SCHOLAR Study Guide

Advanced Higher Computing Science Database design and development

Authored by:

Charlie Love (CompEdNet)

Andy McSwan (Knox Academy)

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2020 by Heriot-Watt University.

This edition published in 2020 by Heriot-Watt University SCHOLAR.

Distributed by the SCHOLAR Forum.

SCHOLAR Study Guide Advanced Higher Computing Science: Database design and development

Advanced Higher Computing Science Course Code: C816 77

Print Production and Fulfilment in UK by Print Trail www.printtrail.com

Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who contributed to the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

The Scottish Qualifications Authority for permission to use Past Papers assessments.

The Scottish Government for financial support.

The content of this Study Guide is aligned to the Scottish Qualifications Authority (SQA) curriculum.

Contents

Topic 1

Analysis

Contents

Prerequisites

From your studies at Higher you should already know how to:

- Identify the end-user requirements of a program, database or website as relates to the design and implementation at Higher level.
- Identify the functional requirements of a program, database or website as relates to the design and implementation at Higher level.

Learning objective

By the end of this topic you should be able to:

- Identify the purpose of a software solution that relates to the design and implementation at this level
- Describe, exemplify, and implement research for:
  - Feasibility studies:
    - Economic
    - Time
    - Legal
    - Technical
  - User surveys
- Describe, exemplify, and implement planning in terms of:
  - Scheduling
  - Resources
  - Gantt charts
- Produce requirement specifications for end-users and develop:
  - End-user requirements
  - Scope, boundaries and constraints
  - Functional requirements in terms of
    - inputs
    - processes
    - processes
- Describe, exemplify, and implement Unified Modelling Language (UML):
  - Use case diagrams:
    - Actors
    - Use cases
    - Relationships

You may have already covered the material in this topic as it applies to Software, Database and Web design and development.

1.1 Introduction

Analysis is the process of understanding what it is that the client, users and stakeholders want from the system that is being developed. There needs to be a detailed list of the requirements for the product. This will form the basis of a contract between the developer and the client and will be legally binding.

We will create the following at the analysis stage:

- A Requirements Specification
- End User Requirements
- The Scope and Boundaries of the project
- A list of Constraints on the project
- The Functional Requirements for the Project

1.1.1 Requirement Specifications

The requirements specification details the scope, boundaries and constraints of the intended software, details the basis of payment for the work to be completed and sets out how the software will be designed, tested, documented and evaluated before hand over to the customer. It may also detail any longer term maintenance agreement between the customer and the developer.

1.1.2 End user requirements

In the Higher Computing Science course we looked at how the end user requirements would be generated by using **Personas**, **User Stories**, **User Scenarios**, **Use Case**, User Devices and Software.

At Advanced Higher level can use these techniques to help generate a list of what the users would expect from the software being created.

1.1.3 Scope and boundaries

The Scope of the project is held in the **product backlog** - it is the definition of the features that the product must have - also known as the requirements of the software. If a feature is no longer required then it is dropped from the product backlog because it is out of scope.

Constraints are limitations that affect the development of the product. A project may be constrained by time (so only a certain number of development sprints can be completed), or may be limited by budget or may be limited by legal position regarding the data that it processes.

It is very important at the outset to establish clearly the scope, boundaries and constraints of the project. Scope and boundaries are opposite sides of the same "coin". Between them, they give a precise description of the extent of the project.

Here is a typical statement about scope and boundaries. You will find similar statements by

searching the web. "The project scope states what will and will not be included as part of the project. Scope provides a common understanding of the project for all stakeholders by defining the project's overall boundaries."

One way of thinking about it is:

- the scope clarifies what the project must cover;
- the boundaries clarify what the project will not cover.

For example, suppose your project was to develop an expert system giving students guidance on job opportunities which they should consider after graduating from University.

The scope of the project would be to create an expert system. Then it would be necessary to describe the range of jobs and degrees that would be included in the system, the level of information that would be output by the system (does it suggest contact addresses as well as simply job types), the types of questions that the user will be asked. Does it cover all degrees, or is it only for students with Computing Science degrees, and so on ... All these things will define the boundaries of the system.

Sometimes it is also helpful to spell out exactly what will NOT be covered. So, for example, a clear statement could be made which states that the system will NOT cover advice on jobs for those with medical and veterinary degrees, or jobs overseas.

The scope and boundaries could also refer to technical issues. For example, they might state that the resultant system will run on any computer capable of running any version of Windows after Windows 7, but not on any other operating system.

Defining scopes and boundaries

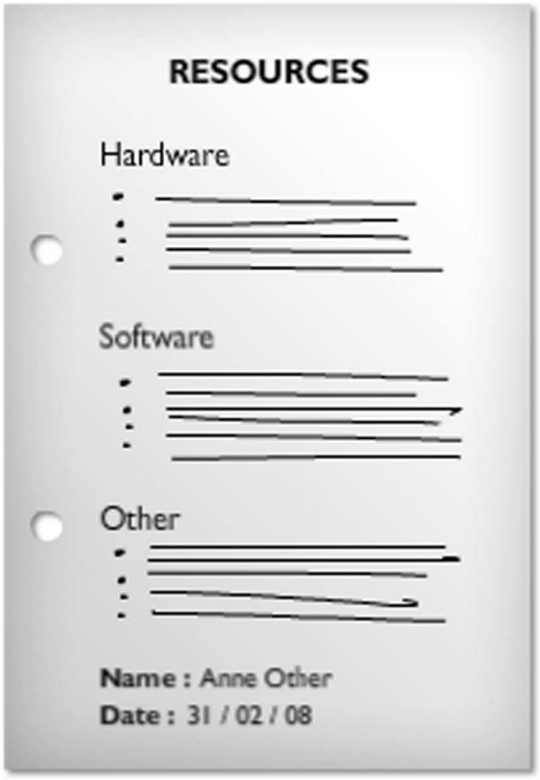
Now, starting from your project proposal, create clear statement of scope and boundaries for your project. Hint: write this as a bulleted list, rather than as a paragraph. This has two benefits - it helps you to clarify your ideas, and it gives you a clear list to use at the end of your project when you are evaluating what you have produced.

Discuss this with your tutor.

Add the scope and boundaries list to your Record of Work.

Don't forget to put your name/initials and the date on the page.

Your Record of Work should now include:



Scope

Examples of scopes for a modular program and a relational database

Scope of a programming task

This development involves creating a modular program.
The deliverables include:

- a detailed design of the program structure
- a test plan with a completed test data table
- a working program
- the results of testing
- an evaluation report

Scope of a relational database task

This development involves creating a relational database.
The deliverables include:

- a detailed design of the database structure
- a test plan with a completed test data table

- a working database
- the results of testing
- an evaluation report

Scope of a website task

This development involves creating a multi-level website.
The deliverables include:

- detailed wireframe designs of each page of the website
- a test plan with a completed test data table
- a working website
- a working website
- an evaluation report

1.1.4 Constraints

Constraints are limitations that affect the development of a product.
A project may be constrained by:

- Time (so only a certain number of development sprints can be completed)
- The scope of the project i.e. what the end result of the project will be and what should the developer deliver (project deliverables) to the client?
- Limited by budget (Cost)
- Legal position regarding the data that it processes i.e. What country will the stored data be held in and does it comply with GDPR

At this stage we should detail what operating systems the project will work on (Windows, iOS, Android, Linux, etc. . .), what environment will be used for development

1.1.5 Functional Requirements

Functional requirements are defined as what the product should do and are split into three sections — Input, Process and Output.

Consider this example of a forum website which allows users to create a user account.

Functional requirements

The new user page should present the user with a form enabling them to create a new account by entering username and email address. On submitting the form, the user receives a welcome email containing a link to confirm the account.

This link should take them to their account details where they can perform the following tasks:

- Inputs
  - enter /change personal information;
  - upload an avatar image;
  - change forum preferences;
  - change password;
  - delete account;
  - confirm their account details by pressing a submit button.
- Process
  - The welcome email will warn the user that their account will expire if they do not go to the link and confirm their details within a certain time period.
  - The site should reject usernames or email addresses which belong to existing users.
  - The site should reject obscene or racist usernames.
  - The site should be secure.
  - If the user details are not submitted within 48 hours then user is deleted from the user database.
  - The page contains a form with text entry boxes with maximum and minimum size restrictions.
  - Client side validation is used for email address and date of birth.
  - The username text box entry is a required one, with a minimum size of 8 characters and a maximum of 16.
  - The Email text box is also required and uses client side validation to check for a valid email address.
  - The page allows an avatar upload restricted to 1Mb. The default avatar is allocated if no upload occurs.
  - Forum options are check boxes.
  - On submit, the form data is passed to a PHP script which sanitizes the data then the username and email address are checked against existing entries in the user database. Usernames are also checked against the database table containing unacceptable values and an appropriate rejection advice response is sent if a match is found in either case. If the username and email are unique, a new database entry is created for that user and a welcome email is sent including a link to the user configuration page using a further PHP script.
  - On submit, the form data is passed to a PHP script where it is sanitized and the user database is updated.
- Outputs
  - Confirmation of successful
    - account creation
    - change of password
    - deletion of account
  - Database reports will be available to administrators.

1.2 Research: feasibility study, user surveys

Learning objective

By the end of this section you will be able to:

- explain the different types of feasibility;
- explain the use of users surveys to gather information about a project.

Research is a vital part of the initial analysis of what the project sponsor is requesting.

1.2.1 Feasibility study

A key part of the project initialisation is undertaking research to ensure that the project is feasible. Software development projects, depending on their size, can be in development for many years and can have budgets worth millions of pounds. It would be foolish to proceed with any project without assessing if completing it is possible.

The basic purpose of a feasibility study is to work out if the proposed expenditure of time and money is likely to be worthwhile and whether the objectives of the project can be achieved: what some projects set out to do might be totally unrealistic.

The results of the feasibility study will determine which, if any, of a number of possible solutions can be further developed at the design phase.

One result of undertaking a feasibility study may indicate that only a simple solution is required to solve the problem:

- a software fault may be identified that is easily fixed;
- more staff training might be required if particular software is to be used.

The feasibility study is most often carried out by the **project leader** - an experienced member of staff who is able to understand the needs of the project and the various aspects of feasibility that apply to it.

1.2.2 Types of feasibility

A feasibility study will look at four main areas of feasibility: Technical (is it technically possible to create a solution with the available technology), Economic (is it possible to complete the project with the budget available or is the cost of creating the project justified by the financial reward of doing so), Legal (can the solution be created and adhere to existing laws) and Schedule (is there enough time to complete the project, are the right people and resources available when required to deliver the project on time).

1.2.2.1 Technical feasibility

The feasibility study must ascertain what technologies are necessary for the proposed system to work as it should. It may be the case that suitably advanced technology does not yet exist. Unless it is the object of the project to design a system to use such advanced technology, this would rule the project out as being a non-starter. It would be a foolish move for a feasibility study to evaluate

technologies which are either under development or undergoing testing.

Given that suitable technology does exist, the study must establish if the organisation already has the necessary resources. If not, the study must make clear what new resources the organisation would have to acquire. This will also involve determining whether the hardware and software recommended will operate effectively under the proposed workload and in the proposed environmental conditions. The development of a new system involves risks of one kind or another. Every understanding that might be reached could carry the risk of some misunderstanding:

- software companies and their clients often have different vocabularies and consequently they appear to be in perfect agreement until the finished product is supplied;
- management may have unrealistic expectations of computer systems. The feasibility study is where idealism meets reality.

Further issues might include the training of personnel to use the new system, consideration of service contracts, warranty conditions and the establishing of help desk facilities for inexperienced users.

1.2.2.2 Economic feasibility

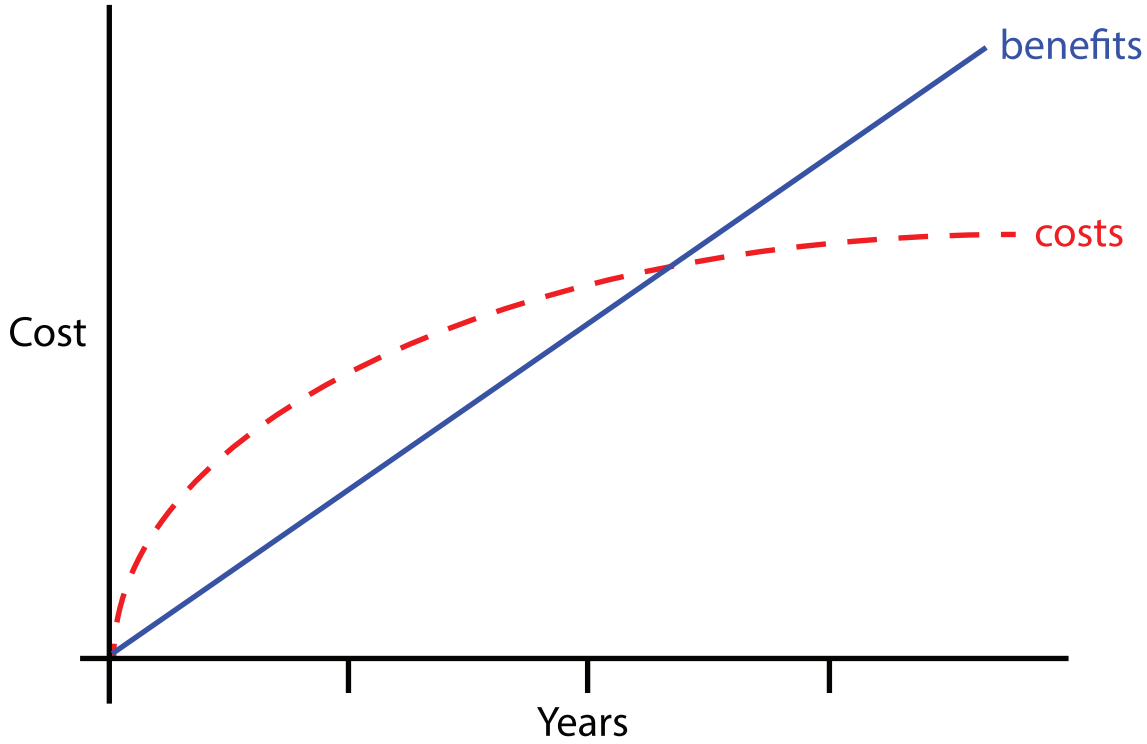
This deals with the cost implications involved. Management will want to know how much each option will cost, what is affordable within the company's budget and what they get for their money. A cost-benefit-analysis is part of the budgetary feasibility study. If the project is not cost-effective then there is no point proceeding.

Setting up a new computer system is an investment and involves capital outlay. The costs of a new system include the costs of acquiring it in the first place (consultancy fees, program development, etc.); the costs of installing it (disruption of current operations, cost of new equipment, alteration of workplace, etc.); and the costs of maintaining it which also includes training.

In the long term management will also want to know the 'break-even point' when the new system stops costing money and starts to make money. This is extremely difficult to quantify. However an accurate estimate of a system's operational life span is a valid option and will rely solely on the knowledge and experience of the systems analyst involved.

Figure 1.1 depicts such a case where the break-even point is at the intersection of the graphs:

Figure 1.1: Break-even point



Tangible benefits that management would certainly be looking for in the new system would be:

- reduced running costs;
- increased operational speed;
- increased throughput of work;
- better reporting facilities.

Note that not all the costs and benefits lend themselves to direct measurement. For example, new systems generally affect the morale of the staff involved, for good or ill. This can only be resolved by competent personnel management practices.

1.2.2.3 Legal feasibility

This has to do with any conflicts that might arise between the proposed system and legal requirements: how would the new system affect contracts and liability, are health and safety issues in place and would the system be legal under such local laws as the UK Data Protection Act? What are the software licensing implications for the new system?

Software licensing can be quite a thorny problem. Licences can be purchased as: client licence (per seat), server licence, network licence or site licence and the period of operation may be annual or perpetual. Software vendors vary in their licensing regulations so this has to be fully investigated.

1.2.2.4 Schedule feasibility

Schedule feasibility may be assessed as part of technical feasibility. Most organisations have an annual schedule of events such as the AGM, end of financial year, main holiday period and so on. Obviously time is a main factor in the development of a new system.

Questions to be asked at this stage might include:

- how long will the proposed system take to develop?
- will it be ready within the specified time-frame?
- when is the best time to install?

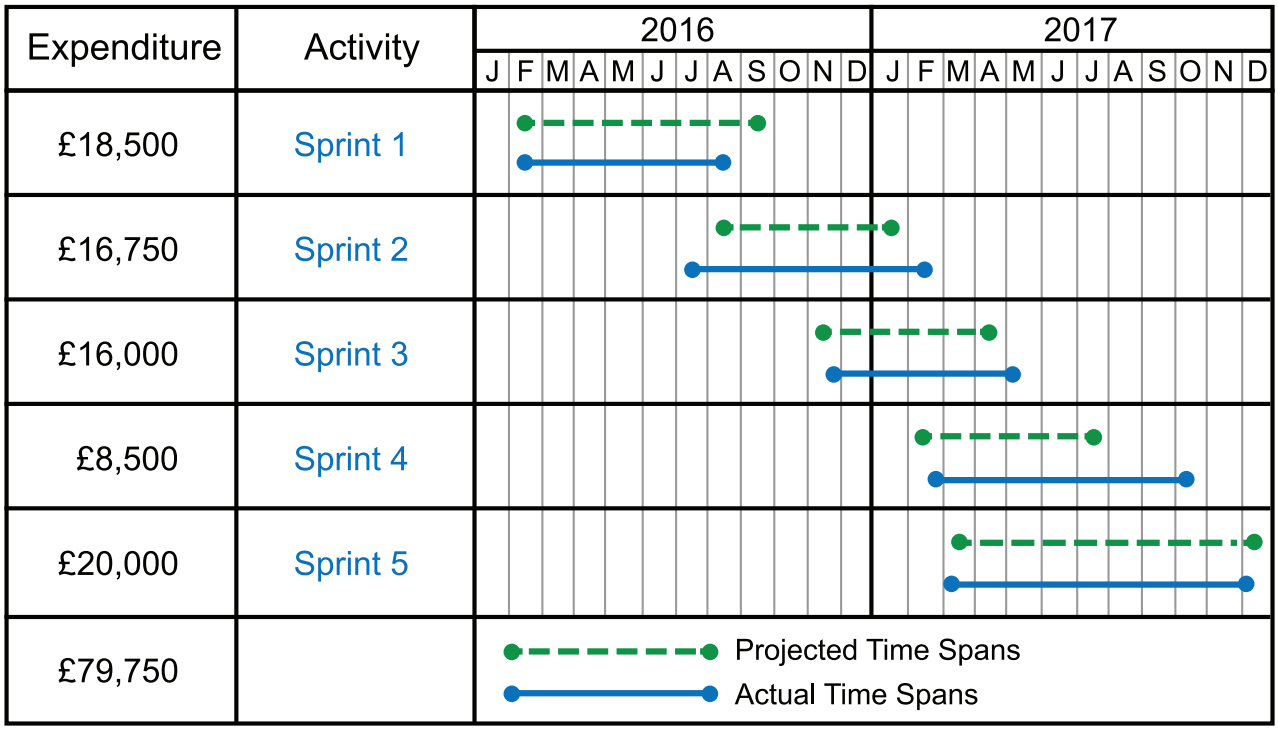
For example, a project might have to start within six months; assuming it would take three months to purchase and install the required hardware and software and a further six months to train the end users. Such a project is not technically feasible because of shortage of time so it would not go ahead unless some of the time constraints were reviewed and changed.

In many cases of project management the scheduling component can be aided by means of a Gantt chart.

A Gantt chart is like a horizontal bar graph used to plan and schedule projects involving several concurrent tasks. The horizontal axis represents the time scale and start and finish times of component parts are graphically represented.

The advantage of a Gantt chart is that it shows, at a glance, the progress of a project as shown in Figure 1.2

Figure 1.2: Example of a Gantt chart



You will look more closely at the creation of a Gantt chart and the concepts of Project management in the Project management Topic of this course.

Activity: Types of feasibility Go online 

Q1:

Match the parts together to generate true statements:

1. A project that cannot be delivered in the time available because of the complexity of the task
2. A project which has insufficient finance to hire the necessary staff to complete the project when required
3. A project that processes banking transactions to indicate spending habits in contravention of data protection laws
4. A project which cannot proceed because the software to link two data sources into the project is not available

a) has an issue with technical feasibility.
b) has an issue with legal feasibility.
c) has an economic feasibility issue.
d) has an issue with schedule feasibility.

1.2.3 User surveys

Often with the development of software, the experience of existing users is vital in shaping what the new software should do better than the old software. The users of a particular program are the best people to identify the existing problems with the software and how it could be improved.

Equally, if the software being developed is entirely new, the target user group should, generally, understand the process that the programme will carry out. For example, a new library system, which is to replace a manual card based system, would be developed following some time spent capturing the experience of existing librarians and how they operate.

When the opinion of a number of users is required, it is often easier to create a survey or questionnaire which will capture the information that the project team require.

1.3 Planning

Project management is the process of planning and controlling the activities of a project to ensure that it is delivered on time, with the required features and within the budget available to the required quality.

The Association Of Project Management define project management as:

"Project management is the application of processes, methods, knowledge, skills and experience to achieve the project objectives."

Projects are different from the day-to-day business of an organisation - a project is a specific piece of work which is defined by what it attempts to achieve. A project is usually judged to be successful if it achieves the **objectives** (normally according to some **acceptance criteria**) within an agreed

timescale and **budget**.

A project plan is more than just a list of tasks to be completed. It is a plan that details individual responsibilities, resources available to complete work, the order of work and the key **milestones**.

1.3.1 Relationship between time, cost, scope and quality

Learning objective

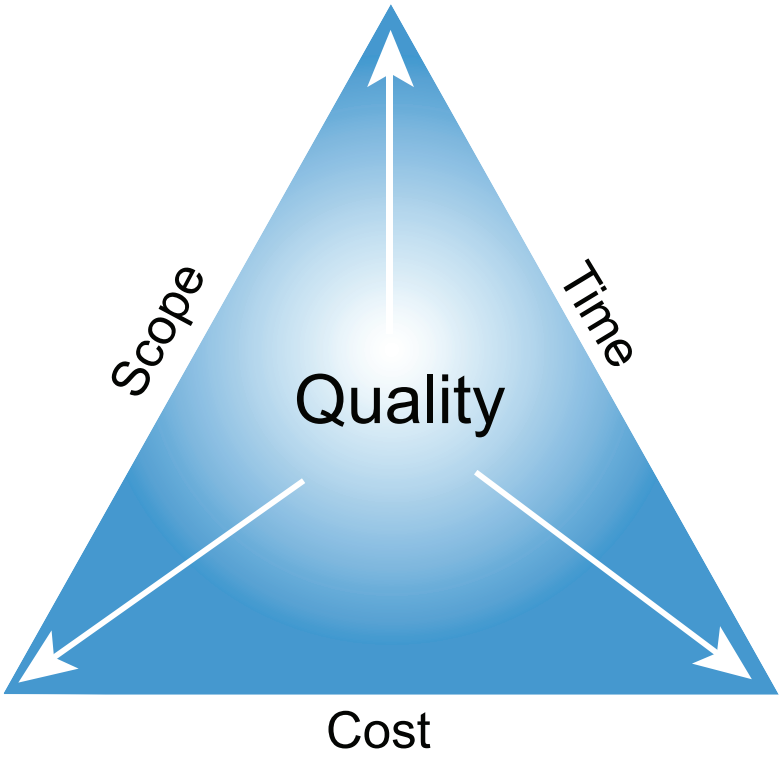
By the end of this section you will be able to:

- describe the relationship between time, cost, scope and quality in the project "triangle".

Before looking more closely at the process of creating a simple project plan, it is important to understand the relationship between **time**, **cost**, **scope** and **quality** when planning a project.

These are the **constraints** of the project. Changes to any one of these constraints has an immediate impact on the others.

Figure 1.3: The Project management triangle



The **Scope** of a project details what needs to be produced and delivered. In the case of a software development project, the system requirements will give us this information. With the **waterfall model** of software development, the requirements of the project are completely known after the analysis phase. Any subsequent changes to the scope will add **time** and/or **cost** to the project because more time and/or more resources (people who have to be paid or over time hours for existing staff) will be required to deliver the project to the same required level of **quality**.

Time is the amount of time it takes to complete the project, to the required **scope** with the available **budget** (cost) to the required quality.

Cost is the budget for the project. If the budget changes then less or more can done. An increase in budget may increase the quality of the work to be done, allow more to be achieved (increased scope) or for the work to be completed in less time.

In a project that uses an **agile Scrum** approach, the time and cost constraints are generally fixed, because the agreement between the developer and the client is for a specific fee for work completed. The reality of large and medium scale projects is that it is almost impossible to understand all of the requirements for the software at the start of the project - using an agile approach reduces the risk to the client as they can change the scope of the project by removing requirements which are no longer needed, adding new ones and by changing the priority of the existing requirements as the project proceeds. This evolving list of requirements is held in the product backlog and is updated prior to each **sprint** in the agile process.

Typically, **Scrum** projects do not have a formal **Project manager** as the team is self-organising and supported by the ScrumMaster, however, in large scale Scrum projects, where there are multiple teams working to produce elements of the project, the role of project manager to act as a buffer between outside organisations and the teams can be useful. This project manager may construct a high-level project management plan based on the **product backlog** and maintain this to estimate completion of the project.

1.4 Unified Modelling Language (UML)

Learning objective

By the end of this section you should be able to use the following design notation associated with programming paradigms:

- Unified Modelling Language (object-oriented programming).

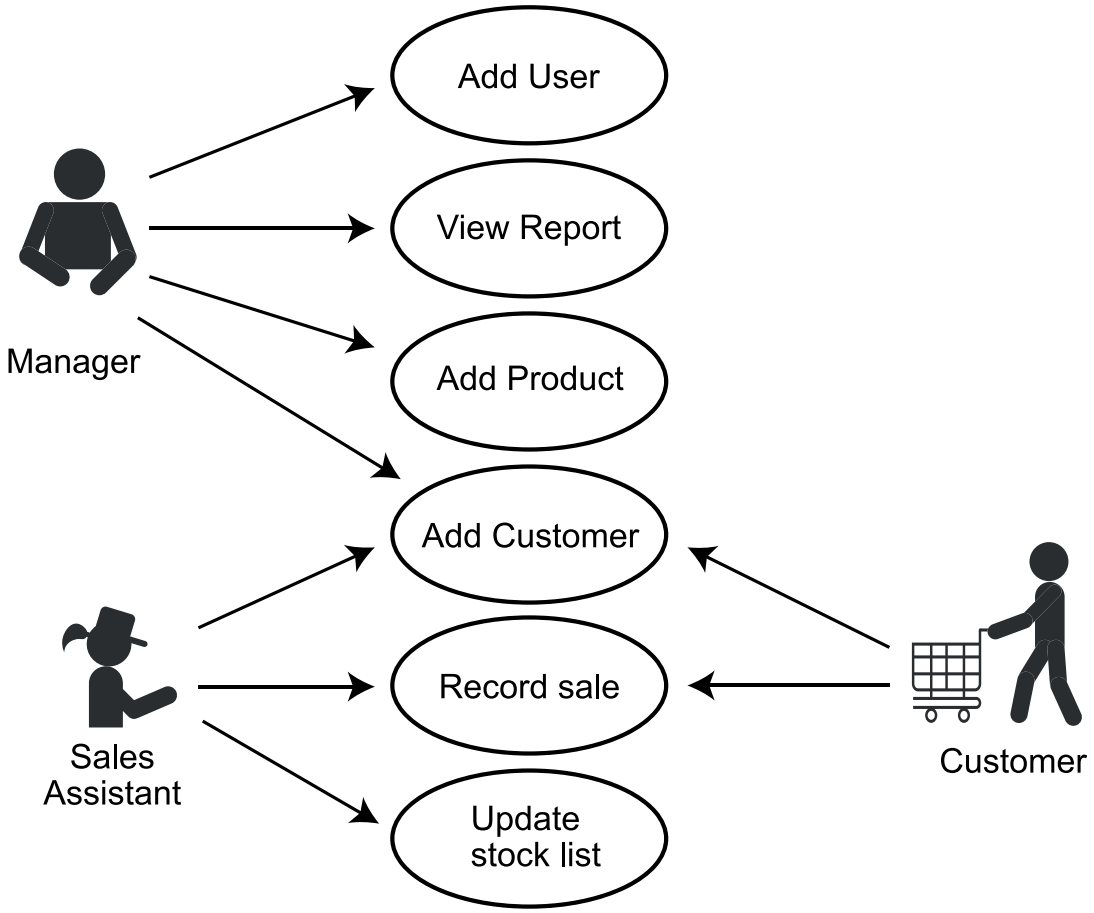
UML is a design notation created specifically to help programmers design systems using the object oriented programming paradigm. As its name suggests it is a formal system for creating models for software design. UML provides a specification for a number of different types of diagram which can be used to represent a software system. Just like pseudocode, UML is programming language independent - it does not dictate which language will be used to translate its diagrams into source code. UML tells you what model elements and diagrams are available and the rules associated with them. It does not tell you what diagrams to create.

There are a number of different types of diagrams specified by UML however for the analysis topic we will only look at **Use Case diagrams**. UML Class Diagrams will be looked at in the design stage of Software Development.

1.4.1 Use Case diagrams

Use case diagrams contain **Actors** (the people or entities who interact with the system) and **Use Cases** which are the procedures which they interact with.

The **Use Cases** in the diagram are identified by the circled text and the **Actor's** interactions with them are identified by the arrows.



A large system will have several different actors. This example might be part of a sales recording system in a shop.

The first stage in creating a Use Case diagram is identifying the actors, by classifying the people who interact with the system, in this case the managers, sales assistants and customers.

The next stage is to identify the data which the actors will interact with and how they change it. The scenario above assumes that the system stores the following data:

- A stock table which stores details of items in stock and their quantities.
- A customer table which stores customer details including previous purchases.
- A staff table which stores staff details and their access privileges to the system.

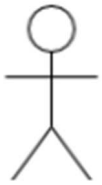
The Customer is involved when a sale is made. If they are a new customer then their details are added to the customer table. When they make a purchase the sale is added to their details.

The Sales assistant also interacts with the customer table but also needs to update the stock list when a sale is made.

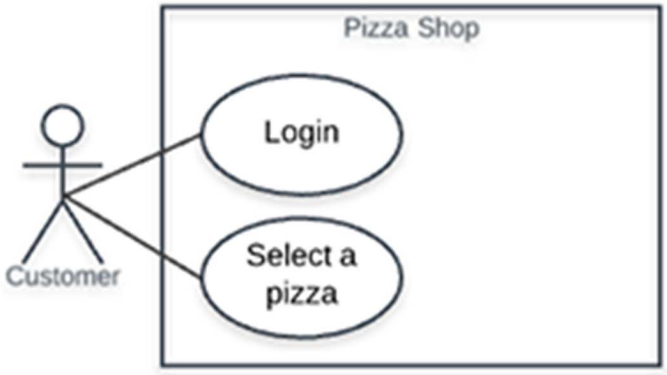
The manager needs to be able to update the staff table if a new sales assistant joins the company, add new products to the stock table and generate reports of sales.

Use case diagrams contain Actors (the people or entities who interact with the system), Use Cases which are the procedures/actions which the user will interact with and the relationships that exist between actors and use cases.

Actors:



Actors should be general categories of users for a system, i.e. customer, staff or another system that it may need to interact with (i.e. a database that the website needs to connect to). We depict an actor in a Use Case Diagram by using a stick figure and putting the type of actor underneath. Each actor must have at least one use case associated with it.



There are two types of actors in use case diagrams, primary and secondary. The primary actors are the ones who start the system. These actors are placed on the left side of the system boundary.

The secondary actors are those users who react to something happening in the system and are displayed on the right side of the system boundary.

Use Cases:

The Use Cases in the diagram are identified by an ellipse with text in the middle explain a task the system is to do. Each individual action that the user can perform in the system will get it's own use case.

Relationships:

- Association
- Include
- Extend
- Generalisation of an Actor
- Generalisation of a Use Case

Association— Solid line shows a basic communication between the actors and the system

Include:

- This is a use case that is not directly accessed by the actors, it shows a dependency between

a **Base Use Case** and a **Dependent Use Case**.

- It is indicated by a dashed line with an arrow that points to the **Dependent Use Case** with the word include.
- The relationship should also be identified in the diagram using double angle brackets and the word <<**include**>>.

Extend:

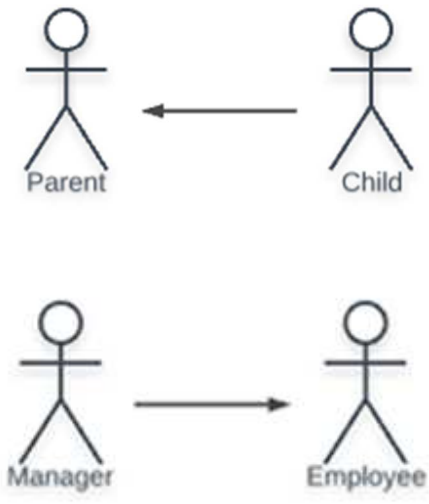
- Similar to an include the **Extend Use Case** will not be directly accessed by an actor and will have a **Base Use Case** but with this relationship the **Extend Use Case** may not always be actioned, certain criteria needs to be met before it can be ran.
- This is also indicated by a dashed line and an arrowhead but this time it will point to the **Base Use Case**.
- The relationship should also be identified in the diagram using double angle brackets and the word <<**extend**>>.

Generalisation (inheritance)

There are two types of **Generalisation** we need to look at, generalisation of actors and generalisation of use cases.

Generalisation of Actors

For actors generalisation uses the idea of inheritance to allow new actor (child) who need to make use of the same basic use cases as the parent actor and can then be given access to any additional use cases they need.

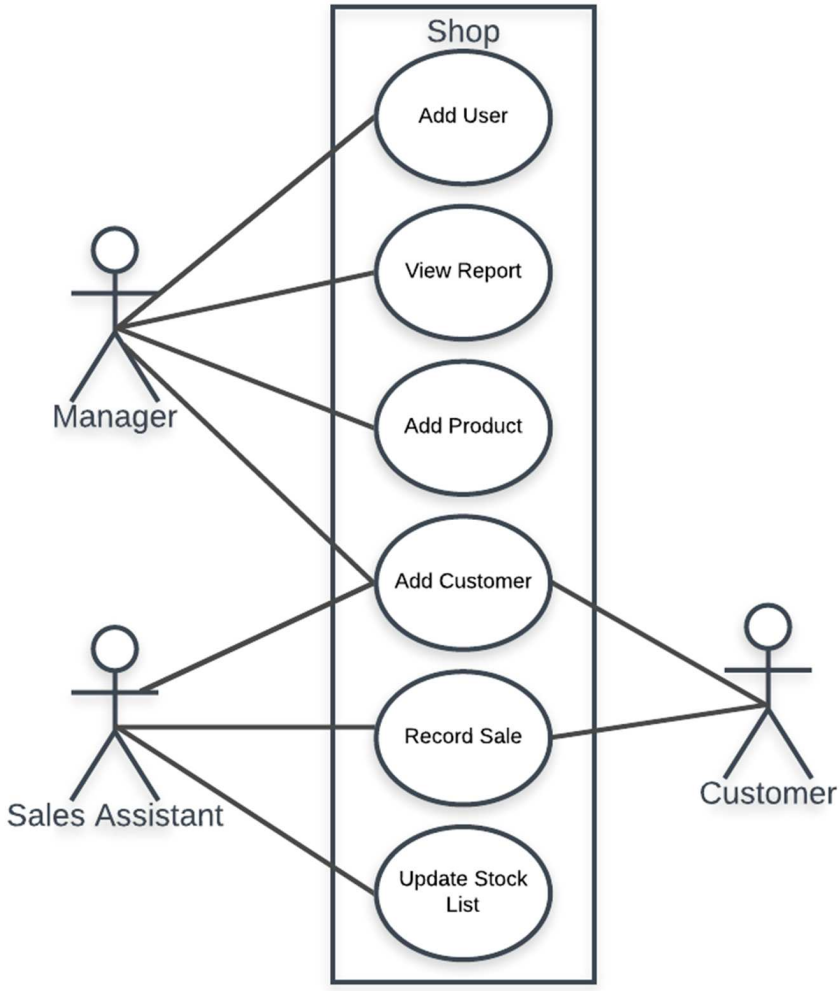


Generalisation of Use Cases

This is Indicated by a solid line and an arrowhead pointing at the **General Use Case**.

Diagrams (Basic example)

The Actor's interactions (associations) with them are identified by solid lines. This is a type of **relationship**.



A large system will have several different actors. This example might be part of a sales recording system in a shop.

The first stage in creating a Use Case diagram is identifying the actors, by classifying the people who interact with the system into categories, in this case the managers, sales assistants and customers.

The next stage is to identify the data which the actors will interact with and how they change it. The scenario above assumes that the system stores the following data:

- A stock table which stores details of items in stock and their quantities.
- A customer table which stores customer details including previous purchases.
- A staff table which stores staff details and their access privileges to the system.

The Customer is involved when a sale is made. If they are a new customer then their details are added to the customer table. When they make a purchase the sale is added to their details.

The Sales assistant also interacts with the customer table but also needs to update the stock list when a sale is made.

The manager needs to be able to update the staff table if a new sales assistant joins the company, add new products to the stock table and generate reports of sales.

Review of Use Case Diagrams Go online

Q2:

Fill in the gaps with the word from the word list.

Word list: Generalisation, Actors, system, ellipse, actioned, Actor, Extend, Use Case, boundary, Include

_____ diagrams are made using _____ that will interact with the system. The system is indicated by a _____ _____ where all the use cases will be placed. A Use Case in the diagram will be represented by an _____.

There are 5 relationships we need to use in a Use Case Diagram:

Association

_____ — this is a use case that is always _____ but not directly by the Actor.

_____ — This Use Case is not directly accessed by the Actor and doesn't always run.

Generalisation of an _____

_____ of a Use Case

1.5 End of topic test

End of topic test: Analysis Go online

Try the end of topic test: Analysis.

Q3: What is an actor in a use case diagram?

a) A person or entity that interacts with the system.
b) An object derived from a class.
c) A relationship between a person and a system.
d) A method.

. . . . . .

Q4: A use case diagram illustrates:

a) interactions between objects.
b) how an object is derived from a class.
c) how actors interact with the system.
d) relationships between classes.

. . . . . .

Q5: There are four types of feasibility: legal, schedule,. . . .

a) sequential and iterative
b) social and economic
c) economic and technical
d) economic and logistical

. . . . . .

Q6: Which four constraints are applied to any project?

- Management
- Sequence
- Milestones
- Rentals
- Product backlog
- Time
- Developer resource
- Scope
- Legal
- Compiler time
- Cost

. . . . . .

Q7: A requirements specification will consist of:

- End User Requirements
- System Specifications
- Use Case Diagram
- Scope
- Boundaries
- Constraints

Topic 2

Design

Contents

Prerequisites

From your studies at Higher, you should already know:

- about entity-relationship diagrams and how to construct them to show entities, attributes and relationships;
- how to identify the cardinality of a relationship using an entity-occurrence diagram;
- how to describe and give examples of compound keys;
- how to create a data dictionary detailing the attributes of three or more entities including types, size, validation and keys;
- the attribute types: text, number, date, time and Boolean;
- the types of validation:
  - presence check;
  - restricted choice;
  - field length;
  - range;
- how to design queries detailing the tables/queries used as the source, fields, search criteria, sort order, calculations and grouping used.

Learning objective

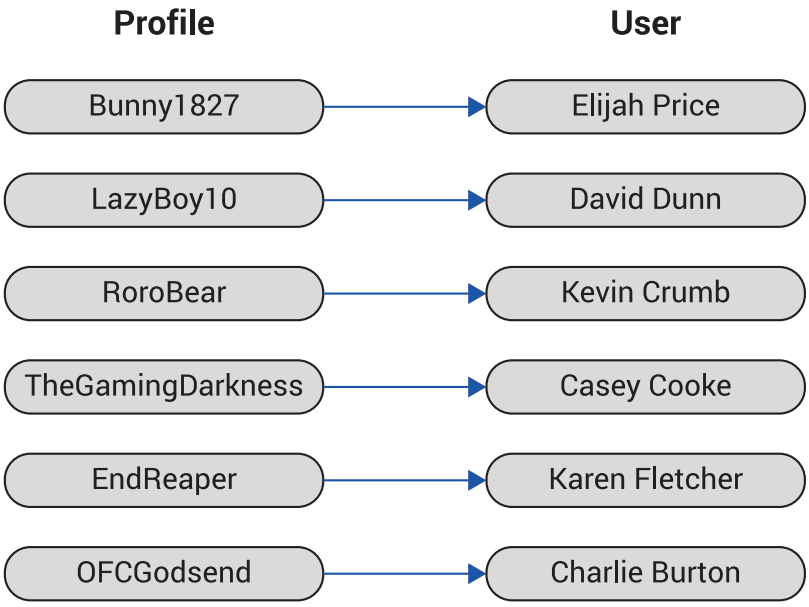
By the end of this topic, you should be able to:

- create entity-relationship diagrams detailing entity type (strong/weak) and relationship participation (mandatory/optional);
- identify relationship participation from an entity-occurrence diagram;
- create and use surrogate keys;
- create data dictionaries, that make use of SQL data types, keys and validation;
- use the SQL data types: varchar, integer, float, date and time;
- create queries that use the "having" operator.

2.1 Revision

Quiz: Revision Go online

Q1: The following entity-occurrence diagram shows the links between instances in two entities.



What is the relationship between the two entities?

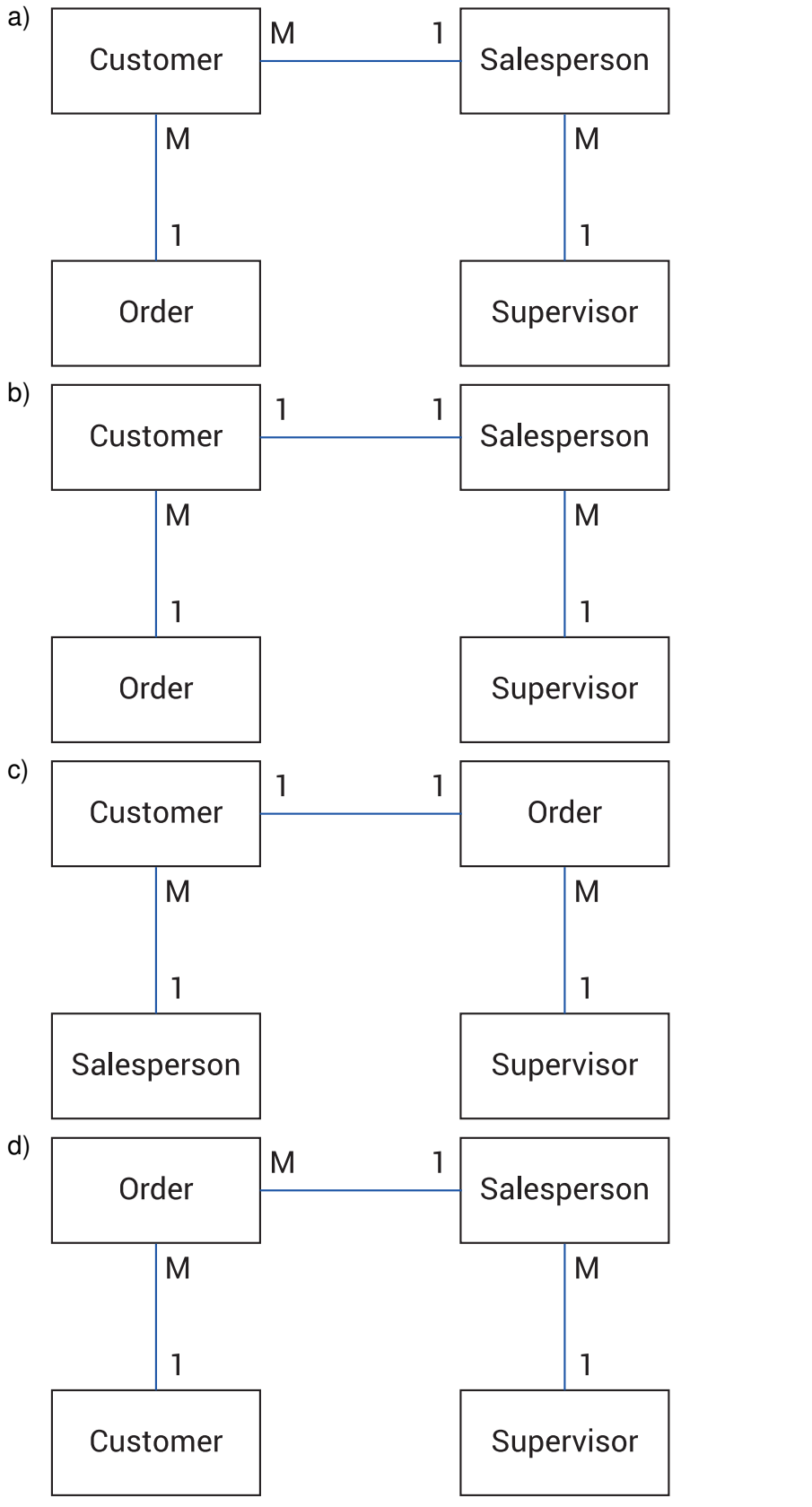
a) One-to-many
b) Many-to-many
c) One-to-one
d) Indeterminate

. . . . . .

Q2: Examine the data model which contains the following tables.

Customer	Order	Salesperson	Supervisor
customerid customername customeraddress customerphone creditlimit	ordercode itemname price customerid salesid saledate	salesid grade dateemployed supervisorid	supervisorid dept

Which is the correct entity-relationship diagram for the model?

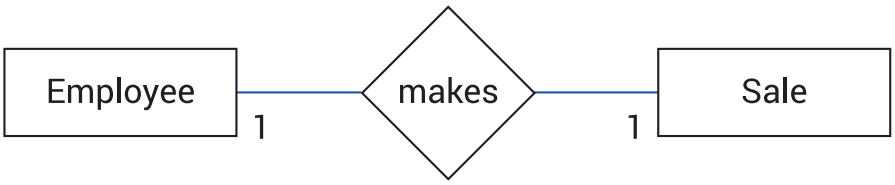

a)
Customer
M
1
Salesperson
M
1
M
1
Order
Supervisor
b)
Customer
1
1
Salesperson
M
1
M
1
Order
Supervisor
c)
Customer
1
1
Order
M
1
M
1
Salesperson
Supervisor
d)
Order
M
1
Salesperson
M
1
M
1
Customer
Supervisor

Q3: Consider the following statements.

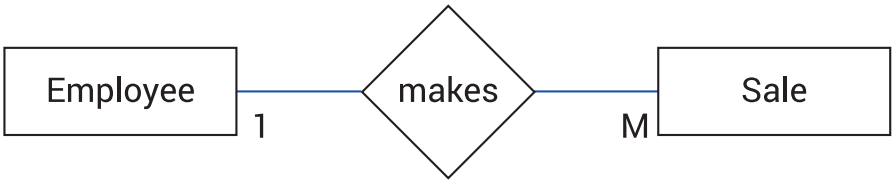
- An employee can make several sales throughout the day.
- Each sale is made by only one employee.

Which entity-relationship diagram represents the statements?

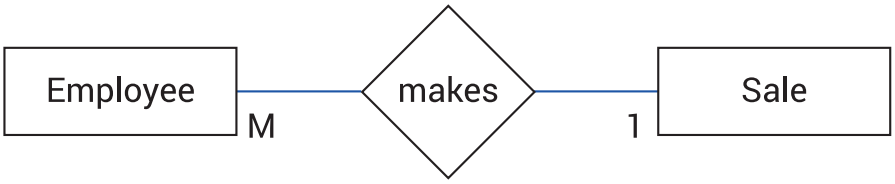
a)



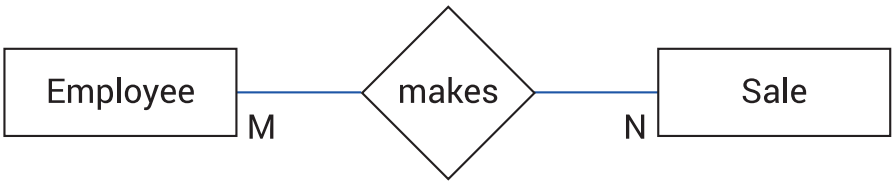
b)



c)

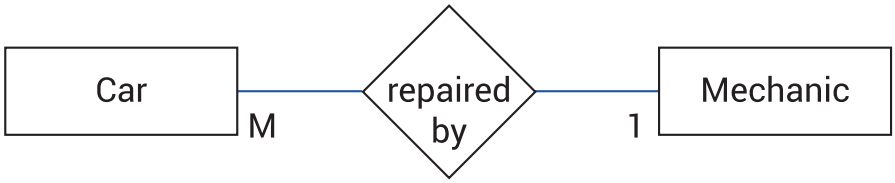


d)



. . . . . .

Q4: Consider the following entity-relationship diagram and complete the statement that it represents.



A car is repaired by:

a) one mechanic, but a mechanic can repair several cars.
b) one mechanic and a mechanic only repairs one car.
c) several mechanics and each mechanic repairs only one car.
d) several mechanics and each mechanic repairs several cars.

. . . . . .

Q5: The following database table is used by a company.

Business

cid	businessname	address	city	postcode	country
12	Island Trading	Garden House, Crowther Way	Cowes	PO31 7PJ	UK
13	Königlich Essen	Maubelstr 90	Brandenburg	14776	Germany
14	Island Trading	67, avenue de l'Europe	Versailles	78000	France
15	La maison d'Asie	1 rue Alsace - Lorraine	Toulouse	31000	France
16	Laughing Bacchus Wine Cellars	1900 Oak St	Vancouver	V3F 2K1	Canada
17	Lazy K Kountry	12 Orchestra Terrace	Walla Walla	99362	USA
18	Königlich Essen	Magazinweg 7	Frankfurt a M	60528	Germany
19	Let's Stop N Shop	87 Polk St, Suite 5	San Francisco	94117	USA
20	Let's Stop N Shop	Carrera 50 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	3508	Venezuela
21	Let's Stop N Shop	Ave. 5 do Mayo Porlamar	I. de Margarita	4980	Venezuela

Which would be the correct query to only produce a list of businesses and the number of locations they have?

a)

Table(s) and/or query	Business
Fields and/or calculations	businessname, COUNT(businessname)
Search criteria	
Grouping	BY businessname
Sort order	businessname ASC

b)

Table(s) and/or query	Business
Fields and/or calculations	businessname, COUNT(cid)
Search criteria	
Grouping	BY cid
Sort order	businessname ASC

c)

Table(s) and/or query	Business
Fields and/or calculations	businessname, COUNT(country)
Search criteria	
Grouping	BY country
Sort order	country ASC

d)

Table(s) and/or query	Business
Fields and/or calculations	country, COUNT(cid)
Search criteria	
Grouping	BY businessname
Sort order	businessname ASC

2.2 Entity-relationship diagrams

Entity-relationship diagrams (or ERD) are used to illustrate the logical structure of a database system. An ERD is a graphical representation of the entities within the database system that is being developed for a **client** and shows how individual entities are related to other entities in the system.

The benefits of using an ERD to represent the structure of the database system include the following.

- It is non-technical and can be easily understood by non-experts. This is important since the analyst must confirm that the representation of the system is correct.
- It is unambiguous as there is only one way of interpreting a well-drawn entity-relationship diagram.

2.2.1 Components of entity-relationship diagrams

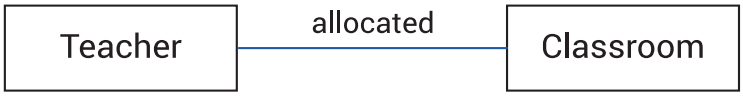
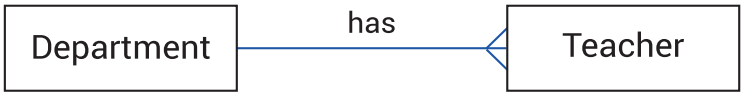
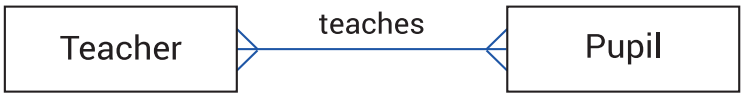
You will know from your studies at Higher that ERDs consist of:

- **entities**;
- **attributes**;
- **relationships**.

As you will also know from Higher, the following symbols are used in the construction of entity-relationship diagrams.

	An *entity* is an object that you want to store information about.
	A *relationship* shows how two entities share information.
	*Attributes* are used to describe an entity. They are the data items that will become the columns in a database table.

Cardinality tells us how many instances of an entity relate to one instance of another entity. You are already familiar from Higher with relationships, which are summarised in the following table.

	*One-to-one* • Each teacher is allocated to one classroom. • Each classroom is allocated to one teacher.
	*One-to-many* • Each department has several teachers. • Each teacher is in one department.
	*Many-to-many* • Each teacher teaches many pupils. • Each pupil has many teachers.

In addition, at Advanced Higher level, you will also be required to construct entity-relationship diagrams that recognise strong and weak entities and mandatory and optional relationships.

	A *weak entity* depends on another entity to exist.
	To connect a weak entity with its owner entity, you need to use a *weak relationship*.
	The end of a relationship marked with a crossed line is a *mandatory relationship*. This means the entity is required in the relationship.
	The end of a relationship marked with a circle is an *optional relationship*. This means the entity is optional in the relationship.

2.2.2 Cardinality of relationships

The cardinality of a relationship specifies the number of **entity occurrences** that take part in a particular relationship. Relationship cardinality can be:

- one-to-one (1:1);
- one-to-many (1:M);
- many-to-many (M:N).

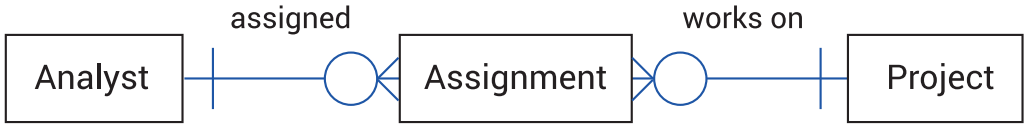
	A UK citizen can have a unique passport number but some don't have a passport. Every passport must belong to a citizen. It is mandatory for the passport to have a citizen and it is optional a citizen to have a passport. This is an example of a one-to-one relationship.
	An aeroplane carries several passengers at a time but any one passenger can only be on one aeroplane at any time. This is an example of a one-to-many relationship.
	An analyst may work on several projects at any one time and a project may require several analysts to work together. This is an example of a many-to-many relationship.

It is important to note that a many-to-many relationship cannot be implemented in a database system. The solution to this problem is to introduce a linking entity.

Examples

1. Linking entity: Analyst and project

In this *analyst* and *project* example, the many-to-many relationship between *analyst* and *project* must be resolved by introducing a third entity *assignment*.

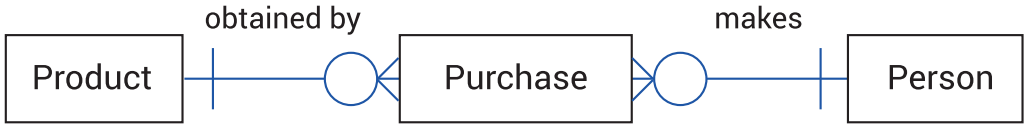


The resolves the many-to-many relationship with two one-to-many relationships. The *assignment* entity will record which analysts are allocated to which projects. It may only contain a compound key of both *analystID* and *projectID*.

. . . . . .

2. Linking entity: Product and person

In this example of a *product* being bought by a *person*, it would be resolved with an entity *purchase*.



Quiz: Cardinality Go online

These questions concern the cardinality of five common relationships.

The first question of each pair asks which description of cardinality best describes the relationship.

For the second question of each pair, explain your decision(s) on paper and then compare your answers with the model answer.

Q6: Parent and child.

a) One-to-one
b) One-to-many
c) Many-to-many

. . . . . .

Q7: Explain your answer for parent and child.

. . . . . .

Q8: Player and team.

a) One-to-one
b) One-to-many
c) Many-to-many

. . . . . .

Q9: Explain your answer for player and team.

. . . . . .

Q10: Pupil and subject.

a) One-to-one
b) One-to-many
c) Many-to-many

. . . . . .

Q11: Explain your answer for pupil and subject.

. . . . . .

Q12: Owner and pet.

a) One-to-one
b) One-to-many
c) Many-to-many

. . . . . .

Q13: Explain your answer for owner and pet.

. . . . . .

Q14: Husband and wife.

a) One-to-one
b) One-to-many
c) Many-to-many

. . . . . .

Q15: Explain your answer for husband and wife.

2.2.3 Strong and weak entities

There are two different types of entity:

- a strong entity does not rely on another entity for identification. Instead, it has enough attributes of its own to make a unique **primary key**;
- a weak entity cannot form a unique primary key on its own. Instead, it must make use of the primary key of another entity to form a unique identifier.

In other words, a weak entity depends on another entity to exist. The entity that supports a weak entity by providing a **foreign key** is often referred to as the **owner entity**. A weak entity cannot stand alone and would not be queried on its own. It may however be queried along with the strong entity it is related to.

Examples

1. Car rental

Consider the following entities:

Car (regnumber, make, model)
Rental (date, location, regnumber)

Car is a strong entity since the attribute *regnumber* forms a unique primary key.

However, *Rental* is a weak entity since its primary key is formed by combining *date* with *regnumber* which is the primary key of the *Car* entity. In other words, the primary key of a weak entity is formed using a foreign key.

. . . . . .

2. Hotel booking

Consider a hotel company which has thousands of rooms across hundreds of hotels.

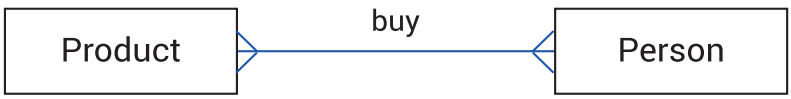
Room (roomnumber, hotelid, roomtype)
Hotel (hotelid, hotelname, hoteladdress, telephonenumber)

The room cannot exist without the hotel. If the hotel were to be removed, then the room would also be removed. This makes the *Hotel* a strong entity and the *Room* a weak entity.

2.2.4 Strong and weak relationships

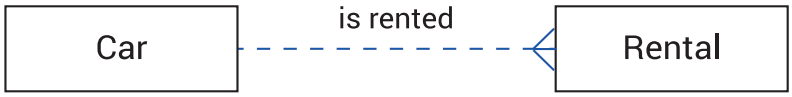
A **strong relationship** is when two entities are related but each stands alone and neither entity relies on the other for part of its primary key. These entities are connected by a strong relationship. A strong relationship is recorded in an entity-relationship diagram using a solid line with the nature of the relationship labelled, normally above it.

For example, a person buys a product online. Both the person and the product are strong entities. Each one has a primary key which does not depend on the other entity. An entity-relationship diagram for this would be:



A **weak relationship** links a strong entity and weak entity. Reviewing the example of *Car* and *Rental*

given previously, the entity-relationship diagram would be:



The weak relationship is shown using a dashed line. The **identifying relationship** is written in text above the relationship line.

2.2.5 Relationship participation

A relationship in an entity-relationship diagram can be either mandatory or optional.

If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of a mandatory relationship is the statement "every project must be managed by a single department".

However, if at least one instance of an entity is not required, the relationship is optional. An example of an optional relationship is the statement, "employees may be assigned to work on projects".

Optionality is shown on an ERD by placing a circle or a perpendicular bar on the relationship line.

- Mandatory relationships are shown by a bar next to the entity that is required.
- Optional relationships are shown by placing a circle next to the entity that is optional.

Weak entities are by definition optional.

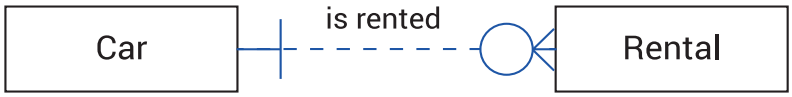
Examples

1. Car rental

A *Car* can exist without any *Rental* being taken for it.

The *Car* is mandatory and the *Rental* is optional.

This would be shown in an ERD as:

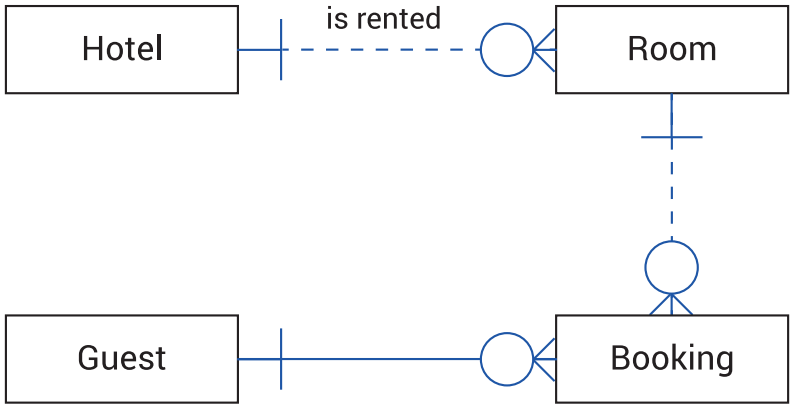


A weak entity in an ERD can be identified as being an optional entity with a weak relationship.

2. Hotel booking

Consider the *Room* and *Hotel* from earlier.

To this, add the concept of a *Guest* who makes a *Booking*.



The *Hotel* is mandatory and must exist if there is a *Room* in the *Hotel*.

A *Guest* is mandatory as a *Booking* must have a *Guest*, but *Booking* is optional as a *Guest* could exist without having made an initial *Booking*.

Each *Booking* has a unique booking ID which makes the *Booking* entity strong.

A *Booking* cannot exist without a *Room* being booked, so *Room* is mandatory for each *Booking* made.

A *Room* can exist without anyone booking it, so *Booking* is optional.

Quiz: Relationships - Authors and books Go online

A system comprises the following:

- an author may write several books;
- a book can have several authors;
- each book must have an author;
- an author may not have yet written a book.

Q16: Identify two entities.

. . . . . .

Q17: What is the relationship?

a) writes
b) can have
c) books

. . . . . .

Q18: What is the cardinality?

a) 1:1
b) 1:M
c) M:N

. . . . . .

Q19: What is the participation?

a) *Author*: mandatory, *Book* mandatory
b) *Author*: mandatory, *Book* optional
c) *Author*: optional, *Book* mandatory
d) *Author*: optional, *Book* optional

. . . . . .

Q20: Draw an entity-relationship diagram which includes resolving the relationship if necessary. Remember to indicate if relationships are strong or weak.

If the relationship is a many-to-many relationship, resolve this by introducing an appropriate third entity.

Quiz: Relationships - Customers and orders Go online

A system comprises the following:

- a customer may make several orders;
- an order is for only one customer.

Q21: Identify two entities.

. . . . . .

Q22: What is the relationship?

a) orders
b) makes
c) customer

. . . . . .

Q23: What is the cardinality?

a) 1:1
b) 1:M
c) M:N

. . . . . .

Q24: What is the participation?

a) *Order*: mandatory, *Customer* mandatory
b) *Order*: mandatory, *Customer* optional
c) *Order*: optional, *Customer* mandatory
d) *Order*: optional, *Customer* optional

. . . . . .

Q25: Draw an entity-relationship diagram which includes resolving the relationship if necessary. Remember to indicate if relationships are strong or weak.

If the relationship is a many-to-many relationship, resolve this by introducing an appropriate third entity.

Quiz: Relationships - Headteachers and schools Go online

A system comprises the following:

- a headteacher manages at least one or more schools;
- a school is managed by at least one or more headteachers.

Q26: Identify two entities.

. . . . . .

Q27: What is the relationship?

a) teaches in
b) subscribes
c) manages
d) jobs

. . . . . .

Q28: What is the cardinality?

a) 1:1
b) 1:M
c) M:N

. . . . . .

Q29: What is the participation?

a) *Headteacher*: mandatory, *School* mandatory
b) *Headteacher*: mandatory, *School* optional
c) *Headteacher*: optional, *School* mandatory
d) *Headteacher*: optional, *School* optional

. . . . . .

Q30: Draw an entity-relationship diagram which includes resolving the relationship if necessary. Remember to indicate if relationships are strong or weak.

Quiz: Relationships - Airports and flights Go online

A system comprises the following:

- a flight connects two airports;
- an airport can be used for several different flights.

Q31: Identify two entities.

. . . . . .

Q32: What is the relationship?

a) flightpath
b) aeroplane
c) pilot
d) connects

. . . . . .

Q33: What is the cardinality?

a) 1:1
b) 1:M
c) M:N

. . . . . .

Q34: What is the participation?

a) *Flight*: mandatory, *Airport* mandatory
b) *Flight*: mandatory, *Airport* optional
c) *Flight*: optional, *Airport* mandatory
d) *Flight*: optional, *Airport* optional

. . . . . .

Q35: Draw an entity-relationship diagram which includes resolving the relationship if necessary. Remember to indicate if relationships are strong or weak.

2.3 Entity-occurrence modelling

At Advanced Higher level you will be required to apply the **entity-occurrence** modelling technique to identify relationships from given data sets.

From your studies at Higher, you will know that entity-occurrence modelling is a process of matching rows from one entity set to the rows in another. This 'linking' of the data items names generates a mapping which identifies the relationship between the two.

The first step in constructing a model of any system is to examine the entities and their relationships. It is relatively easy to identify the entities involved.

Each collection of entities is an entity set. Each of these entity sets will deal with a specific collection of entities: borrower, loan, book, etc.

All entities in an entity set must have the same attributes and each attribute must have a specific domain. Remember, a domain is the set of permitted values for an attribute and therefore defines what the attribute can and cannot store.

Entity-occurrence diagrams are achieved by looking at specific relationship occurrences and linking entity-occurrences with lines, one line for each occurrence of the relationship.

2.3.1 Creating an entity-occurrence diagram

Entity-occurrence modelling is a technique which helps to identify the relationship between two entities. In order to successfully use entity-occurrence modelling, a systems analyst must have a full understanding of the system being analysed, and sample data with which to work from.

The following data has been taken from the library system used in Higher Computing Science.

Identify the entities and their primary keys

The first process is to identify a primary key for each entity. In this case, borrower has a primary key of *borrowerid*, loan has a primary key of *loanid*.

Create a list of primary key values

With the primary keys for each entity decided the next step is to write down the primary key values so that the entity-occurrence diagram can be constructed.

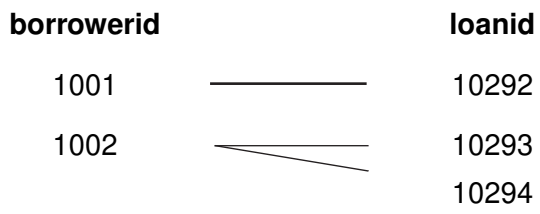
borrowerid	loanid
1001	10292
1002	10293
	10294

Link primary keys

Once the list of the primary key values is completed lines are drawn to illustrate how the values link together. This line is one occurrence of the relationship between the two entities.

Look at the first record for *borrowerid* '1001' in the borrower data in the sample, and look at the value for *loanid*. On the entity-occurrence diagram, we draw a line between the *borrowerid* value and the *loanid* value.

This line indicates one 'link' between the *borrower* entity set and the *loan* entity set. To complete the entity-occurrence diagram we add 'links' for all the remaining records from the sample data. To put it another way, we draw a line to represent each instance of the relationship between the two entities sets.



What does this tell us about the relationship between *borrower* and *loan*? Each borrower has one or more loans linked to it and each loan has one borrower linked to it.

One borrower has many loans and one loan has one borrower. The relationship between *borrower* and *loan* is one-to-many. The pattern of the lines in the entity relationship occurrence diagram can be used to identify the nature of the relationship between two entities sets.

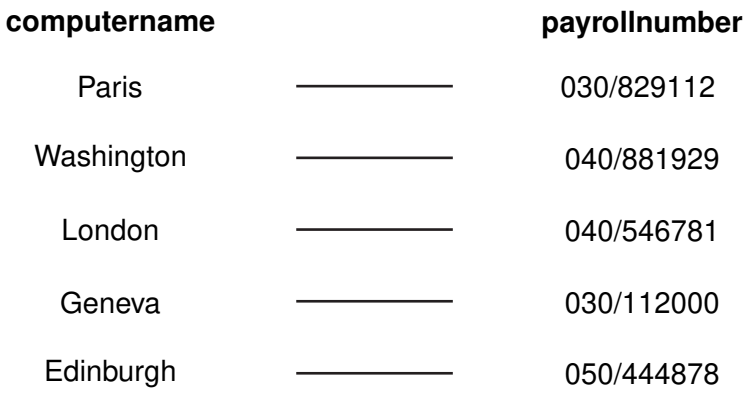
When an entity-occurrence diagram is created, one of three patterns will appear in the lines. There is one pattern for each of the three relationship cardinalities:

- one-to-one;
- one-to-many;
- many-to-many.

2.3.2 Entity-occurrence diagram: One-to-one relationship

- A company has a policy of allocating each employee a computer workstation connected to the company network.
- Each computer has a unique name.
- Each employee has a unique payroll number.

The entity-occurrence diagram created is:

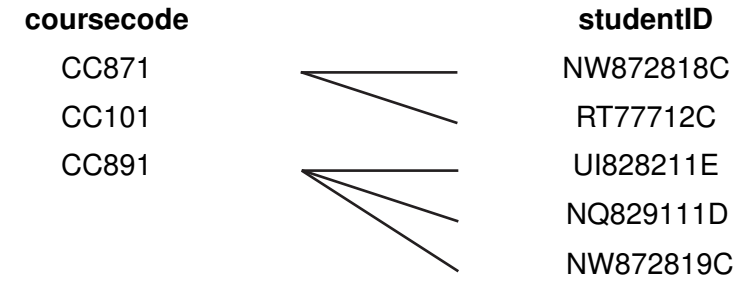


One *computername* has one *payrollnumber* and one *payrollnumber* has one *computername*. This is, therefore, an example of a one-to-one relationship.

2.3.3 Entity-occurrence diagram: One-to-many relationship

- Courses at Westhill University each have a unique course code.
- Students at the university each have a unique Student ID.
- Students are only allowed to register for one course.
- At least 20 students enrol in each course at the university.

The entity-occurrence diagram created is:

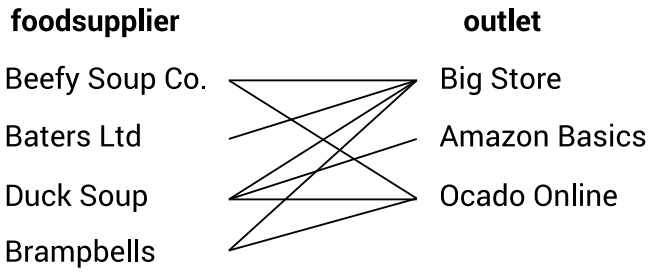


Each course has more than one student and each student has one course. This is, therefore, a one-to-many relationship.

2.3.4 Entity-occurrence diagram: Many-to-many relationship

- Food suppliers have numerous outlets.
- Outlets have numerous food suppliers.

So, for example, Beefy Soup Co. sell soup to Big Store and Ocado Online. Big Store buys soup from Beefy Soup Co., Baters Ltd., Duck Soup and Brampbells.



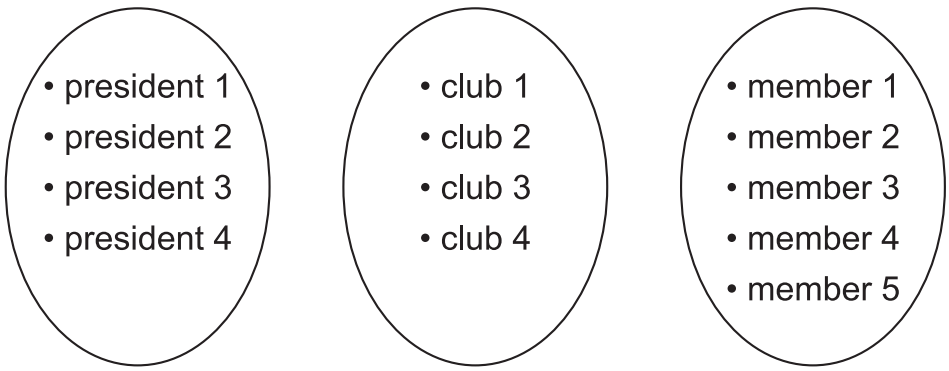
Each food supplier has one or more outlets and each outlet has one or more food suppliers. This is, therefore, a many-to-many relationship.

2.3.5 More about entity-occurrence diagrams

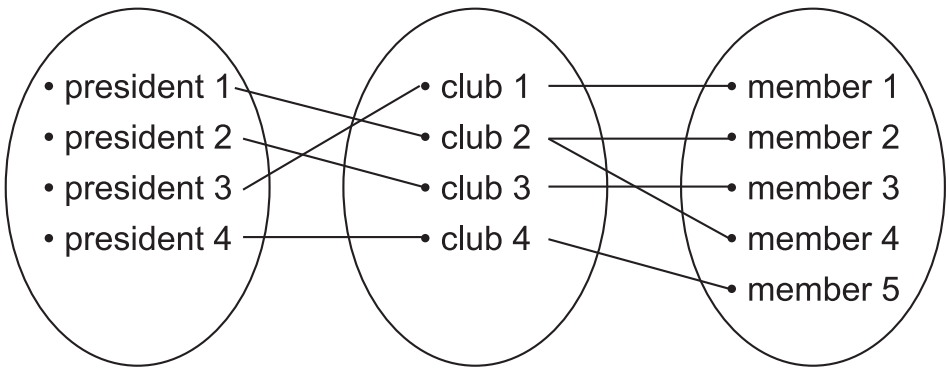
In some cases, you may not be presented with sample data to create your entity-occurrence diagram. You may have to use information about the system to understand the data using dummy values.

For example: You are told that:

- Each rugby club has one president.
- Each president belongs to only one rugby club.
- Each rugby club has many members.
- A member can only belong to one club.



As shown, you would lay this out as having three entity sets and a number of values within them. Then join the values together following the logic for the system.



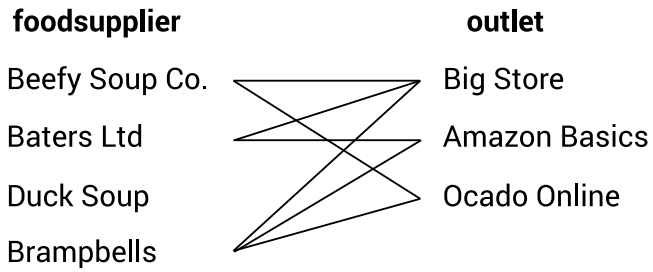
This shows that the relationship between president and club is 1:1 (one-to-one) and that the

relationship between club and member is 1:M (one-to-many)

2.3.6 Modelling participation

If may be possible to determine the participation of a relationship when creating an entity-occurrence diagram.

For example:

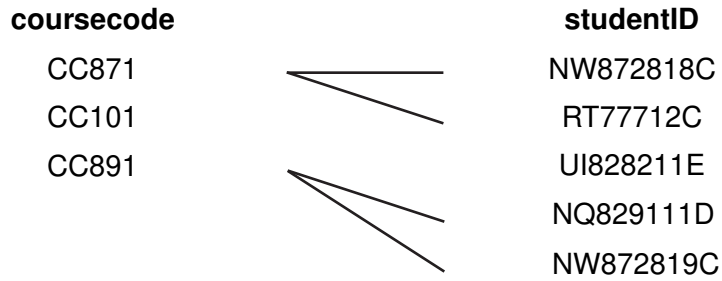


In this example, not all rows of *foodsupplier* are linked to a row of *outlet* but all rows in *outlet* are linked to a row in *foodsupplier*.

This means that, in the relationship between foodsupplier and outlet, foodsupplier is mandatory, but outlet is optional.

Why? Because a food supplier can exist WITHOUT being linked to an outlet. Therefore the outlet is Optional. Similarly, because ALL rows of *outlet* are linked to *foodsupplier*, this indicates that *foodsupplier* is mandatory - a row in *outlet* cannot exist without being linked to a row in *foodsupplier*.

Now look at this example.



In this example, a *coursecode* is not linked to a *studentID* and a *studentID* is not linked to a *coursecode*. This means that both sides of the relationship are optional because both entities can exist without being linked to the other.

Quiz: Entity-occurrence diagram Go online

Q36: From the following three tables of data, create an entity-occurrence diagram.

Shopper

shopperID
25251
25252
25253
25254

Basket

shopperID	ItemCode
25251	E828
25251	L212
25251	P900
25252	E828
25252	L212
25253	P900
25253	E828
25253	T22
25253	L212

Item

ItemCode
C182
D298
E671
E828
D298
P900
T22
K872
L212

. . . . . .

Q37: What is the cardinality between each entity?

a) *Shopper* M:N *Basket* M:1 *Item*
b) *Shopper* 1:M *Basket* M:1 *Item*
c) *Shopper* 1:1 *Basket* M:N *Item*
d) *Shopper* 1:1 *Basket* M:1 *Item*

. . . . . .

Q38: What is the cardinality between each entity?

a) *Shopper* optional - optional *Basket* optional - optional *Item*
b) *Shopper* mandatory - optional *Basket* mandatory - mandatory *Item*
c) *Shopper* mandatory - optional *Basket* optional - mandatory *Item*
d) *Shopper* optional - optional *Basket* optional - mandatory *Item*

2.4 Surrogate key

Weak entities and relationships require multiple attributes to be used as part of a compound key. This is more complex to manage than a single attribute primary key. Complex multi-attribute primary keys require additional processing by the database system.

The database design can be improved if a multi-attribute key is replaced with a unique single attribute primary key. This replacement key is called a **surrogate key**.

Typically, a surrogate key is an arbitrary value, which is unique for each row in the entity set. Often this is an id or a number e.g. *studentid* or *orderno*.

The introduction of a surrogate key changes a weak entity to a strong entity and makes the relationships to that entity strong also.

Consider the earlier *Hotel* and *Room* example.

Room (roomnumber, hotelid*, roomtype)
Hotel (hotelid, hotelname, address, telephonenumber)

Room is a weak entity. It depends on *Hotel* for part of its primary key.

If we add a surrogate key, which will be unique for each row, in the entity set e.g. hotelroomID this makes the entity and the associated relationship strong.

Room (hotelroomID, roomnumber, hotelid*, roomtype)
Hotel (hotelid, hotelname, address, telephonenumber)

Surrogate keys are often added to entities to reduce the complexity of complicated compound keys. When designing your own databases, it will often be helpful to add a surrogate key to an entity to make managing it easier.

2.5 Replacing meaningful primary keys with surrogate keys to improve design

In some cases, a valid primary key exists in an entity. For example:

Ship (shipname, shipport, vesselsize)

Here *shipname* is unique. Each ship has a different name. *shipname* is a meaningful primary key. It has a meaning within the business rules that define the system.

When designing a system, it is preferable to replace meaningful primary keys with arbitrary ids or numbers. This typically makes the database more flexible to cope with changes (e.g. two ships with the same name!).

So, the preferred implementation would be:

Ship (shipid, shipname, shipport, vesselsize)

Quiz: Surrogate keys Go online

Q39: The following is a database system.

Ship (shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (shipname*, companyid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

Surrogate keys have been used to simplify this database model.

Which is the correct model now?

a) Ship (shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (bookingid, shipname*, companyid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

b) Ship (shipid, shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (shipid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

c) Ship (shipid, shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (bookingid, shipid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

d) Ship (shipid, shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (containerid*, dateout, shipid*)
Container (containerid, contents, originated, destination)

. . . . . .

Q40: The following is a database system.

Customer (licenceno, firstname, lastname, address, mobile, email)
Rental (rentaldate, regno*, licenceno*, milesout, milesin)
Car (regno, make, model, rentcost)
Service (regNo*, garagename*, servicedate, mileage, cost)
Garage (garagename*, address)
Servicepart (regno*, garagename*, partcode*, servicedate)
Part (partcode, partdescription)

Surrogate key(s) have been used to simplify this database model.

Which is the correct model now?

a) Customer (licenceno, firstname, lastname, address, mobile, email)
Rental (rentalid, rentaldate, regno*, licenceno*, milesout, milesin)
Car (carid, regno, make, model, rentcost)
Service (carid*, garagename*, servicedate, mileage, cost)
Garage (garagename, address)
Servicepart (regno*, garagename*, partcode*, servicedate)
Part (partcode, partdescription)

b) Customer (licenceno, firstname, lastname, address, mobile, email)
Rental (rentalid, rentaldate, regno*, licenceno*, milesout, milesin)
Car (regno, make, model, rentcost)
Service (serviceid, regno*, garagename*, servicedate, mileage, cost)
Garage (garagename, address)
Servicepart (serviceid*, partcode*, servicedate)
Part (partcode, partdescription)

c) Customer (customerid, licenceno, firstname, lastname, address, mobile, email)
Rental (rentalid, rentaldate, regno*, customerid*, milesout, milesin)
Car (carid, regno, make, model, rentcost)
Service (serviceid, carid *, garageid*, servicedate, mileage, cost)
Garage (garageid, garagename, address)
Servicepart (servicepartid, partcode*, serviceid*)
Part (partcode, partdescription)

d) Customer (customerid, licenceno, firstname, lastname, address, mobile, email)
Rental (rentaldate, regno*, customerid*, milesout, milesin)
Car (regno, make, model, rentcost)
Service (serviceid, regno*, garagename*, servicedate, mileage, cost)
Garage (garagename, address)
Servicepart (servicepartid, partcode*, serviceid*)
Part (partcode, partdescription)

2.6 Data dictionary

Database systems are only as good as the data they store. Without a mutually agreed-upon set of data elements with clearly defined names and definitions, the validity and reliability of the data contained in a system are suspect at best and must be discounted at worst. The **data dictionary** is the foundation of a database system and the central building block that supports communication across processes.

A data dictionary is a catalogue of all data items in a system. The data dictionary stores details and descriptions of all of data items. For this reason it is often referred to as metadata; that is, data about data.

A data dictionary is used to fully describe all data items that are held in a system. Without a data dictionary the development of large systems becomes difficult. The data dictionary is an effective solution to the problem of remembering a large amount of detail. The main purpose of a data dictionary is to provide a source of reference in which the analyst, the user, the designer can look up and find out the content of a system and any other relevant information.

Once developed, a data dictionary is a vital reference point for all developers who work on the system as it is being developed. It also acts as a reference point for any technical staff whose job it will be to support end users working with the system after it is has been implemented.

2.6.1 Creating a data dictionary

Once many-to-many relationships have been resolved, the analyst is now aware of all entities and attributes that are required in the system. The entity-relationship diagram graphically shows the relationships that exist between the entities and even indicates what attributes are stored in each entity. However, it does not give any indication of the type or size of each attribute nor does it show the restrictions that apply to an attribute or where else in the system the attribute is being used. These details are held in a properly developed data dictionary.

The details of each data time that must be included in a data dictionary are:

- entity name;
- attribute name;
- primary and foreign keys;
- attribute type;
- attribute size;
- **validation**.

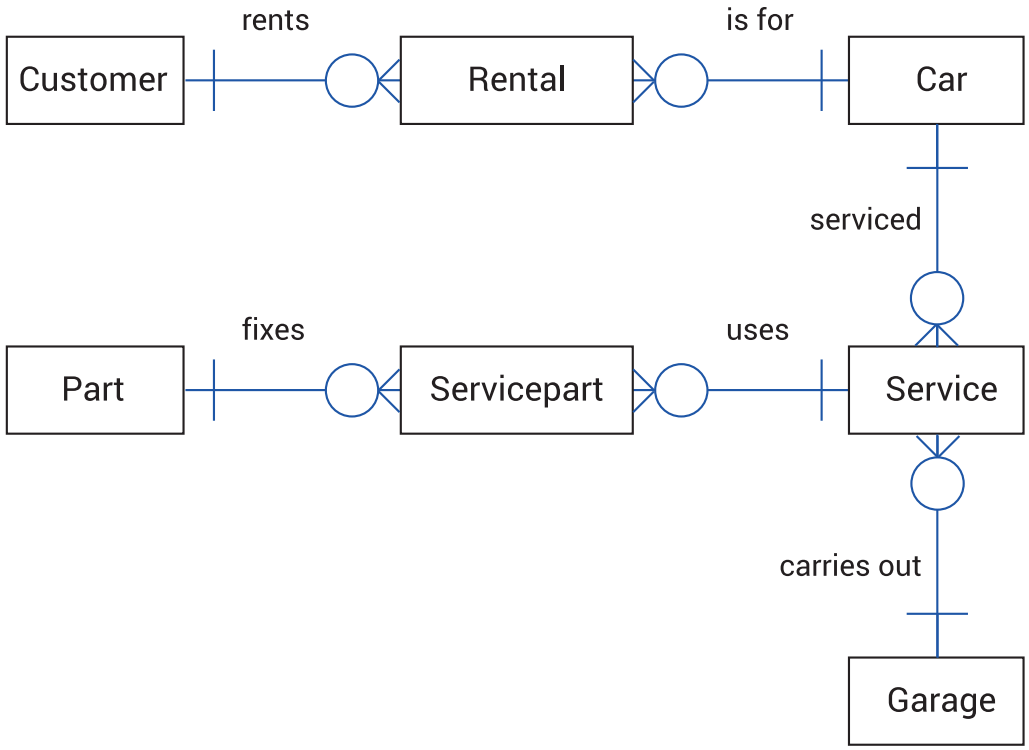
When combined, each of these details forms a detailed description of each data item in the system. The details are recorded in a data dictionary like the blank one shown.

Entity:					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**

Given that the database designed will be implementing the database using SQL, the data dictionary should make use of datatypes specific for this implementation.

We will create a data dictionary using the following expanded version of the car rental system you saw earlier. This is fully resolved; there are no many-to-many relationships and surrogate keys have been added to remove compound keys, making all entities and relationships strong.

Customer (customerid, licenceno, firstname, lastname, address, mobile, email)
Rental (rentalid, rentaldate, regno*, customerid*, milesout, milesin)
Car (carid, regno, make, model, rentcost)
Service (serviceid, carid*, garageid*, servicedate, mileage, cost)
Garage (garageid, garagename, address)
Servicepart (servicepartid, partcode*, serviceid*)
Part (partcode, partdescription)



In this system a customer rents one or more cars, each car can be rented by one or more customers (but not at the same time), cars are serviced at regularly (to ensure they are running correctly), this service is carried out by a garage. One or more parts may be need to service the car.

The data dictionary for the Rentals systems will be:

Entity: Customer

Attribute name	Key	Type	Size	Required	Validation
customerid	PK	INT		Yes	
licenceno		VARCHAR	16	Yes	unique for each customer
firstname		VARCHAR	26	Yes	
lastname		VARCHAR	26	Yes	
address		VARCHAR	150	Yes	
mobile		VARCHAR	12	Yes	
email		VARCHAR	320	No	

Entity: Rental

Attribute name	Key	Type	Size	Required	Validation
rentalid	PK	INT		Yes	autoincrement
rentaldate		DATE		Yes	
carid	FK	INT		Yes	existing carid from Car
customerid	FK	INT		Yes	existing customerid from Customer
milesout		MEDIUMINT		Yes	
milesin		MEDIUMINT		No	> milesout

Entity: Car

Attribute name	Key	Type	Size	Required	Validation
carid	PK	INT		Yes	
regno		VARCHAR	12	Yes	unique for each car
make		VARCHAR		Yes	
model		VARCHAR		Yes	
rentcost		FLOAT		Yes	range: >=25.00 and <= 350.00

Entity: Service

Attribute name	Key	Type	Size	Required	Validation
serviceid	PK	INT		Yes	
carid	FK	INT		Yes	existing carid from Car
garageid	FK	INT		Yes	existing garageid from Garage
servicedate		DATE		Yes	
mileage		INTEGER		No	
cost		FLOAT		No	

Entity: Garage					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
garageid	PK	INT		Yes	
garagename		VARCHAR		Yes	
address		VARCHAR		Yes	

Entity: Servicepart					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
servicepartid	PK	INT		Yes	
partcode	FK	VARCHAR		Yes	existing partcode from Part
serviceid	FK	INTEGER		Yes	existing servicepartid from Service

Entity: Part					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
partcode	PK	VARCHAR		Yes	
partdescription		VARCHAR		Yes	

2.6.2 Attribute name, data type, data size and key

Attribute name

The first four columns of the data dictionary are concerned with the name, cardinality, type and size of each attribute. It is a requirement that all attribute names, within an entity, are unique (you can't have two attributes in the same entity with the same name!)

Data type

SQL **data types** indicated in a data dictionary include:

- **VARCHAR**
  The VARCHAR data type is used because this is very efficient when storing text; it only stores the number of characters used rather than storing the whole length of the field according to the declared size.
- **INTEGER**
  The Integer data types has five different variants in SQL. These different variants all store integers, the difference between them is the number of bytes of storage that they use for this.

Variant	Storage (bytes)	Minimum value signed	Minimum value unsigned	Maximum value signed	Maximum value unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63} - 1$	$2^{64} - 1$

- **FLOAT**
  Real number / floating point values are stored in the database using the FLOAT data type.
- **DATE**
  The date type in SQL records any date in the range '1000-01-01' to '9999-12-31'. It uses dates in the format: YYYY-MM-DD.
- **TIME**
  The time type in SQL records values in the format 'hh:mm:ss'. It can be used to store times which go beyond those in the 24hr clock. This means as well as times in the day, it can also be used to store durations.

  Important: Be careful about assigning abbreviated values to a TIME column. SQL interprets abbreviated TIME values with colons as time of the day. That is, '11:12' means '11:12:00', not '00:11:12'.

Care should be taken when setting data types for values such as telephone number. Such fields often contain spaces, parenthesis '()' or the plus symbol '+' so must be stored as VARCHAR.

Data size

The **size of a data** item refers to the maximum number of characters that will be allowable. This only needs to be considered when the data type is VARCHAR. Some attributes, for example firstname and lastname, have international standards which define what size you should make them (e.g. firstname/lastname may the cardholder name length from ISO IEC 7813 which states they should have minimum length of 2 and a maximum length of 26 - this is the standard that is used by the financial industry for credit and debit cards).

Email is an example which has specific technical restrictions. An email address is limited to 64 characters before the @ and 255 characters after. This means that the technical maximum size for an email address is 320 characters (64 + 1 + 255). The 1 is for the @.

Telephone numbers internationally will be no more than 15 digits (without any internal extension numbers e.g. +44 9922 888112 ext 1234). This is specified in international standard E.164 (see https://en.wikipedia.org/wiki/E.164).

For the VARCHAR data type, the size defines the maximum number of characters that are being stored. Remember, VARCHAR stores characters efficiently so only uses the storage needed. Other data types used by SQL may not store data so efficiently.

Key

An attribute is marked as PK (primary key) or FK (foreign key). Usually, there will be no need to use both PK and FK as compound and/or multi-attribute primary keys will have been resolved by

implementing a surrogate key.

Note that the data type and size of any foreign keys must match the type and size set for the original primary value.

2.6.3 Validation

The validation columns (Required and Validation) of the data dictionary should indicate any restrictions that should be applied to the data item when values are being entered into the system. Validation checks automatically check any input values to make sure that they are sensible. Common validation checks that are indicated in a data dictionary include:

- **presence check** (determined by the required column);
- **range check**;
- **restricted choice**;
- **lookup existing** (primary) value of any foreign key data item.

Presence check

A presence check is used to indicate whether or not a null value can be used. Values for certain data items are not always known at the time when the data is entered into the system. For example, in a powertool rental system, the actual return date of any powertool hired will not be known until the powertool is actually returned. In this case, return date cannot be entered when the rental is made and as a result, must be left blank. This is indicated in a data dictionary by setting the required column to 'No'.

If the value is set to 'Yes', then a row cannot be added to the databases without this value being entered.

Range check

A range check is used to make sure that values entered are within certain pre-defined upper and lower limits. These limits are specified in the data dictionary. For example, quantity ordered in an ordering system must always be greater than zero. Many analysts set restrictions on values that can be entered in price fields in order to reduce errors. Most number, date and time fields have range check set in order to minimise data input errors.

Restricted choice

A restricted choice is used to indicate that only certain values can be entered for a given data item. The list of possible values is specified in the data dictionary. For example, in a video rental system, the rating of the video can be one of a few possibilities: U, PG, E, 15, 18. A restricted choice can only be applied when a restricted list of input values can be defined.

Lookup existing values

When a data item represents a foreign key that is being used to link one entity to another, any value entered in the foreign key normally must already exist in the related primary key. The validation column of the data dictionary is used to indicate that a primary key value must exist for any value entered in the foreign data item.

Format of data items

Although not a validation check in that is does not automatically check for input errors, a data

dictionary can also be used to indicate the format of a particular data item. This is particularly useful when the data item represents a code that has a fixed format. For example, telephone numbers and postcodes within the UK all have a fixed format.

Analysts can take advantage of this knowledge by using it to restrict the pattern of characters that are entered into a data item with a known format. If the format of a data item is to follow a fixed pattern, the pattern can be specified in the data dictionary.

Warning about VARCHAR fields

Note that there is little you can do to restrict the possibility of input errors when a VARCHAR field is set. This is due to the fact that it is difficult to predict all possible combinations of characters that can be entered. The best that an analyst can do is to set the size appropriately and make use of restricted choice if it is sensible to do so. The format of a VARCHAR value that represents a code can also be specified.

2.6.4 Data dictionary practice

Example : Example data dictionary

In the example extract from a data dictionary for the Rental entity:

Entity: Rental					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
rentalid	PK	INT		Yes	autoincrement
rentaldate		DATE		Yes	
carid	FK	INT		Yes	existing carid from Car
customerid	FK	INT		Yes	existing customerid from Customer
milesout		MEDIUMINT		Yes	
milesin		MEDIUMINT		No	$>$ milesout

rentalid is a surrogate key, it is the primary key, is an integer and has been set to autoincrement.

rentaldate is a DATE so will be in the format 'YYYY-MM-DD' and it is required so a value must be entered.

carid is a foreign key from the CAR entity. It has the same type (INT) as the primary key to which it is related. It is required and has to be set to a value that exists in the CAR entity (this enforces referential integrity).

customerid is also a foreign key, this time from the CUSTOMER entity. . It also has the same type (INT) as the primary key to which it is related. It is required and has to be set to a value that exists in the CUSTOMER entity.

milesout is recorded when the car is collected by the customer. It is not known when the rental

is initial recorded so Required is 'no'. Similarly, *milesin* is recorded when the car is returned and Required is also 'no'. Both attributes have a data type of MEDIUMINT which uses three bytes of storage meaning that the largest number that can be stored is 16,777,215.

Quiz: Data dictionary Go online

Q41: Review the following data model and create a data dictionary for it. Some sample data is given to support this task.

Ship (shipid, shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (bookingid, shipid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

Ship

shipid	shipname	companyid	vesselcapacity	regport
9237	MOL Triumph	827	12,000	Panama
2312	MV Barzan	800	13,000	Dublin
2441	CSCL Globe	750	9,000	Edinburgh

Company

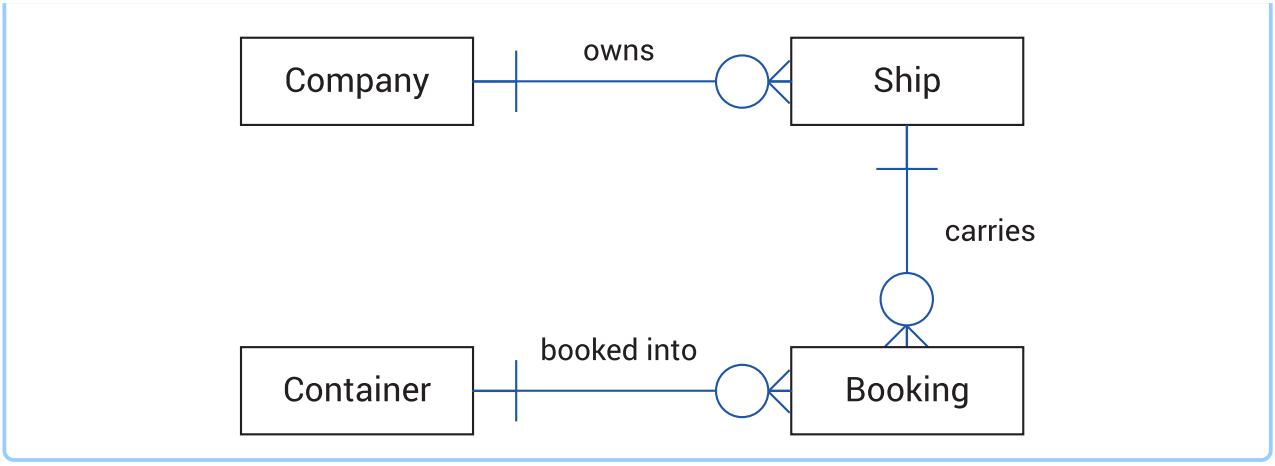
companyid	coname	telephone	email
827	MOL	+507 244 2383	shipping@molplc.net
800	BigShipCo	+353 010 299 921	info@bigship.co
750	Mesk Shipping	+44 131 982 8282	booking@mesk.us

Booking

bookingid	shipid	containerid	dateout
26611	9237	1092119	2020-05-01
26612	2312	8922121	2020-05-01
26613	2441	3998281	

Container

containerid	contents	originated	destination
1092119	Car parts	Delhi, India	Grangemouth, Scotland
2312	Clothing	Hong Kong, China	Liverpool, England
3998281	Furniture	Mersin, Turkey	Belfast, Northern Ireland



2.7 Designing queries

From your studies at Higher level you will be familiar with designing queries making use of the following design layout.

Field(s) and/or calculations(s)	
Table(s) and/or query(-ies)	
Search criteria	
Grouping	
Sort order	

This design layout is extended at Advanced Higher level to include an SQL HAVING clause.

Field(s) and/or calculations(s)	
Table(s) and/or query(-ies)	
Search criteria	
Grouping	
Having	
Sort order	

2.7.1 HAVING clause

The HAVING clause is used in SELECT statements. It acts as a filter for group of rows or aggregated data (e.g. MIN, MAX, AVG, SUM, COUNT). It is often used with the GROUP BY clause so that data is selected based on a specified condition.

If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause.

For example, a query is carried out on the following data table.

Location

locationid	location	access	city
762	Grassmarket	Full access	Edinburgh
763	Royal Mile	Restricted	Edinburgh
764	Arthur's Seat	On foot only	Edinburgh
765	Dundee Law	Limited vehicle	Dundee
766	Gowie Park	Full access	Dundee
767	Glasgow Cathedral	Full access	Glasgow
768	Glasgow City Chambers	Full access	Glasgow
769	Cochrane Street	Full access	Glasgow
770	Necropolis	Limited vehicle	Glasgow
771	Cluny Castle	Full access	Aberdeen
771	Dunnotar Castle	Restricted	Aberdeen

If the query:

```
SELECT city, Count(*) AS 'number of locations' FROM Location GROUP BY city
```

was carried out the resulting table would be:

city	number of locations
Aberdeen	2
Dundee	2
Edinburgh	3
Glasgow	4

But what if we wanted just the grouped rows where the 'number of locations' was three or more? This is where the HAVING clause can be used.

```
1 SELECT city, Count(*) AS 'number of locations'
2 FROM Location
3 GROUP BY city
4 HAVING COUNT(*) >=3;
```

This will show only the rows where the count of the grouped items is equal to three or greater.

city	number of locations
Edinburgh	3
Glasgow	4

The design for this query would be:

Field(s) and/or calculations(s)	city, Count(*)
Table(s) and/or query(-ies)	Location
Search criteria	
Grouping	city
Having	COUNT(*) >= 3
Sort order	

A further example is based on the Shipping system from earlier.

A query is required to total the capacity of all ships booked with containers for "Grangemouth, Scotland" with a dateout of "2020-05-01".

Field(s) and/or calculations(s)	SUM(vesselcapacity)
Table(s) and/or query(-ies)	Ship, Booking, Container
Search criteria	destination = "Grangemouth, Scotland"
Grouping	dateout
Having	dateout = "2020-05-01"
Sort order	

2.7.2 Logical operators

You will also be required to design queries using new logical operators. These are:

NOT

The NOT operator reverses or negates a condition in a query.

Field(s) and/or calculations(s)	SUM(vesselcapacity)
Table(s) and/or query(-ies)	Ship, Booking, Container
Search criteria	NOT (destination = "Grangemouth, Scotland")
Grouping	dateout
Having	dateout = "2020-05-01"
Sort order	

BETWEEN

The BETWEEN operator is a logical operator that allows you to specify whether a value is in a range or not. It works for dates and numeric values.

For example:

Field(s) and/or calculations(s)	*
Table(s) and/or query(-ies)	Booking
Search criteria	dateout BETWEEN "2020-04-28" AND "2020-05-03"
Grouping	
Having	
Sort order	

IN

The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

For example:

Field(s) and/or calculations(s)	*
Table(s) and/or query(-ies)	Booking
Search criteria	WHERE bookingid IN (26611, 26612, 26687)
Grouping	
Having	
Sort order	

2.7.3 Subqueries

A subquery is a query nested within another query. The subquery is used to generate an answer table that will be used in the main query as a condition for the data being retrieved.

Subqueries can be used with SELECT, INSERT, UPDATE and DELETE statements. Subqueries can also be used with the range of operators that you have used at Higher and Advanced Higher levels (=, <, <=, >, >=, IN, BETWEEN and the ANY and EXISTS operators that we will look at shortly).

Key things to know about subqueries:

- Subqueries must be enclosed in parentheses. These are the **'('** and **')'** characters.
- A subquery is usually added as part of the WHERE clause but can also appear in the SELECT clause or the FROM clause.
- If the subquery returns a single row value then you can use comparison operators to compare two values (such as =, <, <=, >, >=).

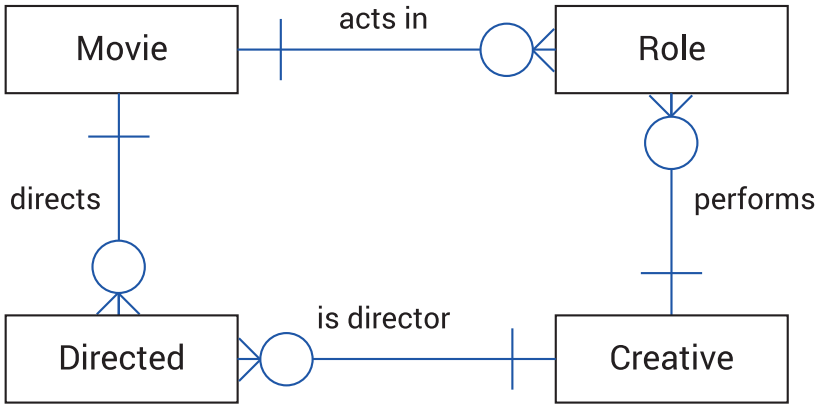
- If the subquery returns multiple row values then multi-row operators, (such as ANY, IN) can be used.

A subquery can be treated as an ***inner query***, which is an SQL query placed as part of another query, which is called the ***outer query***. The inner query retrieves data or executes first before the outer query so that the result of the inner query can be passed to the outer query.

For the following questions, we will use the *Movies* database.

Code 1: Database download

- Movies database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/6FD22C38-3174-685C-D559-1F657D5270B8/movies.zip



Movie (id, title, yr, budget, gross)
Role (roleid, movieid, creativeid, ord)
Creative (creativeid, name)
Directed (directorid, movieid, creativeid)

In this database, there is a table of creatives, these are people involved in film making. Creatives can take on one or more acting roles in movies. In fact, the same creative could perform multiple roles in the same movie and appear in multiple movies. A creative could also direct a movie or be one of a number of directors directing the movie.

Each movie has an id, title, a year of release, a budget to make the movie and a gross (which is the figure it made at the box office before deductions for making the move etc.).

Each role has a roleid, a movieid (to link it back to the movie the role is in), a creativeid (to link it to the creative) and ord. Ord is the order of the role in the film's credits. The main star would have a value of one, a co-star a value of two and so on.

2.7.4 A subquery example

In this database are details of the movies that creatives have been in. For example, for this query:

Field(s) and/or calculations(s)	*
Table(s) and/or query(-ies)	Creative, Role, Movie
Search criteria	Creative.creativeid = Role.creativeid AND Creative.name = "Bill Murray" AND Movie.id = Role.movieid
Grouping	
Having	
Sort order	

produces a list of all of the movies from the database that "Bill Murray" has appeared in which can be viewed in the online materials.

The following query:

Field(s) and/or calculations(s)	*
Table(s) and/or query(-ies)	Creative, Role, Movie
Search criteria	Creative.creativeid = Role.creativeid AND Creative.name = "Dan Aykroyd" AND Movie.id = Role.movieid
Grouping	
Having	
Sort order	

produces a list of all of the movies from the database that "Dan Aykroyd" has appeared in which can be viewed in the online materials.

But what is required is a list of movies in which both "Bill Murray" and "Dan Aykroyd" have appeared together.

An approach to this would be to find all the movies with "Bill Murray" and then see if "Dan Aykroyd" has also had a role in them.

The query to retrieve "Bill Murray" movies is the inner query.

Field(s) and/or calculation(s)	Movie.id, title, yr, budget, gross			
Table(s) and/or query(-ies)	Creative, Role, Movie			
Search criteria	Creative.name = "Dan Aykroyd"			
	AND Movie.id IN	**Inner query**	**Field(s) and/or calculation(s)**	Movie.id
			Table(s)	Creative, Role, Movie
			Search criteria	Creative.name = "Bill Murray"
Grouping				
Having				
Sort order				

The answer table produced by the inner query produces an answer table of Movie.id values e.g.

Movie.id
22027
22014
21808
21671
...
11638

The outer query uses this list with an IN operator and matches all the movies which have "Dan Aykroyd" and a Movie.id value that appears in this inner query list.

The resulting answer table is:

id	title	yr	budget	gross
13523	Ghostbusters II	1989	25000000	215394738
15786	Mr. Mike's Mondo Video	1979	NULL	NULL
13522	Ghostbusters	1984	30000000	291632124
17368	She's Having a Baby	1988	NULL	NULL
22027	The Rutles: All You Need Is Cash	1978	200000	910000
22028	Ghostbusters: Afterlife	2021	180000000	760000000

This query could be amended to show the movies that they have not appeared in together e.g.

Field(s) and/or calculation(s)	Movie.id, title, yr, budget, gross			
Table(s) and/or query(-ies)	Creative, Role, Movie			
Search criteria	Creative.name = "Dan Aykroyd"			
	AND Movie.id NOT IN	**Inner query**	**Field(s) and/or calculation(s)**	Movie.id
			Table(s)	Creative, Role, Movie
			Search criteria	Creative.name = "Bill Murray"
Grouping				
Having				
Sort order				

Let's try another example.

We want to see all the movies, in order of budget descending, that had a bigger budget than Steven Spielberg's biggest budget movie.

Field(s) and/or calculation(s)	*			
Table(s) and/or query(-ies)	Movie			
Search criteria	budget >	**Inner query**	**Field(s) and/or calculation(s)**	MAX(budget)
			Table(s)	Movie, Directed, Creative
			Search criteria	name = "Steven Spielberg"
Grouping				
Having				
Sort order	budget DESC			

2.7.4.1 ANY operator

The ANY operator must follow a comparison operator. It means return TRUE if the comparison is TRUE for any of the values in the columns the subquery returns.

So, to find out if "Judi Dench" has been in a "Pirates of the Caribbean" movie the query could be:

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Creative, Role, Movie</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">Creative.name = "Judi Dench"</td></tr>
<tr><td rowspan="3">AND Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie</td></tr>
<tr><td>Search criteria</td><td>title LIKE "%Pirates of the Caribbean%"</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

This will find ANY movie where Movie.id appears in ANY of the values returned by the inner query. "= ANY" behaves in the same way as IN in this case.

Top tip

Note the use of % around the search term here. The percent sign represents zero, one, or multiple characters.

So the query `title LIKE "%Pirates of the Caribbean%"` will therefore search for any movie names that contain "Pirates of the Caribbean" in the title.

Let's try another example.

Let's see if ANY of the movies directed by "James Cameron" had a bigger production budget than ANY of the movies by directors "Steven Spielberg", "George Lucas" or "Christopher Nolan".

Field(s) and/or calculation(s)	title			
Table(s) and/or query(-ies)	Creative, Directed, Movie			
Search criteria	Creative.name = "James Cameron"			
	AND budget > ANY	**Inner query**	**Field(s) and/or calculation(s)**	budget
			Table(s)	Movie, Directed, Creative
			Search criteria	name = “Christopher Nolan” OR name = “Steven Spielberg” OR name = “George Lucas”
Grouping				
Having				
Sort order				

A further example would be to find the total gross for directors who also stared in their own movies where the totalgross is over 120 million.

Field(s) and/or calculation(s)	SUM(gross) AS totalgross, name			
Table(s) and/or query(-ies)	Creative, Directed, Movie			
Search criteria	Movie.id = ANY	**Inner query**	**Field(s) and/or calculation(s)**	Directed.movieid
			Table(s)	Directed, Role
			Search criteria	Role.ord = 1 AND Role.creativeid = Directed.creativeid
Grouping	Creative.creativeid			
Having	SUM(gross) > 120000000			
Sort order	totalgross DESC			

2.7.4.2 EXISTS operator

When a subquery is used with the EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE.

For example:

```
SELECT * FROM table WHERE EXISTS (subquery);
```

If the subquery returns any rows, EXISTS subquery will return TRUE, otherwise, it will return FALSE.

Using this we could find out if "Tilda Swinton" has ever been in a film with a budget of 100,000,000 or more.

Field(s) and/or calculation(s)	*			
Table(s) and/or query(-ies)	Movie			
Search criteria	EXISTS	**Inner query**	**Field(s) and/or calculation(s)**	movieid
			Table(s)	Role, Creative
			Search criteria	name = “Tilda Swinton” AND budget >= 100000000 AND Movie.id = Role.movieid
Grouping				
Having				
Sort order	budget DESC			

This matches the Movie.id from the movie in the outer query with Role.movieid in the inner query.

We can use the NOT operator to negate an EXISTS subquery. Let's search for movies with "Patrick Stewart" which did not gross more than double their budgets.

Field(s) and/or calculation(s)	title, name, yr, budget, gross			
Table(s) and/or query(-ies)	Movie, Role, Creative			
Search criteria	name = "Patrick Stewart" AND			
	NOT EXISTS	**Inner query**	**Field(s) and/or calculation(s)**	Movie.id
			Table(s)	Movie
			Search criteria	Movie.id = Role.movieid AND (Movie.budget * 2) > Movie.gross
Grouping				
Having				
Sort order	budget DESC			

2.7.5 Combining subqueries

An outer query can contain more than one subquery. For example, to find all the movies with a budget larger than the maximum budget for any "Hugh Jackman" or "Jason Statham" movie in which "Johnny Depp" played a role.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, yr, budget, gross</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie</td></tr>
<tr><td rowspan="6">Search criteria</td><td rowspan="3">Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Creative, Role, Movie</td></tr>
<tr><td>Search criteria</td><td>name = “Johnny Depp”</td></tr>
<tr><td rowspan="3">AND budget ></td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>MAX(budget)</td></tr>
<tr><td>Table(s)</td><td>Creative, Role, Movie</td></tr>
<tr><td>Search criteria</td><td>name = “Hugh Jackman”
OR name = “Jason Statham”</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">budget DESC</td></tr>
</table>

Another example could show the total number of cast members for each movie which was directed by "Ridley Scott" and had "Russell Crowe" in the cast.

In this case, a SELECT statement and aliases are used to add a subquery as an additional table called "Castlist" with "castid" and "castsize". castid is the Movie.id and castsize is a count of the number of roles in the movie. The Castlist answer table can be equi-joined with the movie table using the condition castid = Movie.id.

Field(s) and/or calculation(s)	title, yr, budget, name, castsize			
Table(s) and/or query(-ies)	Movie, Directed, Creative,	**Inner query**	**Field(s) and/or calculation(s)**	Movie.id AS castid, COUNT(*) AS castsize
			Table(s)	Role, Creative, Movie
			Search criteria	
			Grouping	Movie.id
Search criteria	Movie.id = castid AND			
	name = “Ridley Scott” AND			
	Movie.id = ANY	**Inner query**	**Field(s) and/or calculation(s)**	Movie.id
			Table(s)	Movie, Role, Creative
			Search criteria	name = “Russell Crowe”
Grouping				
Having				
Sort order	castsize DESC			

Quiz: Query design Go online

Design queries, using the layout shown previously, for each of the following questions.

Q42: Show the title, budget and castsize for movies with more than six people in roles.
(HINT: COUNT of roles, GROUP BY and HAVING can be used.)

. . . . . .

Q43: Show title and gross for all movies with a budget between 10,000,000 and 20,000,000 which include "Tom Hanks" in a role.

. . . . . .

Q44: Show movie titles for all movies which have roles with either "James Earl Jones", "Samuel L. Jackson" or "Lawrence Fishburne".

. . . . . .

Q45: Use a subquery to display the name, budget, gross and main start of all movies directed by "Francis Ford Coppola" or "Martin Scorsese".

. . . . . .

Q46: Use a subquery to display the title and gross of "Star Wars" movies not directed by "George Lucas".

. . . . . .

Q47: Display the movie with the highest budget and the director.

2.8 Summary

Summary

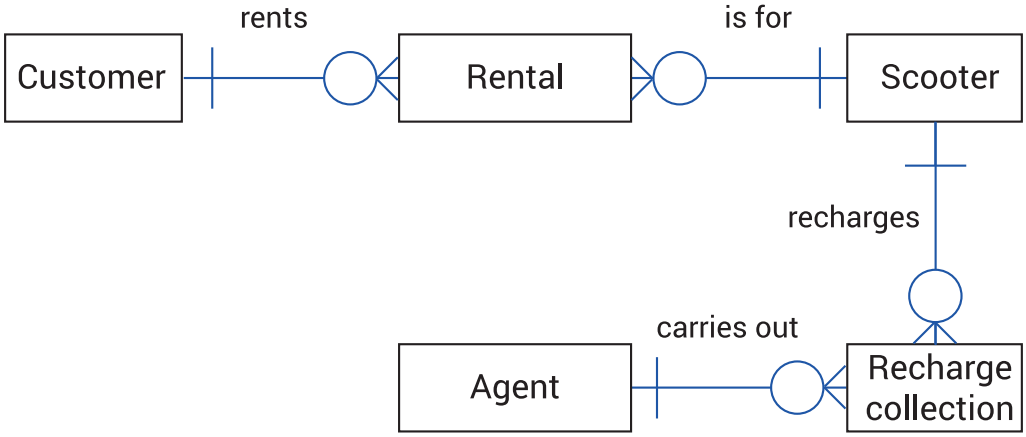
You should now be able to:

- create entity-relationship diagrams detailing entity type (strong/weak) and relationship participation (mandatory/optional);
- identify relationship participation from an entity-occurrence diagram;
- create and use surrogate keys;
- create data dictionaries, that make use of SQL data types, keys and validation;
- Use the SQL data types: varchar, integer, float, date and time;
- create queries that use the "having" operator.

2.9 End of topic test

End of topic test: Design Go online

Q48: Which entities are mandatory in the following entity-relationship diagram?



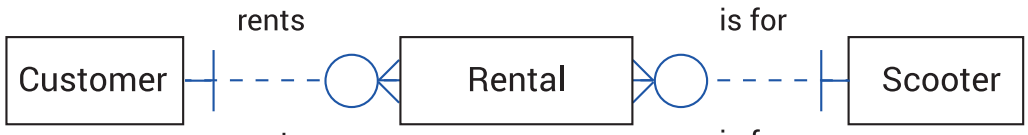
a) Customer, Rental, Recharge Collection, Agent
b) Rental, Recharge collection
c) Agent, Scooter, Customer, Recharge collection
d) Agent, Scooter, Customer

. . . . . .

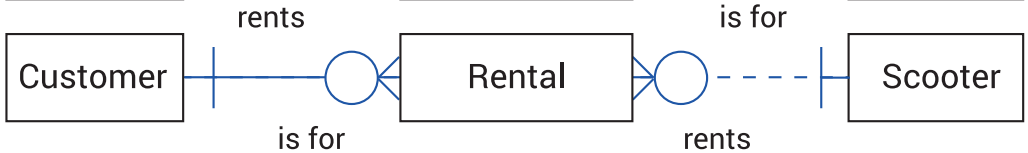
Q49: Which is the correct entity-relationship diagram for the following entities?

Customer (customerid, mobileno)
Rental (customerid, scooterid, rentaldate, rentaltimein, rentaltimeout)
Scooter (scooterid, make, model, serialnumber)

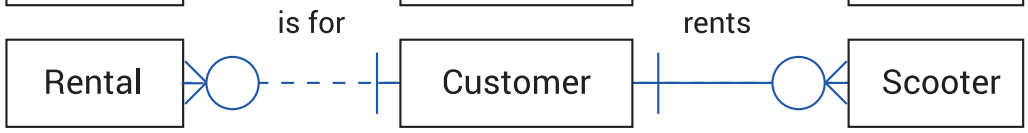
a)



b)



c)



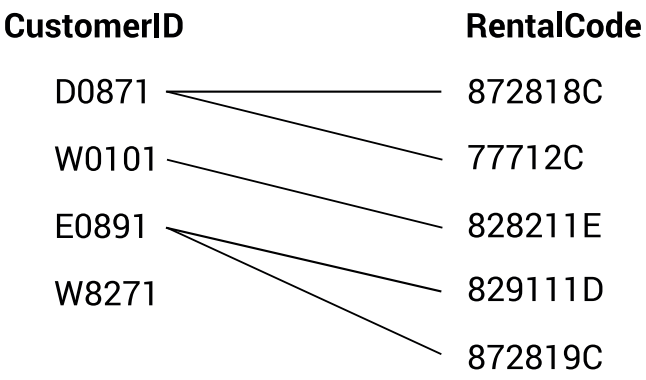
. . . . . .

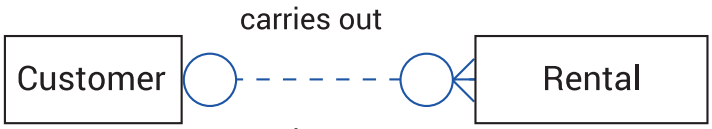
Q50: Which is the correct definition of a weak entity?

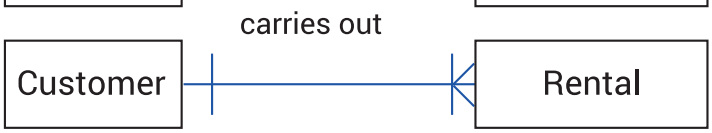
a) It has a minimum of two relationships and a foreign key.
b) It is an entity that cannot be uniquely identified by its attributes alone and therefore requires at least one foreign key from another entity to create a primary key.
c) It provides a foreign key to support the forming of a key in a strong entity.
d) It links foreign and primary keys so that relationship can be established between two or more entities.

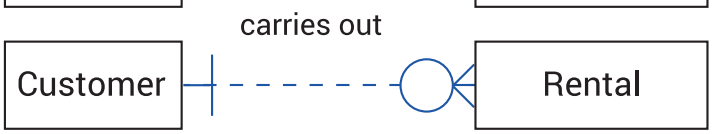
. . . . . .

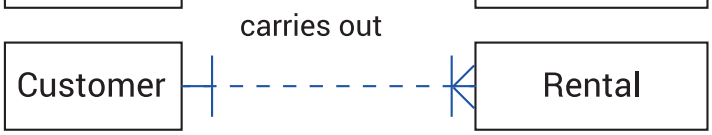
Q51: Which entity-relationship diagram is correct for the following entity-occurrence diagram?



a)


b)


c)


d)


. . . . . .

Q52: A data dictionary is used to represent entities, attributes and their domain constraints. Which of the following is not a data type used at Advanced Higher level?

a) VARCHAR
b) BLOB
c) INT
d) FLOAT

. . . . . .

Q53: What is an inner query?

a) A query with a WHERE condition which connects two attributes of the same entity.
b) A query which only has one WHERE condition.
c) A query which is nested inside another query and generates an answer table which the outer query uses.
d) A query which generates an answer table referencing all the entities within the FROM clause.

Topic 3

Implementation

Contents

Prerequisites

From your studies at Higher you should already know how to:

- describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables;
- use UPDATE, SELECT, DELETE, INSERT statements making use of wildcards, aggregate functions (MIN, MAX, AVG, SUM, COUNT), computed values, alias, GROUP BY, ORDER BY and WHERE;
- read and explain code that makes use of the above SQL.

Learning objective

By the end of this topic you should be able to:

- implement a relational database using SQL Data Definition Language (DDL) and Data Manipulation Language (DML) to match the design;
- describe, exemplify, and implement SQL operations to create databases and tables (CREATE DATABASE, CREATE TABLE);
- define the following constraints when creating tables:
  - primary key;
  - foreign key;
  - not null;
  - check;
  - auto increment.
- define the following data types when creating tables or within queries:
  - varchar;
  - integer;
  - float;
  - date;
  - time.
- describe, exemplify, and implement SQL operations to delete databases and tables (DROP DATABASE, DROP TABLE);
- use Data Manipulation Language (DML) to INSERT INTO, UPDATE and DELETE FROM tables in a database;
- describe, exemplify, and implement:
  - SQL SELECT statements that make use of the HAVING clause;
  - statements that make use of logical operators (NOT, BETWEEN, IN, ANY, EXISTS);
  - subqueries used with the WHERE clause of SELECT statements.
- read and explain code that uses the SQL at this level.

3.1 Revision

Quiz: Revision Go online

Q1: With SQL how can you delete the records where the "firstname" is "Peter" in a table named "developer"?

a) `DELETE ROW firstname ='Peter' FROM developer;`
b) `DELETE FROM developer WHERE firstname = 'Peter';`
c) `DELETE 'Peter' FROM developer;`
d) `DELETE firstname = 'Peter' FROM developer;`

.....

Q2: With SQL how can you select all the records from a table named "developer" where the value of the column "firstname" starts with an "a"?

a) `SELECT * FROM developer WHERE firstname LIKE 'a%';`
b) `SELECT * FROM developer WHERE firstname LIKE '%a';`
c) `SELECT * FROM developer WHERE firstname ='%a%';`
d) `SELECT * FROM developer WHERE firstname ='a';`

.....

Q3: Which of the following produces an answer set which shows all the clients with a maxrent of less than 800?

a) `SELECT maxrent FROM client WHERE maxrent < 800;`
b) `SELECT * FROM client WHERE maxrent IS NOT 800;`
c) `SELECT * FROM client WHERE maxrent < 800;`
d) `SELECT * FROM client WHERE maxrent > 800;`

.....

Q4: A database table is shown as:

Table: Car

type	model	year	listprice	serviceneed
SUV	A716	2018	60000	2
SUV	X75	2018	72000	3
Coupe	XF80	2017	35000	2
Coupe	Radar 2	2016	75000	2
SUV	Sport 57	2016	73000	2
Coupe	EOS91	2018	42000	4
Coupe	MX81	2017	39000	5

This query is run:

```
SELECT type, MAX(listprice) AS "most expensive"
FROM Car
GROUP BY type;
```

Select the correct answer table generated by this query.

a)

type	most expensive
SUV	60000
Coupe	39000

b)

type	most expensive
SUV	35000
Coupe	75000

c)

type	most expensive
SUV	73000
Coupe	75000

d)

type	most expensive
SUV	72000
Coupe	42000

. . . . . .

Q5: A query is run against following two tables:

Table: Competition

competitionid	country	venue	eventdate
1	UK	Southport	13/05/2019
2	Switzerland	Bern	29/08/2019
3	Canada	Montreal	08/09/2019
...	...	...	...

Table: Entryticket

competitionid	team	placing	prizevalue
1	GoGames	Gold	20
1	XForceLite	Silver	15
1	TrashCans	Bronze	10
2	XForceLite	Gold	30
2	Question101	Silver	10
2	TrashCans	Bronze	0
3	GoGames	Silver	40
...	...	...	...

The query:

```
SELECT country, SUM(prizevalue)
FROM Competition, Entryticket
WHERE Competition.competitionid = Entryticket.competitionid
GROUP BY country;
```

What is the purpose of the GROUP BY line of the SQL statement?

a) To enforce referential integrity.
b) To allow aggregation of data using COUNT.
c) To group results by country so that each country only appears once.
d) To establish a one-to-many grouped relationship.

3.2 Introduction

To implement and retrieve data from a database at Advanced Higher level will require that you use both **Data Definition Language (DDL)** and **Data Manipulation Language (DML)**.

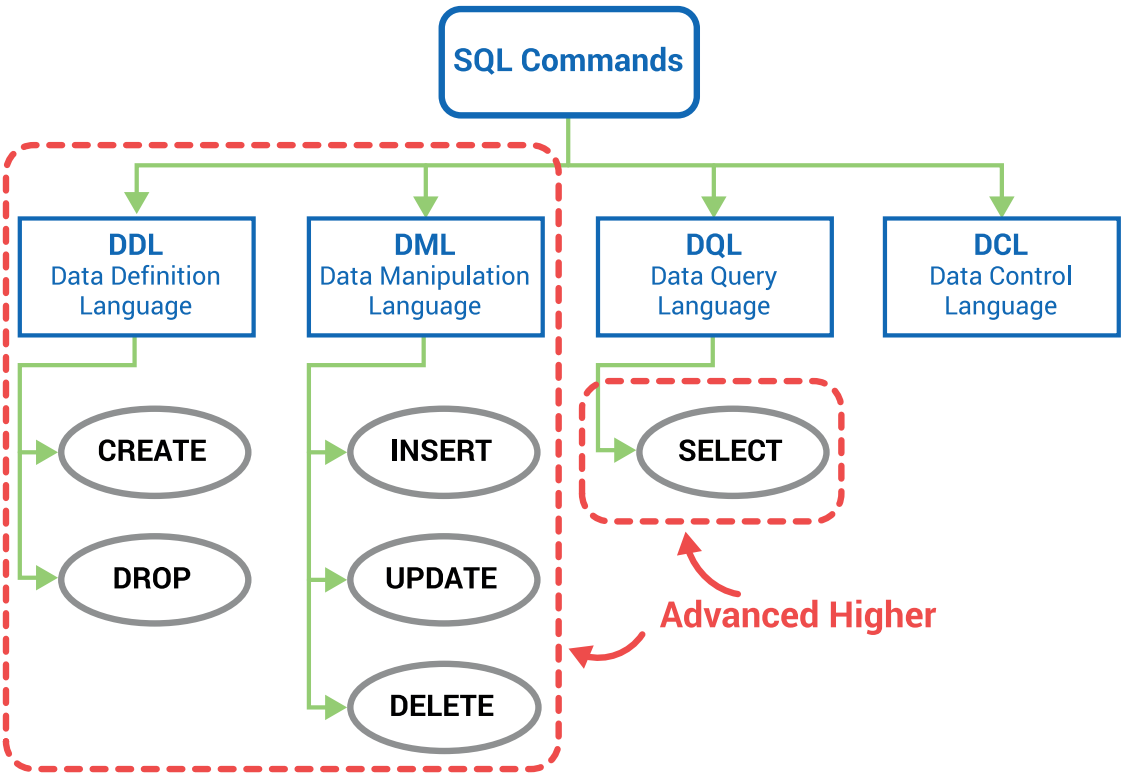
As the name suggests DDL is the collection of **SQL** commands that are concerned with creating and configuring a database and the tables within it.

DML is concerned with the inserting, deleting and modification of data in a database.

We will look at both of these in more detail later on in this topic.

There are two further categories of SQL which are **Data Query Language (DQL)** and **Data Control Language (DCL)**. You are not required to know these additional two categories for Advanced Higher level (but you will use the SELECT keyword which is from DQL).

DCL contains commands which control access to the database and the tables within.



3.3 Data Definition Language (DDL)

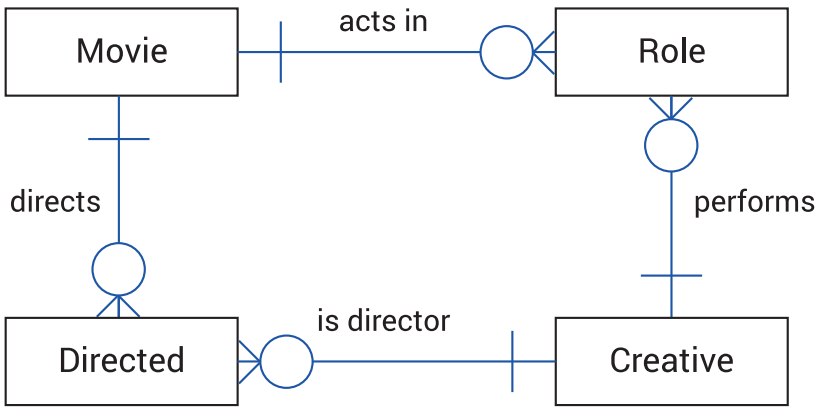
Learning objective

By the end of this section you should be able to:

- implement a relational database using SQL Data Definition Language (DDL) to match the design;
- describe, exemplify, and implement SQL operations to create databases and tables (CREATE DATABASE, CREATE TABLE);
- define the following constraints when creating tables:
  - primary key;
  - foreign key;
  - not null;
  - check;
  - auto increment.
- define the following data types when creating tables or within queries:
  - varchar;
  - integer;
  - float;
  - date;
  - time.
- describe, exemplify, and implement SQL operations to delete databases and tables (DROP DATABASE, DROP TABLE);

In this section we will create the ***Movies*** database that we introduced in the ***Subqueries*** section of the **Design** topic.

Figure 3.1: Entity-relationship diagram for the Movies database



Movie (id, title, rating, yr, budget, gross)
Role (roleid, movieid, creativeid, ord)
Creative (creativeid, name)
Directed (directorid, movieid, creativeid)

For implementation, this database is being extended to include a ***rating*** attribute for the movie. This is the average user review rating from the IMDB website for the movie.

To create this database, you can refer to the **data dictionary**:

Figure 3.2: Data dictionary for the Movies database

Entity: Movie

Attribute name	Key	Type	Size	Required	Validation
id	PK	INT		Yes	autoincrement
title		VARCHAR	50	Yes	
rating		FLOAT	3,1	Yes	>=0 and <= 10
yr		DATE		Yes	
budget		INT		No	
gross		BIGINT		No	

Entity: Role

Attribute name	Key	Type	Size	Required	Validation
roleid	PK	INT		Yes	
movieid	FK	INT		Yes	Valid id from Movie
creativeid	FK	INT		Yes	Valid creativeid from Creative
ord		INT		Yes	

Entity: Creative

Attribute name	Key	Type	Size	Required	Validation
creativeid	PK	INT		Yes	
name		VARCHAR	50	Yes	

Entity: Directed					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
directorid	PK	INT		Yes	
movieid	FK	INT		Yes	Valid id from Movie
creativeid	FK	INT		Yes	Valid creativeid from Creative

This exercise assumes that you are using a tool with command line access to an SQL engine such as MySQL, MSACCESS SQL, Oracle or similar. For example, the following exercises can be completed using the SQL tab from within the web based phpMyAdmin application. This application is commonly distributed with *AMP stacks.

The first action is to use a DDL **CREATE DATABASE** statement to create the ***Movies*** database.

3.3.1 CREATE DATABASE

Example : Create a database

To create the ***Movies*** database enter the SQL:

```
CREATE DATABASE Movies;
```

This will create an empty database called "Movies".

In order to use this database, it has to be selected.

Example : Select a database

We can select the database by either clicking on the database name (in your web based editor e.g. phpMyAdmin) or by using the MySQL Command:

```
USE Movies;
```

3.3.2 CREATE TABLE

A database table is created using the information available in the data dictionary. This details the **constraints** and data types to be applied to each **column** when it is implemented.

The columns in a table are the attributes from the data dictionary.

In the following sections we will look at how to create a database table based on the constraints defined in the data dictionary.

3.3.3 Constraints

Constraints for a table will specify whether an attribute:

- is a primary and/or foreign key;

- is allowed to have a null value;
- has a validation rule or rules set against it;
- is set to have a new auto increment value automatically set for each new **row** added.

Primary key

To specify that a column is the **primary key**, the PRIMARY KEY constraint is used in the CREATE TABLE statement.

```
CREATE TABLE table_name (
  column datatype PRIMARY KEY,
  ...
);
```

When the primary key has more than one column, the PRIMARY KEY is added as a table constraint.

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  ...,
  PRIMARY KEY (column1, column2)
);
```

Foreign key

To specify that a column is a **foreign key**, the FOREIGN KEY constraint is used in the CREATE TABLE statement.

```
CREATE TABLE table1 (
  column1 datatype PRIMARY KEY,
  column2 datatype,
  ...,
  FOREIGN KEY (column2) REFERENCES table2 (column1)
);
```

This creates a table with a foreign key, column2, which is linked to column1 of table2.

Where a table has multiple foreign keys, each is set as a different table constraint.

```
CREATE TABLE table1 (
  column1 datatype PRIMARY KEY,
  column2 datatype,
  column3 datatype,
  ...,
  FOREIGN KEY (column2) REFERENCES table2 (column1),
  FOREIGN KEY (column3) REFERENCES table3 (column1)
);
```

Not null

In SQL, by default, a column can hold NULL values. A null value means that a data value does not exist in the database.

If a value is required for a column, then that column must be constrained to not allow NULL values.

```
CREATE TABLE table_name (
  column datatype NOT NULL,
  ...
);
```

Check

The CHECK constraint is used to limit the value that can be placed in a column.

```
CREATE TABLE table_name (
  column datatype CHECK (expression),
  ...
);
```

The expression can be one or more rules relating to the column.

Auto increment

Auto increment automatically generates a new unique number when a row is added to the table. Typically, this is used to generate a unique arbitrary value to act as a primary key.

```
CREATE TABLE table_name (
  column datatype PRIMARY KEY AUTO_INCREMENT,
  ...
);
```

3.3.4 Data types

You were introduced to SQL data types in the **Design** topic. The following **data types** are used when creating tables or within queries:

- varchar;
- integer;
- float;
- date;
- time.

Varchar

The **VARCHAR** data type is used because this is very efficient when storing text; it only stores the number of characters used rather than storing the whole length of the field according to the declared size.

Integer

The **INTEGER** data types have five different variants in SQL. These different variants all store integers, the difference between them is the number of bytes of storage that they use for this.

Variant	Storage (bytes)	Minimum value signed	Minimum value unsigned	Maximum value signed	Maximum value unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63} - 1$	$2^{64} - 1$

Float

Real number / floating point values are stored in the database using the **FLOAT** data type.

Date

The **DATE** type in SQL records any date in the range '1000-01-01' to '9999-12-31'. It uses dates in the format: YYYY-MM-DD.

Time

The **TIME** type in SQL records values in the format 'hh:mm:ss'. It can be used to store times which go beyond those in the 24hr clock. This means as well as times in the day, it can also be used to store durations.

Top tip

Be careful about assigning abbreviated values to a TIME column. SQL interprets abbreviated TIME values with colons as time of the day. That is, '11:12' means '11:12:00', not '00:11:12'.

3.3.4.1 Setting the size of data types

A size is set for data types that support the use of a size, by using a value or values in braces "()" after the data type.

VARCHAR

The VARCHAR data type uses size to set the maximum number of characters that it can contain e.g. VARCHAR(4).

A field of VARCHAR stores text efficiently and typically uses only the space required e.g. if a column is VARCHAR(4) and a value "ab" is stored in it, it will use storage for two characters rather than four.

FLOAT

The FLOAT data type makes use of two values when setting a size. These are:

- display length;
- number of decimals.

So, a column declared as FLOAT(4,2) with the value 23.2 stored in it will be displayed as 23.20. The display length is the total number of digits and the number of decimals defines the number of digits shown after the decimal point. If a size is not set i.e. if the data type is just set to FLOAT, then the default of FLOAT(10,2) is used.

INT, DATE and TIME

For the purpose of Advanced Higher, INT, DATE and TIME data types do not make use of a size. You may see in your database, once you have declared an integer column, that the size defaults to a value as shown in the following table.

Variant	Default size	Minimum value signed	Maximum value signed
TINYINT	4	-128	127
SMALLINT	6	-32768	32767
MEDIUMINT	8	-8388608	8388607
INT	11	-2147483648	2147483647
BIGINT	32	-2^{63}	$2^{63} - 1$

The default size is the number of digits with the sign e.g. so -28171 as a SMALLINT has a size of 6.

3.3.5 Creating tables for the Movies database

In the following activities you will create the tables for the ***Movies*** database.

Here is a reminder of the **entity-relationship diagram** and the data dictionary for the Movies database that we looked at earlier:

- Entity-relationship diagram for the Movies database: Figure 3.1
- Data dictionary for the Movies database: Figure 3.2

Top tip

When creating your tables for your database, remember to start with tables which do not include foreign keys. You cannot add a FOREIGN KEY constraint if the related primary key is not already established in the database.

Let's begin with the ***Creative*** table.

Creating the Creative table Go online

Execute this SQL in your relational database system

```
1 CREATE TABLE Creative (
2   creativeid INT PRIMARY KEY AUTO_INCREMENT,
3   name VARCHAR(50) NOT NULL
4 );
```

This creates a table called ***Creative*** with two columns: *creativeid* and *name*.

- *creativeid* is an INT data type size 6 and the primary key. It will be automatically incremented each time a new row is added to the table;
- *name* is VARCHAR with a maximum length of 50 characters.

NOT NULL is not required for the PRIMARY KEY as primary key columns cannot have NULL values.

We will create the ***Movie*** table next.

Creating the Movie table Go online

Execute this SQL in your relational database system

```
1 CREATE TABLE Movie (
2   id INT PRIMARY KEY AUTO_INCREMENT,
3   title VARCHAR(50) NOT NULL,
4   rating FLOAT(3,1) NOT NULL CHECK(rating >=0 AND rating <=10),
5   yr INT NOT NULL,
6   budget INT,
7   gross BIGINT
8 );
```

This creates a table called ***Movie***, with the columns *id*, *title*, *rating*, *yr*, *budget* and *gross*.

- *id* is an integer value, primary key and will auto increment;
- *title* is VARCHAR with a maximum length of 50 characters;
- *rating* is a FLOAT value that can display 3 digits and one decimal place - this covers ratings from 0.0 to 10.0 in this case (even if all ratings, apart from 10.0 would only need FLOAT(2,1)!).

  The rating also has a CHECK constraint which requires that any rating entered is greater than or equal to 0 and less than or equal to 10.

- *yr* is an integer (using DATE would not be efficient as we would need to store a day and a year as well!);
- *budget* is an INT value;
- *gross* is a BIGINT as some modern movies have made more than can be stored in the range of INT.

Now that the tables which have primary keys, that are used as foreign keys, are in place the remaining two tables can be created: the ***Role*** table and the ***Directed*** table.

Creating the Role table Go online

Execute this SQL in your relational database system

```
CREATE TABLE Role (
  roleid INT PRIMARY KEY AUTO_INCREMENT,
  movieid INT NOT NULL,
  creativeid INT NOT NULL,
  ord INT NOT NULL,
  FOREIGN KEY (movieid) REFERENCES Movie(id),
  FOREIGN KEY (creativeid) REFERENCES Creative(creativeid)
);
```

This creates the ***Role*** table with a unique primary key that auto increments.

The *foreign keys* movieid and createiveid each are NOT NULL and are referenced back to their source tables and columns.

This just leaves us with the ***Directed*** table to create.

Creating the Directed table Go online

Q6: Write the SQL to create the ***Directed*** table for the ***Movies*** database. Refer to the data dictionary in Figure 3.2 for column names and constraints.

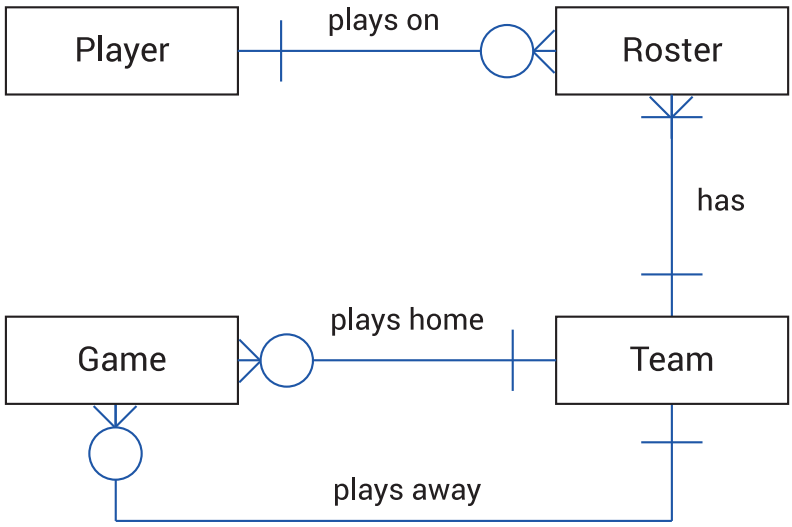
3.3.6 Creating database and tables practice

In this section you will complete two practical activities where you can put into practice what you have learned by creating databases and tables.

Creating database and tables for the Esports league database Go online

An Esports league consists of teams of players, where players can change teams and each team has a current roster of their players. Teams play games against each other and the score for each side playing is recorded. Each game consists of the best of three rounds. The duration of each game is also recorded.

Figure 3.3: Entity-relationship diagram for the Esports league



The data dictionary for the Esports league is shown below.

Figure 3.4: Data dictionary for the Esports league database

Entity: Team

Attribute name	Key	Type	Size	Required	Validation
teamid	PK	INT		Yes	
handle		VARCHAR	15	Yes	

Entity: Roster

Attribute name	Key	Type	Size	Required	Validation
rosterid	PK	INT		Yes	
teamid	FK	INT		Yes	
playerid	FK	INT			
position		TINYINT		Yes	
joindate		DATE		Yes	

Entity: Player					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
playerid	PK	INT		Yes	
handle		VARCHAR	15	Yes	
playerstatus		VARCHAR	7	Yes	Either "Active" or "Retired"

Entity: Game					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
gameid	PK	INT		Yes	
hometeamid	FK	INT		Yes	
awayteamid	FK	INT		Yes	Not equal to hometeamid
gamedate		DATE		Yes	
duration		TIME		Yes	
homescore		TINYINT		Yes	>=0 and <=2
awayscore		TINYINT		Yes	>=0 and <=2

Q7: Write the SQL to create the database and each of the four tables for the **Esports league** using the information given in the entity relationship diagram and data dictionary.

Creating database and tables for the Shipping database Go online

The data dictionary for a Shipping database is shown below.

Figure 3.5: Data dictionary for the Shipping database

Entity: Company

Attribute name	Key	Type	Size	Required	Validation
companyid	PK	INT		Yes	
coname		VARCHAR	15	Yes	
telephone		VARCHAR	15	Yes	
email		VARCHAR	320	Yes	

Entity: Container

Attribute name	Key	Type	Size	Required	Validation
containerid	PK	INT		Yes	
contents		VARCHAR	20	Yes	
originated		VARCHAR	40	Yes	
destination		VARCHAR	40	Yes	

Entity: Ship

Attribute name	Key	Type	Size	Required	Validation
shipid	PK	INT		Yes	
shipname		VARCHAR	25	Yes	
companyid	FK	INT		Yes	Existing companyid from Company
vesselcapacity		MEDIUMINT		Yes	>=8000 and <=32000
regport		VARCHAR	20	Yes	

Entity: Booking					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
bookingid	PK	INT		Yes	
shipid	FK	INT		Yes	Existing shipid from Ship
containerid	FK	INT		Yes	Existing containerid from Container
dateout		DATE		Yes	
datein		DATE		Yes	Must be after dateout

Q8: Write the SQL to create the database and each of the four tables for the **Shipping** using the information given in the data dictionary.

3.3.7 DROP statement

For the purpose of Advanced Higher level, the DROP statement is used to remove a table or a database.

Examples

1. Remove a table

To remove a table called *Platform*, the following statement would be used:

```
DROP TABLE Platform;
```

2. Remove a database

To remove a database called *Mydatabase*, the SQL statement would be:

```
DROP DATABASE Mydatabase;
```

Key point

Warning: There is no undo or "Are you sure Y/N" generally with SQL.

Once you DROP a table or database it is gone (unless you have exported a backup).

3.4 Data Manipulation Language (DML)

Learning objective

By the end of this section you should be able to:

- use Data Manipulation Language (DML) to INSERT INTO, UPDATE and DELETE FROM tables in a database.

Data manipulation language is used to INSERT INTO, UPDATE and DELETE FROM tables of a database.

3.4.1 Insert

As you know from Higher level, the INSERT command is used to add a row or rows to a table.

The format of the INSERT INTO command in DML is:

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

Key point

If you are adding values for all the columns of a table, you do not need to specify the column names in the SQL query.

However, make sure the order of the values is in the same order as the columns in the table.

3.4.2 Update

The UPDATE command is used to amend rows in a table. It can be used to update multiple rows at the same time and can make use of WHERE clauses to select the data to be updated.

The syntax of UPDATE command is:

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

Key point

Remember: The WHERE clause in the UPDATE statement is really important. If you forget it and just run UPDATE without it, you will update ***every*** row in your table and lose all your data.

3.4.3 Delete

As you know from Higher level, the DELETE command is used to delete rows in a table. It can be used to delete multiple rows at the same time and makes use of WHERE clauses to select the data

to be deleted.

The syntax of the DELETE FROM command in DML is:

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

Key point

Remember: The WHERE clause in the DELETE FROM statement is really important. If you forget it and just run DELETE FROM without it, you will delete ***every*** row in your table.

3.5 SELECT statements

Learning objective

By the end of this section you should be able to:

- describe, exemplify, and implement:
  - SQL SELECT statements that make use of the HAVING clause;
  - statements that make use of logical operators (NOT, BETWEEN, IN, ANY, EXISTS);
  - subqueries used with the WHERE clause of SELECT statements.
- read and explain code that uses the SQL at this level.

Each of the examples in this section require that you use a specific prepopulated database. These can be downloaded as zip files here:

Code 2: Database downloads for activities

- Film_locations database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/4DF93285-51E4-0FB5-ABDA-F503C8D86802/film_locations.zip
- Shipping database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/387E5AA2-AD10-EA6F-22FD-6E3B2DCEECAC/shipping.zip
- Movies database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/6FD22C38-3174-685C-D559-1F657D5270B8/movies.zip

Import these databases into your database server before you attempt the activities that follow.

3.5.1 HAVING clause of the SELECT statement

We have already learned in the **Design** topic that the HAVING clause is used in SELECT statements. It acts as a filter for group of rows or aggregated data (e.g. MIN, MAX, AVG, SUM, COUNT). It is often used with the GROUP BY clause so that data is selected based on a specified condition.

We will see how the HAVING clause is used in the following activities.

Using the HAVING clause: Film_locations database Go online

This activity uses the Film_locations database that you downloaded at the start of this section (see Code 2).

A query is required which will show cities with three or more filming locations. For this query the locations are first grouped and counted by city and then the HAVING clause is used to filter the results so that only those with a count of 3 or more are shown.

The design for this query would be:

Field(s) and/or calculation(s)	city, Count(*)
Tables(s) and/or query(-ies)	Location
Search Criteria	
Grouping	city
Having	COUNT(*) >= 3
Sort order	

The SQL statement to enter in your database system is:

```
1 SELECT city, Count(*)
2 FROM Location
3 GROUP BY city
4 HAVING COUNT(*) >=3;
```

Using the HAVING clause: Shipping database Go online

This activity uses the Shipping database that you downloaded at the start of this section (see Code 2).

A query is required to total the capacity of all ships booked with containers for 'Grangemouth, Scotland' with a *dateout* of "2020-05-01".

The design for this query would be:

Field(s) and/or calculation(s)	SUM(vesselcapacity)
Tables(s) and/or query(-ies)	Ship, Booking, Container
Search Criteria	destination = "Grangemouth, Scotland"
Grouping	dateout
Having	dateout = "2020-05-01"
Sort order	

The SQL statement to enter in your database system is:

```
SELECT SUM(vesselcapacity)
FROM Ship, Booking, Container
WHERE Booking.containerid = Container.containerid AND
  Booking.shipid = Ship.shipid AND
    destination = "Grangemouth, Scotland"
GROUP BY dateout
HAVING dateout = "2020-05-01";
```

Key point

Important: It's important to note that the equi-joins between tables are not shown in the query design. These are assumed when multiple tables appear in the query design.

When this is translated into SQL the equi-joins between the tables must be added for the SQL to function as required.

3.5.2 Logical operators

The logical operators NOT, BETWEEN and IN are used in SQL queries at this level.

NOT

The NOT operator reverses or negates a condition in a query.

In this case, it shows the sum of vessel capacity with a destination other than "Grangemouth, Scotland".

Field(s) and/or calculation(s)	SUM(vesselcapacity)
Tables(s) and/or query(-ies)	Ship, Booking, Container
Search Criteria	NOT (destination = "Grangemouth, Scotland")
Grouping	dateout
Having	dateout = "2020-05-01"
Sort order	

Using NOT: Shipping database Go online

This activity uses the Shipping database that you downloaded at the start of this section (see Code 2).

The SQL statement to enter in your database system is:

```
SELECT SUM(vesselcapacity)
FROM Ship, Booking, Container
WHERE Booking.containerid = Container.containerid AND
  Booking.shipid = Ship.shipid AND
    NOT(destination = "Grangemouth, Scotland")
GROUP BY dateout
HAVING dateout = "2020-05-01";
```

BETWEEN

The BETWEEN operator is a logical operator that allows you to specify whether a value is in a range or not. It works for dates and numeric values.

For example:

Field(s) and/or calculation(s)	*
Tables(s) and/or query(-ies)	Booking
Search Criteria	dateout BETWEEN "2020-04-28" AND "2020-05-03"
Grouping	
Having	
Sort order	

Using BETWEEN: Shipping database Go online

This activity uses the Shipping database that you downloaded at the start of this section (see Code 2).

The SQL statement to enter in your database system is:

```
SELECT *
FROM Booking
WHERE dateout BETWEEN "2020-04-28" AND "2020-05-03";
```

Key point

Note that dates are shown in the format YYYY-MM-DD. It is important to use this format in your queries.

IN

The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

For example:

Field(s) and/or calculation(s)	*
Tables(s) and/or query(-ies)	Booking
Search Criteria	WHERE bookingid IN (26611, 26612, 26687)
Grouping	
Having	
Sort order	

Using IN: Shipping database Go online

This activity uses the Shipping database that you downloaded at the start of this section (see Code 2).

The SQL statement to enter in your database system is:

```
1 SELECT *
2 FROM Booking
3 WHERE bookingid IN (26611, 26612, 26687);
```

This finds all values that match any of the values within the braces.

3.5.3 Subqueries

As you know from the **Design** topic a **subquery** is a query nested within another query. The subquery is used to generate an answer table that will be used in the main query for the data being retrieved.

Subqueries at Advanced Higher level are just used with the SELECT statement (subqueries can also be used within INSERT, UPDATE and DELETE statements but this is outwith the coverage of this course).

Important points to remember about subqueries:

- Subqueries must be enclosed in parentheses. These are the **'('** and **')'** characters.
- A subquery is usually added as part of the WHERE clause but can also appear in the SELECT clause or the FROM clause.
- If the subquery returns a single row value then you can use comparison operators to compare two values (such as =, <, <=, >, >=).
- If the subquery returns multiple row values then multi-row operators, (such as ANY, IN) can be used.

A subquery can be treated as an **inner query**, which is an SQL query placed as part of another query, which is called the **outer query**. The inner query retrieves data or executes first before the outer query so that the result of the inner query can be passed to the outer query.

Subqueries - activity 1: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

A list of movies in which both "Dan Aykroyd" and "Bill Murray" have appeared together is required.

An approach to this would be to find all the movies with "Bill Murray" and then see if "Dan Aykroyd" has also had a role in them.

The query to retrieve "Bill Murray" movies is the inner query.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">Movie.id, title, yr, budget, gross</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Creative, Role, Movie</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">Creative.name = "Dan Aykroyd"</td></tr>
<tr><td rowspan="3">AND Movie.id IN</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Creative, Role, Movie</td></tr>
<tr><td>Search criteria</td><td>Creative.name = "Bill Murray"</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

The SQL for the inner query is:

```
SELECT Movie.id
FROM Creative, Role, Movie
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  Creative.name = "Bill Murray";
```

This is used in the outer query as follows:

```
SELECT Movie.id, title, yr, budget, gross
FROM Creative, Role, Movie
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  Creative.name = "Dan Aykroyd" AND
  Movie.id IN (
    SELECT Movie.id
    FROM Creative, Role, Movie
    WHERE Movie.id = Role.movieid AND
      Role.creativeid = Creative.creativeid AND
      Creative.name = "Bill Murray"
    );
```

The inner query produces a list of movie ids and this is used in the WHERE clause to match Movie.id with any that appear IN the list.

This query could be amended to show the movies that "Dan Aykroyd" has appeared in which "Bill Murray" does not appear.

For example:

```
SELECT Movie.id, title, yr, budget, gross
FROM Creative, Role, Movie
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  Creative.name = "Dan Aykroyd" AND
  Movie.id NOT IN (
    SELECT Movie.id
    FROM Creative, Role, Movie
    WHERE Movie.id = Role.movieid AND
      Role.creativeid = Creative.creativeid AND
      Creative.name = "Bill Murray"
    );
```

Enter both of these queries into your database system.

Subqueries - activity 2: Movies database

Go online 

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

We want to list all the movies, in order of budget descending, that had a bigger budget than Steven Spielberg's biggest budget movie.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">*</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie</td></tr>
<tr><td rowspan="3">Search criteria</td><td rowspan="3">budget ></td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>MAX(budget)</td></tr>
<tr><td>Table(s)</td><td>Movie, Directed, Creative</td></tr>
<tr><td>Search criteria</td><td>name = "Steven Spielberg"</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">budget DESC</td></tr>
</table>

The inner query is:

```
SELECT MAX(budget)
FROM Movie, Directed, Creative
WHERE Movie.id = Directed.movieid AND
  Directed.creativeid = Creative.creativeid AND
  Creative.name = "Steven Spielberg";
```

This is then used in the outer query, which you can enter into your database server as:

```
SELECT *
FROM Movie
WHERE budget > (
  SELECT MAX(budget)
  FROM Movie, Directed, Creative
  WHERE Movie.id = Directed.movieid AND
    Directed.creativeid = Creative.creativeid AND
    Creative.name = "Steven Spielberg"
);
```

3.5.4 ANY operator

The ANY operator must follow a comparison operator. It means return TRUE if the comparison is TRUE for any of the values in the columns the subquery returns.

ANY operator - activity 1: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

So, to find out if "Judi Dench" has been in a "Pirates of the Caribbean" movie the query could be:

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Creative, Role, Movie</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">Creative.name = "Judi Dench"</td></tr>
<tr><td rowspan="3">AND Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie</td></tr>
<tr><td>Search criteria</td><td>title LIKE "%Pirates of the Caribbean%"</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

This will find ANY movie where Movie.id appears in ANY of the values returned by the inner query. "= ANY" behaves in the same way as IN in this case.

Top tip

Note the use of % around the search term here. The percent sign represents zero, one, or multiple characters.

So the query `title LIKE "%Pirates of the Caribbean%"` will therefore search for any movie names that contain "Pirates of the Caribbean" in the title.

The inner query in SQL is:

```
SELECT Movie.id
FROM Movie
WHERE title LIKE "%Pirates of the Caribbean%";
```

This finds all the movie ids for files with "Pirates of the Caribbean" in the title. This can then be used to match against movies with "Judi Dench".

The complete query (with both the outer query and the inner query) can be entered in your database server as:

```
SELECT title
FROM Creative, Role, Movie
WHERE Creative.creativeid = Role.creativeid AND
  Role.movieid = Movie.id AND
  Creative.name = "Judi Dench" AND
  Movie.id = ANY (
    SELECT Movie.id
    FROM Movie
    WHERE title LIKE "%Pirates of the Caribbean%"
  );
```

Let's try another example.

ANY operator - activity 2: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

Let's see if ANY of the movies directed by "James Cameron" had a bigger production budget than ANY of the movies by directors "Steven Spielberg", "George Lucas" or "Christopher Nolan".

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Creative, Directed, Movie</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">Creative.name = "James Cameron"</td></tr>
<tr><td rowspan="3">AND budget > ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>budget</td></tr>
<tr><td>Table(s)</td><td>Movie, Directed, Creative</td></tr>
<tr><td>Search criteria</td><td>name = “Christopher Nolan”
OR name = “Steven Spielberg”
OR name = “George Lucas”</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

The inner query is:

```
1 SELECT budget
2 FROM Creative, Directed, Movie
3 WHERE Creative.creativeid = Directed.creativeid AND
4   Directed.movieid = Movie.id AND
5   (name="Christopher Nolan" OR
6   name="Steven Spielberg" OR
7   name="George Lucas");
```

The complete query (with both the outer query and the inner query) can be entered in your database server as:

```
SELECT title
FROM Creative, Directed, Movie
WHERE Creative.creativeid = Directed.creativeid AND
  Directed.movieid = Movie.id AND
  Creative.name = "James Cameron" AND
  budget > ANY (
    SELECT budget
    FROM Creative, Directed, Movie
    WHERE Creative.creativeid = Directed.creativeid AND
      Directed.movieid = Movie.id AND
      (name="Christopher Nolan" OR
      name="Steven Spielberg" OR
      name="George Lucas")
  );
```

Let's look at one further example.

ANY operator - activity 3: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

Let's find the total gross for directors who also starred in their own movies where the totalgross is over 120 million.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">SUM(gross) AS totalgross, name</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Creative, Directed, Movie</td></tr>
<tr><td rowspan="3">Search criteria</td><td rowspan="3">Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Directed.movieid</td></tr>
<tr><td>Table(s)</td><td>Directed, Role</td></tr>
<tr><td>Search criteria</td><td>Role.ord = 1 AND
Role.creativeid =
Directed.creativeid</td></tr>
<tr><td>Grouping</td><td colspan="4">Creative.creativeid</td></tr>
<tr><td>Having</td><td colspan="4">SUM(gross) > 120000000</td></tr>
<tr><td>Sort order</td><td colspan="4">totalgross DESC</td></tr>
</table>

The inner query is:

```
SELECT Directed.movieid
FROM Directed, Role
WHERE Role.ord = 1 AND
  Role.creativeid = Directed.creativeid;
```

The complete query (with both the outer query and the inner query) can be entered in your database server as:

```
SELECT SUM(gross) AS totalgross, name
FROM Creative, Directed, Movie
WHERE Creative.creativeid = Directed.creativeid AND
  Directed.movieid = Movie.id AND
  Movie.id = ANY (
    SELECT Directed.movieid
    FROM Directed, Role
    WHERE Role.ord = 1 AND
      Role.creativeid = Directed.creativeid
)
GROUP BY Creative.creativeid
HAVING SUM(gross) > 120000000
ORDER BY totalgross DESC;
```

3.5.5 EXISTS operator

When a subquery is used with the EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE. For example:

```
SELECT * FROM table WHERE EXISTS (subquery);
```

If the subquery returns any rows, EXISTS (subquery) will return TRUE, otherwise, it will return FALSE.

EXISTS operator - activity 1: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

We could find out if "Tilda Swinton" has ever been in a film with a budget of 100,000,000 or more.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">*</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie</td></tr>
<tr><td rowspan="3">Search criteria</td><td rowspan="3">EXISTS</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>movieid</td></tr>
<tr><td>Table(s)</td><td>Role, Creative</td></tr>
<tr><td>Search criteria</td><td>name = “Tilda Swinton” AND
budget >= 100000000 AND
Movie.id = Role.movieid</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">budget DESC</td></tr>
</table>

This matches the Movie.id from the movie in the outer query with Role.movieid in the inner query.

The complete query to enter into your database server is:

```
SELECT *
FROM Movie
WHERE EXISTS (
  SELECT movieid
  FROM Role, Creative
  WHERE Role.creativeid = Creative.creativeid AND
    name = "Tilda Swinton" AND
    budget >= 100000000 AND
    Movie.id = Role.movieid
)
ORDER BY budget DESC;
```

EXISTS operator - activity 2: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

We can use the NOT operator to negate an EXISTS subquery. Let's search for movies with "Patrick Stewart" which did not gross more than double their budgets.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, name, yr, budget, gross</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie, Role, Creative</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">name = "Patrick Stewart" AND</td></tr>
<tr><td rowspan="3">NOT EXISTS</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie</td></tr>
<tr><td>Search criteria</td><td>Movie.id = Role.movieid AND (Movie.budget * 2) > Movie.gross</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">budget DESC</td></tr>
</table>

The complete query to enter into your database server is:

```
SELECT title, name, yr, budget, gross
FROM Role, Creative, Movie
WHERE Role.creativeid = Creative.creativeid AND
  name = "Patrick Stewart" AND Movie.id = Role.movieid AND
  NOT EXISTS
  (
    SELECT *
    FROM Movie
    WHERE Movie.id = Role.movieid AND
      (Movie.budget * 2) > Movie.gross
  )
ORDER BY budget DESC;
```

In this query, the Role.movieid in the inner query is the same value from the outer query. This allows the Role.movieid to be used in the subquery ***without*** the role table being added to the subquery.

3.5.6 Combining subqueries

An outer query can contain more than one subquery.

Let's work through some activities to see how to combine subqueries.

Combining subqueries - activity 1: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

Let's look at a query where we want to find all the movies with a budget larger than the maximum budget for any "Hugh Jackman" or "Jason Statham" movie in which "Johnny Depp" played a role.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, yr, budget, gross</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie</td></tr>
<tr><td rowspan="6">Search criteria</td><td rowspan="3">Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Creative, Role, Movie</td></tr>
<tr><td>Search criteria</td><td>name = “Johnny Depp”</td></tr>
<tr><td rowspan="3">AND budget ></td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>MAX(budget)</td></tr>
<tr><td>Table(s)</td><td>Creative, Role, Movie</td></tr>
<tr><td>Search criteria</td><td>name = “Hugh Jackman”
OR name = “Jason Statham”</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">budget DESC</td></tr>
</table>

The two inner queries are:

```
SELECT Movie.id
FROM Creative, Role, Movie
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  name = "Johnny Depp";
```

and

```
SELECT MAX(budget)
FROM Creative, Role, Movie
WHERE Creative.creativeid = Role.creativeid AND
  Role.movieid = Movie.id AND
  name = "Hugh Jackman" OR name = "Jason Statham");
```

The complete query to enter into your database server is:

```
SELECT title, yr, budget, gross
FROM Movie
WHERE
  Movie.id IN (
    SELECT Movie.id
    FROM Creative, Role, Movie
    WHERE Movie.id = Role.movieid AND
      Role.creativeid = Creative.creativeid AND
      name = "Johnny Depp"
  ) AND
  budget > (
    SELECT MAX(budget)
    FROM Creative, Role, Movie
    WHERE Creative.creativeid = Role.creativeid AND
      Role.movieid = Movie.id AND
      (name = "Hugh Jackman" OR name = "Jason Statham")
  )
ORDER BY budget DESC;
```

Combining subqueries - activity 2: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

We want to show the total number of cast members for each movie which was directed by "Ridley Scott" and had "Russell Crowe" in the cast.

In this case, a SELECT statement and aliases are used to add a subquery as an additional table called "Castlist" with "castid" and "castsize". castid is the Movie.id and castsize is a count of the number of roles in the movie. The Castlist answer table can be equi-joined with the movie table using the condition castid = Movie.id.

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, yr, budget, name, castsize</td></tr>
<tr><td rowspan="4">Table(s) and/or query(-ies)</td><td rowspan="4">Movie, Directed, Creative</td><td rowspan="4">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id AS castid, COUNT(*) AS castsize</td></tr>
<tr><td>Table(s)</td><td>Role, Creative, Movie</td></tr>
<tr><td>Search criteria</td><td></td></tr>
<tr><td>Grouping</td><td>Movie.id</td></tr>
<tr><td rowspan="5">Search criteria</td><td colspan="4">Movie.id = castid AND</td></tr>
<tr><td colspan="4">name = “Ridley Scott” AND</td></tr>
<tr><td rowspan="3">Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie, Role, Creative</td></tr>
<tr><td>Search criteria</td><td>name = “Russell Crowe”</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4">castsize DESC</td></tr>
</table>

The first inner query is:

```
SELECT Movie.id AS castid, COUNT(Role.roleid) AS castsize
FROM Role, Creative, Movie
WHERE Movie.id = Role.movieid AND Role.creativeid =
   Creative.creativeid
GROUP BY Movie.id;
```

The second inner query is:

```
SELECT Movie.id
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  name ="Russell Crowe";
```

The complete query to enter into your database server is:

```
SELECT title, yr, budget, name, castsize
FROM Movie, Directed, Creative,
(
  SELECT Movie.id AS castid, COUNT(Role.roleid) AS castsize
    FROM Role, Creative, Movie
    WHERE Movie.id = Role.movieid AND Role.creativeid =
       Creative.creativeid
    GROUP BY Movie.id
) AS Castlist
WHERE Movie.id = Directed.movieid AND
  Directed.creativeid = Creative.creativeid AND
  Movie.id = Castlist.castid AND
  name = "Ridley Scott" AND
  Movie.id = ANY (
    SELECT Movie.id
    FROM Movie, Role, Creative
    WHERE Movie.id = Role.movieid AND
      Role.creativeid = Creative.creativeid AND
      name ="Russell Crowe"
  )
ORDER BY Castlist.castsize DESC;
```

Notice that the first subquery creates a table which is given an alias of Castlist and has a key of castid (an alias of Movie.id) which is used in an equi-join in the outer query (`Movie.id = Castlist.castid`) to link the result of the subquery to the outer query.

Create your own SQL queries: Movies database Go online

This activity uses the Movies database that you downloaded at the start of this section (see Code 2).

Create SQL queries, using the layout from the previous activities, for each of the following questions.

Q9: Show the title, budget and castsize for movies with more than six people in roles.
(HINT: COUNT of roles, GROUP BY and HAVING can be used.)

. . . . . .

Q10: Show title and gross for all movies with a budget between 10,000,000 and 20,000,000 which include "Tom Hanks" in a role.

. . . . . .

Q11: Show movie titles for all movies which have roles with either "James Earl Jones", "Samuel L. Jackson" or "Lawrence Fishburne".

. . . . . .

Q12: Use a subquery to display the name, budget, gross and main start of all movies directed by "Francis Ford Coppola" or "Martin Scorsese".

. . . . . .

Q13: Use a subquery to display the title and gross of "Star Wars" movies not directed by "George Lucas".

. . . . . .

Q14: Display the movie with the highest budget and the director.

3.6 Learning points

Summary

You should now know how to:

- implement a relational database using SQL Data Definition Language (DDL) and Data Manipulation Language (DML) to match the design;
- describe, exemplify, and implement SQL operations to create databases and tables (CREATE DATABASE, CREATE TABLE);
- define the following constraints when creating tables:
  - primary key;
  - foreign key;
  - not null;
  - check;
  - auto increment.
- define the following data types when creating tables or within queries:
  - varchar;
  - integer;
  - float;
  - date;
  - time.
- describe, exemplify, and implement SQL operations to delete databases and tables (DROP DATABASE, DROP TABLE);
- use Data Manipulation Language (DML) to INSERT INTO, UPDATE and DELETE FROM tables in a database;
- describe, exemplify, and implement:
  - SQL SELECT statements that make use of the HAVING clause;
  - statements that make use of logical operators (NOT, BETWEEN, IN, ANY, EXISTS);
  - subqueries used with the WHERE clause of SELECT statements.
- read and explain code that uses the SQL at this level.

3.7 End of topic test

End of topic test: Implementation Go online

Q15: The two categories of SQL that you should be aware of at Advanced Higher level are:

a) Data Query Language (DQL) and Data Control Language (DCL).
b) Database Command Language (DCL) and Data Manipulation Language (DML).
c) Data Definition Language (DDL) and Data Manipulation Language (DML).
d) Data Control Language (DCL) and Data Definition Language (DDL).

.....

Q16: Which of the following commands are examples of Data Definition Language (DDL)?

a) INSERT INTO, UPDATE, DELETE FROM.
b) SELECT.
c) ANY, EXISTS.
d) CREATE, DROP.

.....

Q17: Which of the following commands are examples of Data Manipulation Language (DML)?

a) INSERT INTO, UPDATE, DELETE FROM.
b) SELECT.
c) ANY, EXISTS.
d) CREATE, DROP.

.....

Q18: The correct SQL to create a database called ***Scholar*** would be:

a) `CREATE Scholar DATABASE;`
b) `CREATE TABLE Scholar;`
c) `CREATE Scholar;`
d) `CREATE DATABASE Scholar;`

.....

Q19: Part of the data dictionary for a database system is as follows:

Entity: Sellers					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
sellerno	PK	INT		Yes	Auto increment
firstname		VARCHAR	15	Yes	
lastname		VARCHAR	15	Yes	
datejoined		DATE		Yes	
rating		TINYINT		Yes	>=0 and <=100
location	FK	INT		Yes	Existing location from Area table

The SQL to create this table would be:

a)

```
1  CREATE TABLE Sellers (
2    sellerno INT PRIMARY KEY AUTO_INCREMENT,
3    firstname VARCHAR(15) NOT NULL,
4    lastname VARCHAR(15) NOT NULL,
5    datejoined DATE NOT NULL,
6    rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
7    location INT NOT NULL,
8    FOREIGN KEY (location) REFERENCES Area(location)
9  );
```

b)

```
1  CREATE TABLE Sellers (
2    sellerno INT PRIMARY KEY AUTO_INCREMENT,
3    firstname VARCHAR(15) NOT NULL,
4    lastname VARCHAR(15) NOT NULL,
5    datejoined DATE NOT NULL,
6    rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
7    location INT NOT NULL FOREIGN KEY REFERENCES Area(location),
8  );
```

c)

```
1  CREATE Sellers TABLE(
2    sellerno INT AUTO_INCREMENT,
3    firstname VARCHAR(15) NOT NULL,
4    lastname VARCHAR(15) NOT NULL,
5    datejoined DATE NOT NULL,
6    rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
7    location INT NOT NULL,
8    PRIMARY KEY (location) REFERENCES Area(location),
9    FOREIGN KEY sellerno REFERENCES self
10 );
```

d)

```
CREATE TABLE Sellers (
  sellerno INT PRIMARY KEY AUTO_INCREMENT,
  firstname VARCHAR(15) NOT NULL,
  lastname VARCHAR(15) NOT NULL,
  datejoined DATE NOT NULL,
  rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
  FOREIGN KEY (location) REFERENCES Area(location) INT NOT NULL
);
```

.....

Q20: What is the effect of the following command?

```
UPDATE masterlist SET salary TO 100000;
```

a) Sets the value for the salary column, for the first record in the table, to 100000.
b) Sets the value for salary, for all rows in the table, to 100000.
c) Sets the value for salary to 100000 for the last row edited.
d) Sets the value of salary for no rows as the statement fails.

.....

Q21: Examine the following SQL query:

```
SELECT employee_id, first_name, last_name, salary
FROM Employees WHERE salary >
  (SELECT AVG(SALARY) FROM Employees);
```

What does this query produce?

a) A list of average salaries for employees.
b) A list of employees who earn less than the average for all employees.
c) A list of employees who earn more than the average for all employees.
d) A list of all employees with the average salary for all employees included in the columns.

.....

Q22: Examine the following SQL query:

```
SELECT lastname, firstname
FROM Employee
WHERE officecode IN (
  SELECT officecode
  FROM Office
  WHERE country = "SCO"
);
```

What does this query produce?

a) A list of employee lastnames and firstnames where the employee officecode starts with "SCO".
b) A list of employee lastnames and firstnames where the employee has an office code which matches one of the offices with a country of "SCO".
c) A list of office codes for employees with a country of "SCO".
d) A list of all employees with country and officecode.

.....

Q23: Examine the following SQL query:

```
1  SELECT lastname, firstname
2  FROM Employee
3  WHERE officecode IN (
4    SELECT officecode
5    FROM Office
6    WHERE country = "SCO"
7  );
```

Identify the inner query.

a) `SELECT lastname, firstname FROM Employee`
b) `WHERE officecode IN ()`
c) `FROM Employee WHERE officecode IN ()`
d) `SELECT officecode FROM Office WHERE country = "SCO"`

.....

Q24: Examine the following SQL query:

```
1  SELECT customernumber, customername
2  FROM Customers
3  WHERE
4    EXISTS (
5      SELECT ordernumber, SUM(priceeach * quantityordered)
6      FROM OrderDetails, Orders
7      WHERE OrderDetails.customernumber = Customers.customerNumber
           AND
8        Orders.ordernumber = OrderDetails.ordernumber
9      GROUP BY ordernumber
10     HAVING SUM(priceeach * quantityOrdered) > 60000
11 );
```

What does this query produce?

a) A list of customer numbers and names where the customer has at least one order with a value of more than 60000.
b) A list of customer numbers and names where the sum of all orders by the customer is more than 60000.
c) A list of all items with order values of more than 60000.
d) A list of customer numbers and names where each customer has spent over 60000 in total across all orders.

Topic 4

Testing and evaluation

Contents

Prerequisites

From you studies at Higher, you should already know:

- how to describe and exemplify testing of SQL operations, involving solutions using three or more linked tables, to ensure they work correctly;
- how to evaluate your solution in terms of fitness for purpose and accuracy of output.

Learning objective

By the end of this topic, you should be able to:

- describe and exemplify how SQL implemented tables match their design;
- describing and exemplifying testing of SQL operations, at this level, to ensure they work correctly;
- evaluate solutions in terms of accuracy of output.

4.1 Revision

Quiz: Revision Go online

Q1: ***Fitness for purpose*** means that:

a) any user can use the solution presented.
b) the solution is still in a testing phase.
c) the solution is good enough to meet the required need.
d) the solution precisely and accurately meets all the requirements.

. . . . . .

Q2: A database consists of two tables: Country and City. Sample data from both tables is shown. The next three questions refer to these tables.

Country

id	name	countrycode	district	info
670	A Coruña (La Coruña)	ESP	Galicia	243402
3097	Aachen	DEU	Nordrhein-Westfalen	243825
3318	Aalborg	DNK	Nordjylland	161161
2760	Aba	NGA	Imo & Abia	298900
1404	Abadan	IRN	Khuzestan	206073
395	Abaetetuba	BRA	Pará	111258
3683	Abakan	RUS	Hakassia	169200
1849	Abbotsford	CAN	British Colombia	105403

City

code	name	capital	code2
ABW	Aruba	129	AW
AFG	Afghanistan	1	AF
AGO	Angola	56	AO
AIA	Anguilla	62	AI
ALB	Albania	34	AL
AND	Andorra	55	AD
ANT	Netherlands Antilles	33	AN
ARE	United Arab Emirates	65	AE

A query has been used to create the following answer table.

number of cities	name
59	Spain

Identify the correct query to produce this answer table.

a)

```
SELECT COUNT(Country), Country.name
FROM City, Country
WHERE City.name = Country.name AND Country.name = 'Spain'
GROUP BY Country.name;
```

b)

```
SELECT COUNT(*) AS 'number of cities', Country.name
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name = 'Spain'
GROUP BY Country.name;
```

c)

```
SELECT COUNT(*) AS 'number of cities', Country.name
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name =
    'Spain';
```

d)

```
SELECT COUNT(*) AS 'number of cities', Country.name
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name = 'Spain'
GROUP BY City.name;
```

.....

Q3: A query has been used to create the following answer table.

id	name	name
2974	Paris	France
3068	Berlin	Germany

Identify the correct query to produce this answer table.

a)

```
SELECT City.id, City.name, Country.name
FROM City, Country
WHERE Country.code = City.countrycode AND (Country.name =
    'Germany' AND Country.name = 'France');
```

b)

```
SELECT City.id, City.name, Country.name
FROM City, Country
WHERE Country.code = City.countrycode AND (Country.name =
    'Germany' OR Country.name = 'France');
```

c)

```
SELECT *
FROM City, Country
WHERE Country.capital = City.id AND (Country.name = 'Germany'
   OR Country.name = 'France');
```

d)

```
SELECT City.id, City.name, Country.name
FROM City, Country
WHERE Country.capital = City.id AND (Country.name = 'Germany'
   OR Country.name = 'France');
```

.....

Q4: A query has been used to create the following answer table:

name	name	info
Afghanistan	Kabul	1780000
Afghanistan	Quandahar	237500
Afghanistan	Herat	186800
Afghanistan	Mazar-e-Sharif	127800

Identify the correct query to produce this answer table.

a)

```
SELECT Country.name, City.name, City.info
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name =
   'Afghanistan'
ORDER BY City.info DESC;
```

b)

```
SELECT Country.name, City.info, City.name
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name =
   'Afghanistan'
ORDER BY City.name ASC;
```

c)

```
SELECT *
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name =
   'Afghanistan'
ORDER BY City.info ASC;
```

d)

```
SELECT Country.name, City.name, City.info
FROM City, Country
WHERE City.id = Country.capital AND Country.name = 'Afghanistan'
ORDER BY City.info DESC;
```

.....

Q5: Accuracy of output means that:

a) all numerical data is shown to at least two decimal places.
b) all data required is present in the given output.
c) the output generated exactly matches the requirements.
d) query results are close enough so that required data can be visually identified.

4.2 Testing

Testing solutions at this level requires that you can check that the databases you have created match their designs and that the structured query language statements you write to create tables, insert/delete/update and select data from these databases are correct.

4.2.1 SQL implemented tables match design

The design that you implement for your database will be based on the **data dictionary** and the **entity-relationship diagram**. You need to be able to take design and check that this matches the database created.

The databases for the activities in this section can be downloaded as zip files here:

Code 3: Database downloads for activities

- Employees database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/215E6E3E-DA66-46A7-9B40-C12BFA726B15/employees.zip
- Reseller database: /vle/asset/Downloads/AHCCMP/Course%20Downloads/37D42341-965C-FD38-4454-84B9E6725F29/reseller.zip

SQL implemented tables match design: Employees database Go online

This activity uses the Employees database that you downloaded at the start of this section (see Code 3).

The data dictionary for the database is:

Entity: Department

Attribute name	Key	Type	Size	Required	Validation
dept_no	PK	VARCHAR	4	yes	
dept_name		VARCHAR	40	yes	

Entity: Dept_emp

Attribute name	Key	Type	Size	Required	Validation
emp_dept_no	PK	INT		yes	
emp_no	FK	INT		yes	emp_no from Employee
dept_no	FK	VARCHAR	4	yes	dept_no from Department
from_date		DATE		yes	
to_date		DATE		no	

Entity: Dept_manager

Attribute name	Key	Type	Size	Required	Validation
manager_id	PK	INT		yes	
emp_no	FK	INT		yes	emp_no from Employee
dept_no	FK	VARCHAR	4	yes	dept_no from Department
from_date		DATE		yes	
to_date		DATE		no	

Entity: Employee

Attribute name	Key	Type	Size	Required	Validation
emp_no	PK	INT		yes	
birth_date		DATE		yes	
first_name		VARCHAR	14	yes	
last_name		VARCHAR	16	yes	
gender		VARCHAR	1	no	Either M or F
hire_date		DATE			

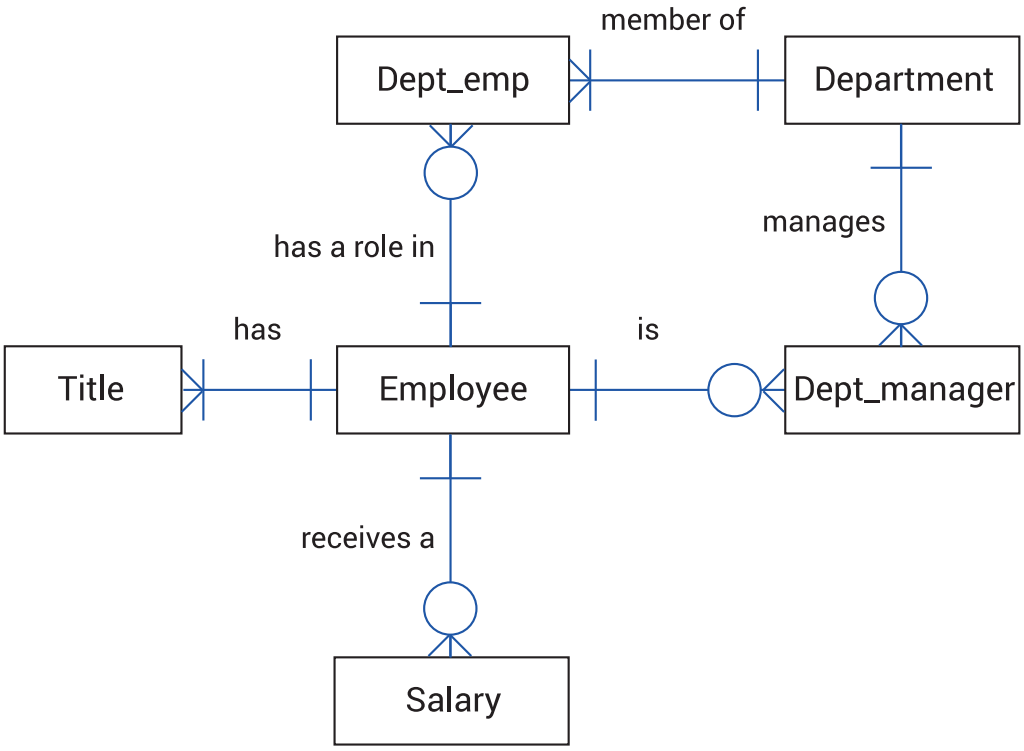
Entity: Salary

Attribute name	Key	Type	Size	Required	Validation
salary_id	PK	INT		yes	
emp_no	FK	INT		yes	emp_no from Employee
salary		INT		yes	
from_date		DATE		yes	
to_date		DATE		no	

Entity: Title

Attribute name	Key	Type	Size	Required	Validation
title_id	PK	INT		yes	
emp_no	FK	INT		yes	emp_no from Employee
title		VARCHAR	50	yes	
from_date		DATE		yes	
to_date		DATE		no	

The ERD for this database is:



You can check the structure of the database by reviewing the downloaded SQL file (Code 3) in a text editor.

For this purpose of this activity, there are a number of errors in the SQL file which you can identify and record as part of your testing that the implementation matches the design.

The first example of this is in the department table. The data dictionary for this table is:

Entity: Department

Attribute name	Key	Type	Size	Required	Validation
dept_no	PK	VARCHAR	4	yes	
dept_name		VARCHAR	40	yes	

And the SQL from the file to create it is:

```
1 CREATE TABLE Department (
2   dept_no VARCHAR(4) NOT NULL,
3   dept_name VARCHAR(40) NULL
4 );
```

This SQL statement contains two errors.

- The first is that the primary key is not set for dept_no.

- The second is that dept_name is required so should not accept a NULL value.

The corrected SQL for this table is:

```
1 CREATE TABLE Department (
2   dept_no VARCHAR(4) NOT NULL PRIMARY KEY,
3   dept_name VARCHAR(40) NOT NULL
4 );
```

Q6: Now identify the eight remaining errors in the SQL file.

SQL implemented tables match design: Reseller database Go online

This activity uses the Reseller database that you downloaded at the start of this section (see Code 3).

The data dictionary for the database is:

Entity: Agent

Attribute name	Key	Type	Size	Required	Validation
agent_code	PK	VARCHAR	6	yes	
agent_name		VARCHAR	40	yes	
working_area		VARCHAR	35	yes	
commission		FLOAT	10,2	yes	
phone_no		VARCHAR	15	no	
country		VARCHAR	25	no	

Entity: Customer

Attribute name	Key	Type	Size	Required	Validation
cust_code	PK	VARCHAR	6	yes	
cust_name		VARCHAR	40	yes	
cust_city		VARCHAR	35	yes	
working_area		FLOAT	10,2	yes	
cust_country		VARCHAR	20	no	
grade		INT		no	
opening_amt		FLOAT	12,2	yes	
receive_amt		FLOAT	12,2	yes	
payment_amt		FLOAT	12,2	yes	
outstanding_amt		FLOAT	12,2	yes	
phone_no		VARCHAR	17	yes	
agent_code	FK	VARCHAR	6	yes	existing agent_code from Agent

Entity: Order					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
ord_num	PK	INT		yes	
ord_amount		FLOAT	12,2	yes	
advance_amount		FLOAT	12,2	yes	
ord_date		DATE		yes	
cust_code	FK	VARCHAR	6	yes	existing cust_code from Customer
agent_code	FK	VARCHAR	6	yes	existing agent_code from Agent
ord_description		VARCHAR	60	yes	

You can check the structure of the database by reviewing the downloaded SQL file (Code 3) in a text editor.

Q7: Identify the eight errors in this implementation.

Remember, as well as the code to CREATE a database and a table, you may be asked to check code to remove tables and databases using the DROP DATABASE and DROP TABLE commands.

4.2.2 SQL operations work correctly at this level

As this level, you should be able to use INSERT, DELETE and UPDATE to add, remove and amend data in tables. These operations should be tested to ensure that they produce the expected results.

SELECT queries are used to gather data from the database. At this level you are expected to make use of and test queries which contain:

- HAVING clause of the SELECT statement;
- subqueries used with the WHERE clause of SELECT statements.

SQL operations work correctly at this level: Quiz Go online

A database consists of six tables: Customer, Country, Order, Product, Orderitem, Supplier:

Customer (id, firstname, lastname, city, countryid*, phone)
Country (id, country)
Order (ordernumber, orderdate, customerid*, totalvalue)
Product (pid, productname, supplierid*, unitprice)
Orderitem (id, ordernumber*, pid*, unitprice, quantity)
Supplier (sid, suppliername)

Key point

Note: underlined is a primary key, * is a foreign key.

For each of the following queries, state if it is correct or not.

Where the query is incorrect, rewrite the query to be correct.

Q8: A query is required to show a list of countries which have at least 100 customers.

```
1 SELECT COUNT(Customer.id), country
2 FROM Customer, Country
3 WHERE Customer.countryid = Country.id
4 GROUP BY country
5 HAVING COUNT(Customer.id) >=100;
```

a) Correct
b) Incorrect

.....

Q9: A query is required to list the number of customers in each country except "Scotland", sorted from high values to low. Only include countries with 20 or more customers.

```
1 SELECT COUNT(Customer.id), country
2 FROM Customer, Country
3 WHERE Customer.countryid = Country.id OR country <> "Scotland"
4 GROUP BY country
5 HAVING COUNT(Customer.id) > 20;
```

a) Correct
b) Incorrect

.....

Q10: Show all customers with average orders of between £2000 and £3000.

```
SELECT AVG(totalvalue), firstname, lastname
FROM Order, Customer
WHERE Order.customerid = Customer.id
GROUP BY firstname, lastname
HAVING totalvalue BETWEEN 2000 AND 3000;
```

a) Correct
b) Incorrect

.....

Q11: A query is required to show all the customers who have placed an order.

```
SELECT Customer.id, firstname, lastname
FROM Customer
WHERE EXISTS (SELECT *
  FROM Order
  WHERE Order.customerid = Customer.id);
```

a) Correct
b) Incorrect

.....

Q12: A query is required to list products where order quantities are more than 100.

```
SELECT productname
FROM Orderitem
WHERE Product.pid = ANY (
  SELECT Orderitem.pid
  FROM Orderitem
      WHERE quantity > 100
);
```

a) Correct
b) Incorrect

4.3 Evaluation

The evaluation of a solution is covered in the **Testing and evaluation** topic in **Software design and development**. There you evaluate a solution in terms of:

- fitness for purpose;
- maintainability:
  - perfective;
  - corrective;
  - adaptive.
- robustness.

In addition, for database design and development you are to include accuracy of output in your evaluation.

4.3.1 Accuracy of output

When evaluating the **accuracy of output**, you are considering if the output generated by your solution, matches the specification. Key questions would be:

- are all the criteria required for the query applied?
- are answer tables grouped using the intended columns?
- are columns sorted correctly in ascending or descending order as required?
- have equi-joins been used to link tables correctly?
- where aliases are used, are they spelt correctly and used consistently?
- are aggregated columns given aliases where required?
- are wildcards used when a limited number of columns as specified? The results may be fit for purpose but the output would not be accurate.

The accuracy of the output from your solution must be evaluated against the specification of requirements. Compare what you actually produced with what you were asked to produce.

4.4 Summary

Summary

You should now be able to:

- describe and exemplify how SQL implemented tables match their design;
- describe and exemplify testing of SQL operations, at this level, to ensure they work correctly;
- evaluate solutions in terms of accuracy of output.

4.5 End of topic test

End of topic test: Testing and evaluation Go online

The questions in this test refer to the following database.

Entity: Pet

Attribute name	Key	Type	Size	Required	Validation
petid	PK	INT		yes	
name		VARCHAR	20	yes	
owner		VARCHAR	20	yes	
species		VARCHAR	20	yes	
sex		VARCHAR	1	no	
birth		DATE		yes	
endofrecord		DATE		no	

Entity: Petevent

Attribute name	Key	Type	Size	Required	Validation
eventid	PK	INT		yes	
pedid	FK	INT		yes	exisiting petid from Pet
eventdate		DATE		yes	
eventtype		VARCHAR	15	yes	
remark		VARCHAR	255	no	

The following is sample data.

Pet

petid	name	owner	species	sex	birth	endofrecord
1	Fluffy	Harold	cat	f	2013-02-04	*NULL*
2	Claws	Gwen	cat	m	2014-03-17	*NULL*
3	Buffy	Harold	dog	f	2019-05-13	*NULL*
4	Fang	Benny	dog	m	2010-08-27	*NULL*
5	Bowser	Diane	dog	m	2019-08-31	2022-07-29
6	Chirpy	Gwen	bird	f	2018-09-11	*NULL*
7	Whistler	Gwen	bird	*NULL*	2017-12-09	*NULL*
8	Slim	Benny	snake	m	2016-04-29	*NULL*

Petevent

eventid	petid	eventdate	eventtype	remark
1	1	2015-05-15	litter	4 kittens, 3 female, 1 male
2	3	2020-06-23	litter	5 puppies, 2 female, 3 male
3	3	2021-06-19	litter	3 puppies, 3 female
4	6	2019-03-21	vet	needed beak straightened
5	8	2017-08-03	vet	broken rib
6	5	2020-10-12	kennel	*NULL*
7	4	2020-10-12	kennel	*NULL*
8	4	2018-08-28	birthday	Gave him a new chew toy
9	2	2018-03-17	birthday	Gave him a new flea collar
10	7	2018-12-09	birthday	First birthday

Q13: To create the Petevent table, the following SQL was used.
Four areas of the SQL have been highlighted as potential errors.

```
CREATE TABLE Petevent (
    eventid INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    petid INT(11) NULL,                                  -- Error 1
    eventdate TIME DEFAULT NULL,                         -- Error 2
    type VARCHAR(15) DEFAULT NULL,
    remark VARCHAR(255) DEFAULT NULL,                    -- Error 3
    FOREIGN KEY(petid) REFERENCES Pet(name)              -- Error 4
);
```

Which are actual errors?

a) Error 1, Error 2, Error 4
b) Error 2, Error 3, Error 4
c) Error 1, Error 3, Error 4
d) Error 2, Error 2, Error 3

. . . . . .

Q14: A query is required to display all the pets with two or more Petevent entries. Which query will produce the required answer table?

a)

```
1 SELECT Pet.petid, name
2 FROM Pet, Petevent
3 WHERE Pet.petid = Petevent.petid
4 GROUP BY Pet.petid
5 HAVING COUNT(Pet.name) > 2;
```

b)

```
1 SELECT COUNT(Pet.petid), name
2 FROM Pet
3 WHERE Pet.petid = Petevent.petid
4 GROUP BY Pet.name
5 HAVING COUNT(Pet.petid) >= 2;
```

c)

```
1 SELECT COUNT(*), *
2 FROM Pet, Petevent
3 WHERE Pet.petid = Petevent.petid
4 GROUP BY Pet.petid
5 HAVING COUNT(Pet.petid) >= 2;
```

d)

```
1 SELECT COUNT(Pet.petid), name
2 FROM Pet, Petevent
3 WHERE Pet.petid = Petevent.petid
4 GROUP BY Pet.petid
5 HAVING COUNT(Pet.petid) >= 2;
```

.....

Q15: A query is required to show events for pets who are dogs.
Which query will produce the required answer table?

a)

```
1 SELECT *
2 FROM Petevent
3 WHERE petid IN (
4   SELECT petid
5   FROM Pet
6   WHERE species = 'dog'
7 );
```

b)

```
1 SELECT *
2 FROM Petevent
3 WHERE petid = ANY (
4   SELECT *
5   FROM Pet
6   WHERE species = 'dog'
7 );
```

c)

```
1 SELECT *
2 FROM Petevent, Pet
3 WHERE Pet.petid = Petevent.petid AND petid IN (
4   SELECT petid
5   FROM Pet
6 );
```

d)

```
1 SELECT *
2 FROM Petevent
3 WHERE petid NOT EXISTS  (
4   SELECT petid
5   FROM Pet
6   WHERE species = 'dog'
7 );
```

.....

Q16: A query is intended to produce a list of petid, name, birth and owner for dogs with a kennel event type. The query that was designed produces the following answer table.

petid	name	owner	species	sex	birth
5	Bowser	Diane	dog	m	2019-08-31
4	Fang	Benny	dog	m	2010-08-27
endofrecord	**eventid**	**petid**	**eventdate**	**eventtype**	**remark**
2022-07-29	6	5	2020-10-12	kennel	*NULL*
NULL	7	4	2020-10-12	kennel	*NULL*

This answer table is not accurate because:

a) it contains details of other species other than dogs.
b) it contains details of events which are not "kennel".
c) it contains additional rows for the dogs selected which do not meet the requirements.
d) it contains additional columns in excess of those requested for the query.

. . . . . .

Q17: A new bird called "Polly" is to be added to the pets. The details for Polly are:

name	Polly
owner	Morag
species	bird
sex	*NULL*
birth	2020-08-18

The correct query to INSERT a record for Polly is:

a)

```
1 INSERT INTO Pet(petid, name, owner, species, birth)
2 VALUES (NULL, 'Polly', 'Morag', 'Bird', '2020-08-18');
```

b)

```
1 INSERT INTO Pet(petid, name, owner, species, sex, birth,
     endofrecord)
2 VALUES ('', 'Polly', 'Morag', 'Bird', NULL, '2020-08-18', NULL);
```

c)

```
1 INSERT INTO Pet(name, owner, species, birth)
2 VALUES ('Polly', 'Morag', 'Bird', '2020-08-18');
```

d)

```
1 INSERT INTO Pet(name, owner, species, birth, sex)
2 VALUES ('Polly, 'Morag', 'Bird', date(), 'm');
```

Topic 5

Database design and development test

Database design and development test Go online

Q1: Which of the following is the correct PHP syntax to connect to a server and select a database?

a) `$conn = connect ($server, $username, $password) for $database;`
b) `$conn = mysqli_connect($server, $username, $password,$database);`
c) `$conn = ($server, $username, $password,$database);`
d) `$conn = new connection($server, $username, $password,$database);`

.....

Q2: Which of the following PHP code is required to execute a query stored in a variable and store the result?

a) `$result = $mysql_query($conn, $query);`
b) `$result->fetch.object($conn, $query);`
c) `$result = $mysqli_prepare($conn, $query);`
d) `$result->mysqli_error($query);`

.....

Q3: Identify the correct explanation for this section of code.

```
1 $querystring = "SELECT * FROM game";
2 if ($result = mysqli_query($conn, $querystring)) {
3     printf("Select returned %d rows.\n", mysqli_num_rows($result));
4     mysqli_close($conn);
5 }
```

a) Set $querystring to hold an SQL statement. Execute the query using the server and database previously connected to and stored in $conn. In the if statement the result of the query are returned in $result. If $result is true, exit the if statement without closing the database connection. If $result is false, display how many rows have been retrieved and then close the database connection.
b) Set $querystring to hold an SQL statement. Execute this query using the server and database held in the variable $conn. The results of the query will be stored in the variable $result. If the query fails then the if statement is exited and the code is completed, if the query is successful, a message will be shown, stating how many rows of data have been retrieved and the connection is closed.
c) Set $querystring to hold an SQL statement. Execute this query using the server and database held in the variable $conn. The results of the query are compared with the value in $result. If the values match then the contents of the if statement are executed, displaying how many rows of data have been retrieved and the connection is closed. If the values do not match then the if statement is exited.
d) The results of the query are stored in $querystring and compared with $result within the IF statement. If the values do not match then the contents of the if statement are executed, displaying how many rows of data have been retrieved and the connection is closed. If the values do match then the if statement is exited.

. . . . . .

Q4: The PHP command mysqli_num_rows:

a) fetches a result row as a numeric array and as an associative array.
b) returns the number of rows in a result set.
c) returns the number of columns for the most recent query.
d) performs query against a database.

. . . . . .

Q5: A database consists of six tables: Customer, Country, Order, Product, Orderitem, Supplier

Customer (id, firstname, lastname, city, countryid*, phone)
Country(id, country)
Order (ordernumber, orderdate, customerid*, totalvalue)
Product (pid, productname, supplierid*, unitprice)
Orderitem (ordernumber*, pid*, unitprice, quantity)
Supplier (*sid*, suppliername)

Identify the weak entity in this database.

a) Customer
b) Supplier
c) Orderitem
d) Product

. . . . . .

Q6: A weak entity can be removed from a database through the addition of a:

a) primary key.
b) compound key.
c) foreign key.
d) surrogate key.

. . . . . .

Q7: Which is the correct definition of a weak entity?

a) A weak entity has a minimum of two relationships and a foreign key.
b) A weak entity is an entity that cannot be uniquely identified by its attributes alone and therefore requires at least one foreign key from another entity to create a primary key.
c) A weak entity provides a foreign key to support the forming of a key in a strong entity.
d) A weak entity links foreign and primary keys so that relationship can be established between two or more entities.

. . . . . .

Q8: An extract from a data dictionary for a database is shown below:

Entity: Movie					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
id	PK	INT		Yes	Autoincrement
title		VARCHAR	50	Yes	
rating		FLOAT	3,1	Yes	>=0 and <=10
yr		DATE		Yes	
budget		INT		No	
gross		BIGINT		No	

Identify the SQL statement to create this table.

a)

```
CREATE TABLE Movie (
  id INT,
  title VARCHAR(50) NOT NULL,
  rating FLOAT(3,1) NOT NULL,
  yr INT NOT NULL,
  budget INT,
  gross BIGINT,
  CONSTRAINT id AUTO_INCREMENT PRIMARY KEY,
  CONSTRAINT CHECK(rating >=0 AND rating <=10)
);
```

b)

```
CREATE TABLE Movie (
  id INT,
  title VARCHAR(50),
  rating FLOAT(3,1),
  yr INT,
  budget INT,
  gross BIGINT,
  CONSTRAINT id AUTO_INCREMENT PRIMARY KEY,
  CONSTRAINT CHECK(rating >=0 AND rating <=10),
  CONSTRAINT title, rating, yr NOT NULL
);
```

c)

```
CREATE TABLE Movie (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL,
  rating FLOAT(3,1) NOT NULL CHECK(rating >=0 AND rating <=10),
  yr INT NOT NULL,
  budget INT,
  gross BIGINT
);
```

d)

```
CREATE TABLE Movie (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(20) NULL,
  rating FLOAT(2,1) NULL CHECK(rating >=0 AND rating <=10),
  yr INT NOT NULL,
  budget INT,
  gross BIGINT
);
```

.....

Q9: An extract from a data dictionary is shown below.

Entity: Roster					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
rosterid	PK	INT		Yes	AUTO_ INCREMENT
teamid	FK	INT		Yes	
playerid	FK	INT			
position		TINYINT		Yes	
joindate		DATE		Yes	

Identify the SQL statement to insert a record with the following values.

teamid	23
playerid	55
position	Divinity Support
joindate	24 September 2019

a)

```
INSERT INTO Roster (rosterid, teamid, playerid, position,
   joindate)
VALUES (NULL, 23, 55,"Divinity Support","2019-09-24");
```

b)

```
1 INSERT INTO Roster COLUMNS (teamid, playerid, position,
     joindate)
2 VALUES (23, 55,"Divinity Support","2019-09-24") AUTO_INCREMENT
     rosterid;
```

c)

```
1 INSERT INTO Roster
2 VALUES (NULL, 23, 55,"Divinity Support","2019-09-24");
```

d)

```
1 INSERT INTO Roster (teamid, playerid, position, joindate)
2 VALUES (23, 55,"Divinity Support","2019-09-24");
```

.....

Q10: A database called *Encrypted_info* is to be deleted from the database server. Identify the command to complete this operation.

a) `DROP TABLE AND DATABASE Encrypted_info;`
b) `DROP TABLE *; DROP Encrypted_info();`
c) `DROP DATABASE Encrypted_info;`
d) `DROP FILE Encrypted info;`

.....

Q11: A system has the following entities and attributes:

Accountant (acc_id, acc_fname, acclname, date_hired, office_no)
Customer (customer_id, cfname, clname, street, city, postcode, contact_no)
Project (acc_id*, customer_id*, tax year, iscomplete)

The correct ERD for this system is:

a)

b)

c)

d)

.....

Q12: A data dictionary is used to:

a) illustrate the relationships between entities in a relational database system.
b) record details of information flow between the entities in a relational database system.
c) illustrate the sequencing of events that can be carried out within a relational database system.
d) record details of all data items within a relational database system.

. . . . . .

Q13: A database contains the following entities and attributes:

Movie (id, title, yr, budget, gross)
Role (roleid, movieid*, creativeid*, ord)
Creative (creativeid, name)
Directed (directorid, movieid*, creativeid*)

A query is required to show the name and number of movies directed by any director who directed more than five movies between 2000 and 2010.

Identify the correct query design to answer this query.

a)

Field(s) and/or calculation(s)	SUM(*), Creative.name
Tables(s) and/or query(-ies)	Movie, Directed, Creative
Search Criteria	yr FROM 2000 TO 2010
Grouping	Creative.name
Having	COUNT(*) $>$ 5
Sort order	

b)

Field(s) and/or calculation(s)	COUNT(Creative.creativeid), Creative.name
Tables(s) and/or query(-ies)	Movie, Directed, Creative
Search Criteria	yr BETWEEN 2000 AND 2010
Grouping	Creative.name
Having	COUNT(Creative.creativeid) $>$ 5
Sort order	

c)

Field(s) and/or calculation(s)	COUNT(*), Creative.name
Tables(s) and/or query(-ies)	Movie, Directed, Creative
Search Criteria	yr BETWEEN 2000 AND 2010
Grouping	Creative.name
Having	COUNT(*) $>$ 5
Sort order	

d)

Field(s) and/or calculation(s)	Creative.name
Tables(s) and/or query(-ies)	Movie, Directed, Creative
Search Criteria	yr BETWEEN 2000 AND 2010
Grouping	COUNT(*)
Having	COUNT(*) $>$ 5
Sort order	

. . . . . .

Q14: This question refers to the following database tables

Entity: Dept_manager

Attribute name	Key	Type	Size	Required	Validation
manager_id	PK	INT		Yes	
emp_no	FK	INT		Yes	emp_no from Employee
dept_no	FK	VARCHAR	4	Yes	dept_no from Department
from_date		DATE		Yes	
to_date		DATE		No	

Entity: Employee

Attribute name	Key	Type	Size	Required	Validation
emp_no	PK	INT		Yes	
birth_date		DATE		Yes	
first_name		VARCHAR	14	Yes	
last_name		VARCHAR	16	Yes	
gender		VARCHAR	1	No	Either M or F
hire_date		DATE			

A query is required to find the birth_date, first_name and last_name for all employees in department management roles.

Identify the correct SQL query to display this data.

a)

```
SELECT *
FROM Dept_manager
WHERE EXISTS (
  SELECT birth_date, first_name, last_name FROM Employee
  WHERE Employee.emp_no = Dept_manager.emp_no
);
```

b)

```
SELECT birth_date, first_name, last_name
FROM Employee
WHERE EXISTS (
  SELECT * FROM Dept_manager
  WHERE Employee.emp_no = Dept_manager.emp_no
);
```

c)

```
SELECT birth_date, first_name, last_name
FROM Dept_manager
WHERE EXISTS (
  SELECT * FROM Employee
  WHERE Employee.emp_no = Dept_manager.emp_no
);
```

d)

```
SELECT birth_date, first_name, last_name
FROM Employee
WHERE EXISTS (
  SELECT * FROM Dept_manager
  WHERE Employee.emp_no = Dept_manager.emp_no
);
```

.....

Q15: The next two questions use the following database tables:

Entity: Product

Attribute name	Key	Type	Size	Required	Validation
productid	PK	INT		Yes	
productname		VARCHAR	25	Yes	
supplierid	FK	INT		Yes	Existing supplierid from Supplier
categoryid	FK	INT		Yes	Existing categoryid from Category
unit		VARCHAR	20	Yes	
price		FLOAT	6,2	Yes	

Entity: OrderDetail

Attribute name	Key	Type	Size	Required	Validation
orderdetailid	PK	INT		Yes	
orderid	FK	INT		Yes	Existing orderid from Order
productid	FK	INT		Yes	Existing productid from Product
quantity		INT		Yes	Must be >= 1

Identify the SQL query which will display a list of productname where the product has sold in an order quantity of 10 in at least one order.

a)

```
SELECT *
FROM Product
WHERE productid = ANY (
  SELECT *
  FROM OrderDetail
  WHERE quantity >= 10
);
```

b)

```
1 SELECT productname
2 FROM Product
3 WHERE productid = ANY (
4   SELECT productid
5   FROM OrderDetail
6   WHERE quantity >= 10
7 );
```

c)

```
1 SELECT productname
2 FROM Product
3 WHERE productname = ANY (
4   SELECT productname
5   FROM OrderDetail
6   WHERE quantity >= 10
7 );
```

d)

```
1 SELECT productname
2 FROM Product, OrderDetail
3 WHERE productid = ANY (
4   SELECT *
5   FROM OrderDetail
6   WHERE quantity >= 10
7 ) AND Product.productid = OrderDetail.productid;
```

.....

Q16: This question uses the Product and OrderDetail database tables from the previous question.

A design for a query is required which will produce a list of productids and productnames where the total orders are more than 1000.

a)

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">productid, productname</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Product</td></tr>
<tr><td rowspan="4">Search Criteria</td><td rowspan="4">EXISTS</td><td rowspan="4">Inner query</td><td>Field(s) and/or calculation(s)</td><td>productid</td></tr>
<tr><td>Table(s)</td><td>OrderDetail</td></tr>
<tr><td>Search Criteria</td><td>SUM(quantity) > 1000</td></tr>
<tr><td>Grouping</td><td>productid</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

b)

Field(s) and/or calculation(s)	productid, productname			
Table(s) and/or query(-ies)	Product, OrderDetail			
Search Criteria	EXISTS	Inner query	Field(s) and/or calculation(s)	*
			Table(s)	OrderDetail
			Search Criteria	COUNT(quantity) > 1000
			Grouping	productid
Grouping				
Having				
Sort order				

c)

Field(s) and/or calculation(s)	productid, productname			
Table(s) and/or query(-ies)	Product			
Search Criteria	NOT EXISTS	Inner query	Field(s) and/or calculation(s)	*
			Table(s)	OrderDetail
			Search Criteria	SUM(quantity) > 1000
			Grouping	productname
Grouping				
Having				
Sort order				

d)

Field(s) and/or calculation(s)	productid, productname			
Table(s) and/or query(-ies)	Product, OrderDetail			
Search Criteria	EXISTS	Inner query	Field(s) and/or calculation(s)	productid, productname
			Table(s)	Product, OrderDetail
			Search Criteria	COUNT(quantity) > 1000
			Grouping	productid
Grouping				
Having				
Sort order				

.....

Q17: Which of the following are examples of Data Manipulation Language (DML)?

a) INSERT INTO, UPDATE, DELETE FROM.
b) SELECT.
c) ANY, EXISTS.
d) CREATE, DROP.

. . . . . .

Q18: Part of data dictionary for a database system is shown below:

Entity: Sellers

Attribute name	Key	Type	Size	Required	Validation
sellerno	PK	INT		Yes	Autoincrement
firstname		VARCHAR	15	Yes	
lastname		VARCHAR	15	Yes	
datejoined		DATE		Yes	
rating		TINYINT		Yes	>=0 and <=100
location	FK	INT		Yes	Existing location from Area table

The SQL to create this table would be:

a)

```
1 CREATE TABLE Sellers (
2   sellerno INT PRIMARY KEY AUTO_INCREMENT,
3   firstname VARCHAR(15) NOT NULL,
4   lastname VARCHAR(15) NOT NULL,
5   datejoined DATE NOT NULL,
6   rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
7   location INT NOT NULL,
8   FOREIGN KEY (location) REFERENCES Area(location)
9 );
```

b)

```
1 CREATE TABLE Sellers (
2   sellerno INT PRIMARY KEY AUTO_INCREMENT,
3   firstname VARCHAR(15) NOT NULL,
4   lastname VARCHAR(15) NOT NULL,
5   datejoined DATE NOT NULL,
6   rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
7   location INT NOT NULL,
8   FOREIGN KEY (location) REFERENCES Area(location)
9 );
```

c)

```
CREATE Sellers TABLE(
  sellerno INT AUTO_INCREMENT,
  firstname VARCHAR(15) NOT NULL,
  lastname VARCHAR(15) NOT NULL,
  datejoined DATE NOT NULL,
  rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
  location INT NOT NULL,
  PRIMARY KEY (location) REFERENCES Area(location),
  FOREIGN KEY sellerno REFERENCES self
);
```

d)

```
CREATE TABLE Sellers (
  sellerno INT PRIMARY KEY AUTO_INCREMENT,
  firstname VARCHAR(15) NOT NULL,
  lastname VARCHAR(15) NOT NULL,
  datejoined DATE NOT NULL,
  rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
  FOREIGN KEY (location) REFERENCES Area(location) INT NOT NULL
);
```

.....

Q19: What is an inner query?

a) A query with a WHERE condition which connects two attributes of the same entity.
b) A query which only has one WHERE condition.
c) A query which is nested inside another query and generates an answer table which the outer query uses.
d) A query which generates an answer table referencing all the entities within the FROM clause.

.....

Q20: This question refers to the following database:

Entity: Movie

Attribute name	Key	Type	Size	Required	Validation
id	PK	INT		Yes	Autoincrement
title		VARCHAR	50	Yes	
rating		FLOAT	3,1	Yes	>=0 and <=10
yr		DATE		Yes	
budget		INT		No	
gross		BIGINT		No	

Entity: Role

Attribute name	Key	Type	Size	Required	Validation
roleid	PK	INT		Yes	
movieid	FK	INT	50	Yes	Valid id from Movie
creativeid	FK	INT	3,1	Yes	Valid creativeid from Creative
ord		INT		Yes	

Entity: Creative

Attribute name	Key	Type	Size	Required	Validation
creativeid	PK	INT		Yes	
name		VARCHAR	50	Yes	

A SQL query is required to show title and gross for all movies with a budget between 10,000,000 and 20,000,000 which include "Tom Hanks" in a role.

a)

```
1 SELECT *
2 FROM Movie, Role, Creative
3 WHERE Movie.id = Role.movieid AND
4   Creative.creativeid = Role.creativeid AND
5   budget IN(10000000, 20000000)
6   AND name = "Tom Hanks";
```

b)

```
SELECT title, gross
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Creative.creativeid = Role.creativeid AND
  budget > 10000000 AND 20000000
  AND name = "Tom Hanks";
```

c)

```
SELECT title, gross
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Creative.creativeid = Role.creativeid AND
  EXISTS (
    SELECT budget
    FROM Movie
    WHERE budget BETWEEN 10000000 AND 20000000 )
  AND name = "Tom Hanks";
```

d)

```
SELECT title, gross
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Creative.creativeid = Role.creativeid AND
  budget BETWEEN 10000000 AND 20000000
  AND name = "Tom Hanks";
```

Glossary

Acceptance criteria

the agreed critera which have to be met before the product can be handed over to the client.

Accuracy of output

the output generated by a solution, matches the specification exactly, based on the data available

Actor

Individuals who interact with a system in a Use Case Diagram or a Sequence Diagram.

Agile

an approach to software development that is flexible and focuses on working software over documentation and complete sets of requirements.

Attribute

a single detail or heading used to identify one data value held in a database system.

Budget

the amount of money available to complete a project.

Cardinality

refers to the type of relationships used to link two entities. In particular, it refers to the number of instances of one entity that are related to one instance of a second entity. Examples include one-to-one, one-to-many and many-to-many.

Client

the person or company that initiates the development of a new system by producing a project proposal.

Column

single heading in a table used to identify one data value held in a database system

Compound primary key

a primary key formed using two or more attributes Sometimes called a concatenated key.

Concatenated key

see ***compound primary key***

Constraints

the variables that limit the activities of the project: scope, time, cost and quality.

Cost

the financial value of a resource used to deliver part of a project (may also apply to the whole project).

CREATE DATABASE

a command used to create a database of a given name. The name follows the command e.g. CREATE DATABASE mydatabase

Data Definition Language (DDL)

as the name suggests DDL is the collection of SQL commands that are concerned with creating and configuring a database and the tables within it. These are CREATE and DELETE statements at Advanced Higher level. A CHECK can also be used to add validation to a column

Data dictionary

used to record the details and descriptions of every data item stored within a database system.

Data Manipulation Language (DML)

DML is concerned with the inserting, deleting and modification of data in a database. This includes INSERT INTO, UPDATE and DELETE FROM statements at Advanced Higher level

Data size

entry in a data dictionary that specifies the maximum number of characters that can be stored by a particular data item.

Data type

an entry in a data dictionary that identifies the type of value that will be stored in the data item concerned.

Date

the date type in SQL records any date in the range '1000-01-01' to '9999-12-31'. It uses dates in the format: YYYY-MM-DD.

Entity

used to store a group of related attributes in a database system.

Entity integrity

an entity with a non-null primary key that has no repeating groups.

Entity occurrence

an entity can contain a number of individual items that all share the same properties. An entity occurrence is one particular item within an entity.

Entity-relationship diagram

an entity-relationship diagram shows the relationship among entity sets.

Float

a real number / floating point value is stored in the database using the float data type.

Foreign key

primary value from one entity that has been copied into another entity

Identifying relationship

relationship between a weak entity and its owner entity. The relationship is needed to provide the weak entity with a unique identifier.

Inner query

a subquery is an inner query which exists within an outer query

Integer

the integer data type has five different variants in SQL which allow numbers of different ranges to be stored.

Item name

entry in a data dictionary that is used to identify the particular data item. It is good practice to ensure that all data items within a system are unique.

Lookup existing

entry in a data dictionary that is used to make sure that a value entered in a particular data item already exists elsewhere in the system.

Milestones

a significant stage or event in the development of a project.

Objectives

what the project is intended to achieve.

Optionality

refers to the type of relationship between two entities. The optionality of a relationship identifies whether the relationship is optional or mandatory.

Outer query

where a subquery is used it exists within an outer query. The subquery provides data that is used by the outer query

Owner entity

this term is sometimes used to refer to an entity that supports a weak entity by supplying a foreign key value.

Presence check

type of validation that is used to make sure that a value is entered into a particular data item.

Primary key

a single attribute or combination of attributes that uniquely identifies a single row of data held in a table.

Product backlog

is the single most important artifact in the Scrum approach to agile development. The product backlog is, in essence, an incredibly detailed analysis document, which outlines every requirement for a system, project, or product.

Project manager

the person in overall charge of the planning and execution of a particular project.

Quality

the standard of something as measured against other things of a similar kind

Range check

type of validation that is used to make any value entered lies between a fixed upper and lower limit.

RDBMS

Relational Database Management System. This refers to the software product that is used to implement a relational database system. The RDBMS provides the developers with the tools needed to build the system.

Referential integrity

where two related entities are implemented to ensure that a foreign key in one entity cannot contain a value that doesn't already exist as a primary key in the other entity.

Relational data model

a method of organising the storage of data within a system. In the relational model, data is held in several separate tables that are linked using relationships.

Relationship

a relationship shows how two entities share information.

Restricted choice

type of validation that specifies that values that can be used for a particular data item. Only values that have been specified can be entered.

Row

a row in a table contains all the data about that one instance of an entity

Scope

the detail of what a project is to achieve. Changes to the scope of the project have a direct impact on the time and cost of the project.

SCRUM

is an iterative and incremental agile software development framework for managing product development.

Sprint

a period of development when a fixed set of product items (the sprint backlog) is developed.

SQL

Structured Query Language

Strong relationship

a type of relationship that exists between two entities where neither entity provides a unique identifying value for the other.

Subquery

a Subquery or Inner query is a query within another SQL query. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved

Time

the time type in SQL records values in the format 'hh:mm:ss'. It can be used to store times which go beyond those in the 24hr clock. This means as well as times in the day, it can also be used to store durations. Be careful about assigning abbreviated values to a *time* column. SQL interprets abbreviated *time* values with colons as time of the day. That is, '11:12' means '11:12:00', not '00:11:12'.

Timescale

a period of time to complete an activity.

Use cases

is an approach used in system analysis to identify, clarify, and organize system requirements. Use cases are made up of sets of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

User persona

is a representation of the goals and behavior of a hypothesized group of users. In most cases, personas are created from data collected from interviews with users.

User scenarios

describe the stories and context behind why a specific user or user group comes to your site or use your application. They note the goals and questions to be achieved and sometimes define the possibilities of how the user(s) can achieve them on the site.

User stories

are short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

Validation

rules in the data dictionary that are used to restrict the values that can be entered. Examples include ***presence check***, ***range check***, ***restricted choice*** and ***lookup***.

VARCHAR

this data type is used because this is very efficient when storing text; it only stores the number of characters used rather than storing the whole length of the field according to the declared size.

Waterfall model

is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.

Weak relationship

an entity that depends on another entity to exist. It cannot form a unique primary key of its own.

Answers to questions and activities

Topic 1: Analysis

Activity: Types of feasibility (page 12)

Q1:

A project that cannot be delivered in the time available because of the complexity of the task	has an issue with schedule feasibility.
A project which has insufficient finance to hire the necessary staff to complete the project when required	has an economic feasibility issue.
A project that processes banking transactions to indicate spending habits in contravention of data protection laws	has an issue with legal feasibility.
A project which cannot proceed because the software to link two data sources into the project is not available	has an issue with technical feasibility.

Review of Use Case Diagrams (page 19)

Q2:

Use Case diagrams are made using **Actors** that will interact with the system. The system is indicated by a **system boundary** where all the use cases will be placed. A Use Case in the diagram will be represented by an **ellipse**.

There are 5 relationships we need to use in a Use Case Diagram:

Association

Include — this is a use case that is always **actioned** but not directly by the Actor.

Extend — This Use Case is not directly accessed by the Actor and doesn't always run.

Generalisation of an **Actor**

Generalisation of a Use Case

End of topic test: Analysis (page 19)

Q3: a) A person or entity that interacts with the system.

Q4: c) how actors interact with the system.

Q5: c) economic and technical

Q6:

- Time
- Scope
- Legal
- Cost

Q7:

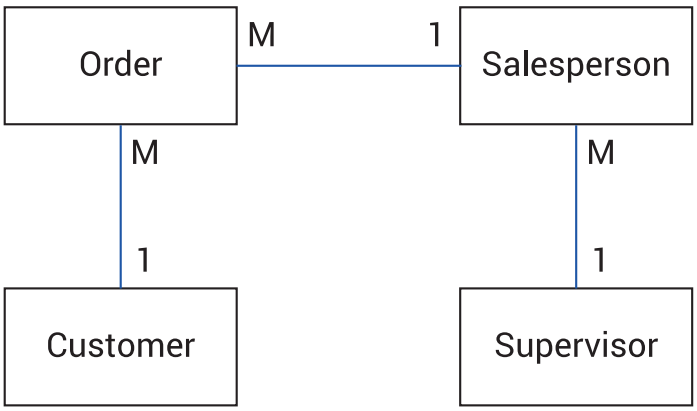
- End User Requirements
- Scope
- Boundaries
- Constraints

Topic 2: Design

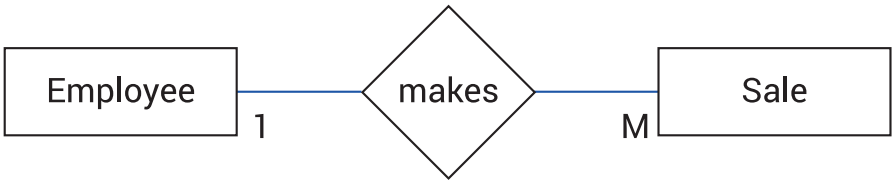
Quiz: Revision (page 23)

Q1: c) One-to-one

Q2: d)



Q3: b)



Q4: a) one mechanic, but a mechanic can repair several cars.

Q5: a)

Table(s) and/or query	Business
Fields and/or calculations	businessname, COUNT(businessname)
Search criteria	
Grouping	BY businessname
Sort order	businessname ASC

Quiz: Cardinality (page 30)

Q6: c) Many-to-many

Q7: Each child has 2 parents and each parent can have more than one child.

Q8: b) One-to-many

Q9: Each team has several players but each player can only be a member of one team at a time.

Q10: c) Many-to-many

Q11: Each pupil has many subjects and each subject is studied by many pupils.

Q12: b) One-to-many

Q13: Each pet has one owner but each owner may have several pets.

Q14: a) One-to-one

Q15: A husband has one wife and a wife has one husband - each has one of the other at a time.

Quiz: Relationships - Authors and books (page 34)

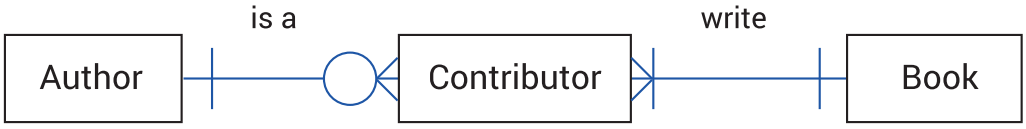
Q16: *Author* and *Book*

Q17: a) writes

Q18: c) M:N

Q19: b) *Author*: mandatory, *Book* optional

Q20:



Quiz: Relationships - Customers and orders (page 35)

Q21: *Customer* and *Order*

Q22: a) orders

Q23: b) 1:M

Q24: c) *Order*: optional, *Customer* mandatory

Q25:

Quiz: Relationships - Headteachers and schools (page 36)

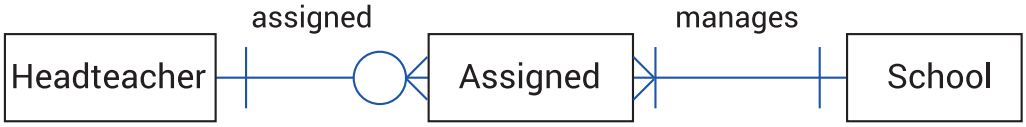
Q26: *Headteacher* and *School*

Q27: c) manages

Q28: c) M:N

Q29: a) *Headteacher*: mandatory, *School* mandatory

Q30:



Quiz: Relationships - Airports and flights (page 37)

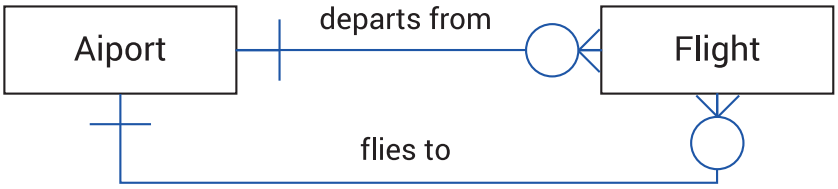
Q31: *Airport* and *Flight*

Q32: d) connects

Q33: c) M:N

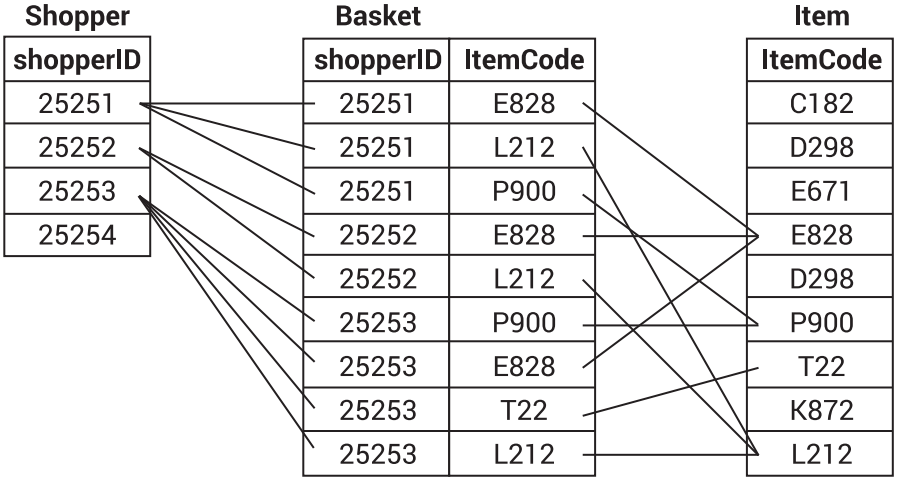
Q34: c) *Flight*: optional, *Airport* mandatory

Q35:



Quiz: Entity-occurrence diagram (page 43)

Q36:



Shopper
shopperID
25251
25252
25253
25254

Basket	
shopperID	ItemCode
25251	E828
25251	L212
25251	P900
25252	E828
25252	L212
25253	P900
25253	E828
25253	T22
25253	L212

Item
ItemCode
C182
D298
E671
E828
D298
P900
T22
K872
L212

Q37: b) *Shopper* 1:M *Basket* M:1 *Item*

Q38: c) *Shopper* mandatory - optional *Basket* optional - mandatory *Item*

Quiz: Surrogate keys (page 45)

Q39: c)

Ship (shipid, shipname, companyid*, vesselcapacity, regport)
Company (companyid, coname, telephone, email)
Booking (bookingid, shipid*, containerid*, dateout)
Container (containerid, contents, originated, destination)

Q40: c)

Customer (customerid, licenceno, firstname, lastname, address, mobile, email)
Rental (rentalid, rentaldate, regno*, customerid*, milesout, milesin)
Car (carid, regno, make, model, rentcost)
Service (serviceid, carid *, garageid*, servicedate, mileage, cost)
Garage (garageid, garagename, address)
Servicepart (servicepartid, partcode*, serviceid*)
Part (partcode, partdescription)

Quiz: Data dictionary (page 54)

Q41:

Entity: Ship					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
shipid	PK	INT		Yes	
shipname		VARCHAR		Yes	
companyid	FK	INT		Yes	existing companyid from Company
vesselcapacity		MEDIUMINT		Yes	
regport		VARCHAR	20	Yes	

Entity: Company					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
companyid	PK	INT		Yes	
coname		VARCHAR		Yes	
telephone		VARCHAR	15	Yes	
email		VARCHAR	320	Yes	

Entity: Booking					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
bookingid	PK	INT		Yes	
shipid	FK	INT		Yes	existing shipid from Ship
containerid	FK	INT		Yes	existing containerid from Container
dateout		DATE		Yes	

Entity: Container					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
containerid	PK	INT		Yes	
contents		VARCHAR	20	Yes	
originated		VARCHAR	40	Yes	
destination		VARCHAR	40	Yes	

Quiz: Query design (page 68)

Q42:

Field(s) and/or calculations(s)	title, budget, COUNT(Role.roleid) AS castsize
Table(s) and/or query(-ies)	Movie, Role
Search criteria	
Grouping	title
Having	COUNT(*) $>$ 6
Sort order	

Q43:

Field(s) and/or calculations(s)	title, gross
Table(s) and/or query(-ies)	Movie, Role, Creative
Search criteria	budget BETWEEN 10000000 AND 20000000 AND name = "Tom Hanks"
Grouping	
Having	
Sort order	

Q44:

Field(s) and/or calculations(s)	title
Table(s) and/or query(-ies)	Movie, Role, Creative
Search criteria	name IN ("James Earl Jones", "Samuel L. Jackson","Lawrence Fishburne")
Grouping	
Having	
Sort order	

Q45:

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, budget, gross, name</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie, Role, Creative</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">Role.ord = 1</td></tr>
<tr><td rowspan="3">Movie.id = ANY</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie, Role, Creative</td></tr>
<tr><td>Search criteria</td><td>name IN (”Francis Ford Coppola”, “Martin Scorsese”)</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

Q46:

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">title, gross</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Movie</td></tr>
<tr><td rowspan="4">Search criteria</td><td colspan="4">title LIKE “Star Wars%”</td></tr>
<tr><td rowspan="3">Movie.id NOT IN</td><td rowspan="3">Inner query</td><td>Field(s) and/or calculation(s)</td><td>Movie.id</td></tr>
<tr><td>Table(s)</td><td>Movie, Directed, Creative</td></tr>
<tr><td>Search criteria</td><td>name = George Lucas”</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

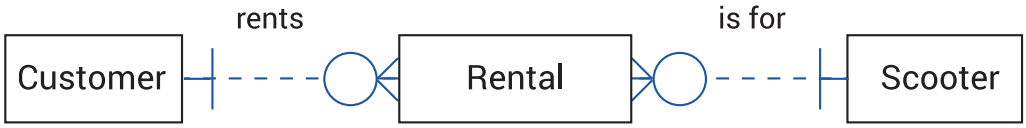
Q47:

Field(s) and/or calculation(s)	title, name			
Table(s) and/or query(-ies)	Movie, Directed, Creative			
Search criteria	budget =	**Inner query**	**Field(s) and/or calculation(s)**	MAX(budget)
			Table(s)	Movie
			Search criteria	
Grouping				
Having				
Sort order				

End of topic test: Design (page 69)

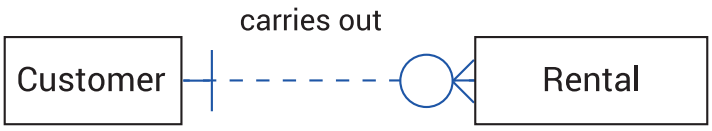
Q48: d) Agent, Scooter, Customer

Q49: a)



Q50: b) It is an entity that cannot be uniquely identified by its attributes alone and therefore requires at least one foreign key from another entity to create a primary key.

Q51: c)



Q52: b) BLOB

Q53: c) A query which is nested inside another query and generates an answer table which the outer query uses.

Topic 3: Implementation

Quiz: Revision (page 73)

Q1: b) `DELETE FROM developer WHERE firstname = 'Peter';`

Q2: a) `SELECT * FROM developer WHERE firstname LIKE 'a%';`

Q3: c) `SELECT * FROM client WHERE maxrent < 800;`

Q4: c)

type	most expensive
SUV	73000
Coupe	75000

Q5: c) To group results by country so that each country only appears once.

Creating the Directed table (page 85)

Q6:

```
CREATE TABLE Directed (
  directorid INT PRIMARY KEY AUTO_INCREMENT,
  movieid INT NOT NULL,
  creativeid INT NOT NULL,
  FOREIGN KEY (movieid) REFERENCES Movie(id),
  FOREIGN KEY (creativeid) REFERENCES Creative(creativeid)
);
```

Creating database and tables for the Esports league database (page 86)

Q7:

```
CREATE DATABASE Esports;

CREATE TABLE Team (
  teamid INT PRIMARY KEY AUTO_INCREMENT,
  handle VARCHAR(15) NOT NULL
);

CREATE TABLE Player (
  playerid INT PRIMARY KEY AUTO_INCREMENT,
  handle VARCHAR(15) NOT NULL,
  playerstatus VARCHAR(7) NOT NULL
    CHECK (playerstatus = "Active" OR playerstatus ="Retired")
);

CREATE TABLE Roster (
  rosterid INT PRIMARY KEY AUTO_INCREMENT,
  teamid INT NOT NULL,
  playerid INT NOT NULL,
  position TINYINT NOT NULL,
  joindate DATE NOT NULL,
  FOREIGN KEY (teamid) REFERENCES Team(teamid),
  FOREIGN KEY (playerid) REFERENCES Player(playerid)
);

CREATE TABLE Game (
  gameid INT PRIMARY KEY AUTO_INCREMENT,
  hometeamid INT NOT NULL,
  awayteamid INT NOT NULL CHECK(awayteamid<>hometeamid),
  gamedate DATE NOT NULL,
  durataion TIME NOT NULL,
  homescore TINYINT NOT NULL
    CHECK (homescore >=0 AND homescore <=2),
  awayscore TINYINT NOT NULL
    CHECK (awayscore >=0 AND awayscore <=2),
  FOREIGN KEY (hometeamid) REFERENCES Team(teamid),
  FOREIGN KEY (awayteamid) REFERENCES Team(teamid)
);
```

Creating database and tables for the Shipping database (page 88)

Q8:

```
CREATE DATABASE Shipping;

CREATE TABLE Company (
  companyid INT PRIMARY KEY AUTO_INCREMENT,
  coname VARCHAR(15) NOT NULL,
  telephone VARCHAR(15) NOT NULL,
  email VARCHAR(320) NOT NULL
);

CREATE TABLE Container (
  containerid INT PRIMARY KEY AUTO_INCREMENT,
  contents VARCHAR(20) NOT NULL,
  originated VARCHAR(40) NOT NULL,
  destination VARCHAR(40) NOT NULL
);

CREATE TABLE Ship (
  shipid INT PRIMARY KEY AUTO_INCREMENT,
  shipname VARCHAR(25) NOT NULL,
  companyid INT NOT NULL,
  vesselcapacity MEDIUMINT NOT NULL
    CHECK(vesselcapacity>=8000 AND vesselcapacity<=32000),
  regport VARCHAR(20) NOT NULL,
  FOREIGN KEY (companyid) REFERENCES Company(companyid)
);

CREATE TABLE Booking (
  bookingid INT PRIMARY KEY AUTO_INCREMENT,
  shipid INT NOT NULL,
  containerid INT NOT NULL,
  dateout DATE NOT NULL,
  datein DATE CHECK(datein > dateout),
  FOREIGN KEY (shipid) REFERENCES Ship(shipid),
  FOREIGN KEY (containerid) REFERENCES Container(containerid)
);
```

Create your own SQL queries: Movies database (page 111)

Q9:

```
SELECT title, budget, COUNT(Role.roleid) AS castsize
FROM Movie, Role
WHERE Movie.id = Role.movieid
GROUP BY title HAVING count(*) > 6;
```

Q10:

```
SELECT *
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Creative.creativeid = Role.creativeid AND
  budget BETWEEN 10000000 AND 20000000
  AND name = "Tom Hanks";
```

Q11:

```
SELECT title
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  name IN ("James Earl Jones", "Samuel L. Jackson","Lawrence
     Fishburne");
```

Q12:

```
SELECT title, budget, gross, name
FROM Movie, Role, Creative
WHERE ord = 1 AND
  Movie.id = Role.movieid AND
  Role.creativeid = Creative.creativeid AND
  Movie.id IN (
    SELECT Movie.id
      FROM Movie, Directed, Creative
      WHERE Movie.id = Directed.movieid AND
      Directed.creativeid = Creative.creativeid AND
      (name = "Francis Ford Coppola" OR name = "Martin Scorsese")
);
```

Q13:

```
SELECT title, gross
FROM Movie
WHERE title LIKE "Star Wars%" AND
  Movie.id NOT IN (
    SELECT Movie.id
    FROM Movie, Directed, Creative
    WHERE Movie.id = Directed.movieid AND
      Directed.creativeid = Creative.creativeid AND
      name = "George Lucas"
  );
```

Q14:

```
SELECT title, name
FROM Movie, Directed, Creative
WHERE budget = (
  SELECT max(budget)
  FROM Movie ) AND
  Movie.id = Directed.movieid AND
  Directed.creativeid = Creative.creativeid;
```

End of topic test: Implementation (page 113)

Q15: c) Data Definition Language (DDL) and Data Manipulation Language (DML).

Q16: d) CREATE, DROP.

Q17: a) INSERT INTO, UPDATE, DELETE FROM.

Q18: d) `CREATE DATABASE Scholar;`

Q19: a)

```
CREATE TABLE Sellers (
  sellerno INT PRIMARY KEY AUTO_INCREMENT,
  firstname VARCHAR(15) NOT NULL,
  lastname VARCHAR(15) NOT NULL,
  datejoined DATE NOT NULL,
  rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
  location INT NOT NULL,
  FOREIGN KEY (location) REFERENCES Area(location)
);
```

Q20: b) Sets the value for salary, for all rows in the table, to 100000.

Q21: c) A list of employees who earn more than the average for all employees.

Q22: b) A list of employee lastnames and firstnames where the employee has an office code which matches one of the offices with a country of "SCO".

Q23: d) `SELECT officecode FROM Office WHERE country = "SCO"`

Q24: a) A list of customer numbers and names where the customer has at least one order with a value of more than 60000.

Topic 4: Testing and evaluation

Quiz: Revision (page 118)

Q1: c) the solution is good enough to meet the required need.

Q2: b)

```
SELECT COUNT(*) AS 'number of cities', Country.name
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name = 'Spain'
GROUP BY Country.name;
```

Q3: d)

```
SELECT City.id, City.name, Country.name
FROM City, Country
WHERE Country.capital = City.id AND (Country.name = 'Germany' OR
   Country.name = 'France');
```

Q4: a)

```
SELECT Country.name, City.name, City.info
FROM City, Country
WHERE City.countrycode = Country.code AND Country.name = 'Afghanistan'
ORDER BY City.info DESC;
```

Q5: c) the output generated exactly matches the requirements.

SQL implemented tables match design: Employees database (page 122)

Q6:

1. birth_date in Employee is TIME, should be DATE
2. firstname in Employee is VARCHAR(10), should be VARCHAR(14)
3. hire_date in Employee is NULL, should be NOT NULL
4. emp_dept_id in Dept_emp cannot be NULL, should be NOT NULL
5. Dept_emp table is missing both foreign keys, should include the text

   ```
   FOREIGN KEY (dept_no) REFERENCES Department(dept_no),
   FOREIGN KEY (emp_no) REFERENCES Employee(emp_no)
   ```

6. manager_id in Dept_manager is VARCHAR, should be INT
7. Foreign key dept_no references dept_name in Department, should be Department(dept_no)
8. to_date in Dept_manager is NOT NULL, should be NULL

SQL implemented tables match design: Reseller database (page 126)

Q7:

1. agent_code in Agent is NULL, should be NOT NULL
2. agent_code in Agent is not a primary key, it should include PRIMARY KEY
3. commission in Agent is FLOAT(8,2), it should be FLOAT(10,2)
4. country is missing from Agent, it should be

```
country VARCHAR(25) NULL
```

5. grade in Customer is VARCHAR(10), should be INT
6. outstanding_amt in Customer is NULL, should be NOT NULL
7. phone_no in Customer is VARCHAR(5), should be VARCHAR(17)
8. agent_code in Customer as foreign key incorrectly references Order(ord_num), should be Agent(agent_code)

SQL operations work correctly at this level: Quiz (page 128)

Q8: a) Correct

Q9: b) Incorrect

The correct query is:

```
SELECT COUNT(Customer.id), country
FROM Customer, Country
WHERE Customer.countryid = Country.id AND country <> "Scotland"
GROUP BY country
HAVING COUNT(Customer.id) >= 20;
```

Q10: b) Incorrect

The correct query is:

```
SELECT AVG(totalvalue), firstname, lastname
FROM Order, Customer
WHERE Order.customerid = Customer.id
GROUP BY Customer.id
HAVING AVG(totalvalue) BETWEEN 2000 AND 3000;
```

Q11: a) Correct

Q12: b) Incorrect

The correct query is:

```
SELECT productname
FROM Product
WHERE Product.pid IN (
  SELECT Orderitem.pid
  FROM Orderitem
  WHERE quantity > 100
);
```

End of topic test: Testing and evaluation (page 132)

Q13: a) Error 1, Error 2, Error 4

Q14: d)

```
SELECT COUNT(Pet.petid), name
FROM Pet, Petevent
WHERE Pet.petid = Petevent.petid
GROUP BY Pet.petid
HAVING COUNT(Pet.petid) >= 2;
```

Q15: a)

```
SELECT *
FROM Petevent
WHERE petid IN (
  SELECT petid
  FROM Pet
  WHERE species = 'dog'
);
```

Q16: d) it contains additional columns in excess of those requested for the query.

Q17: c)

```
INSERT INTO Pet(name, owner, species, birth)
VALUES ('Polly', 'Morag', 'Bird', '2020-08-18');
```

Topic 5: Database design and development test

Database design and development test (page 138)

Q1: b) `$conn = mysqli_connect($server, $username, $password,$database);`

Q2: a) `$result = $mysql_query($conn, $query);`

Q3: b) Set $querystring to hold an SQL statement. Execute this query using the server and database held in the variable $conn. The results of the query will be stored in the variable $result. If the query fails then the if statement is exited and the code is completed, if the query is successful, a message will be shown, stating how many rows of data have been retrieved and the connection is closed.

Q4: b) returns the number of rows in a result set.

Q5: c) Orderitem

Q6: d) surrogate key.

Q7: b) A weak entity is an entity that cannot be uniquely identified by its attributes alone and therefore requires at least one foreign key from another entity to create a primary key.

Q8: c)

```
CREATE TABLE Movie (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL,
  rating FLOAT(3,1) NOT NULL CHECK(rating >=0 AND rating <=10),
  yr INT NOT NULL,
  budget INT,
  gross BIGINT
);
```

Q9: d)

```
INSERT INTO Roster (teamid, playerid, position, joindate)
VALUES (23, 55,"Divinity Support","2019-09-24");
```

Q10: c) `DROP DATABASE Encrypted_info;`

Q11: a)

Q12: d) record details of all data items within a relational database system.

Q13: c)

Field(s) and/or calculation(s)	COUNT(*), Creative.name
Tables(s) and/or query(-ies)	Movie, Directed, Creative
Search Criteria	yr BETWEEN 2000 AND 2010
Grouping	Creative.name
Having	COUNT(*) $>$ 5
Sort order	

Q14: d)

```
SELECT birth_date, first_name, last_name
FROM Employee
WHERE EXISTS (
  SELECT * FROM Dept_manager
  WHERE Employee.emp_no = Dept_manager.emp_no
);
```

Q15: b)

```
SELECT productname
FROM Product
WHERE productid = ANY (
  SELECT productid
  FROM OrderDetail
  WHERE quantity >= 10
);
```

Q16: a)

<table>
<tr><td>Field(s) and/or calculation(s)</td><td colspan="4">productid, productname</td></tr>
<tr><td>Table(s) and/or query(-ies)</td><td colspan="4">Product</td></tr>
<tr><td rowspan="4">Search Criteria</td><td rowspan="4">EXISTS</td><td rowspan="4">Inner query</td><td>Field(s) and/or calculation(s)</td><td>productid</td></tr>
<tr><td>Table(s)</td><td>OrderDetail</td></tr>
<tr><td>Search Criteria</td><td>SUM(quantity) > 1000</td></tr>
<tr><td>Grouping</td><td>productid</td></tr>
<tr><td>Grouping</td><td colspan="4"></td></tr>
<tr><td>Having</td><td colspan="4"></td></tr>
<tr><td>Sort order</td><td colspan="4"></td></tr>
</table>

Q17: a) INSERT INTO, UPDATE, DELETE FROM.

Q18: a)

```
CREATE TABLE Sellers (
  sellerno INT PRIMARY KEY AUTO_INCREMENT,
  firstname VARCHAR(15) NOT NULL,
  lastname VARCHAR(15) NOT NULL,
  datejoined DATE NOT NULL,
  rating TINYINT NOT NULL CHECK (rating >=0 AND rating <=100),
  location INT NOT NULL,
  FOREIGN KEY (location) REFERENCES Area(location)
);
```

Q19: c) A query which is nested inside another query and generates an answer table which the outer query uses.

Q20: d)

```
SELECT title, gross
FROM Movie, Role, Creative
WHERE Movie.id = Role.movieid AND
  Creative.creativeid = Role.creativeid AND
  budget BETWEEN 10000000 AND 20000000
  AND name = "Tom Hanks";
```