end of your script to create the COURIER\_COMPANY table.

CREATE TABLE COURIER\_COMPANY (

ABN BIGINT,

CompanyName NVARCHAR(50) NOT NULL,

Phone NVARCHAR(10) NOT NULL,

CONSTRAINT COURIER\_COMPANY\_pk PRIMARY KEY (ABN)

)

CREATE UNIQUE NONCLUSTERED INDEX COURIER\_COMPANY\_idx ON COURIER\_COMPANY(CompanyName);

GO

Save the script file. Your script should now look as follows:

1 -- Two dashes denote that what follows is a single line comment.

2 -- Comments are ignored by the query parser.

3 /\* A forward slash followed by an asterisk denotes the beginning of a

4 block comment. This is the traditional C style comment. The end of a block

5 comment is denoted by an asterisk followed by a forward slash

6 \*/

7 -- ===========================================================================

8 -- This script creates the Courier Company Governance Database

9 -- Author: Your name

10 -- Date: today’s date

11 -- Version: 1.0

12 -- ===========================================================================

13 USE master;

14 GO

15 IF EXISTS ( SELECT name

16 FROM master.dbo.sysdatabases

17 WHERE name = N'CourierCompanyGovernance'

18 )

19 DROP DATABASE CourierCompanyGovernance;

20 GO

21 CREATE DATABASE CourierCompanyGovernance;

22 GO

23 USE CourierCompanyGovernance;

24 GO

25 CREATE TABLE DRIVER (

26 Licence\_No NVARCHAR(8),

27 FirstName NVARCHAR(30) NOT NULL,

28 LastName NVARCHAR(30) NOT NULL,

29 Phone NCHAR(10) NOT NULL,

30 EmployingCompanyABN BIGINT NOT NULL,

31 CONSTRAINT DRIVER\_pk PRIMARY KEY (Licence\_No)

32 );

33 GO

34 CREATE TABLE COURIER\_COMPANY (

35 ABN BIGINT,

36 CompanyName NVARCHAR(50) NOT NULL,

37 Phone NVARCHAR(10) NOT NULL,

38 CONSTRAINT COURIER\_COMPANY\_pk PRIMARY KEY (ABN)

39 )

40 CREATE UNIQUE NONCLUSTERED INDEX COURIER\_COMPANY\_idx ON COURIER\_COMPANY(CompanyName);

41 GO

You may recall that the candidate keys for the COURIER\_COMPANY table are the company’s ABN and name. This is because they are both unique and every courier company that operates must have both an ABN and a name (so they can’t be null).

The decision was made to use the ABN as the primary key of the COURIER\_COMPANY table, however, the company name will often be used to search this table, so it has been designated an alternate key.

Alternate keys are implemented by creating a unique, usually non-clustered, index on the alternate key column.

* Line 40 indicates the definition of the index.

Go ahead and create the remaining three tables, VEHICLE, VEHICLE\_TYPE and ACCREDITATION by adding the following CREATE TABLE statements to the end of your script file:

CREATE TABLE VEHICLE (

Registration\_No NVARCHAR(6),

OwningCompanyABN BIGINT NOT NULL,

VehicleType\_ID INT NOT NULL,

DateOfManufacture DATE NOT NULL,

DateOfLastMaintenance DATE NOT NULL,

CONSTRAINT VEHICLE\_pk PRIMARY KEY (Registration\_No)

);

GO

CREATE TABLE VEHICLE\_TYPE (

VehicleType\_ID INT,

FuelType NVARCHAR(30) NOT NULL,

FuelCapacity SMALLINT NOT NULL,

[Range] SMALLINT NOT NULL,

CONSTRAINT VEHICLE\_TYPE\_pk PRIMARY KEY (VehicleType\_ID)

);

GO

CREATE TABLE ACCREDITATION (

VehicleType\_ID INT,

Licence\_No NVARCHAR(8),

ExpiryDate DATE,

CONSTRAINT ACCREDITATION\_pk PRIMARY KEY (VehicleType\_ID, Licence\_No, ExpiryDate)

);

GO

Save the script file. Your script should now look as follows:

1 -- Two dashes denote that what follows is a single line comment.

2 -- Comments are ignored by the query parser.

3 /\* A forward slash followed by an asterisk denotes the beginning of a

4 block comment. This is the traditional C style comment. The end of a block

5 comment is denoted by an asterisk followed by a forward slash

6 \*/

7 -- ===========================================================================

8 -- This script creates the Courier Company Governance Database

9 -- Author: Your name

10 -- Date: today’s date

11 -- Version: 1.0

12 -- ===========================================================================

13 USE master;

14 GO

15 IF EXISTS ( SELECT name

16 FROM master.dbo.sysdatabases

17 WHERE name = N'CourierCompanyGovernance'

18 )

19 DROP DATABASE CourierCompanyGovernance;

20 GO

21 CREATE DATABASE CourierCompanyGovernance;

22 GO

23 USE CourierCompanyGovernance;

24 GO

25 CREATE TABLE DRIVER (

26 Licence\_No NVARCHAR(8),

27 FirstName NVARCHAR(30) NOT NULL,

28 LastName NVARCHAR(30) NOT NULL,

29 Phone NCHAR(10) NOT NULL,

30 EmployingCompanyABN BIGINT NOT NULL,

31 CONSTRAINT DRIVER\_pk PRIMARY KEY (Licence\_No)

32 );

33 GO

34 CREATE TABLE COURIER\_COMPANY (

35 ABN BIGINT,

36 CompanyName NVARCHAR(50) NOT NULL,

37 Phone NVARCHAR(10) NOT NULL,

38 CONSTRAINT COURIER\_COMPANY\_pk PRIMARY KEY (ABN)

39 )

40 CREATE UNIQUE NONCLUSTERED INDEX COURIER\_COMPANY\_idx ON COURIER\_COMPANY(CompanyName);

41 GO

42 CREATE TABLE VEHICLE (

43 Registration\_No NVARCHAR(6),

44 OwningCompanyABN BIGINT NOT NULL,

45 VehicleType\_ID INT NOT NULL,

46 DateOfManufacture DATE NOT NULL,

47 DateOfLastMaintenance DATE NOT NULL,

48 CONSTRAINT VEHICLE\_pk PRIMARY KEY (Registration\_No)

49 );

50 GO

51 CREATE TABLE VEHICLE\_TYPE (

52 VehicleType\_ID INT,

53 FuelType NVARCHAR(30) NOT NULL,

54 FuelCapacity SMALLINT NOT NULL,

55 [Range] SMALLINT NOT NULL,

56 CONSTRAINT VEHICLE\_TYPE\_pk PRIMARY KEY (VehicleType\_ID)

57 );

58 GO

59 CREATE TABLE ACCREDITATION (

60 VehicleType\_ID INT,

62 Licence\_No NVARCHAR(8),

63 ExpiryDate DATE,

64 CONSTRAINT ACCREDITATION\_pk PRIMARY KEY (VehicleType\_ID, Licence\_No, ExpiryDate)

65);

66 GO

* Lines 42 to 66 create the columns and the primary keys for the VEHICLE, VEHICLE\_TYPE and ACCREDITATION tables.

The next step is to create the foreign key constraints to implement the relationships between the tables.

## Create the constraints

The following describes some of the more basic options of the syntax for the ALTER TABLE statement. With the ALTER TABLE statement you can do things like change the name of a column, its data type and other properties. You can also ADD a new column, or a CONSTRAINT, or you can DROP (delete) constraints and columns from the table.

ALTER TABLE { database\_name.schema\_name.table\_name | schema\_name.table\_name | table\_name }

{

ALTER COLUMN column\_name

{

[ type\_schema\_name. ] type\_name  
 ...

| ADD

{

<table\_constraint>  
 ...  
[ CONSTRAINT constraint\_name ]  
 ...  
 FOREIGN KEY  
 ( column [ ,...n ] )  
 REFERENCES referenced\_table\_name [ ( ref\_column [ ,...n ] ) ]

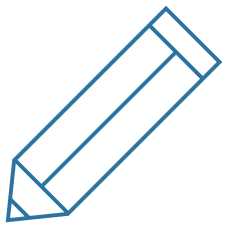
The foreign key constraints in your data model are the mechanism by which you implement the one-to-many relationships between the tables in your database. A foreign key column is placed in the many table of two related tables. Its values reference the values of the primary key column in the table on the one side of the relationship.

So, the limitation or restriction (constraint) that the database server applies to the foreign key column is that no values can be inserted into the foreign key column if a corresponding value does not exist in the referenced primary key.

An error will occur if this is attempted. Foreign key constraints are how the database enforces the concept of referential integrity. That is:

* each foreign key value must reference an existing and valid value in the primary key of the referenced table, or the foreign key value can be null.

So, using the data model for your Courier Company Governance System depicted in Figure 32, you can use the many sides of each relationship lineCrows foot line diagram as discussed in using the crows foot ERD. to identify which table has the foreign key column in it.

 Practice activity

## Activity 4.3: Add constraints using DDL

Now add the following foreign key constraints to the end of your script:

ALTER TABLE VEHICLE ADD CONSTRAINT VEHICLE\_TYPE\_VEHICLE\_fk FOREIGN KEY (VehicleType\_ID) REFERENCES VEHICLE\_TYPE (VehicleType\_ID);

GO

ALTER TABLE DRIVER ADD CONSTRAINT COURIER\_COMPANY\_DRIVER\_fk FOREIGN KEY (EmployingCompanyABN) REFERENCES COURIER\_COMPANY (ABN);

GO

ALTER TABLE VEHICLE ADD CONSTRAINT COURIER\_COMPANY\_VEHICLE\_fk FOREIGN KEY (OwningCompanyABN) REFERENCES COURIER\_COMPANY (ABN);

GO

ALTER TABLE ACCREDITATION ADD CONSTRAINT DRIVER\_ACCREDITATION\_fk FOREIGN KEY (Licence\_No) REFERENCES DRIVER (Licence\_No);

GO

ALTER TABLE ACCREDITATION ADD CONSTRAINT VEHICLE\_TYPE\_ACCREDITATION\_fk FOREIGN KEY (VehicleType\_ID) REFERENCES VEHICLE\_TYPE (VehicleType\_ID);

GO

Save the script file. Your script should now look as follows:

1 -- Two dashes denote that what follows is a single line comment.

2 -- Comments are ignored by the query parser.

3 /\* A forward slash followed by an asterisk denotes the beginning of a

4 block comment. This is the traditional C style comment. The end of a block

5 comment is denoted by an asterisk followed by a forward slash

6 \*/

7 -- ===========================================================================

8 -- This script creates the Courier Company Governance Database

9 -- Author: Your name

10 -- Date: today’s date

11 -- Version: 1.0

12 -- ===========================================================================

13 USE master;

14 GO

15 IF EXISTS ( SELECT name

16 FROM master.dbo.sysdatabases

17 WHERE name = N'CourierCompanyGovernance'

18 )

19 DROP DATABASE CourierCompanyGovernance;

20 GO

21 CREATE DATABASE CourierCompanyGovernance;

22 GO

23 USE CourierCompanyGovernance;

24 GO

25 CREATE TABLE DRIVER (

26 Licence\_No NVARCHAR(8),

27 FirstName NVARCHAR(30) NOT NULL,

28 LastName NVARCHAR(30) NOT NULL,

29 Phone NCHAR(10) NOT NULL,

30 EmployingCompanyABN BIGINT NOT NULL,

31 CONSTRAINT DRIVER\_pk PRIMARY KEY (Licence\_No)

32 );

33 GO

34 CREATE TABLE COURIER\_COMPANY (

35 ABN BIGINT,

36 CompanyName NVARCHAR(50) NOT NULL,

37 Phone NVARCHAR(10) NOT NULL,

38 CONSTRAINT COURIER\_COMPANY\_pk PRIMARY KEY (ABN)

39 )

40 CREATE UNIQUE NONCLUSTERED INDEX COURIER\_COMPANY\_idx ON 41 COURIER\_COMPANY(CompanyName);

41 GO

42 CREATE TABLE VEHICLE (

43 Registration\_No NVARCHAR(6),

44 OwningCompanyABN BIGINT NOT NULL,

45 VehicleType\_ID INT NOT NULL,

46 DateOfManufacture DATE NOT NULL,

47 DateOfLastMaintenance DATE NOT NULL,

48 CONSTRAINT VEHICLE\_pk PRIMARY KEY (Registration\_No)

49 );

50 GO

51 CREATE TABLE VEHICLE\_TYPE (

52 VehicleType\_ID INT,

53 FuelType NVARCHAR(30) NOT NULL,

54 FuelCapacity SMALLINT NOT NULL,

55 [Range] SMALLINT NOT NULL,

56 CONSTRAINT VEHICLE\_TYPE\_pk PRIMARY KEY (VehicleType\_ID)

57 );

58 GO

59 CREATE TABLE ACCREDITATION (

60 VehicleType\_ID INT,

61 Licence\_No NVARCHAR(8),

62 ExpiryDate DATE,

63 CONSTRAINT ACCREDITATION\_pk PRIMARY KEY (VehicleType\_ID, Licence\_No,

64 ExpiryDate)

65 );

66 GO

67 ALTER TABLE VEHICLE ADD CONSTRAINT VEHICLE\_VEHICLETYPE\_fk FOREIGN KEY (VehicleType\_ID)

68 REFERENCES VEHICLE\_TYPE (VehicleType\_ID);

69 GO

70 ALTER TABLE DRIVER ADD CONSTRAINT DRIVER\_COURIERCOMPANY\_fk FOREIGN KEY

71 (EmployingCompanyABN) REFERENCES COURIER\_COMPANY (ABN);

72 GO

73 ALTER TABLE VEHICLE ADD CONSTRAINT VEHICLE\_COURIER\_COMPANY\_fk FOREIGN KEY

74 (OwningCompanyABN) REFERENCES COURIER\_COMPANY (ABN);

75 GO

76 ALTER TABLE ACCREDITATION ADD CONSTRAINT ACCREDITATION\_DRIVER\_fk FOREIGN KEY

77 (Licence\_No) REFERENCES DRIVER (Licence\_No);

78 GO

79 ALTER TABLE ACCREDITATION ADD CONSTRAINT ACCREDITATION\_VEHICLETYPE\_fk FOREIGN KEY 80 (VehicleType\_ID) REFERENCES VEHICLE\_TYPE (VehicleType\_ID);

81 GO

* Lines 67 to 80 define the foreign key constraints for the database.

Note that the naming convention used for the names of the foreign key objects (e.g. VEHICLE\_VEHICLETYPE\_fk ) is foreignKeyTable\_referencedTable\_fk.

Run the script to create the database and tables.

# Testing and debugging the database

Often database developers and programmers underestimate the importance of testing during the implementation stage of database development.

A test plan is essential for ensuring that you have included tests for all scenarios. This will generally include the item being tested, the test data used and the expected result. For example, you may want to test whether a *LastName* field is of a reasonable length.

## Test data

Poorly designed test data may not enable you to discover problems with the database. Test data is used for both:

* *positive testing,* to confirm that aspects of the database perform as expected
* *negative testing,* to test the capacity of the database to handle unexpected and extraordinary inputs.

Test data should be created before testing begins. It can be generated by any combination of the following methods:

1. Manually, by either using the database server client software to enter data directly into the tables, or by writing SQL DDL statements (INSERT statements) to be executed as a script to insert the data into their respective tables.
2. Creating a mass copy of data that is in the production environment (an existing live system that has real data it in) and duplicating that data in the test database environment.
3. Creating a mass copy of data from decommissioned legacy systems and moving the data to the new test database environment.
4. Generating test data using automated tools.

Often the creation of test data from existing databases, whether they’re production databases or legacy databases, will require a migration process where the data must be prepared for the test database environment. This may include ensuring that the test data meets the schema of the test database, particularly its datatypes.

## Types of testing

There are several types of tests that can be performed on a relational database. These include the following:

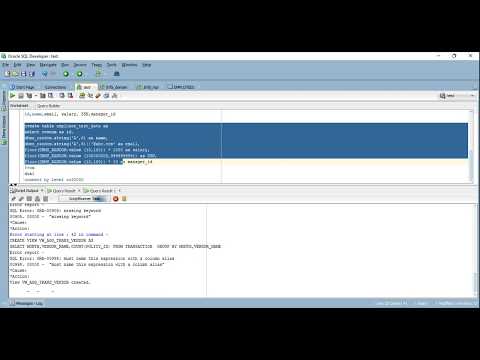
* White box testing is a technique designed to test queries that are known, which have been written to meet the information requirements of the organisation that owns the database.
  + The test data should test as many relationships as possible, including testing aggregation of the data to produce aggregated information.
  + The test data and the tests themselves should also test the creation of new records, particularly in tables related both in 1:M and M:N relationships (using a bridging table). The deletion and updating of data should also be thoroughly tested to ensure that the business rules regarding updates and deletes have been implemented correctly.
  + Negative testing should also be performed using invalid data types, parameter values, joins and various combinations of test data.
* Performance testing should be conducted to ensure that the database provides information at the rate that’s expected by the accessing applications. For example, information produced by a database for a website should be available in the pages of the site within the specified time requirements, otherwise customers will go elsewhere. Performance testing should also be conducted under the anticipated loads that the database should expect to encounter during its normal operational lifetime.
* Security testing should also be conducted to ensure the security of the data. Security testing should encompass the following aspects of data security:
  + Ensure that the confidentiality of the data is maintained.
  + Ensure that the integrity of the data is maintained. This will require checking the database design and the data model to ensure that it meets the requirements of the relational data model and the rules of normalisation.
  + Authentication and authorisation of access to the data. Testing should ensure the people or systems accessing the data are authenticated, i.e. they’re confirmed to be who they say they are. Testing should also confirm that the various roles that have been defined to access the data are authorised to access that data.
* Black box testing should be conducted to test user-stored procedures, functions, triggers and insert, update and delete actions.

 Watch

## Video: Test data generation in SQL

Watch this video to learn how can you generate test data for tables within one minute using only SQL queries.

[Test data generation in SQL within 1 minute](https://youtu.be/Lf8m9lXNPnc) (YouTube 21:25 mins).

[](https://www.youtube.com/watch?v=Lf8m9lXNPnc)

https://youtu.be/Lf8m9lXNPnc

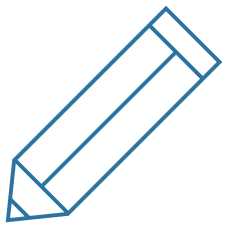
## Video: Creating large tables with random data

Watch this video to learn how to insert a large amount of random data into SQL server tables for performance testing.

[Creating a large table with random data for performance testing Part 61](https://youtu.be/RizVYigF4GI) (YouTube 17:17 mins).

[](https://www.youtube.com/watch?v=RizVYigF4GI)

https://youtu.be/RizVYigF4GI

 Practice activity

## Activity 4.4: Create a test plan

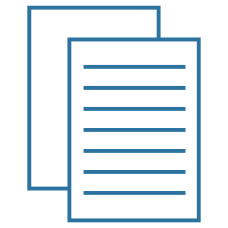
Create a short test plan to test one of the tables that you have created for the courier company governance database. Discuss your plan with other members of your class.

## Debugging and problem solving

As a result of your testing, you may have discovered some flaws or errors in your database design. There are many common design mistakes that can cause errors with your database, including poor planning, incorrect incomplete normalisation, and lack of naming standards, documentation and testing.

However, how do you find a mistake that’s already been made? Using a problem-solving process is a good start. This can include the following steps:

1. Define the problem
2. List all possible solutions
3. Evaluate options
4. Select the best solution
5. Plan to implement your solution
6. Communicate your solution.

 Resources

## Additional resources

Read the article [Ten Common Database Design Mistakes](https://www.red-gate.com/simple-talk/sql/database-administration/ten-common-database-design-mistakes/) to learn more about the types of mistakes you should avoid when designing a database.

Read more about problem-solving techniques at the following websites:

* [Problem solving techniques: Steps and methods](https://credentials.deakin.edu.au/problem-solving-techniques-steps-and-methods/)
* [Methods and Exercises for Effective Problem Solving](https://medium.com/hygger-io/methods-and-exercises-for-effective-problem-solving-3503dfabbd02).

# Populating the tables with data

You are now ready to populate the database with data. There are several ways that this can be done.

## INSERT statement

You can use the INSERT statement (DML) to add records, line by line. This is usually only used with small sets of data as it is time consuming.

The syntax for the INSERT statement is:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

If you’re adding values into every column of the table, you don’t need to specify the column names, however the order of the values must be the same as the order of the columns:

INSERT INTO table\_name

VALUES (value1, value2, value3, ...);

## Import data from a file

When you have large amounts of data, it is more common to import the data from another file, such as a .csv or .txt file.

You can use DML to import the data; the syntax for this will vary depending on the version of SQL that you’re using. For example, SQL Server uses the [BULK INSERT](https://docs.microsoft.com/en-us/sql/t-sql/statements/bulk-insert-transact-sql?view=sql-server-ver15) statement and MySQL uses the [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) statement.

Most DBMS will also have a tool that you can use to import a data file.

 Watch

## Video: Importing data

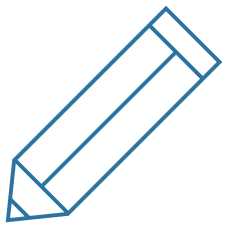
Watch this video from LinkedIn Learning to learn how to import data using the SQL Server Import and Export Wizard.

* [Importing data](https://www.linkedin.com/learning/sql-server-2014-essential-training/importing-data?u=57684225&auth=true) (LinkedIn Learning 04:10 mins).

## Import the results of a query

You can also use queries to shape the data that exists in another database or table so that it matches the structure of the destination database or table.

You can read more about these methods at [How to Import Data into SQL Tables](https://data36.com/how-to-import-data-into-sql-tables/).

 Practice activity

## Activity 4.5: Populate tables

For this activity you will use three different methods to populate the tables. Save the provided .csv files (contained in the *Cl\_Database\_SW\_1of1\_SR1* folder) to a location on your computer.

1. Write and execute the following script to perform a bulk insert of the ACCREDITATION.csv file. Change the local path to the location where you have saved your file.

BULK INSERT ACCREDITATION

FROM ‘c:\ACCREDITATION.csv’

WITH

(

FIELDTERMINATOR = ‘,’,

ROWTERMINATOR = ‘\n’

)

1. Write and execute the following script to insert data into the VEHICLE table. Note you should type at least one of the INSERT statements, then you can copy and paste the rest.

Note the first line of the statements above ‘SET DATEFORMAT YMD;’ tells the server that any dates that are included in the data to be inserted are in the format **Y**ear **M**onth **D**ay.

SET DATEFORMAT YMD;

USE CourierCompanyGovernance;

GO

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('ADB435', 17948738949, 1002, '2018-02-01', '2019-07-04');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('AWQ763', 17948738949, 2003, '2017-07-12', '2019-06-15');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('DSF743', 79834510388, 2004, '2016-09-08', '2019-06-05');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('EGD783', 17948738949, 3003, '2019-09-04', '2019-09-04');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('FHT326', 79834510388, 8001, '2008-12-01', '2019-06-08');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('IOU645', 17948738949, 1003, '2017-10-07', '2019-09-12');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('JHF674', 91876578373, 12002, '2007-01-09', '2019-07-05');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('NGJ946', 79834510388, 3003, '2009-10-23', NULL);

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('POR621', 79834510388, 2004, '2016-09-08', '2019-06-05');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('TRT757', 91876578373, 12001, '2019-04-07', '2019-11-10');

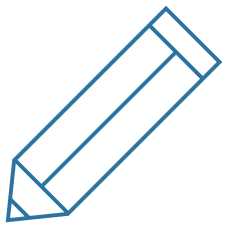
INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('UTY327', 79834510388, 2004, '2016-09-08', '2019-06-05');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('VIV443', 79834510388, 8001, '2008-12-01', '2019-06-09');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('YDF742', 91876578373, 12003, '2003-06-14', '2019-09-11');

INSERT INTO VEHICLE (Registration\_No, OwningCompanyABN, VehicleType\_ID, DateOfManufacture, DateOfLastMaintenance) VALUES ('YRN737', 17948738949, 1002, '2019-04-03', NULL);

1. Use the DBMS to import the remaining COURIER\_COMPANY.csv, DRIVER.csv and VEHICLE\_TYPE.csv files.

 Practice activity

## Activity 4.6: Little Athletics scenario: Create the database

Using the documentation from the activities Activity 1.1: Little Athletics scenario: Create a context diagram, Activity 1.2: Little Athletics scenario: Document the requirements and Activity 3.3: Little Athletics scenario: Create the ERD, write the SQL code to create the Little Athletics database, its tables and its constraints (primary key, foreign key, unique indexes, etc.).

Develop some test data to populate the tables and write the SQL commands to insert the test data into the appropriate tables.

Topic 5: Using SQL

­

# Overview

In this topic you will learn the following:

* Writing SQL code to query databases. You will begin with the most basic queries, or expressions, selecting data from a single table, learning how to specify the columns that you want to list, the rows to be included and the order in which the results are displayed.
* Developing more complex queries selecting data from multiple tables and using logical operators and arithmetic operators to build expressions that will determine the information to be provided by the queries.
* How to use aggregate, formatting and date and time functions to perform more complex operations on the data to extract the desired information.

Again, as in Topic 4, this topic takes a more hands-on practical approach to your learning. It is therefore very important that you write all the code included in this topic to query the Courier Company Governance System database. It is important it is to your development as a programmer that you practise writing the code and that you do not just copy and paste the code.

The activities throughout this resource will assist you in your learning. These activities do not form a part of your final assessment however they will contribute to your understanding of the topic area.

Since you are using SQL as the language for this course, the data source will be a relational database. To learn SQL you will continue to use the Courier Company Governance System database that you have been developing throughout your student workbook.

# Conventions for formatting SQL

SQL is not case sensitive and the commonly observed conventions are as follows:

* SQL keywords, generally coloured blue in SSMS, are written in UPPERCASE.
* Place the individual clauses of a query on a new line, e.g.:

SELECT  
FROM  
WHERE  
GROUP BY  
HAVING  
ORDER BY;

* In SQL server’s version of SQL (Transact SQL or T-SQL) the semicolon is optional, however, you should always include it, just in case you migrate your code to another platform.
* Try to format the layout of your SQL queries so that columns in the SELECT clause, tables in the FROM clause, filtering criteria in the WHERE clause, etc. are placed in a columnar format as follows:

SELECT attribute1,  
 attribute2,  
 attribute3,  
 attribute4  
FROM table1,  
 table2,  
 table3

WHERE table1.PrimaryKey = table2.ForeignKey

AND table2.PrimaryKey = table3.ForeignKey

AND (attribute5 = 'a value' AND attribute6 = 9999)  
OR (attribute7 = 'another value' OR attribute8 = 0000);

Note in the above example the logical operators AND and OR are indented or stacked on top of each other (inside the parentheses).

# Selecting and sorting values

## Displaying all columns

The most commonly written query is one that SELECTS all the columns in the table and displays all the data.

The next query displays all the columns from the DRIVER table. The asterisk (\*) indicates that it must list ALL the columns.

The SELECT clause is where you specify which columns you’d like to return data for.

The FROM clause specifies which table(s) to get the data from. Note that the columns are returned in the same order in which they’re defined in the DRIVER table.

SELECT \*

FROM DRIVER;

Licence\_No FirstName LastName Phone EmployingCompanyABN

---------- ---------- ---------- ---------- ------------

1593PD Donald White 0294765367 91876578373

2765GF John Smith 0297465367 17948738949

28476653 Yu Shoiwheng 0294756456 91876578373

3173BC Mary Dilberry 0247564778 79834510388

43567482 Nina Johansen 0294845748 17948738949

4433JK David Livingston 0294564536 79834510388

57463289 Marium Hashemi 0295847858 91876578373

57463576 Barry Brown 0348473647 79834510388

7465SW Peter Anderton 0392837272 17948738949

75849857 Sharon Branson 0347637748 91876578373

84756748 Andrew Davidson 0376564653 91876578373

## Choosing the columns to be displayed

You can specify the order in which you want the columns listed by specifying the columns in the SELECT clause in the order in which you want them listed, as demonstrated in the next query:

SELECT FirstName,

LastName,

Licence\_No,

EmployingCompanyABN,

Phone

FROM DRIVER;

FirstName LastName Licence\_No EmployingCompanyABN Phone

---------- ---------- ---------- -------------------- ----------

Donald White 1593PD 91876578373 0294765367

John Smith 2765GF 17948738949 0297465367

Yu Shoiwheng 28476653 91876578373 0294756456

Mary Dilberry 3173BC 79834510388 0247564778

Nina Johansen 43567482 17948738949 0294845748

David Livingston 4433JK 79834510388 0294564536

Marium Hashemi 57463289 91876578373 0295847858

Barry Brown 57463576 79834510388 0348473647

Peter Anderton 7465SW 17948738949 0392837272

Sharon Branson 75849857 91876578373 0347637748

Andrew Davidson 84756748 91876578373 0376564653

The next query demonstrates that you don’t have to list all the columns of the table:

SELECT FirstName,

LastName,

Phone

FROM DRIVER;

FirstName LastName Phone

---------- ---------- ----------

Donald White 0294765367

John Smith 0297465367

Yu Shoiwheng 0294756456

Mary Dilberry 0247564778

Nina Johansen 0294845748

David Livingston 0294564536

Marium Hashemi 0295847858

Barry Brown 0348473647

Peter Anderton 0392837272

Sharon Branson 0347637748

Andrew Davidson 0376564653

## Sorting results

The ORDER BY clause is used for specifying the column(s) by which the sort order of the result set is defined. The relational database model stipulates that the order of the records is not important. Physically, the order in which data is written to disk could be random or it could be ordered by the primary key.

So, the order of any SELECT statement might vary each time the query is executed. To avoid this problem, you can use the ORDER BY clause to specify the order in which you want the results listed, as demonstrated in the next query that lists the vehicles from oldest to newest:

SELECT \*

FROM VEHICLE

ORDER BY DateOfManufacture;

Registration\_No OwningCompanyABN VehicleType\_ID DateOfManufacture DateOfLastMaintenance

--------------- -------------------- -------------- ----------------- ---------------------

YDF742 91876578373 12003 2003-06-14 2019-09-11

JHF674 91876578373 12002 2007-01-09 2019-07-05

FHT326 79834510388 8001 2008-12-01 2019-06-08

VIV443 79834510388 8001 2008-12-01 2019-06-09

NGJ946 79834510388 3003 2009-10-23 NULL

POR621 79834510388 2004 2016-09-08 2019-06-05

DSF743 79834510388 2004 2016-09-08 2019-06-05

UTY327 79834510388 2004 2016-09-08 2019-06-05

AWQ763 17948738949 2003 2017-07-12 2019-06-15

IOU645 17948738949 1003 2017-10-07 2019-09-12

ADB435 17948738949 1002 2018-02-01 2019-07-04

YRN737 17948738949 1002 2019-04-03 NULL

TRT757 91876578373 12001 2019-04-07 2019-11-10

EGD783 17948738949 3003 2019-09-04 2019-09-04

Ascending is the default sort direction.

You can also specify more than one column to sort by. For example, you can sort in **descending** order for the fuel capacity; then within the individual groups of fuel capacities, you can sort the range in **ascending** order, as indicated in the next query:

SELECT \*

FROM VEHICLE\_TYPE

ORDER BY FuelCapacity DESC, [Range] ASC;

VehicleType\_ID FuelType FuelCapacity Range

-------------- -------------- ------------ ------

12002 Diesel 1200 3000

12003 Diesel 1200 3500

12001 Diesel 800 2400

8003 Diesel 300 900

8002 Diesel 250 1800

8001 Diesel 210 1500

3002 Diesel 120 1000

3003 Diesel 120 1100

2003 Diesel 120 1200

3001 Diesel 110 900

2002 Diesel 100 800

2001 Diesel 80 600

2005 Hybrid/Diesel 80 1200

2006 Hybrid/Diesel 80 1400

2004 Hybrid/Diesel 50 800

1003 Diesel 50 900

1001 Petrol 40 600

1002 Petrol 40 600

1005 Hybrid/Petrol 30 1500

1004 Hybrid/Petrol 20 1200

If you list the vehicle types from the VEHICLE table, you’ll get a lot of repetition of the VehicleType\_ID, which may be unnecessary. The next query lists all the VehicleType\_IDs in the database, including the duplicates:

SELECT VehicleType\_ID

FROM VEHICLE

ORDER BY VehicleType\_ID;

VehicleType\_ID

--------------

1002

1002

1003

2003

2004

2004

2004

3003

3003

8001

8001

12001

12002

12003

If you want to list the individual vehicle types that you have in the database, rather than the number of each vehicle type, you can use the DISTINCT keyword to eliminate the duplicated values as demonstrated in the next query:

SELECT DISTINCT VehicleType\_ID

FROM VEHICLE

ORDER BY VehicleType\_ID;

VehicleType\_ID

--------------

1002

1003

2003

2004

3003

8001

12001

12002

12003

# Selecting specific values

## Comparing values

To make more complex queries, you will often need to compare values. If the values aren’t of the same data type, this may be difficult. SQL has two useful functions, CAST and CONVERT, that can allow you to change a value’s data type in an expression. These functions are used in some of the expressions below.

 Watch

## Video: CAST() and CONVERT() functions

Watch this video to learn how to convert one data type to another using cast() and convert() functions in SQL Server.

[Cast and Convert functions in SQL Server Part 28](https://youtu.be/8GHUfb5k-a8) (YouTube 17:25 mins).

[](https://www.youtube.com/watch?v=8GHUfb5k-a8)

https://youtu.be/8GHUfb5k-a8

## Filtering results

You can filter the results of a query by specifying a predicate (or a condition) that evaluates to TRUE or FALSE. The most common way to achieve this is using the WHERE clause, however, using a combination of operators will allow you to write more complex expressions to filter subsets of results.

### Using the WHERE clause

Often you will want to select data based on some criteria, such as a person’s last name. The next query demonstrates the use of the WHERE clause to specify which records you want to return.

The single quotes (‘) are required around the value of LastName (‘Branson’) because the LastName column is defined as character data.

SELECT \*

FROM DRIVER

WHERE LastName = 'Branson';

Licence\_No FirstName LastName Phone EmployingCompanyABN

---------- ---------- --------- ---------- --------------------

75849857 Sharon Branson 0347637748 91876578373

When using numeric data, single quotes are not required.

The next query returns data for all vehicle types that have a fuel capacity of 120 litres; note the absence of the single quotes:

SELECT VehicleType\_ID,

FuelType,

FuelCapacity,

[Range]

FROM VEHICLE\_TYPE

WHERE FuelCapacity = 120;

VehicleType\_ID FuelType FuelCapacity Range

-------------- --------- ------------ ------

2003 Diesel 120 1200

3002 Diesel 120 1000

3003 Diesel 120 1100

### Dealing with NULL values

The next query lists all the columns and all the rows from the VEHICLE table. You’ll notice that the last column, DateOfLastMaintenance, has two records that are NULL.

This tells us that those two vehicles, NGJ946 and YRN737, have not yet received any maintenance. How would you list **only** the details of vehicles that have not received any maintenance yet?

SELECT \*

FROM VEHICLE;

Registration\_No OwningCompanyABN VehicleType\_ID DateOfManufacture DateOfLastMaintenance

--------------- -------------------- -------------- ----------------- ---------------------

ADB435 17948738949 1002 2018-02-01 2019-07-04

AWQ763 17948738949 2003 2017-07-12 2019-06-15

DSF743 79834510388 2004 2016-09-08 2019-06-05

EGD783 17948738949 3003 2019-09-04 2019-09-04

FHT326 79834510388 8001 2008-12-01 2019-06-08

IOU645 17948738949 1003 2017-10-07 2019-09-12

JHF674 91876578373 12002 2007-01-09 2019-07-05

NGJ946 79834510388 3003 2009-10-23 **NULL**

POR621 79834510388 2004 2016-09-08 2019-06-05

TRT757 91876578373 12001 2019-04-07 2019-11-10

UTY327 79834510388 2004 2016-09-08 2019-06-05

VIV443 79834510388 8001 2008-12-01 2019-06-09

YDF742 91876578373 12003 2003-06-14 2019-09-11

YRN737 17948738949 1002 2019-04-03 **NULL**

Most databases will support the IS NULL operation rather than = NULL in the WHERE clause, as demonstrated in the next query. IS NULL is the standard for SQL.

SELECT \*

FROM VEHICLE

WHERE DateOfLastMaintenance IS NULL;

Registration\_No OwningCompanyABN VehicleType\_ID DateOfManufacture DateOfLastMaintenance

--------------- -------------------- -------------- ----------------- ---------------------

NGJ946 79834510388 3003 2009-10-23 NULL

YRN737 17948738949 1002 2019-04-03 NULL

The next query lists all the details of vehicles that **have** been maintained using IS NOT NULL in the WHERE clause:

SELECT Registration\_No,

OwningCompanyABN,

VehicleType\_ID,

DateOfManufacture,

DateOfLastMaintenance

FROM VEHICLE

WHERE DateOfLastMaintenance IS NOT NULL;

Registration\_No OwningCompanyABN VehicleType\_ID DateOfManufacture DateOfLastMaintenance

--------------- -------------------- -------------- ----------------- ---------------------

ADB435 17948738949 1002 2018-02-01 2019-07-04

AWQ763 17948738949 2003 2017-07-12 2019-06-15

DSF743 79834510388 2004 2016-09-08 2019-06-05

EGD783 17948738949 3003 2019-09-04 2019-09-04

FHT326 79834510388 8001 2008-12-01 2019-06-08

IOU645 17948738949 1003 2017-10-07 2019-09-12

JHF674 91876578373 12002 2007-01-09 2019-07-05

POR621 79834510388 2004 2016-09-08 2019-06-05

TRT757 91876578373 12001 2019-04-07 2019-11-10

UTY327 79834510388 2004 2016-09-08 2019-06-05

VIV443 79834510388 8001 2008-12-01 2019-06-09

YDF742 91876578373 12003 2003-06-14 2019-09-11

### Using logical operators: AND, OR, NOT

You can combine expressions in the WHERE clause using the logical operators AND and OR. NOT can also be used to reverse the operation. The next query lists the date of manufacture and the date of last maintenance (which is NULL) of the vehicle with registration number YRN737, because you want to know if it’s old enough to have received a service yet:

SELECT DateOfManufacture,

DateOfLastMaintenance

FROM VEHICLE

WHERE DateOfLastMaintenance IS NULL

AND Registration\_No = 'YRN737';

DateOfManufacture DateOfLastMaintenance

----------------- ---------------------

2019-04-03 NULL

The next query lists the details of all VEHICLE\_TYPEs. What if you wanted to list the details of vehicle types that have a range of 1,200 km and either use a Hybrid/Petrol fuel type or have a fuel capacity of 80 litres?

Those records are highlighted in the next result set:

SELECT \*

FROM VEHICLE\_TYPE;

VehicleType\_ID FuelType FuelCapacity Range

-------------- -------------- ------------ ------

1001 Petrol 40 600

1002 Petrol 40 600

1003 Diesel 50 900

1004 Hybrid/Petrol 20 1200

1005 Hybrid/Petrol 30 1500

2001 Diesel 80 600

2002 Diesel 100 800

2003 Diesel 120 1200

2004 Hybrid/Diesel 50 800

2005 Hybrid/Diesel 80 1200

2006 Hybrid/Diesel 80 1400

3001 Diesel 110 900

3002 Diesel 120 1000

3003 Diesel 120 1100

8001 Diesel 210 1500

8002 Diesel 250 1800

8003 Diesel 300 900

12001 Diesel 800 2400

12002 Diesel 1200 3000

12003 Diesel 1200 3500

Combining the logical operators AND and OR in the next query provides the information that you require.

Note the square brackets around the column [Range]. SQL Server requires you to use the square brackets as delimiting characters, in this case because Range is a keyword in the Microsoft .Net Framework.

SELECT \*

FROM VEHICLE\_TYPE

WHERE [Range] = 1200

AND FuelType = 'Hybrid/Petrol'

OR FuelCapacity = 80;

VehicleType\_ID FuelType FuelCapacity Range

-------------- -------------- ------------ ------

1004 Hybrid/Petrol 20 1200

2001 Diesel 80 600

2005 Hybrid/Diesel 80 1200

2006 Hybrid/Diesel 80 1400

When combining complex expressions using the logical operators AND and OR, SQL evaluates the expressions according to the following rules:

* Operations inside the parentheses are prioritised
* then all ANDs
* then all ORs.

Now, if you add parentheses around the fuel type and fuel capacity, the logic of the query changes and produces a different result set as demonstrated in the next query:

SELECT \*

FROM VEHICLE\_TYPE

WHERE [Range] = 1200

AND (FuelType = 'Hybrid/Petrol' OR FuelCapacity = 80);

VehicleType\_ID FuelType FuelCapacity Range

-------------- -------------- ------------ ------

1004 Hybrid/Petrol 20 1200

2005 Hybrid/Diesel 80 1200

So, the two questions being asked are as follows:

* The query **without** the parentheses is asking ‘*list all the details of vehicle types that have a range of 1200 km and a fuel type of Hybrid/Petrol. Also list any other vehicle types that have a fuel capacity of 80 litres regardless of the range and fuel type.’*
* The query **with** the parentheses is asking *‘list all the details of vehicle types that have a range of 1200 km and either a fuel type of Hybrid/Petrol or a fuel capacity of 80 litres.’*

The difference is quite subtle however the two result sets demonstrate this critical difference.

### Using logical operators: IN, BETWEEN and LIKE

The next query lists all the details of the courier companies in the database:

SELECT \*

FROM COURIER\_COMPANY;

ABN CompanyName Phone

------------ -------------------------------- ----------

17948738949 Lockwood Couriers & Taxi Trucks 0295676789

32874653678 Abc Express 0287465367

37067463537 GoParcel 0284765367

53004085616 A1 Freight 0238476378

57087489589 Go-Go Errands 0294653678

72034374637 A2B Couriers & Taxi Trucks 0376578398

73023003457 Budget Courier Systems 0237678956

79834510388 Rocket Couriers 0298765463

89653008761 Barnetts Couriers 0287645786

90131456438 Bluewater Couriers 0294765367

91876578373 Bestever Transport 0387647584

In a production database, the list of companies would be much longer and retrieving the details of a company with *go* in the company name would be difficult if you had to manually check the list.

The solution is to use the LIKE operator. The LIKE operator searches *each* string of characters in *each* row (i.e. all characters and records) of the specified column, looking for a pattern match. This is a very expensive database operation in terms of resources used, so this operation should be used sparingly.

The next query demonstrates the use of the LIKE operator in conjunction with the wildcard character %:

SELECT \*

FROM COURIER\_COMPANY

WHERE CompanyName LIKE '%go%';

ABN CompanyName Phone

------------ ------------- ----------

37067463537 GoParcel 0284765367

57087489589 Go-Go Errands 0294653678

The preceding wildcard character % stipulates the selection of records with any number of any characters preceding the pattern *go*. The trailing wildcard character % stipulates the selection of records with any number of any characters following the pattern *go*. The LIKE operator can only be used with character data; it can’t be used with numeric data.

The next query selects drivers that have ‘r’ as the second letter in their last name, using the underscore ‘\_’. The underscore exactly matches any single character.

SELECT \*

FROM DRIVER

WHERE LastName LIKE '\_r%';

Licence\_No FirstName LastName Phone EmployingCompanyABN

---------- ---------- --------- ---------- --------------------

57463576 Barry Brown 0348473647 79834510388

75849857 Sharon Branson 0347637748 91876578373

If you were asked ‘*give me a list of vehicle types that have a minimum fuel capacity of 40 litres and a maximum fuel capacity of 100 litres’* you could use the BETWEEN operator to produce the list, as shown in the next query:

SELECT VehicleType\_ID,

FuelCapacity

FROM VEHICLE\_TYPE

WHERE FuelCapacity BETWEEN 40 AND 100;

VehicleType\_ID FuelCapacity

-------------- ------------

1001 40

1002 40

1003 50

2001 80

2002 100

2004 50

2005 80

2006 80

Notice that the BETWEEN operator is inclusive of the boundary values and is the same as ‘WHERE FuelCapacity >= 40 AND FuelCapacity <= 100’.

You can also use the BETWEEN operator with character data, as the next query demonstrates with the listing of names that are between the letters A and B:

SELECT FirstName,

LastName

FROM DRIVER

WHERE LastName BETWEEN 'A' AND 'B';

FirstName LastName

--------- ---------

Peter Anderton

What happened to the last names that start with a B? Well B is less than Ba, Bb, Bc, Bd, etc. So, since your last names have trailing characters after the first letter B, if you wanted to include last names such as Brown and Branson, you would have to make the boundary value bigger than Br.

So Bs would work, though Bz would be better, because then you’d include all possible letters for the second character of the last name as the next query demonstrates.

SELECT FirstName,

LastName

FROM DRIVER

WHERE LastName BETWEEN 'A' AND 'Bz';

FirstName LastName

--------- ---------

Barry Brown

Peter Anderton

Sharon Branson

Suppose you wanted to list the first and last names of drivers with the following driver’s licence numbers: 1593PD; 28476653; 4433JK; 75849857; 84756748. You could write a query like the following:

SELECT \*

FROM DRIVER

WHERE Licence\_No = '1593PD'

OR Licence\_No = '28476653'

OR Licence\_No = '4433JK'

OR Licence\_No = '75849857'

OR Licence\_No = '84756748';

Licence\_No FirstName LastName Phone EmployingCompanyABN

---------- --------- ---------- ---------- --------------------

1593PD Donald White 0294765367 91876578373

28476653 Yu Shoiwheng 0294756456 91876578373

4433JK David Livingston 0294564536 79834510388

75849857 Sharon Branson 0347637748 91876578373

84756748 Andrew Davidson 0376564653 91876578373

Or you could write a query like the next query where IN allows the comparison of column values to a list.

SELECT \*

FROM DRIVER

WHERE Licence\_No IN ('1593PD', '28476653', '4433JK', '75849857', '84756748');

Licence\_No FirstName LastName Phone EmployingCompanyABN

---------- --------- ---------- ---------- --------------------

1593PD Donald White 0294765367 91876578373

28476653 Yu Shoiwheng 0294756456 91876578373

4433JK David Livingston 0294564536 79834510388

75849857 Sharon Branson 0347637748 91876578373

84756748 Andrew Davidson 0376564653 91876578373

## Extracting a specific value by position

Often you will want to select data based on its ranking or position in a range of values. For example, you might want to know which is the fifth-most fuel-efficient vehicle type in your database. You can use the DENSE\_RANK() function to rank values according to their position in an ordered list.

The next query uses the DENSE\_RANK() function to rank the vehicle types according to the fuel consumption:

SELECT VehicleType\_ID,

FuelType,

FuelCapacity,

[Range],

CAST(FuelCapacity / ([Range] / 100.00) AS DECIMAL(5,2)) AS [FuelConsumption],  
DENSE\_RANK() OVER (ORDER BY (CAST(FuelCapacity / ([Range] / 100.00) AS DECIMAL(5,2))) ASC) AS RankedConsumptionValue

FROM VEHICLE\_TYPE;

VehicleType\_ID FuelType FuelCapacity Range FuelConsumption RankedConsumptionValue

-------------- ------------- ------------ ------ --------------- ----------------------

1004 Hybrid/Petrol 20 1200 1.67 1

1005 Hybrid/Petrol 30 1500 2.00 2

1003 Diesel 50 900 5.56 3

2006 Hybrid/Diesel 80 1400 5.71 4

2004 Hybrid/Diesel 50 800 6.25 5

2005 Hybrid/Diesel 80 1200 6.67 6

1001 Petrol 40 600 6.67 6

1002 Petrol 40 600 6.67 6

2003 Diesel 120 1200 10.00 7

3003 Diesel 120 1100 10.91 8

3002 Diesel 120 1000 12.00 9

3001 Diesel 110 900 12.22 10

2002 Diesel 100 800 12.50 11

2001 Diesel 80 600 13.33 12

8002 Diesel 250 1800 13.89 13

8001 Diesel 210 1500 14.00 14

8003 Diesel 300 900 33.33 15

12001 Diesel 800 2400 33.33 15

12003 Diesel 1200 3500 34.29 16

12002 Diesel 1200 3000 40.00 17

You can use a Common Table Expression (CTE) to extract a specific value by its position. For example, if you wanted to know which vehicle type had the fifth-best fuel consumption figures (the vehicle type highlighted in green above) you could write a CTE query that uses the DENSE\_RANK() function as demonstrated in the query below:

1 WITH RankedFuelConsumption AS

2 (

3 SELECT VehicleType\_ID,

4 FuelType,

5 FuelCapacity,

6 [Range],

7 CAST(FuelCapacity / ([Range] / 100.00) AS DECIMAL(5,2)) AS [FuelConsumption],  
8 DENSE\_RANK() OVER (ORDER BY (CAST(FuelCapacity / ([Range] / 100.00) AS   
9 DECIMAL(5,2))) ASC) AS RankedConsumptionValue

10 FROM VEHICLE\_TYPE

11 )

12 SELECT VehicleType\_ID,

13 FuelType,

14 FuelCapacity,

15 [Range],

16 FuelConsumption

17 FROM RankedFuelConsumption

18 WHERE RankedFuelConsumption.RankedConsumptionValue = 5;

VehicleType\_ID FuelType FuelCapacity Range FuelConsumption

-------------- -------------- ------------ ------ ----------------

2004 Hybrid/Diesel 50 800 6.25

* Line one defines the name of the CTE to be used later in line 17 as a table.
* Lines three to 10 define the actual CTE.

You can think of a CTE as a virtual table, which will have the columns listed in the SELECT list of lines three to nine. In this case, it also uses the DENSE\_RANK() function in lines nine and 10 to rank the derived and ordered fuel consumption values.

* Line seven is included purely for the purpose of displaying the actual fuel consumption value in the output and is used in line 16.

Other clauses that can be used for this purpose include TOP, LIMIT and ROWNUM. Read the web page [SQL TOP, LIMIT or ROWNUM Clause](https://www.w3schools.com/sql/sql_top.asp) to see examples of these clauses.

 Watch

## Video: RANK, DENSE\_RANK and ROW\_NUMBER functions

Watch this video to learn the difference between RANK, DENSE\_RANK and ROW\_NUMBER functions in SQL Server.

[Difference between rank dense rank and row number in SQL](https://youtu.be/MZTSHDFuCUk) (YouTube 04:42 mins).

[](https://www.youtube.com/watch?v=MZTSHDFuCUk)

https://youtu.be/MZTSHDFuCUk

# Selecting from multiple tables

One of the primary goals of relational database theory is to avoid data duplication. Relational databases are quite good at achieving this goal because they allow us to separate the data into tables, with each table representing one thing, then providing a way to relate the tables to each other.

So, in the Courier Company Governance System database, the details of each vehicle type, fuel type, fuel capacity and range are stored only once for each vehicle type in the VEHICLE\_TYPE table, rather than being stored for every vehicle in the VEHICLE table, so avoiding duplication.

SQL allows you to select data from multiple tables by including the list of tables in the FROM clause. However, if you write a query such as the next query, you’ll get a lot of (what appear to be meaningless) records listed (which we won’t list here because the listing will be too long).

Try it and see.

SELECT \*

FROM COURIER\_COMPANY,

DRIVER;

The query above returns 121 rows. Table COURIER\_COMPANY has 11 records in it and table DRIVER, coincidentally, also has 11 records in it. 11 × 11 = 121. So, what did the database server do when it executed the above query? It has taken each record from one table and matched it to each of the records in the other table, producing a somewhat meaningless ‘Product’ (multiplication) of the two tables.

A more meaningful combination of the two tables would be to match the records from the COURIER\_COMPANY table to the records in the DRIVER table, only if the DRIVER EmployingCompanyABN value equals (=) the COURIER\_COMPANY ABN value. You can do this joining operation (or equality operation) in the WHERE clause of the query. You’re effectively joining the two tables based on the values stored in the primary key and the foreign key (the two keys that relate the tables to each other).

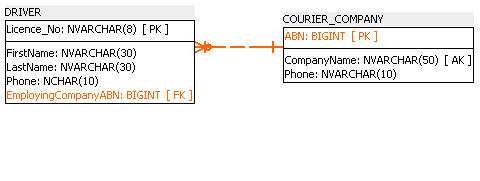


Figure Joining the DRIVER table and the COURIER\_COMPANY table

The next query lists all the columns and rows from the DRIVER and COURIER\_COMPANY tables, joining the tables in the WHERE clause on the foreign key and the primary key. Note that the order of join is not important because the expression doesn’t change if the operands switch sides, i.e. ‘WHERE ABN = EmployingCompanyABN’ is the same as ‘WHERE EmployingCompanyABN = ABN’.

SELECT \*

FROM DRIVER,

COURIER\_COMPANY

WHERE EmployingCompanyABN = ABN;

Licence\_No FirstName LastName Phone EmployingCompanyABN ABN CompanyName Phone

---------- --------- ---------- ---------- -------------------- ----------- ------------------------------- ----------

1593PD Donald White 0294765367 91876578373 91876578373 Bestever Transport 0387647584

2765GF John Smith 0297465367 17948738949 17948738949 Lockwood Couriers & Taxi Trucks 0295676789

28476653 Yu Shoiwheng 0294756456 91876578373 91876578373 Bestever Transport 0387647584

3173BC Mary Dilberry 0247564778 79834510388 79834510388 Rocket Couriers 0298765463

43567482 Nina Johansen 0294845748 17948738949 17948738949 Lockwood Couriers & Taxi Trucks 0295676789

4433JK David Livingston 0294564536 79834510388 79834510388 Rocket Couriers 0298765463

57463289 Marium Hashemi 0295847858 91876578373 91876578373 Bestever Transport 0387647584

57463576 Barry Brown 0348473647 79834510388 79834510388 Rocket Couriers 0298765463

7465SW Peter Anderton 0392837272 17948738949 17948738949 Lockwood Couriers & Taxi Trucks 0295676789

75849857 Sharon Branson 0347637748 91876578373 91876578373 Bestever Transport 0387647584

84756748 Andrew Davidson 0376564653 91876578373 91876578373 Bestever Transport 0387647584

The join will only produce results if there are matched rows. That is, it will only show results if there are drivers in the DRIVER table whose ‘EmployingCompanyABN’ exists in the ‘COURIER\_COMPANY’ table. Those courier companies that have no drivers will not be listed.

The next query shows those courier companies (CompanyName) whose ABN is not in the driver table. That is, this is a listing of companies that have no drivers:

SELECT CompanyName

FROM COURIER\_COMPANY

WHERE ABN NOT IN (SELECT EmployingCompanyABN

FROM DRIVER

);

CompanyName

---------------------------

Abc Express

GoParcel

A1 Freight

Go-Go Errands

A2B Couriers & Taxi Trucks

Budget Courier Systems

Barnetts Couriers

Bluewater Couriers

## Qualifying column names

When the database server prepares a query for execution, it will parse (read) each of the column names listed in the SELECT clause and check each of the tables in the FROM clause for the existence of each column.

The parser will continue to scan **all** the tables in the FROM clause, even after it finds the column. Searching through all the tables can affect the performance of the query, particularly on large databases.

It can also result in an ambiguous column name error if the column name is found in more than one table. The solution to both problems is to always qualify the names of the columns with the name of the table that the column is in. This is done using the ‘.’ dot notation by specifying tableName.columnName.

The next query demonstrates the qualification of column names with the names of the tables that the column is in, listing the first names, last names and company name (employer) of all the drivers:

SELECT DRIVER.FirstName,

DRIVER.LastName,

COURIER\_COMPANY.CompanyName

FROM DRIVER,

COURIER\_COMPANY

WHERE DRIVER.EmployingCompanyABN = COURIER\_COMPANY.ABN;

FirstName LastName CompanyName

-------------------- --------------------------------

Donald White Bestever Transport

John Smith Lockwood Couriers & Taxi Trucks

Yu Shoiwheng Bestever Transport

Mary Dilberry Rocket Couriers

Nina Johansen Lockwood Couriers & Taxi Trucks

David Livingston Rocket Couriers

Marium Hashemi Bestever Transport

Barry Brown Rocket Couriers

Peter Anderton Lockwood Couriers & Taxi Trucks

Sharon Branson Bestever Transport

Andrew Davidson Bestever Transport

## Using aliases

Now that you’ve qualified each column name with the names of the tables that the columns are in, you’ve increased the amount of typing that you have to do. You can use aliases to shorten the table names using the AS keyword in the FROM clause. Using an alias in a SELECT query doesn’t permanently change the actual underlying table name.

The next query is functionally the same as the previous query except that it uses aliases for the table names. You will use aliases for the remaining queries in this course.

SELECT d.FirstName,

d.LastName,

c.CompanyName

FROM COURIER\_COMPANY AS c,

DRIVER AS d

WHERE d.EmployingCompanyABN = c.ABN;

FirstName LastName CompanyName

--------- ---------- ---------------------------------

Donald White Bestever Transport

John Smith Lockwood Couriers & Taxi Trucks

Yu Shoiwheng Bestever Transport

Mary Dilberry Rocket Couriers

Nina Johansen Lockwood Couriers & Taxi Trucks

David Livingston Rocket Couriers

Marium Hashemi Bestever Transport

Barry Brown Rocket Couriers

Peter Anderton Lockwood Couriers & Taxi Trucks

Sharon Branson Bestever Transport

Andrew Davidson Bestever Transport

Suppose that you want to list the driver’s licence number, first name, last name, the name of the company that employs the driver, the vehicle type IDs that the driver is accredited to drive, the registration number of those vehicle types and the names of the companies that own the vehicles. Use the ERD Figure 32 to figure out how to traverse the joins from table to table, primary key to foreign key.

The following query uses the ORDER BY clause to sort the results by the driver’s licence number. Note from the result set produced, that most of the drivers are accredited to drive more than one type of vehicle.

1 SELECT d.Licence\_No,

2 d.FirstName,

3 d.LastName,

4 c.CompanyName AS Employer,

5 v.VehicleType\_ID,

6 v.Registration\_No,

7 c.CompanyName AS VehicleOwner

8 FROM DRIVER AS d,

9 COURIER\_COMPANY AS c,

10 VEHICLE AS v,

11 VEHICLE\_TYPE AS vt,

12 ACCREDITATION AS a

13 WHERE d.EmployingCompanyABN = c.ABN

14 AND c.ABN = v.OwningCompanyABN

15 AND v.VehicleType\_ID = vt.VehicleType\_ID

16 AND vt.VehicleType\_ID = a.VehicleType\_ID

17 AND a.Licence\_No = d.Licence\_No

18 ORDER BY d.Licence\_No;

Let’s take a closer look at the WHERE clause of the query above using the data model to help understand it better.

Line 13 joins the DRIVER table and the COURIER\_COMPANY table on:

‘WHERE d.EmployingCompanyABN = c.ABN’

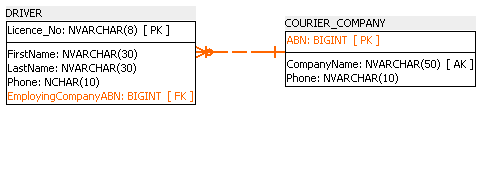


Figure The product of the DRIVER and COURIER\_COMPANY tables, matching those records where the EmployingCompanyABN equals the COURIER\_COMPAN ABN.

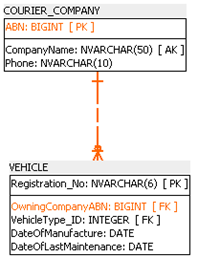


Figure The product of the COURIER\_COMPANY and VEHICLE tables, matching those records where the COURIER\_COMPANY ABN equals the VEHICLE OwningCompanyABN.

Line 15 then joins the COURIER\_COMPANY table to the VEHICLE table on:

‘AND v.VehicleType\_ID = vt.VehicleType\_ID’

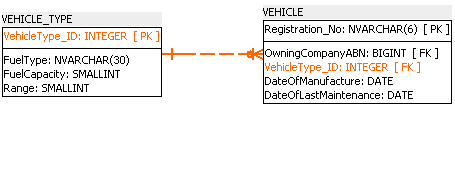


Figure The product of the VEHICLE and VEHICLE\_TYPE tables

Line 16 then joins the VEHICLE\_TYPE to the ACCREDITATION table on:

‘AND vt.VehicleType\_ID = a.VehicleType\_ID’

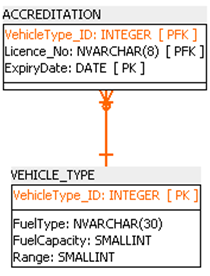


Figure The product of the VEHICLE\_TYPE and ACCREDITATION tables

Line 17 joins the ACCREDITATION table to the DRIVER table on:

‘AND a.Licence\_No = d.Licence\_No’

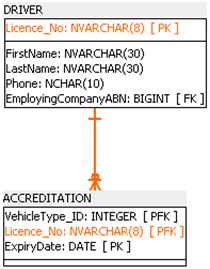


Figure The product of the ACCREDITATION and DRIVER tables

Below is the listing from the query discussed above.

Licence\_No FirstName LastName Employer VehicleType\_ID Registration\_No VehicleOwner

---------- ---------- ---------- -------------------------------- -------------- --------------- -------------------------------

1593PD Donald White Bestever Transport 12002 JHF674 Bestever Transport

1593PD Donald White Bestever Transport 12001 TRT757 Bestever Transport

2765GF John Smith Lockwood Couriers & Taxi Trucks 1002 ADB435 Lockwood Couriers & Taxi Trucks

2765GF John Smith Lockwood Couriers & Taxi Trucks 2003 AWQ763 Lockwood Couriers & Taxi Trucks

2765GF John Smith Lockwood Couriers & Taxi Trucks 3003 EGD783 Lockwood Couriers & Taxi Trucks

2765GF John Smith Lockwood Couriers & Taxi Trucks 1003 IOU645 Lockwood Couriers & Taxi Trucks

2765GF John Smith Lockwood Couriers & Taxi Trucks 1002 YRN737 Lockwood Couriers & Taxi Trucks

28476653 Yu Shoiwheng Bestever Transport 12003 YDF742 Bestever Transport

3173BC Mary Dilberry Rocket Couriers 2004 DSF743 Rocket Couriers

3173BC Mary Dilberry Rocket Couriers 3003 NGJ946 Rocket Couriers

3173BC Mary Dilberry Rocket Couriers 2004 POR621 Rocket Couriers

3173BC Mary Dilberry Rocket Couriers 2004 UTY327 Rocket Couriers

43567482 Nina Johansen Lockwood Couriers & Taxi Trucks 3003 EGD783 Lockwood Couriers & Taxi Trucks

4433JK David Livingston Rocket Couriers 2004 DSF743 Rocket Couriers

4433JK David Livingston Rocket Couriers 8001 FHT326 Rocket Couriers

4433JK David Livingston Rocket Couriers 2004 POR621 Rocket Couriers

4433JK David Livingston Rocket Couriers 2004 UTY327 Rocket Couriers

4433JK David Livingston Rocket Couriers 8001 VIV443 Rocket Couriers

57463289 Marium Hashemi Bestever Transport 12001 TRT757 Bestever Transport

57463576 Barry Brown Rocket Couriers 2004 DSF743 Rocket Couriers

57463576 Barry Brown Rocket Couriers 2004 POR621 Rocket Couriers

57463576 Barry Brown Rocket Couriers 2004 UTY327 Rocket Couriers

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks 1002 ADB435 Lockwood Couriers & Taxi Trucks

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks 2003 AWQ763 Lockwood Couriers & Taxi Trucks

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks 3003 EGD783 Lockwood Couriers & Taxi Trucks

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks 1003 IOU645 Lockwood Couriers & Taxi Trucks

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks 1002 YRN737 Lockwood Couriers & Taxi Trucks

75849857 Sharon Branson Bestever Transport 12001 TRT757 Bestever Transport

75849857 Sharon Branson Bestever Transport 12003 YDF742 Bestever Transport

84756748 Andrew Davidson Bestever Transport 12003 YDF742 Bestever Transport

## Using joins

Tables can be combined in a query in ways other than using a product, which we have been using so far. i.e. ‘WHERE d.EmployingCompanyABN = c.ABN’.

As of the SQL2 ANSI standard (released in 1992) and subsequent releases of the standard, SQL specifies five types of joins:

* INNER JOIN: This returns only the records from both tables in the join statement that have matching values in the joined columns.
* LEFT OUTER JOIN: This returns all the records from the table on the left side of the join statement and only those records from the table on the right side of the join statement that have a matching value in the joined columns.
* RIGHT OUTER JOIN: This returns all of the records from the table on the right side of the join statement and only those records from the table on the left side of the join statement that have a matching value in the joined columns.
* FULL OUTER JOIN: This returns all the records from both tables when there is a matching value in either the left or right side table in the joined columns.
* CROSS JOIN: This returns the Cartesian product of the two tables being joined. i.e. it will match each record from the table on the left to each record from the table on the right. No matching values are specified in a CROSS JOIN.

Note that the keywords INNER and OUTER are optional in the ANSI standard. However, we will continue to use them here for clarity. INNER is the default, so JOIN implies an INNER JOIN. LEFT, RIGHT and FULL imply an OUTER JOIN.

### INNER JOIN

Conceptually, INNER JOINs work in the same manner as products. That is, the related tables are still joined on the Primary Key and Foreign Key columns. The following query describes the syntax for INNER JOINs.

Note that *table1* is considered the table on the left of the join and *table2* is on the right. So, the table’s position in the join statement determines whether it’s on the left or the right. This will be a factor to consider when deciding whether to use a LEFT OUTER JOIN or a RIGHT OUTER JOIN later on.

SELECT column\_name(s)  
FROM table1 INNER JOIN table2  
ON table1.column\_name = table2.column\_name;

Suppose you want a listing of driver’s licence numbers, the driver’s first and last names and the name of the driver’s employing company. You can produce this listing using a product as demonstrated in the next query:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

c.CompanyName

FROM DRIVER AS d,

COURIER\_COMPANY AS c  
WHERE d.EmployingCompanyABN = c.ABN;

This can be converted to the following INNER JOIN query:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

c.CompanyName

FROM DRIVER AS d

INNER JOIN COURIER\_COMPANY AS c

ON d.EmployingCompanyABN = c.ABN;

Licence\_No FirstName LastName CompanyName

---------- --------- ---------- --------------------------------

1593PD Donald White Bestever Transport

2765GF John Smith Lockwood Couriers & Taxi Trucks

28476653 Yu Shoiwheng Bestever Transport

3173BC Mary Dilberry Rocket Couriers

43567482 Nina Johansen Lockwood Couriers & Taxi Trucks

4433JK David Livingston Rocket Couriers

57463289 Marium Hashemi Bestever Transport

57463576 Barry Brown Rocket Couriers

7465SW Peter Anderton Lockwood Couriers & Taxi Trucks

75849857 Sharon Branson Bestever Transport

84756748 Andrew Davidson Bestever Transport

Where previously the tables were listed in the FROM clause separated by a comma, they are now separated by the INNER JOIN keywords. The joining conditions that were previously defined in the WHERE clause are now listed following the second table name and preceded with the ON keyword.

#### Using INNER JOINs with more than two tables

You can also use joins for queries that list data from more than two tables.

The next query selects data from all the tables in the database and lists the driver’s licence number, the driver’s last name, the ID of the vehicle types they’re accredited to drive, the expiry date of the driver’s accreditation, the vehicle registration number and the company names of those companies that own those types of vehicle. Note the introduction of the use of parentheses to control the joining of the tables.

SELECT d.Licence\_No,

d.LastName,

a.VehicleType\_ID,

a.ExpiryDate,

v.Registration\_No,

c.CompanyName

FROM ((((DRIVER AS d

INNER JOIN ACCREDITATION AS a ON d.Licence\_No = a.Licence\_No)

INNER JOIN VEHICLE\_TYPE AS vt ON a.VehicleType\_ID = vt.VehicleType\_ID)

INNER JOIN VEHICLE AS v ON vt.VehicleType\_ID = v.VehicleType\_ID)

INNER JOIN COURIER\_COMPANY AS c ON v.OwningCompanyABN = c.ABN);

Licence\_No LastName VehicleType\_ID ExpiryDate Registration\_No CompanyName

---------- ----------- -------------- ---------- --------------- --------------------------------

2765GF Smith 1002 2020-01-01 ADB435 Lockwood Couriers & Taxi Trucks

2765GF Smith 1002 2020-01-01 YRN737 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1002 2020-03-05 ADB435 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1002 2020-03-05 YRN737 Lockwood Couriers & Taxi Trucks

2765GF Smith 1003 2020-01-01 IOU645 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1003 2020-03-05 IOU645 Lockwood Couriers & Taxi Trucks

2765GF Smith 2003 2020-01-01 AWQ763 Lockwood Couriers & Taxi Trucks

7465SW Anderton 2003 2020-03-05 AWQ763 Lockwood Couriers & Taxi Trucks

3173BC Dilberry 2004 2020-05-16 DSF743 Rocket Couriers

3173BC Dilberry 2004 2020-05-16 POR621 Rocket Couriers

3173BC Dilberry 2004 2020-05-16 UTY327 Rocket Couriers

4433JK Livingston 2004 2020-03-03 DSF743 Rocket Couriers

4433JK Livingston 2004 2020-03-03 POR621 Rocket Couriers

4433JK Livingston 2004 2020-03-03 UTY327 Rocket Couriers

57463576 Brown 2004 2020-04-18 DSF743 Rocket Couriers

57463576 Brown 2004 2020-04-18 POR621 Rocket Couriers

57463576 Brown 2004 2020-04-18 UTY327 Rocket Couriers

2765GF Smith 3003 2020-01-01 EGD783 Lockwood Couriers & Taxi Trucks

2765GF Smith 3003 2020-01-01 NGJ946 Rocket Couriers

3173BC Dilberry 3003 2020-05-16 EGD783 Lockwood Couriers & Taxi Trucks

3173BC Dilberry 3003 2020-05-16 NGJ946 Rocket Couriers

43567482 Johansen 3003 2020-06-10 EGD783 Lockwood Couriers & Taxi Trucks

43567482 Johansen 3003 2020-06-10 NGJ946 Rocket Couriers

7465SW Anderton 3003 2020-03-05 EGD783 Lockwood Couriers & Taxi Trucks

7465SW Anderton 3003 2020-03-05 NGJ946 Rocket Couriers

4433JK Livingston 8001 2020-03-03 FHT326 Rocket Couriers

4433JK Livingston 8001 2020-03-03 VIV443 Rocket Couriers

1593PD White 12001 2020-06-05 TRT757 Bestever Transport

57463289 Hashemi 12001 2020-03-04 TRT757 Bestever Transport

75849857 Branson 12001 2019-12-26 TRT757 Bestever Transport

1593PD White 12002 2020-06-05 JHF674 Bestever Transport

28476653 Shoiwheng 12003 2020-04-05 YDF742 Bestever Transport

75849857 Branson 12003 2019-12-30 YDF742 Bestever Transport

84756748 Davidson 12003 2019-12-28 YDF742 Bestever Transport

Each INNER JOIN is executed on a single pair of tables. Since more than two tables have been included in this query, parentheses have been used to combine (JOIN) the third table, VEHICLE\_TYPE, to the result of combining the first two tables, DRIVER and ACCREDITATION.

This combined result is then joined to the fourth table, VEHICLE. Then finally the result of this combination is JOINed to the fifth table COURIER\_COMPANY.

The use of parentheses to control the joining logic (or precedence) of the tables in a query can become quite complex and sophisticated. Further investigation of more complex joining logic is beyond the scope of the units in this course.

At this stage of your learning you should focus on the technique described here, that is, joining the first pair of tables, then joining each subsequent table to the results of previous combinations.

From this point forward, you should always write your SQL queries using JOINs rather than products (where possible). JOINs are the most recent method incorporated into the SQL Standard and JOINs also execute more efficiently. All the previous queries demonstrated in this Student Workbook could have been written using JOINs instead of products.

The following query uses an INNER JOIN to return a listing that will only include records if it is TRUE that the driver’s first name is David and the driver’s last name is Livingston (one of the queries discussed earlier in these course notes), ordered by the Vehicle Type ID:

SELECT d.FirstName,

d.LastName,

c.CompanyName AS Employer,

v.VehicleType\_ID

FROM ((((DRIVER AS d

INNER JOIN COURIER\_COMPANY AS c ON d.EmployingCompanyABN = c.ABN

AND (d.FirstName = 'David' AND d.LastName = 'Livingston'))

INNER JOIN VEHICLE AS v ON c.ABN = v.OwningCompanyABN)

INNER JOIN VEHICLE\_TYPE AS vt ON v.VehicleType\_ID = vt.VehicleType\_ID)

INNER JOIN ACCREDITATION AS a ON vt.VehicleType\_ID = a.VehicleType\_ID

AND a.Licence\_No = d.Licence\_No)

ORDER BY v.VehicleType\_ID;

FirstName LastName Employer VehicleType\_ID

--------- ---------- --------------- --------------

David Livingston Rocket Couriers 2004

David Livingston Rocket Couriers 2004

David Livingston Rocket Couriers 2004

David Livingston Rocket Couriers 8001

David Livingston Rocket Couriers 8001

Note that the predicates used in the previous query ‘AND (d.FirstName = 'David' AND d.LastName = 'Livingston')‘ are attached to the JOIN of the DRIVER and COURIER\_COMPANY tables.

The effect of this is to reduce the size of the result set produced by combining those two tables to only those records that meet the criteria of the predicates. This will then reduce the number of records to be processed (or matched) when the next table (VEHICLE) is joined.

Having said that, however, modern RDBMS incorporate query-optimisation algorithms. For most database applications you won’t have to worry about the location of the predicates in the query, since the query optimiser of the database engine will determine the most efficient execution plan.

### OUTER JOIN

There are three types of OUTER JOIN that we will discuss here, LEFT OUTER JOIN, RIGHT OUTER JOIN and FULL OUTER JOIN.

#### LEFT OUTER JOIN

You may have noticed that in the Courier Company Governance database, not all types of vehicle are owned by all courier companies. If you wanted a listing of all vehicle types and the registration numbers of those types of vehicle, you could use a LEFT OUTER JOIN query as demonstrated below:

SELECT vt.VehicleType\_ID,

v.Registration\_No

FROM (VEHICLE\_TYPE AS vt

LEFT OUTER JOIN VEHICLE AS v ON vt.VehicleType\_ID = v.VehicleType\_ID)

VehicleType\_ID Registration\_No

-------------- ---------------

1001 NULL

1002 ADB435

1002 YRN737

1003 IOU645

1004 NULL

1005 NULL

2001 NULL

2002 NULL

2003 AWQ763

2004 DSF743

2004 POR621

2004 UTY327

2005 NULL

2006 NULL

3001 NULL

3002 NULL

3003 EGD783

3003 NGJ946

8001 FHT326

8001 VIV443

8002 NULL

8003 NULL

12001 TRT757

12002 JHF674

12003 YDF742

Note the definition of a LEFT OUTER JOIN given earlier:

*LEFT OUTER JOIN: This returns all the records from the table on the left side of the join statement and only those records from the table on the right side of the join statement that have a matching value in the joined columns.*

So in the result set of the previous query, you can see that ***ALL*** the vehicle type IDs are listed and that for some of them, there are no vehicles that match the type in the VEHICLE table (those rows coloured green indicating NULL for the registration number).

Now let’s say that you wanted to list the vehicle types and the names of the companies that own each vehicle type. Remember, that not all types of vehicle are owned by a company.

The following query lists the vehicle type and the owning company names:

SELECT vt.VehicleType\_ID,

cc.CompanyName

FROM ((VEHICLE\_TYPE AS vt

LEFT OUTER JOIN VEHICLE AS v ON vt.VehicleType\_ID = v.VehicleType\_ID)

LEFT OUTER JOIN COURIER\_COMPANY AS cc ON v.OwningCompanyABN = cc.ABN)

VehicleType\_ID CompanyName

-------------- --------------------------------

1001 NULL

1002 Lockwood Couriers & Taxi Trucks

1002 Lockwood Couriers & Taxi Trucks

1003 Lockwood Couriers & Taxi Trucks

1004 NULL

1005 NULL

2001 NULL

2002 NULL

2003 Lockwood Couriers & Taxi Trucks

2004 Rocket Couriers

2004 Rocket Couriers

2004 Rocket Couriers

2005 NULL

2006 NULL

3001 NULL

3002 NULL

3003 Lockwood Couriers & Taxi Trucks

3003 Rocket Couriers

8001 Rocket Couriers

8001 Rocket Couriers

8002 NULL

8003 NULL

12001 Bestever Transport

12002 Bestever Transport

12003 Bestever Transport

In the previous query, a LEFT OUTER JOIN has been used to select ***ALL*** the records from the VECHICLE\_TYPE table (which is on the left of the JOIN statement) and those records from the VEHICLE table whose VehicleType\_IDs match.

A second LEFT OUTER JOIN is then used to select ***ALL*** rows from the combination of the first two tables and only those records from the third table COURIER\_COMPANY where the company ABNs match.

Again, you can see that some VEHICLE\_TYPES are not owned by any companies (those coloured green and indicating NULL for the company name in the result set above).

#### RIGHT OUTER JOIN

*Do we have drivers accredited to drive each type of vehicle in our database?* You can answer this question by listing ***ALL*** the vehicle type IDs and the Licence numbers of those drivers accredited to drive each vehicle type.

In the following query you’ll use a RIGHT OUTER JOIN to answer this question:

SELECT d.FirstName,

d.LastName,

vt.VehicleType\_ID

FROM ((DRIVER AS d

RIGHT OUTER JOIN ACCREDITATION AS a ON d.Licence\_No = a.Licence\_No)

RIGHT OUTER JOIN VEHICLE\_TYPE AS vt ON a.VehicleType\_ID = vt.VehicleType\_ID)

FirstName LastName VehicleType\_ID

---------- ---------- --------------

NULL NULL 1001

John Smith 1002

Peter Anderton 1002

John Smith 1003

Peter Anderton 1003

NULL NULL 1004

NULL NULL 1005

NULL NULL 2001

NULL NULL 2002

John Smith 2003

Peter Anderton 2003

Mary Dilberry 2004

David Livingston 2004

Barry Brown 2004

NULL NULL 2005

NULL NULL 2006

NULL NULL 3001

NULL NULL 3002

John Smith 3003

Mary Dilberry 3003

Nina Johansen 3003

Peter Anderton 3003

David Livingston 8001

NULL NULL 8002

NULL NULL 8003

Donald White 12001

Marium Hashemi 12001

Sharon Branson 12001

Donald White 12002

Yu Shoiwheng 12003

Sharon Branson 12003

Andrew Davidson 12003

Those rows coloured green in the above result set indicate the Vehicle Types that have no drivers accredited to drive them.

If you wanted to list only the Vehicle Type IDs, you can modify the above query to return only those records where the driver’s first name and driver’s last name is NULL, returning the list of vehicle types with no accredited drivers:

SELECT vt.VehicleType\_ID

FROM ((DRIVER AS d

RIGHT OUTER JOIN ACCREDITATION AS a ON d.Licence\_No = a.Licence\_No)

RIGHT OUTER JOIN VEHICLE\_TYPE AS vt ON a.VehicleType\_ID = vt.VehicleType\_ID)

WHERE d.FirstName IS NULL and d.LastName IS NULL

VehicleType\_ID

--------------

1001

1004

1005

2001

2002

2005

2006

3001

3002

8002

8003

Note that, as mentioned previously, the choice between the use of a LEFT OUTER JOIN or a RIGHT OUTER JOIN will be determined by the table’s position in the join statement.

In the previous example you wanted a listing of ***ALL*** the records from the VEHICLE\_TYPE table which is positioned on the right side of the join statement in the query, hence the decision to use a RIGHT OUTER JOIN.

#### FULL OUTER JOIN

The following query uses a FULL OUTER JOIN to return a listing of ***ALL*** vehicle types, driver’s accredited to drive each type and the name of the company that owns vehicles of the type:

SELECT d.Licence\_No,

d.LastName,

vt.VehicleType\_ID,

cc.CompanyName

FROM ((((VEHICLE\_TYPE AS vt

LEFT OUTER JOIN ACCREDITATION AS a ON vt.VehicleType\_ID = a.VehicleType\_ID)

LEFT OUTER JOIN DRIVER AS d ON a.Licence\_No = d.Licence\_No)

FULL OUTER JOIN VEHICLE AS v ON vt.VehicleType\_ID = v.VehicleType\_ID)

LEFT OUTER JOIN COURIER\_COMPANY AS cc ON v.OwningCompanyABN = cc.ABN)

Licence\_No LastName VehicleType\_ID CompanyName

---------- ---------- -------------- --------------------------------

NULL NULL 1001 NULL

2765GF Smith 1002 Lockwood Couriers & Taxi Trucks

2765GF Smith 1002 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1002 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1002 Lockwood Couriers & Taxi Trucks

2765GF Smith 1003 Lockwood Couriers & Taxi Trucks

7465SW Anderton 1003 Lockwood Couriers & Taxi Trucks

NULL NULL 1004 NULL

NULL NULL 1005 NULL

NULL NULL 2001 NULL

NULL NULL 2002 NULL

2765GF Smith 2003 Lockwood Couriers & Taxi Trucks

7465SW Anderton 2003 Lockwood Couriers & Taxi Trucks

3173BC Dilberry 2004 Rocket Couriers

3173BC Dilberry 2004 Rocket Couriers

3173BC Dilberry 2004 Rocket Couriers

4433JK Livingston 2004 Rocket Couriers

4433JK Livingston 2004 Rocket Couriers

4433JK Livingston 2004 Rocket Couriers

57463576 Brown 2004 Rocket Couriers

57463576 Brown 2004 Rocket Couriers

57463576 Brown 2004 Rocket Couriers

NULL NULL 2005 NULL

NULL NULL 2006 NULL

NULL NULL 3001 NULL

NULL NULL 3002 NULL

2765GF Smith 3003 Lockwood Couriers & Taxi Trucks

2765GF Smith 3003 Rocket Couriers

3173BC Dilberry 3003 Lockwood Couriers & Taxi Trucks

3173BC Dilberry 3003 Rocket Couriers

43567482 Johansen 3003 Lockwood Couriers & Taxi Trucks

43567482 Johansen 3003 Rocket Couriers

7465SW Anderton 3003 Lockwood Couriers & Taxi Trucks

7465SW Anderton 3003 Rocket Couriers

4433JK Livingston 8001 Rocket Couriers

4433JK Livingston 8001 Rocket Couriers

NULL NULL 8002 NULL

NULL NULL 8003 NULL

1593PD White 12001 Bestever Transport

57463289 Hashemi 12001 Bestever Transport

75849857 Branson 12001 Bestever Transport

1593PD White 12002 Bestever Transport

28476653 Shoiwheng 12003 Bestever Transport

75849857 Branson 12003 Bestever Transport

84756748 Davidson 12003 Bestever Transport

Note that, coincidentally, the vehicle types for which there are no drivers are also the same types that are not owned by any of the companies.

In the previous query, ***ALL*** the records from the VEHICLE\_TYPE table are returned using a LEFT OUTER JOIN to the ACCREDITATION table. Then ***ALL*** results of this combination are returned using a LEFT OUTER JOIN to the DRIVER table. Then a FULL OUTER JOIN is used to maintain in the listing ***ALL*** vehicle types and ***ALL*** vehicles. Finally ***ALL*** the results of this combination are joined to the COURIER\_COMPANY table using a LEFT OUTER JOIN.

So the vehicle types coloured green in the above query are those for which there are no accredited drivers and, coincidentally, none of the companies own those types of vehicles.

### CROSS JOIN

As mentioned earlier, a CROSS JOIN returns the Cartesian product of the two tables being joined. Each record from the left table will be matched to each of the records in the right table. So if table T1 has the values 1, 2, and 3 in it and table T2 has the values A, B and C in it, the Cartesian product would be all combinations of 1, 2, 3 and A, B, C. as indicated below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tables | | |  |  |  |
|  |  |  |  |  |  |
| **T1** |  | **T2** |  | **Result** | |
| 1 |  | A | SELECT \*  FROM T1  CROSS JOIN T2 | 1 | A |
| 2 |  | B | 1 | B |
| 3 |  | C | 1 | C |
|  |  |  | 2 | A |
|  |  |  | 2 | B |
|  |  |  | 2 | C |
|  |  |  |  | 3 | A |
|  |  |  |  | 3 | B |
|  |  |  |  | 3 | C |

The most practical applications of a CROSS JOIN are to return all possible combinations of two entities (or attributes of two entities) or to performance-test a database. If a WHERE clause is used in a CROSS JOIN query to select records where the primary key equals the foreign key, then the CROSS JOIN query will produce the same results as an INNER JOIN.

In our Courier Company Governance System database, one application of a CROSS JOIN would be to produce a listing of all drivers and every possible combination of vehicle type accreditation they could hold, as illustrated by the following query.

Since there are 11 records in the DRIVER table and 20 records in the VEHICLE\_TYPE table, the Cartesian product of the two tables returns 220 rows (11 x 20), too many to display here.

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

vt.VehicleType\_ID

FROM DRIVER AS d

CROSS JOIN VEHICLE\_TYPE AS vt

A more meaningful query would be to combine the results of the previous CROSS JOIN query with a sub-query to list the vehicle type accreditations that none of the drivers in the system currently have. So, drivers (like Donald White) could be encouraged to upgrade their accreditations for vehicle types that nobody else is yet accredited to drive.

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

vt.VehicleType\_ID AS AccreditationsNotHeld

FROM DRIVER AS d

CROSS JOIN VEHICLE\_TYPE AS vt

WHERE vt.VehicleType\_ID NOT IN

( SELECT vt.VehicleType\_ID

FROM ((DRIVER AS d

INNER JOIN ACCREDITATION AS a ON d.Licence\_No = a.Licence\_No)

INNER JOIN VEHICLE\_TYPE AS vt ON a.VehicleType\_ID = vt.VehicleType\_ID)

)

AND d.FirstName = 'Donald'

AND d.LastName = 'White'

ORDER BY d.Licence\_No;

Licence\_No FirstName LastName AccreditationsNotHeld

---------- --------- -------- ---------------------

1593PD Donald White 1001

1593PD Donald White 1004

1593PD Donald White 1005

1593PD Donald White 2001

1593PD Donald White 2002

1593PD Donald White 2005

1593PD Donald White 2006

1593PD Donald White 3001

1593PD Donald White 3002

1593PD Donald White 8002

1593PD Donald White 8003

# Expressions using calculations

## Operations on numeric values

Mathematical equations in queries and expressions use arithmetic operators to calculate values. The following table shows the syntax for arithmetic operators in Microsoft Access.

Table 2 Arithmetic operators in SQL

| Operator | Description | Example |
| --- | --- | --- |
| + | Add | SubTotal+GST |
| - | Subtract | TotalPrice-TradeDiscount |
| \* | Multiply | QuantityOrdered\*TotalPrice |
| / | Divide | TotalPrice/NumberOrdered |
| % | Modulo - find the remainder of a division equation | NumberGuests % NumberRooms |

You can perform calculations on numeric values stored in a table by using a mathematical expression that also evaluates to a numeric result. For example, if you wanted to calculate the approximate litres of fuel consumption per 100 km for each vehicle type, you could write a query such as the following:

SELECT vt.VehicleType\_ID,

vt.FuelType,

vt.FuelCapacity,

vt.[Range],

vt.FuelCapacity / ([Range] / 100) AS [Approx. Litres Of Fuel Per 100 kms]

FROM VEHICLE\_TYPE AS vt

ORDER BY [Approx. Litres Of Fuel Per 100 kms];

VehicleType\_ID FuelType FuelCapacity Range Approx. Litres Of Fuel Per 100 kms

-------------- -------------- ------------ ------ --------------------------------

1004 Hybrid/Petrol 20 1200 1

1005 Hybrid/Petrol 30 1500 2

1003 Diesel 50 900 5

2006 Hybrid/Diesel 80 1400 5

2004 Hybrid/Diesel 50 800 6

2005 Hybrid/Diesel 80 1200 6

1001 Petrol 40 600 6

1002 Petrol 40 600 6

3003 Diesel 120 1100 10

2003 Diesel 120 1200 10

3001 Diesel 110 900 12

3002 Diesel 120 1000 12

2002 Diesel 100 800 12

2001 Diesel 80 600 13

8002 Diesel 250 1800 13

8001 Diesel 210 1500 14

8003 Diesel 300 900 33

12001 Diesel 800 2400 33

12003 Diesel 1200 3500 34

12002 Diesel 1200 3000 40

This is an approximation because if you consider the first row of the result set and do a manual calculation, the answer is not 1 (as indicated):

20 ÷ (1200 ÷ 100) = 1.66666…

The reason for this is that *FuelCapacity* and *Range* are stored in the table as SMALLINTs (whole numbers) and you’re dividing the Range by 100, which is also interpreted by the database server as an INTEGER.

So, the result produced is **rounded down** to the nearest INTEGER. If you want a more accurate measure of fuel consumption, you should modify the expression so that the result is produced as a DECIMAL. This can be done by dividing the Range by 100.00 (instead of 100). So, in the next query the expression becomes:

FuelCapacity ÷ (Range ÷ 100.00) = Litres of fuel per 100 km.

SELECT vt.VehicleType\_ID,

vt.FuelType,

vt.FuelCapacity,

vt.[Range],

vt.FuelCapacity / ([Range] / 100.00) AS [Litres Of Fuel Per 100 kms]

FROM VEHICLE\_TYPE AS vt

ORDER BY [Litres Of Fuel Per 100 kms];

VehicleType\_ID FuelType FuelCapacity Range Litres Of Fuel Per 100 kms

-------------- -------------- ------------ ------ --------------------------

1004 Hybrid/Petrol 20 1200 1.66666666666666

1005 Hybrid/Petrol 30 1500 2.00000000000000

1003 Diesel 50 900 5.55555555555555

2006 Hybrid/Diesel 80 1400 5.71428571428571

2004 Hybrid/Diesel 50 800 6.25000000000000

2005 Hybrid/Diesel 80 1200 6.66666666666666

1001 Petrol 40 600 6.66666666666666

1002 Petrol 40 600 6.66666666666666

2003 Diesel 120 1200 10.00000000000000

3003 Diesel 120 1100 10.90909090909090

3002 Diesel 120 1000 12.00000000000000

3001 Diesel 110 900 12.22222222222222

2002 Diesel 100 800 12.50000000000000

2001 Diesel 80 600 13.33333333333333

8002 Diesel 250 1800 13.88888888888888

8001 Diesel 210 1500 14.00000000000000

8003 Diesel 300 900 33.33333333333333

12001 Diesel 800 2400 33.33333333333333

12003 Diesel 1200 3500 34.28571428571428

12002 Diesel 1200 3000 40.00000000000000

Now that’s probably a little too accurate. You can format the output of the column displaying the fuel consumption in many ways. In this case you’ll use the CAST() function (rather than the FORMAT() function) because you’re sorting the results based on the actual numeric value.

The FORMAT() function would be easier to use but would convert the output to character data and disrupt the sort order. The next query demonstrates the use of the CAST() function.

Note the specification of the size of the DECIMAL type of (5,2). This stipulates a decimal value with five significant figures and two decimal places (e.g. 999.99).

SELECT vt.VehicleType\_ID,

vt.FuelType,

vt.FuelCapacity,

vt.[Range],

CAST((vt.FuelCapacity / ([Range] / 100.00)) AS DECIMAL(5,2)) AS [Litres Of Fuel Per 100 kms]

FROM VEHICLE\_TYPE AS vt

ORDER BY [Litres Of Fuel Per 100 kms] ;

VehicleType\_ID FuelType FuelCapacity Range Litres Of Fuel Per 100 kms

-------------- ------------- ------------ ------ --------------------------

1004 Hybrid/Petrol 20 1200 1.67

1005 Hybrid/Petrol 30 1500 2.00

1003 Diesel 50 900 5.56

2006 Hybrid/Diesel 80 1400 5.71

2004 Hybrid/Diesel 50 800 6.25

2005 Hybrid/Diesel 80 1200 6.67

1001 Petrol 40 600 6.67

1002 Petrol 40 600 6.67

2003 Diesel 120 1200 10.00

3003 Diesel 120 1100 10.91

3002 Diesel 120 1000 12.00

3001 Diesel 110 900 12.22

2002 Diesel 100 800 12.50

2001 Diesel 80 600 13.33

8002 Diesel 250 1800 13.89

8001 Diesel 210 1500 14.00

8003 Diesel 300 900 33.33

12001 Diesel 800 2400 33.33

12003 Diesel 1200 3500 34.29

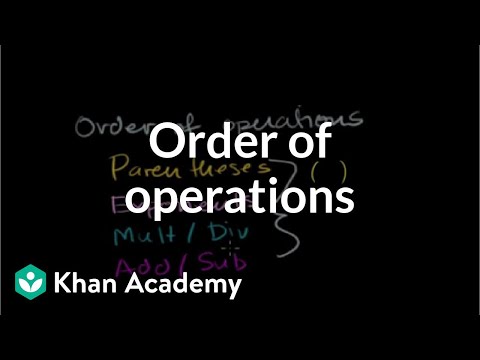
12002 Diesel 1200 3000 40.00

 Watch

## Video: Order of operations

Something to be aware of when using mathematical equations is the order of operations. This means the priority that is given to the mathematical operators in an equation. If you’re not sure what this is, watch this video.

[Introduction to order of operations](https://www.youtube.com/watch?v=ClYdw4d4OmA) (YouTube 09:39 mins)

[](https://www.youtube.com/embed/ClYdw4d4OmA?feature=oembed)

https://www.youtube.com/watch?v=ClYdw4d4OmA

## Operations on date and time values

The next query lists the details of those drivers whose accreditations were due to expire between the 25th December 2019 and the 1st January 2020 inclusive.

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

a.VehicleType\_ID,

a.ExpiryDate

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

AND a.ExpiryDate BETWEEN '2019-12-25' AND '2020-01-01';

Licence\_No FirstName LastName VehicleType\_ID ExpiryDate

---------- ---------- --------- -------------- ----------

2765GF John Smith 1002 2020-01-01

2765GF John Smith 1003 2020-01-01

2765GF John Smith 2003 2020-01-01

2765GF John Smith 3003 2020-01-01

75849857 Sharon Branson 12001 2019-12-26

75849857 Sharon Branson 12003 2019-12-30

84756748 Andrew Davidson 12003 2019-12-28

The next query derives the approximate age (in years) of each vehicle using the DATEDIFF() function to determine the difference between the vehicle's date of manufacture and the current date (using the GETDATE() function).

SELECT v.Registration\_No,

v.DateOfManufacture,

DATEDIFF(year, v.DateOfManufacture, GETDATE()) AS VehicleAgeYears

FROM VEHICLE AS v;

Registration\_No DateOfManufacture VehicleAgeYears

--------------- ----------------- ---------------

ADB435 2018-02-01 1

AWQ763 2017-07-12 2

DSF743 2016-09-08 3

EGD783 2019-09-04 0

FHT326 2008-12-01 11

IOU645 2017-10-07 2

JHF674 2007-01-09 12

NGJ946 2009-10-23 10

POR621 2016-09-08 3

TRT757 2019-04-07 0

UTY327 2016-09-08 3

VIV443 2008-12-01 11

YDF742 2003-06-14 16

YRN737 2019-04-03 0

The GETDATE() function returns the current system date. The DATEDIFF() function returns the difference between two dates – a start date and an end date – and has the following syntax:

DATEDIFF( datepart, startdate, enddate)

The DATEDIFF() function can also calculate the difference between two time values. For example, the difference in seconds between two given times could be determined as follows:

DATEDIFF( second, starttime, endtime)

A more accurate determination of the age can be achieved by calculating the difference in days and then dividing by 365.0 to return a decimal value, as demonstrated in the next query:

SELECT v.Registration\_No,

v.DateOfManufacture,

(DATEDIFF(day, v.DateOfManufacture, GETDATE()) / 365.0) AS VehicleAgeYears

FROM VEHICLE AS v;

Registration\_No DateOfManufacture VehicleAgeYears

--------------- ----------------- ----------------

ADB435 2018-02-01 1.873972

AWQ763 2017-07-12 2.432876

DSF743 2016-09-08 3.273972

EGD783 2019-09-04 0.284931

FHT326 2008-12-01 11.049315

IOU645 2017-10-07 2.194520

JHF674 2007-01-09 12.945205

NGJ946 2009-10-23 10.156164

POR621 2016-09-08 3.273972

TRT757 2019-04-07 0.695890

UTY327 2016-09-08 3.273972

VIV443 2008-12-01 11.049315

YDF742 2003-06-14 16.520547

YRN737 2019-04-03 0.706849

Now that’s too accurate. Use the FORMAT() function in the next query to format the output to two decimal places:

SELECT v.Registration\_No,

v.DateOfManufacture,  
 FORMAT((DATEDIFF(day, v.DateOfManufacture, GETDATE()) / 365.0), '###.##') AS   
 VehicleAgeYears  
FROM VEHICLE AS v;

Registration\_No DateOfManufacture VehicleAgeYears

--------------- ----------------- ---------------

ADB435 2018-02-01 1.87

AWQ763 2017-07-12 2.43

DSF743 2016-09-08 3.27

EGD783 2019-09-04 .28

FHT326 2008-12-01 11.05

IOU645 2017-10-07 2.19

JHF674 2007-01-09 12.95

NGJ946 2009-10-23 10.16

POR621 2016-09-08 3.27

TRT757 2019-04-07 .7

UTY327 2016-09-08 3.27

VIV443 2008-12-01 11.05

YDF742 2003-06-14 16.52

YRN737 2019-04-03 .71

You can list the day of the week that the vehicles were manufactured using the DATENAME() function. The DATENAME() function returns character data and has the following syntax:

DATENAME(datepart, date)

This is demonstrated in the next query:

SELECT v.Registration\_No,

v.DateOfManufacture,

DATENAME(dw, v.DateOfManufacture) AS DayOfTheWeekBuilt

FROM VEHICLE AS v;

Registration\_No DateOfManufacture DayOfTheWeekBuilt

--------------- ----------------- -----------------

ADB435 2018-02-01 Thursday

AWQ763 2017-07-12 Wednesday

DSF743 2016-09-08 Thursday

EGD783 2019-09-04 Wednesday

FHT326 2008-12-01 Monday

IOU645 2017-10-07 Saturday

JHF674 2007-01-09 Tuesday

NGJ946 2009-10-23 Friday

POR621 2016-09-08 Thursday

TRT757 2019-04-07 Sunday

UTY327 2016-09-08 Thursday

VIV443 2008-12-01 Monday

YDF742 2003-06-14 Saturday

YRN737 2019-04-03 Wednesday

The next query lists details of vehicles that received their maintenance on a Sunday.

SELECT v.Registration\_No,

v.OwningCompanyABN,

v.VehicleType\_ID,

v.DateOfLastMaintenance

FROM VEHICLE AS v

WHERE DATENAME(dw, DateOfLastMaintenance) = 'Sunday';

Registration\_No OwningCompanyABN VehicleType\_ID DateOfLastMaintenance

--------------- ----------------- -------------- ---------------------

TRT757 91876578373 12001 2019-11-10

VIV443 79834510388 8001 2019-06-09

You can format the output of date and time values listed in your query. There are many ways that you can achieve this. The first technique is to use the CONVERT() function, which provides some pre-defined styles for date and time output. The syntax for the CONVERT() function is as follows:

CONVERT(data\_type [( length ) ], expression [ , style ])

The next query converts the driver’s accreditation expiry date to the British/French date format of dd/mm/yyyy using a style value of 103:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

a.VehicleType\_ID,

CONVERT(NVARCHAR, a.ExpiryDate, 103) AS AccreditationExpiryDate

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

AND (d.Licence\_No = '43567482'

OR d.LastName = 'Livingston');

Licence\_No FirstName LastName VehicleType\_ID AccreditationExpiryDate

---------- --------- ------------------------- ------------------------

4433JK David Livingston 2004 03/03/2020

43567482 Nina Johansen 3003 10/06/2020

4433JK David Livingston 8001 03/03/2020

You can also use the FORMAT() function to control the output format of date and time values from your queries. The next query outputs the date in the format WeekdayName, MonthName dd, yyyy:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

a.VehicleType\_ID,

FORMAT(a.ExpiryDate, 'dddd, MMMM dd, yyyy') AS AccreditationExpiryDate

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

AND (d.Licence\_No = '43567482'

OR d.LastName = 'Livingston');

Licence\_No FirstName LastName VehicleType\_ID AccreditationExpiryDate

---------- --------------------- -------------- ------------------------

4433JK David Livingston 2004 Tuesday, March 03, 2020

43567482 Nina Johansen 3003 Wednesday, June 10, 2020

4433JK David Livingston 8001 Tuesday, March 03, 2020

## Operations on text values

You could achieve the same result (in a somewhat more convoluted way) using the DATENAME() function to extract the parts of a date value and then use the character concatenation operator (+) to build an expression that formats the date in the desired format.

You can also control the format of the driver’s name as demonstrated in the next query. Note the apostrophe (‘) in the alias for the [Driver’s Name]. Using an apostrophe for anything other than denoting character data (strings) in any code is courting catastrophe.

SELECT d.Licence\_No,

d.LastName + ', ' + d.FirstName AS [Driver's Name],

a.VehicleType\_ID,

DATENAME(dw, a.ExpiryDate) + ', ' +

DATENAME(MONTH, a.ExpiryDate) + ' ' +

DATENAME(dd, a.ExpiryDate) + ', ' +

DATENAME(year, a.ExpiryDate) AS AccreditationExpiryDate

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

AND (d.Licence\_No = '43567482'

OR d.LastName = 'Livingston');

Licence\_No Driver's Name VehicleType\_ID AccreditationExpiryDate

---------- ------------------ -------------- ------------------------

4433JK Livingston, David 2004 Tuesday, March 3, 2020

43567482 Johansen, Nina 3003 Wednesday, June 10, 2020

4433JK Livingston, David 8001 Tuesday, March 3, 2020

You can use conditional statements such as the CASE statement (often referred to in programming languages as a SELECT CASE or sometimes a SWITCH statement) to apply conditional control over the display of data. For example, the value of the VehicleType\_ID in the VEHICLE\_TYPE table has semantic values as follows:

‘1- - -’ = One Tonne Utility.

‘2- - -’ = Two Tonne Utility.

‘3- - -‘ = Three Tonne Tipper Truck.

‘8- - -‘ = Eight Tonne Single Axle Pantech Truck.

’12- - - ‘ = Twelve Tonne Articulated Semi-Trailer.

Since this has not been included in your table, you can write SQL to include this information using a CASE statement demonstrated in the next query. The syntax for the CASE statement is as follows:

CASE input\_expression

WHEN when\_expression THEN result\_expression [ ...n ]

[ ELSE else\_result\_expression ]

END

SELECT v.VehicleType\_ID,

CASE

WHEN v.VehicleType\_ID LIKE '1\_\_\_'

THEN 'One Tonne Utility'

WHEN v.VehicleType\_ID LIKE '2\_\_\_'

THEN 'Two Tonne Utility'

WHEN v.VehicleType\_ID LIKE '3\_\_\_'

THEN 'Three Tonne Tipper Truck'

WHEN v.VehicleType\_ID LIKE '8\_\_\_'

THEN 'Eight Tonne Single Axle Pantech Truck'

WHEN v.VehicleType\_ID LIKE '12\_\_\_'

THEN 'Twelve Tonne Articulated Semi-Trailer'

ELSE 'Unknown Vehicle Type'

END AS VehicleTypeDescription,

v.FuelType,

v.FuelCapacity,

v.[Range]

FROM VEHICLE\_TYPE v

VehicleType\_ID VehicleTypeDescription FuelType FuelCapacity Range

-------------- ------------------------------------- ------------- ------------ ------

1001 One Tonne Utility Petrol 40 600

1002 One Tonne Utility Petrol 40 600

1003 One Tonne Utility Diesel 50 900

1004 One Tonne Utility Hybrid/Petrol 20 1200

1005 One Tonne Utility Hybrid/Petrol 30 1500

2001 Two Tonne Utility Diesel 80 600

2002 Two Tonne Utility Diesel 100 800

2003 Two Tonne Utility Diesel 120 1200

2004 Two Tonne Utility Hybrid/Diesel 50 800

2005 Two Tonne Utility Hybrid/Diesel 80 1200

2006 Two Tonne Utility Hybrid/Diesel 80 1400

3001 Three Tonne Tipper Truck Diesel 110 900

3002 Three Tonne Tipper Truck Diesel 120 1000

3003 Three Tonne Tipper Truck Diesel 120 1100

8001 Eight Tonne Single Axle Pantech Truck Diesel 210 1500

8002 Eight Tonne Single Axle Pantech Truck Diesel 250 1800

8003 Eight Tonne Single Axle Pantech Truck Diesel 300 900

12001 Twelve Tonne Articulated Semi-Trailer Diesel 800 2400

12002 Twelve Tonne Articulated Semi-Trailer Diesel 1200 3000

12003 Twelve Tonne Articulated Semi-Trailer Diesel 1200 3500

Note the use of the underscore ( \_ ) with the LIKE logical operator above. The underscore denotes a place holder for which any single character can be substituted. Each string in the query above contains three underscores.

You can use an inline IIF statement (an Immediate IF) to return either of two arguments based on the results of the evaluation of a Boolean expression. The syntax for the IIF is as follows:

IIF ( Boolean\_expression, actionIFTrue, actionIFFalse)

If you wanted to check that the expiry date of the driver’s accreditations is not due for at least 14 days and display a message accordingly, you could use an IIF as indicated in the next query:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

a.ExpiryDate,

IIF ( DATEDIFF(DAY, '2019-12-18', a.ExpiryDate) <= 14, 'Accreditation due to expire within two weeks!', 'Not Due to expire.') AS AccreditationStatus

FROM ACCREDITATION a,

DRIVER d

WHERE a.Licence\_No = d.Licence\_No

Licence\_No FirstName LastName ExpiryDate AccreditationStatus

---------- ---------- ----------- ---------- ---------------------------------------------

2765GF John Smith 2020-01-01 Accreditation due to expire within two weeks!

7465SW Peter Anderton 2020-03-05 Not Due to expire.

2765GF John Smith 2020-01-01 Accreditation due to expire within two weeks!

7465SW Peter Anderton 2020-03-05 Not Due to expire.

2765GF John Smith 2020-01-01 Accreditation due to expire within two weeks!

7465SW Peter Anderton 2020-03-05 Not Due to expire.

3173BC Mary Dilberry 2020-05-16 Not Due to expire.

4433JK David Livingston 2020-03-03 Not Due to expire.

57463576 Barry Brown 2020-04-18 Not Due to expire.

2765GF John Smith 2020-01-01 Accreditation due to expire within two weeks!

3173BC Mary Dilberry 2020-05-16 Not Due to expire.

43567482 Nina Johansen 2020-06-10 Not Due to expire.

7465SW Peter Anderton 2020-03-05 Not Due to expire.

4433JK David Livingston 2020-03-03 Not Due to expire.

1593PD Donald White 2020-06-05 Not Due to expire.

57463289 Marium Hashemi 2020-03-04 Not Due to expire.

75849857 Sharon Branson 2019-12-26 Accreditation due to expire within two weeks!

1593PD Donald White 2020-06-05 Not Due to expire.

28476653 Yu Shoiwheng 2020-04-05 Not Due to expire.

75849857 Sharon Branson 2019-12-30 Accreditation due to expire within two weeks!

84756748 Andrew Davidson 2019-12-28 Accreditation due to expire within two weeks!

## Using aggregate functions

Aggregate functions perform calculations by grouping values from multiple rows into a single aggregated row. Calculations include average AVG, maximum MAX(), minimum MIN(), COUNT() and SUM().

You can list the lowest fuel consumption by using the MIN() aggregate function as demonstrated in the query below:

SELECT MIN(CAST((vt.FuelCapacity / ([Range] / 100.00)) AS decimal(5,2))) AS [Lowest Fuel Consumption Ltrs/100 kms]

FROM VEHICLE\_TYPE AS vt;

Lowest Fuel Consumption Ltrs/100 kms

------------------------------------

1.67

The next query lists the highest fuel consumption value using the MAX() aggregate function:

SELECT MAX(CAST((vt.FuelCapacity / ([Range] / 100.00)) AS decimal(5,2))) AS [Highest Fuel Consumption Ltrs/100 kms]

FROM VEHICLE\_TYPE AS vt;

Highest Fuel Consumption Ltrs/100 kms

---------------------------------------

40.00

You can count the number of distinct vehicle types using the COUNT() function as demonstrated in the next query:

SELECT COUNT(DISTINCT VehicleType\_ID) AS NoOfVehicleTypes

FROM VEHICLE\_TYPE;

NoOfVehicleTypes

----------------

20

The next query counts the number of drivers in the DRIVER table, again using the COUNT() function:

SELECT COUNT(\*) AS NoOfDrivers

FROM DRIVER;

NoOfDrivers

-----------

11

### The GROUP BY clause

If you wanted to know how many vehicle accreditations each driver has, you could join the DRIVER and ACCREDITATION tables and list the VehicleType\_IDs for each vehicle the driver is accredited to drive, as indicated in the query below:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

a.VehicleType\_ID

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

ORDER BY d.LastName;

Licence\_No FirstName LastName VehicleType\_ID

---------- --------- ---------- --------------

7465SW Peter Anderton 1002

7465SW Peter Anderton 1003

7465SW Peter Anderton 2003

7465SW Peter Anderton 3003

75849857 Sharon Branson 12001

75849857 Sharon Branson 12003

57463576 Barry Brown 2004

84756748 Andrew Davidson 12003

3173BC Mary Dilberry 2004

3173BC Mary Dilberry 3003

57463289 Marium Hashemi 12001

43567482 Nina Johansen 3003

4433JK David Livingston 2004

4433JK David Livingston 8001

28476653 Yu Shoiwheng 12003

2765GF John Smith 1002

2765GF John Smith 1003

2765GF John Smith 2003

2765GF John Smith 3003

1593PD Donald White 12001

1593PD Donald White 12002

You can see that Peter Anderton is accredited to drive four different vehicle types, Sharon Branson two, Barry Brown one and so on. You’ll also notice that the values in each of the three columns of the highlighted rows are all the same.

So, if you apply the COUNT() function to the VehicleType\_ID column and you collapse the values of the other three columns (using the GROUP BY clause) into a single row (which you can do because the values are the same for each row) you can list a single row for each driver with a count of how many vehicles they’re accredited to drive.

The GROUP BY clause is used in conjunction with aggregate functions to group rows that have the same values in each column.

Note: The general rule for the use of the GROUP BY clause is that any columns in the SELECT clause that are *not* being aggregated *must* be included in the GROUP BY clause. The next query demonstrates this rule:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

COUNT(a.VehicleType\_ID) AS NoOfVehicleAccreditations

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

GROUP BY d.Licence\_No,

d.FirstName,

d.LastName

ORDER BY d.LastName,   
NoOfVehicleAccreditations DESC;

Licence\_No FirstName LastName NoOfVehicleAccreditations

---------- ---------- ----------- -------------------------

7465SW Peter Anderton 4

75849857 Sharon Branson 2

57463576 Barry Brown 1

84756748 Andrew Davidson 1

3173BC Mary Dilberry 2

57463289 Marium Hashemi 1

43567482 Nina Johansen 1

4433JK David Livingston 2

28476653 Yu Shoiwheng 1

2765GF John Smith 4

1593PD Donald White 2

### Using the HAVING clause

Suppose you wanted to list the details of only those drivers that have more than two vehicle accreditations. The next query demonstrates the use of the HAVING clause to filter a subset of a sequence using a predicate:

SELECT d.Licence\_No,

d.FirstName,

d.LastName,

COUNT(a.VehicleType\_ID) AS NoOfVehicleAccreditations

FROM DRIVER AS d,

ACCREDITATION AS a

WHERE d.Licence\_No = a.Licence\_No

GROUP BY d.Licence\_No,

d.FirstName,

d.LastName

HAVING COUNT(a.VehicleType\_ID) > 2

ORDER BY d.LastName,

NoOfVehicleAccreditations DESC;

Licence\_No FirstName LastName NoOfVehicleAccreditations

---------- --------- --------- -------------------------

7465SW Peter Anderton 4

2765GF John Smith 4

The HAVING clause is used to filter the results produced by aggregated data or functions.

The query below demonstrates the use of the AVG() aggregate function to return the average age of the vehicles in the VEHICLE table.

SELECT AVG(DATEDIFF(year, v.DateOfManufacture, GETDATE())) AS AverageVehicleAge

FROM VEHICLE AS v

AverageVehicleAge

-----------------

5

Now, let’s count how many vehicles each courier company owns in the next query below:

SELECT c.ABN,

c.CompanyName,

COUNT(v.Registration\_No) AS NoOfVehiclesOwned

FROM COURIER\_COMPANY AS c,

VEHICLE AS v

WHERE c.ABN = v.OwningCompanyABN

GROUP BY c.ABN,

c.CompanyName

ORDER BY NoOfVehiclesOwned DESC;

ABN CompanyName NoOfVehiclesOwned

----------- ------------------------------- -----------------

79834510388 Rocket Couriers 6

17948738949 Lockwood Couriers & Taxi Trucks 5

91876578373 Bestever Transport 3

 Watch

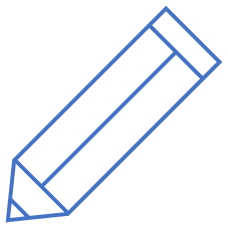
## Video: GROUP BY

Watch this video to learn about grouping rows using GROUP BY, filtering groups and difference between WHERE and HAVING clause in SQL Server.

[Group by in sql server - Part 11](https://youtu.be/FKSSOpQe5Jc) (YouTube 17:43 mins).

[](https://www.youtube.com/watch?v=FKSSOpQe5Jc)

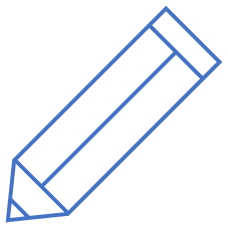
https://youtu.be/FKSSOpQe5Jc

 Practice activity

## Activity 5.1: Little Athletics scenario: Create and populate the database

For the following activities you will use the Little Athletics Database. Although you already created the database with some test data in previous activities, use the provided script (*CreateDatabaseLittleAthletics.sql* contained in the *Cl\_Database\_SW\_1of1\_SR1* folder*)* to create the Little Athletics Databasefor these activities.

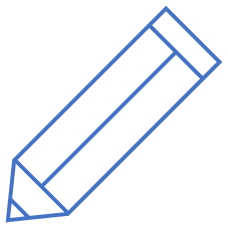
Load the SQL script *CreateDatabaseLittleAthletics.sql* into the SQL Server Management Studio IDE and execute the script to create the Little Athletics database. The script will create each of the tables for the database, the primary key constraints, the foreign key constraints and load the database with some data.

 Practice activity

## Activity 5.2: Single table queries

Using the Little Athletics database, write the SQL queries to perform the following:

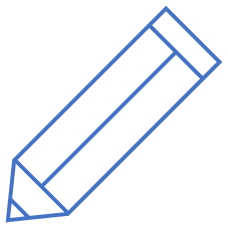
1. List the Carnival Name and Carnival Location of each Carnival.
2. List the Location Name of each Location in NSW.
3. List the first name and last name of each competitor who has the word ‘Street’ as part of their address.
4. List the first and last names, date of birth and suburb for each female competitor (females have a gender of ‘F’) who lives in either NSW or VIC.
5. List the first and last names and date of birth of each competitor who has cancelled their membership of Little Athletics. Sort the results into descending order on YearCancelled.
6. List the event type of each event. Do not list duplicate event types.
7. List the Event\_ID, Competitor\_ID, Place and ResultMetric for event entries with ResultMetric values between 3.71 and 4.12.
8. List the first and last names of all competitors who live in the ACT.
9. List the Competitor\_ID and first and last names of every Competitor who is still a member of Little Athletics and joined after 2018. Order by the Competitor\_ID. Note: YearCancelled is the year that the competitor cancelled their membership of Little Athletics.
10. List the first and last names and date of birth of each competitor who is no longer a member of Little Athletics but had joined in either 2010, 2012, 2014, or 2015.

 Practice activity

## Activity 5.3: Retrieving information from multiple tables

Write the SQL queries to perform the following:

1. List the Event\_id and EventType for each event, together with the EventDescription (*hint – join the EVENT and EVENT\_TYPE tables).*
2. List the first name and last name of each competitor who has been awarded a first place. Do not display the name of a competitor more than once.
3. For each Event that a competitor has entered, list the first and last names of the competitor together with the EventType and the competitor’s placing in that event (if any).
4. For each Event, list the EventType, the CarnivalName, the LocationName, the address (street, suburb, state, etc.) and the name and address of the judge judging the event.
5. For all events at the carnival named *‘Little Athletics ACT Championships’*, list the date of the carnival, the judge’s name and the EventType.
6. For each first place in an event, list the EventType and the ResultMetric for the first place.
7. List the Event\_id, Competitor\_ID, place and ResultMetric. Sort the results into order on Event\_id then Competitor\_ID.
8. For each event in which a first place has been won, list the name of the carnival, the EventType and the first place ResultMetric.
9. List the competitor’s first and last names, the carnival name, the EventType and place for all competitors in events judged by Deborah Donovan.
10. List the Event\_id and EventType for all events entered by Lilian Bradnock or Joelle Postan and judged by Miranda Morrison.

 Practice activity

## Activity 5.4: Retrieving information from multiple tables

Write the SQL queries to perform the following:

1. List the EventType of each event and the number of times that EventType was on.
2. List the name of each Carnival and the number of different times it has been held, in order of the highest numbers listed first.
3. List the number of judges grouped by first name.
4. For each Carnival, list the Carnival name, the start date and the number of events held at that Carnival.
5. For each Competitor who has won an event, list the Competitor’s first and last names, Competitor\_ID and their average ResultMetric.
6. For each Competitor who has an average ResultMetric of at least 110, list the Competitor’s first and last names, their Competitor\_ID and their average ResultMetric.
7. Produce a table similar to that below, showing, for first and second place, the number of times each was achieved, the total of the ResultMetric, the greatest ResultMetric, the smallest ResultMetric and the average ResultMetric.

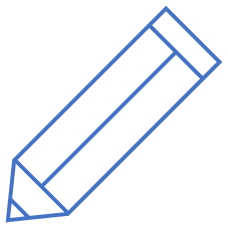
Place TimesAchieved TotalResultMetric Maximum Minimum Average

----- ------------- ----------------- ------- ------- -------

1 8 604.330 239.640 3.710 75.541250

2 8 608.680 240.320 3.820 76.085000

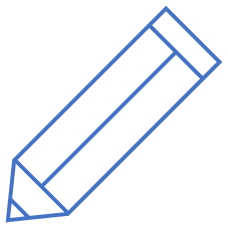
1. List the Competitor\_ID, first names and last names for each competitor who has been placed in the top two more than once.
2. List the first and last names of each competitor and the number of times they have competed in an event judged by Raymond Demarco of Leichardt, Sydney.

 Practice activity

## Activity 5.5: Working with dates

Write the SQL queries to perform the following:

1. List the Carnival Name of each Carnival together with the start date that it was held. Format the date into the exact format:  
   *Monday 13 July, 2020*
2. List the Event Type of each Event hosted at a Carnival held in 2020.
3. List the first and last names of each competitor, the Event Type and the Carnival Name that the Event was in and the competitor’s age at the time of the Carnival (subtracting the year that the competitor was born in from the year that the Carnival was held will give a reasonable approximation of the Competitor’s age at the time).
4. List the Event Type of each Event that has been held in January together with the number of times it was held in that month. Make sure your query uses “January” and not “1” to specify the month. Sort by the Event Types which have been held most often first.
5. List the Competitor\_ID, first and last names and the number of years between the year that the competitor joined Little Athletics (YearJoined) and 2020. Sort the results so that the competitor with the longest membership appears first.
6. List the Event\_ID and Event Type for any events that have been held on a Saturday.
7. List the number of days between the *Little Athletics SA State Championships 2020* held at *Kensington Oval* and the *Adelaide City Little Athletics Carnival 2020* held at the *Enfield Harriers Amateur Athletic Club* in 2019.
8. List the ID and first and last names of each competitor who has competed at a Carnival. Also list the number of years between their last competition and 2025.

 Practice activity

## Activity 5.6: Alias tables and other types of joins

Write the SQL queries to perform the following:

1. Produce a table similar to that below showing, for each event entry with at least a first, second and third placing, the Event\_ID and the ResultMetric for first, second and third, on one line.

Event\_ID First Second Third

-------- -------- ------- --------

1 12.500 12.550 12.700

2 17.200 17.250 17.500

6 67.940 67.990 68.020

10 3.710 3.820 4.120

28 48.090 48.910 49.290

29 37.140 37.250 37.590

30 178.110 180.590 185.240

31 239.640 240.320 242.590

1. Rewrite the previous query to produce a table similar to that below, which now includes the EventType, the name (concatenated first and last names) of the first, second and third place getters.

Event\_ID EventType FirstPlaceGetter First SecondPlaceGetter Second ThirdPlaceGetter Third

-------- ------------------------- ----------------- ------- ------------------- -------- ---------------- -------

1 70 metres Lilian Bradnock 12.500 Viola Reavell 12.550 Olympe Avraham 12.700

2 100 metres Viola Reavell 17.200 Lilian Bradnock 17.250 Olympe Avraham 17.500

6 400 metres Lilian Bradnock 67.940 Viola Reavell 67.990 Olympe Avraham 68.020

10 Long Jump (sand) Windy Adolthine 3.710 Jarret Emmens 3.820 Terrell Petrak 4.120

28 400 metres Jon Bingell 48.090 Juditha Tewkesbury 48.910 Colet Gilligan 49.290

29 300 metre hurdles (76cm) Wally MacKimm 37.140 Armin Anthes 37.250 Joelle Postan 37.590

30 700 metre Race walk Armin Anthes 178.110 Wally MacKimm 180.590 Hermann Cuchey 185.240

31 1500 metres Jon Bingell 239.640 Juditha Tewkesbury 240.320 Colet Gilligan 242.590

1. For each Judge, list their first and last names, the Event\_Id and EventType of each event they have judged. Include judges who have not judged any events.
2. List the Event\_ID, EventType, Competitor\_Id and first and last names of competitors who have notified Little Athletics of their intention to enter an event. i.e. competitors who have made an entry notification. Competitors who have never made any entry notifications should also be listed.
3. Produce a table similar to that below showing, for each event record, the Event\_Id, the EventType, the Place, the Competitor\_ID and the concatenated first and last names of the record holder.

Event\_ID EventType Place Competitor\_ID Record Holder's Name

-------- ------------------------ ----- ------------- --------------------

1 70 metres 1 32 Lilian Bradnock

2 100 metres 1 47 Viola Reavell

6 400 metres 1 32 Lilian Bradnock

10 Long Jump (sand) 1 37 Windy Adolthine

28 400 metres 1 21 Jon Bingell

29 300 metre hurdles (76cm) 1 10 Wally MacKimm

30 700 metre Race walk 1 45 Armin Anthes

31 1500 metres 1 21 Jon Bingell

# Summary

You have now completed the resources for the cluster Database, covering the units ICTDBS403 Create Basic Database, ICTPRG402 Apply Query Language and ICTSAD501 Model Data Objects.

In this workbook you learned about:

* analysing requirements
* database environments
* data modelling
* creating a database
* using SQL to query a database.

# Appendix 1: Little Athletics scenario

You have been commissioned by Little Athletics to build an information system to help manage the day-to-day operations of the organisation regarding their athletics carnivals. Little Athletics provides opportunities for young people to compete in athletics carnivals at various locations around Australia.

You have met with the senior management of the organisation and some of the carnival organisers and officials and they have discussed their requirements with you:

1. **Competitors**

The organisers would like to store contact information for each of the children who compete at Little Athletics carnivals, so that they can inform them of upcoming events. When an athlete decides that they’re going to compete in an event, they’ll notify the organisers and their entry will be recorded in the system. Organisers need to know the athlete’s full name, address, date of birth, gender, email, two phone numbers; home and mobile and the events in which they intend to compete. Most athletes will compete in more than one event. The competitors' date of birth will determine which age group they can compete in. The year that the competitor joined Little Athletics and the year that they cancelled their membership (if applicable) of Little Athletics must also be recorded.

At the time of the event, the athletes participation in the event is to be confirmed in the system. Up to 50% of athletes that notify Little Athletics of their intention to enter an event never actually compete in the event.

1. **Carnivals (often referred to as ‘Meets’)**

Details of each carnival must be emailed to the competitors at least two months prior to the carnival. Carnivals may be held for two or three days or could span a week (for state or national carnivals). The details include the location of the Carnival, the start date and the end date. Carnivals are held at the various athletics field locations around Australia. The location’s address and contact phone are provided to the competitors who have notified Little Athletics of their intention to enter an event. Athletics Carnivals could be one of several types, such as regional carnivals held in the various regions around Australia, State Carnivals held in each of the Australian States and National Championships, held at the national level.

1. **Events**

A carnival will host multiple events in the various disciplines. The types of events include, but are not limited to, high jump, 100m, 200m, 400m, 800m and 1500m running events, long jump, shotput, discus and hurdles. The organisers would like to be able to record a competitor’s result for an event, the type of event entered, the placing in that event and the competitor’s time/distance/height/weight achieved. The result is recorded as a numeric value. In addition to tracking the Athlete’s event entry results, Little Athletics keeps track of the National record for each type of Event. When a competitor breaks a record, the new record (time/distance/height achieved) is recorded for that event type, as is the name of the record holder.

1. **Age Groups**

Events are organised by age groups. The age groups begin at under 6’s and continue through to under 17’s in yearly increments. i.e. U6’s, U7’s, U8’s, U9’s, etc. The competitor’s age that they turn during a calendar year will determine which age group they can compete in. For example, if the competitor is born in 2010, they will turn the age of 10 during 2020, so they are eligible to compete in the Under 10 age group in 2020; this applies whether their birthday is 1 January or 31 December. If the competitor were born in 2014 they would be eligible to compete in the Under 6 age group during the 2020 year. Competitors must compete in their own age group only – they can’t compete in an older or younger age group.

# Appendix 2: Links

## Websites

Table Website URL

|  |  |
| --- | --- |
| Reference | URL |
| Introduction to Web Accessibility | <https://www.w3.org/WAI/fundamentals/accessibility-intro/> |
| Basic Data Structure Models Explained With a Common Example | <https://www.researchgate.net/publication/291448084_BASIC_DATA_STRUCTURE_MODELS_EXPLAINED_WITH_A_COMMON_EXAMPLE> |
| LucidChart | <https://www.lucidchart.com/pages/> |
| Chapter 27 - User-defined types | <https://crate.io/docs/sql-99/en/latest/chapters/27.html#type-predicate> |
| Chapter 12 - Reference types | <https://crate.io/docs/sql-99/en/latest/chapters/12.html> |
| Vertabelo.com | <https://www.vertabelo.com/> |
| SQL Power Architect | <http://www.bestofbi.com/downloads/architect/1.0.8/SQL-Power-Architect-Setup-Windows-jdbc-1.0.8.jar> |
| Ten Common Database Design Mistakes | <https://www.red-gate.com/simple-talk/sql/database-administration/ten-common-database-design-mistakes/> |
| Problem solving techniques: Steps and methods | <https://credentials.deakin.edu.au/problem-solving-techniques-steps-and-methods/> |
| Methods and Exercises for Effective Problem Solving | <https://medium.com/hygger-io/methods-and-exercises-for-effective-problem-solving-3503dfabbd02> |
| BULK INSERT | <https://docs.microsoft.com/en-us/sql/t-sql/statements/bulk-insert-transact-sql?view=sql-server-ver15> |
| LOAD DATA | <https://dev.mysql.com/doc/refman/8.0/en/load-data.html> |
| How to Import Data into SQL Tables | <https://data36.com/how-to-import-data-into-sql-tables/> |
| SQL TOP, LIMIT or ROWNUM Clause | <https://www.w3schools.com/sql/sql_top.asp> |

## LinkedIn Learning videos

Table : LinkedIn Learning videos

|  |  |
| --- | --- |
| Reference | URL |
| Effective listening and questioning techniques | https://www.linkedin.com/learning-login/ share?forceAccount=true&redirect=https%3A%2F%2F www.linkedin.com%2Flearning%2Fcollections %2F6585328392747446272%3Ftrk% 3Dshare\_collection\_url&account=57684225&auth=true |
| Communication Foundations | https://www.linkedin.com/learning/communication-foundations-2/welcome?u=57684225&auth=true |
| Software Design: Developing Effective Requirements | https://www.linkedin.com/learning/software-design-developing-effective-requirements?u=57684225&auth=true |
| Structuring formal requirements documents | https://www.linkedin.com/learning/project-management-foundations-requirements-2015/structuring-formal-requirements-documents?u=57684225&auth=true |
| Reviewing essential technical writing skills | https://www.linkedin.com/learning/project-management-foundations-requirements-2015/reviewing-essential-technical-writing-skills?u=57684225&auth=true |
| Understanding flat file databases | https://www.linkedin.com/learning/database-foundations-core-concepts/understanding-flat-file-databases?u=57684225 |
| What is cloud computing? | https://www.linkedin.com/learning/introduction-to-cloud-computing-for-it-pros/what-is-cloud-computing?u=57684225 |
| Overview of storage as a service | https://www.linkedin.com/learning/introduction-to-cloud-computing-for-it-pros/overview-of-storage-as-a-service?u=57684225 |
| Introduction to XML | https://www.linkedin.com/learning/microsoft-sql-server-2016-essential-training/introduction-to-xml?u=57684225 |
| Understanding hierarchical databases | https://www.linkedin.com/learning/database-foundations-core-concepts/understanding-hierarchical-databases?u=57684225 |
| Data mining | https://www.linkedin.com/learning/big-data-in-the-age-of-ai/data-mining?u=57684225 |
| 2. Fundamentals of Data Understanding | https://www.linkedin.com/learning/learning-data-analytics-2/learning-to-identify-data?u=57684225&auth=true |
| Introduction to data modelling | https://www.linkedin.com/learning/filemaker-relational-database-design/introduction-to-data-modeling?u=57684225 |
| Understanding business rules | https://www.linkedin.com/learning/learning-data-analytics-2/understanding-business-rules?u=57684225&auth=true |
| Importing data | https://www.linkedin.com/learning/sql-server-2014-essential-training/importing-data?u=57684225&auth=true |

# Image attributions

Table Copyright and attributions

|  |  |  |
| --- | --- | --- |
| Image | Page # | Attribution |
| Cover | 1 | © Getty Images copied under licence  Credit: Colin Anderson Productions pty ltd  Creative #: [532029221](https://www.gettyimages.com.au/detail/photo/businessman-using-computer-in-virtual-server-room-royalty-free-image/532029221?adppopup=true) |
| Topic 1 cover | 9 | © Getty Images copied under licence  Credit: Thomas Barwick Creative #:[481347421](https://www.gettyimages.com.au/detail/photo/coworkers-discussing-project-on-digital-tablet-royalty-free-image/481347421?adppopup=true)Credit: Thomas Barwick Creative #:[481347421](https://www.gettyimages.com.au/detail/photo/coworkers-discussing-project-on-digital-tablet-royalty-free-image/481347421?adppopup=true) |
| Topic 2 cover | 24 | © Getty Images copied under licence  Credit: Laurence Dutton Creative #: [885690024](https://www.gettyimages.com.au/detail/photo/touch-screen-analyser-royalty-free-image/885690024?adppopup=true) |
| Topic 3 cover | 39 | © Getty Images copied under licence  Credit: Maskot Creative #: [1159379063](https://www.gettyimages.com.au/detail/photo/high-angle-view-of-female-it-expert-working-on-royalty-free-image/1159379063?adppopup=true)Credit: Maskot Creative #: [1159379063](https://www.gettyimages.com.au/detail/photo/high-angle-view-of-female-it-expert-working-on-royalty-free-image/1159379063?adppopup=true) |
| Topic 4 cover | 97 | © Getty Images copied under licence  Credit: 10'000 Hours Creative #: [1084167640](https://www.gettyimages.com.au/detail/photo/open-plan-office-with-people-working-at-desks-royalty-free-image/1084167640?adppopup=true) |
| Topic 5 cover | 126 | © Getty Images copied under licence  Credit: mirsad sarajlic Creative #: [1154381413](https://www.gettyimages.com.au/detail/photo/sql-programming-language-royalty-free-image/1154381413?adppopup=true) |
| All other figures |  | © TAFE NSW 2019 |