
SCHOLAR Study Guide

Higher Computing Science

Unit 3: Database design and development

Authored by:

Ian King (Kelso High School)

Mark Tennant (Community School of Auchterarder)

Charlie Love (CompEdNet)

Andy McSwan (Knox Academy)

Reviewed by:

Jeremy Scott (George Heriot's School)

Previously authored by:

Jennifer Wilson (Denny High School)

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2018 by Heriot-Watt University.

This edition published in 2018 by Heriot-Watt University SCHOLAR.

Copyright © 2018 SCHOLAR Forum.

Members of the SCHOLAR Forum may reproduce this publication in whole or in part for educational purposes within their establishment providing that no profit accrues at any stage, Any other use of the materials is governed by the general copyright statement that follows.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without written permission from the publisher.

Heriot-Watt University accepts no responsibility or liability whatsoever with regard to the information contained in this study guide.

Distributed by the SCHOLAR Forum.

SCHOLAR Study Guide Higher Computing Science: Unit 3

Higher Computing Science Course Code: C816 76

ISBN 978-1-911057-28-4

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.

All brand names, product names, logos and related devices are used for identification purposes only and are trademarks, registered trademarks or service marks of their respective holders.

Contents

1	Analysis	1
1.1	Introduction	3
1.2	End users	3
1.3	Functional requirements	5
1.4	Learning points	7
1.5	End of topic test	8
2	Design	9
2.1	Introduction	11
2.2	The relational model	23
2.3	Relationships	28
2.4	Entity-occurrence diagrams	33
2.5	Compound keys	40
2.6	Entity relationship diagrams	41
2.7	Solutions to queries	44
2.8	Learning points	51
2.9	End of topic test	52
3	Implementation	53
3.1	Application software	55
3.2	Introduction	59
3.3	Example database	72
3.4	SQL Wildcards	74
3.5	Table and column aliases	76
3.6	Using sub-queries	77
3.7	SQL aggregate functions (MIN, MAX, AVG, SUM, COUNT)	80
3.8	Computed values	82
3.9	GROUP BY	83
3.10	ORDER BY	85
3.11	Learning points	86
3.12	End of topic test	87
4	Testing and evaluation	93
4.1	Testing SQL queries	95
4.2	Evaluating SQL queries	100
4.3	Learning points	101
4.4	End of topic test	102
5	End of unit test	105
	Glossary	119

Answers to questions and activities

122

Topic 1

Analysis

Contents

1.1 Introduction	3
1.2 End users	3
1.2.1 End user requirements	4
1.3 Functional requirements	5
1.3.1 Types of functional requirements	5
1.3.2 Identifying functional requirements	6
1.3.3 Example functional requirements	6
1.4 Learning points	7
1.5 End of topic test	8

Prerequisites

From your studies at National 5 you should already know about:

- end user requirements;
- functional requirements.

Learning objective

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

- identify and define the end user from a description of a development project;
- explain the term functional requirements;
- define a number of functional requirements from the following areas:
 - interface requirements;
 - business requirements;
 - legal requirements;
 - security requirements.

1.1 Introduction

Data is anything in a form that can be used by a computer. A program is a set of instructions that set out how a computer should perform a particular task or tasks, so data is everything else that isn't a program!

A **database** is used to store data. A database is an organised store of information so that the data can be easily accessed, managed and updated.

Databases can be found in most aspects of modern life: mobile phones, cars software systems, online shopping, bus ticketing systems and supermarket checkouts all depend on database systems to function.

There are many different types of database management systems (DBMS). These are the tools that allow you to manage the data you store. These tools adopt different methods to structure the data: flat-file, distributed, object-oriented and so on; however, the method used in Higher Computing Science is relational.

Later in this unit, you will use a **relational database management system** (RDBMS) to execute queries to generate answer tables and manage your data.

1.2 End users

Read the following scenario for a database system.

Bob's Super Sweets is a sweet shop. The owner, Bob, has decided that he needs a database system to manage all the orders that the shop receives via emails and telephone calls. The staff in the shop will be able to enter the details of a customer (or select them if already in the system) and create the order for the customer. Once the order is created, the staff can add the sweets and the weight ordered to the order. The system will then calculate the cost of the order and print out an invoice to send to the customer, with the order, for payment.

Who do you think the end user is? Is it Bob, the customer or the staff in the shop?

When identifying the **end user**, you have to consider who will actually use the system. The end user is different from the user who supports/maintains the system. The end user, typically, won't have technical knowledge of the system. Also, the end user is usually not the person or company paying for the system (known as the **system owner**).

In the preceding scenario, the end users are the staff in the shop. They will use the system and the functional requirements of the system will need to be created to meet their needs.

Activity: End user scenarios

Go online



Read the following scenarios for database systems and identify the end users.

Q1: Alex Vardy owns a business that trades in used vehicles. He wants to have a database system that will allow the staff in his showroom to enter the details of vehicles that are bought and sold. When a vehicle is bought, the seller will be entered into the database with the details of the vehicle bought by the business. Similarly, when a vehicle is sold, the details of

the sale and who the buyer is will be entered. The system will create all of the paperwork associated with the trading in vehicles.

Identify the end user(s).

- a) The seller of the vehicle.
 - b) Alex Vardy, the owner of the business.
 - c) The buyer of a vehicle.
 - d) The showroom staff.
-

Q2: Amir Kartal owns a flat and wants a database system as part of his web site for renting out the flat to holiday makers. He will be able to view and edit the bookings made online and print out who has booked the flat and for when. He will use the printouts to make sure that a cleaner is booked to clean the flat after each holidaymaker leaves. Holidaymakers will be able to use the system and book the flat online. They can also print out details of their stay.

Identify the end user(s).

- a) Both Amir and the holidaymakers.
 - b) Amir.
 - c) The holiday makers.
 - d) The cleaners.
-

Q3: A library has decided to set up a database system to allow visitors to search for books and locate them on the shelves within the library. The system is to enhance the service provided by the librarians who work there by allowing visitors to self-serve rather than wait to speak with a librarian.

Identify the end users.

- a) The library owners.
- b) The librarians.
- c) Visitors to the library
- d) Visitors and librarians.

1.2.1 End user requirements

Systems are designed to meet the needs of the end users as well as the requirements of the business/organisation or person commissioning the system. The system should allow end users to carry out the required functions in the most efficient manner.

End users normally have a specific set of skills or experience with technology. This experience will be considered when the interface for the system is designed and developed to ensure that the system is user friendly, reducing the number of user interactions so that the system can be used efficiently and presenting the user with the required commands/options in an accessible way.

These 'interface requirements' are just one element of the functional requirements of the system.

1.3 Functional requirements

All systems have functional and non-functional requirements. Functional requirements specify exactly what the system should do while non-functional requirements describe how the system works.

Example : Functional requirements example

- 'Calculate and display the number of cars sold for a specific range of dates' is a functional requirement - it details something that the system must do.
- 'The system will be reliable and available online 99.9% of the time' is a non-functional requirement - it details how the system should perform but not something that it should do.

When analysing a database problem in Higher Computing Science, you are only required to identify the functional requirements of the solution to be developed.

1.3.1 Types of functional requirements

Functional requirements define specific actions that the solution must perform. There are many possible aspects of the solution that functional requirements may detail. Some of these are detailed as follows.

Interface requirements

What are the user interface requirements for the database system? These can be detailed as functional requirements based on the data entry and output required.

Example : Interface requirements example

The system must provide the main menu options:

- manage or add new customers;
- manage or create new orders;
- customer reports (select from available customers on file).

Business requirements

Business requirements determine the business rules applied within the system.

Example : Business requirements example

The system must calculate VAT on all sales using the current VAT rate. The system will have settings screen which will allow the current VAT rate to be modified and saved.

Legal requirements

All systems are required to meet the relevant laws for the data they hold. This means that rules, such as the General Data Protection Regulation (GDPR), must be applied as required.

Example : Legal requirements example

The system must timestamp all personal data stored and provide a maintenance screen which will allow data to be deleted after a defined retention period. This period can be modified via a settings screen.

Security requirements

All systems should have appropriate security requirements applied to them. These requirements ensure that data is kept securely and that users have the appropriate level of access to the system.

Example : Security requirements example

The system must have three levels of access:

1. Level 1 will allow read only access to order and customer data.
2. Level 2 will allow the modification and input of data for all areas of the system.
3. Level 3 will allow access to all aspects of the system including setup/configuration screens and user management.

1.3.2 Identifying functional requirements

Functional requirements are identified by a systems analyst as part of the analysis of the system. They will be added to the **requirements specification** or the **product backlog** depending on whether an iterative development process or an agile one has been adopted.

The process of identifying functional requirements is the same for each approach. The systems analyst will complete the following sequence of activities.

- End user or system owner request a requirement: The analyst receives the requirement request.
- Analyse: the systems analyst examines the requirement; why it is needed and what is required to deliver it. The analyst will have to ensure that the requirement is fully understood before it can be documented. This may include observation of existing operations, reviewing documents produced by existing processes or modelling the requirement to ensure that it is correctly understood.
- Use case: a use case is a list of actions or events detailing the interactions, between an end user and the system, to achieve the functional requirement. It is a way to document what must happen.
- Incorporate: once the functional requirement is documented, it can be added to the requirements specification or product backlog for implementation.

1.3.3 Example functional requirements

A systems analyst has prepared the following partial functional requirements for Bob's Sweet Shop system. This is written as a number of simple use cases.

- The system must have an interface to access the following options: manage customers,

manage orders, manage stock, manage users, modify settings.

- The manage customers options must allow search, viewing, creation or modification of customer information.
- The manage orders option must allow search, viewing, creation or modification of orders.
- There must be a print option for each order that will generate an invoice for the order.
- When creating or editing an order, it will be possible to select a stock item and enter the weight of that item ordered. The system must calculate the value for each line in an order using the stock item price and the weight.
- The system will calculate the total cost of an order and include any additional charges such as VAT and delivery.

1.4 Learning points

Summary

You should now know that:

- end users can be identified and defined from a description of a development project;
- the term functional requirements can be explained;
- a number of functional requirements from the following areas can be defined:
 - interface requirements;
 - business requirements;
 - legal requirements;
 - security requirements.

1.5 End of topic test

End of Topic 1 test

Go online



Q4: Which of the following is **not** a functional requirement?

- a) The system must allow a password to be set with a minimum of eight characters.
- b) The system must allow a password to be set with at least one upper-case character, one number and one text symbol.
- c) The system will be protected by an authentication system to prevent unauthorised access.
- d) The system must allow a user to login to the system using a username and password.

.....

Q5: Who is normally responsible for identifying and documenting the functional requirements?

- a) End user
- b) Programmer
- c) Systems analyst
- d) System owner

.....

Q6: How are functional requirements different from non-functional requirements?

- a) Functional and non-functional requirements explain the quality requirements of system. Non-functional requirements state which parts of the system should be measured and the functional requirements detail how each part should be measured.
- b) Functional requirements define what the system must do. Non-functional requirements define how the system should behave.
- c) Functional requirements detail how the modules of code in the system should be linked to each other. Non-functional requirements detail how the documentation will be written.
- d) Functional requirements are defined by the system owner only whereas non-functional requirements can be suggested by anyone associated with the system.

Topic 2

Design

Contents

2.1	Introduction	11
2.1.1	Entities and attributes	11
2.1.2	Relationships	11
2.1.3	Primary and foreign keys	13
2.1.4	Attribute types and size	15
2.1.5	Validation	17
2.1.6	Data dictionary	18
2.1.7	Entity relationship diagrams	20
2.1.8	Design a solution to a query	21
2.2	The relational model	23
2.3	Relationships	28
2.3.1	One-to-one	28
2.3.2	Many-to-many	30
2.4	Entity-occurrence diagrams	33
2.5	Compound keys	40
2.6	Entity relationship diagrams	41
2.7	Solutions to queries	44
2.8	Learning points	51
2.9	End of topic test	52

Prerequisites

From your studies at National 5 you should already know how to:

- use simple entity relationship diagrams, making use of a one-to-many relationship between entities and showing their attributes;
- construct a data dictionary and know its purpose;
- identify attribute types including text, number, date, time and Boolean and attribute sizes where required;
- specify validation rules for data;
- design a query based on two tables, fields, search criteria and sort order.

Learning objective

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

- create and describe entity-relationship diagrams using three or more entities;
- exemplify relationships and their cardinality as one-to-one, one-to-many or many-to-many;
- use entity-occurrence diagrams to identify and present the relationship between two entities;
- describe, and explain through examples, a compound key;
- describe, and explain through examples, a data dictionary with three or more entities including attribute names, types, size and validation;
- design, create and explain a design of solution to a query making use of tables, queries, fields, search criteria, sort order, calculations and grouping.

2.1 Introduction

In Higher Computing Science we use the relational database model. The model details how data is to be organised, how it can be accessed and how it can be manipulated. This topic deals with how solutions to database problems are designed. This introduction is a recap of what you will have covered as part of the National 5 course.

2.1.1 Entities and attributes

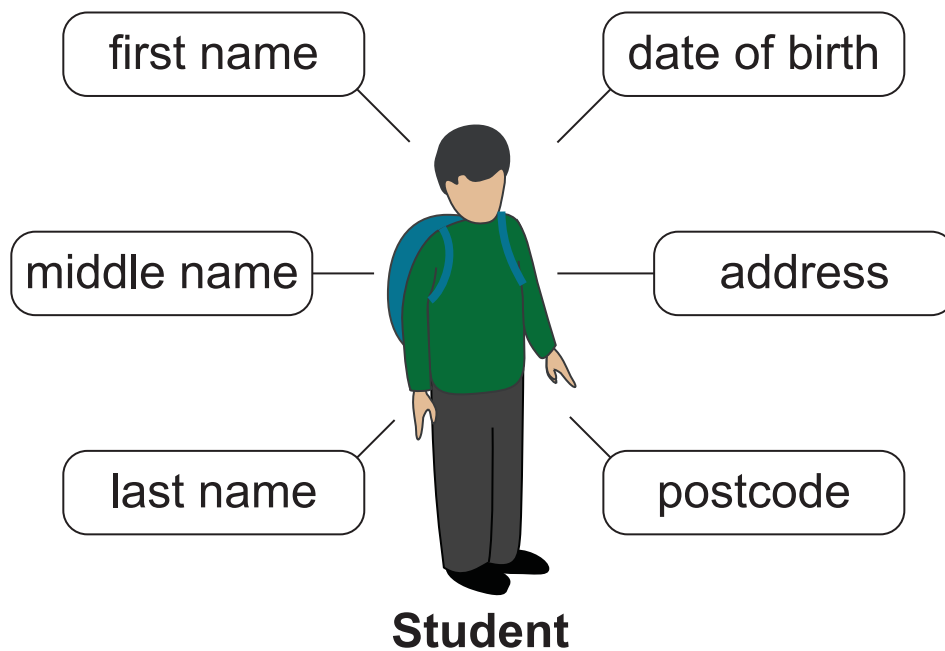
During the design of a database we refer to entities and attributes.

An **entity** is something that exists - it is something that we want to store information about. It could be a physical thing (such as a book or a person) or it could be something abstract (such as a reservation at a restaurant or a payment for rent on a home).

A collection of entities (all the students in a school, for example) is called an **entity set**.

Attributes are the information that we hold about an entity. For example, a student in school is an entity and the attributes that may be stored for a student are: first name, middle name, last name, date of birth, address, postcode and so on.

Figure 2.1: Student entity with attributes



2.1.2 Relationships

Relationships show how two entities are related to each other. You should be familiar with the relationship type one-to-many.

For example, one doctor will have many registered patients, but each patient will have only one registered doctor.

Table 2.1: One-to-many relationship

DoctorID	DoctorName	PatientID	PatientName
81	Dr Paul Johnson	891201	Mrs Morag Proudfoot
		090120	Miss Claire Shaw
		210092	Mr Henry Ford
142	Dr Wendy Reid	093791	Mr Ivor Barr
		103792	Miss Iona Jones
		210198	Mr Ian Hosack
223	Dr Vanessa Smith	210200	Mr David Chisholm
		781902	Mr Gordon Buck
226	Dr James Brown	110202	Mr Rob Elliott
		093794	Miss Louise Flower
		210199	Mr Nick Spencer

This is a one-to-many relationship. An entity on one side can be associated with multiple entities on the other side BUT the entities on the other side are only associated with one entity opposite them, i.e Dr James Brown has two registered patients: Miss Louise Flower and Mr Nick Spencer AND Miss Louise Flower is only registered with ONE doctor, Dr James Brown.

A further example would be accounts on a social media platform and posts made. Each account can create one or more posts but each post is created by only one account.

Table 2.2: Another one-to-many relationship

User		Post	
UserID	AccountName	PostID	PostContent
72618	NKSL	281728	Using Google Tour...
		281721	DAY #21 on...
72619	GameTag101	281722	Cycling through apocolyptic...
		281731	It is almost exactly...
72620	WhoRYou	281732	Want to get back into...
		281749	I had a lot of fun...
72621	HappyHarry	281700	You know you've been...
		281735	All organised and very...
		281767	Well done Stirling...
		281792	More slow runners needed...
		281793	Tune in to amazing...

There are other types of relationship which we will cover later in this topic.

2.1.3 Primary and foreign keys

When an entity follows the rules for the relational model, it should have a **primary key**. This is a value which is unique for the entity. In the example of social media posts which follows, each user has a unique UserID and each post has a unique PostID.

UserID is the primary key of the User entity. PostID is the primary key of the Post entity.

Table 2.3: One-to-many relationship with primary and foreign keys

User		Post		
<u>UserID</u>	AccountName	<u>UserID</u> *	<u>PostID</u>	PostContent
72618	NKSL	72618	281728	Using Google Tour...
		72618	281721	DAY #21 on...
72619	GameTag101	72619	281722	Cycling through apocalyptic...
		72619	281731	It is almost exactly...
72620	WhoRYou	72620	281732	Want to get back into...
		72620	281749	I had a lot of fun...
		72620	281700	You know you've been...
72621	HappyHarry	72621	281735	All organised and very...
		72621	281767	Well done Stirling...
		72621	281792	More slow runners needed...
		72621	281793	Tune in to amazing...

When listing the attributes of an entity we use an underline to indicate the primary key.

A **foreign key** is a primary key value from one entity included in a related entity. The purpose of the foreign key is to establish the relationship between the two entities. A foreign key is typically shown in notation using an asterisk *.

In the preceding example, the primary key of User, UserID has a related foreign key in the Post entity. The two entities are now linked by primary/foreign key values that they have in common.

The posts for 'WhoRYou' can be found by using the unique value of UserID for this user, 72620, and looking for the same value in the foreign key within the posts table. In the example above, this shows that there are three posts by this user.

Activity: Primary and foreign keys

Go online



How a foreign key is used to relate data in separate entities:

Two separate entities:

Student_num	Fname	Sname
20001	Gerry	Sutherland
20002	Fiona	McGhee
20003	Sally	Fultness

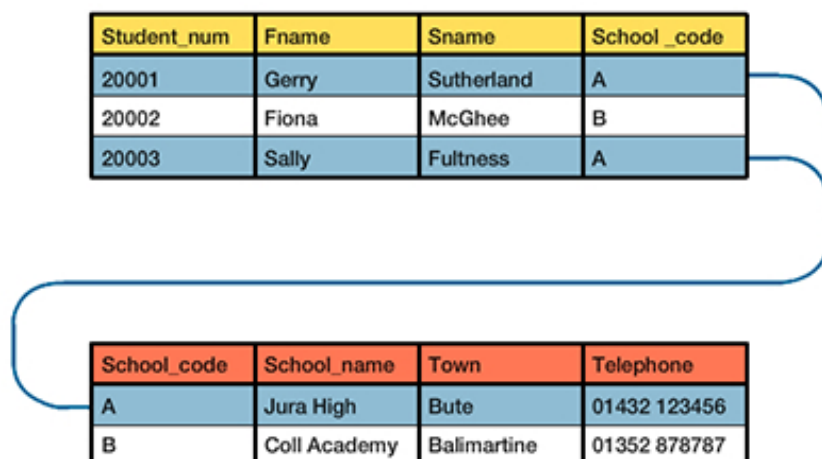
School_num	Town	Telephone
Jura High	Bute	01432 123456
Coll Academy	Ballimartine	01352 878787

'School_code' keys are added to each entity:

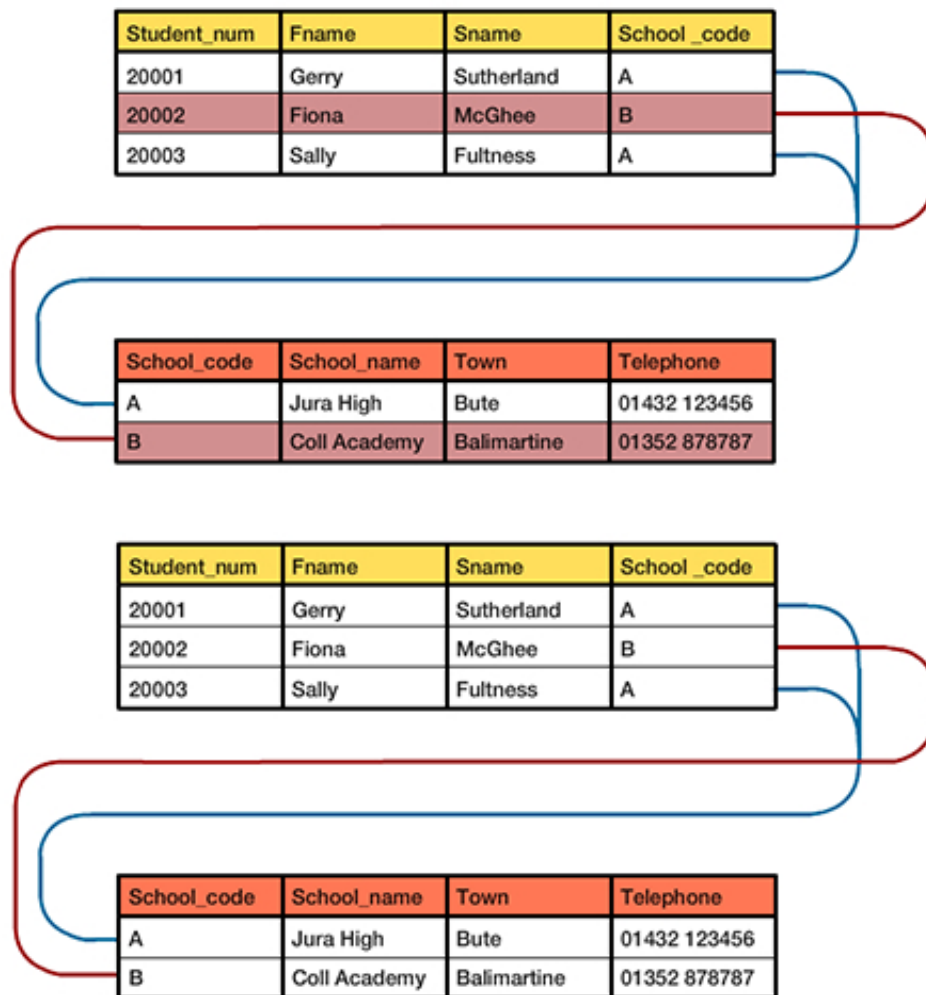
Student_num	Fname	Sname	School_code
20001	Gerry	Sutherland	A
20002	Fiona	McGhee	B
20003	Sally	Fultness	A

School_code	School_num	Town	Telephone
A	Jura High	Bute	01432 123456
B	Coll Academy	Ballimartine	01352 878787

Key A relationship:



Key B relationship:



School_code is the foreign key in the upper entity and is used in the lower entity as a primary key providing a link between the two entities.

2.1.4 Attribute types and size

Attributes hold specific types of data. For the National 5 and Higher Computing Science courses these types are defined as follows:

- **text:** Any data that is represented using a string of text. For example, the attribute lastname would normally be stored as text;
- **number:** Any data that is represented using number values. These can be integer or decimal values. Floating point representation may also be used to store very large or very small numbers;
- **date:** Any data that is of date format. For example: date of birth: '24/09/2003' is a date format. It is worth noting that many SQL software solutions display dates in the format YYYY/MM/DD. In this case, the date above would be '2003/09/24';

- **time:** Data about the time is stored in a time type. This can be more than the time of day e.g. 16:00:00, it can also hold an amount of time (duration) e.g. 561:01:02 (this is 561 hours, 1 minute and 2 seconds);
- **Boolean:** Attributes which are Boolean are either True or False.

Activity: Attribute types and size

Go online



Q1: Select the data types that you would use to store the following:

Attribute	Example	Data type
Surname	Toner	
First name	Geraldo	
Postcode	TD7 0EG	
SQA candidate number	2311447853	
Enrollment fee paid	£27.50	
All documents received	True	
Enrolment date	23/03/14	
Course begins at	19:30	
Qualification credit value	24	

Data types:

- Text
- Number
- Date
- Time
- Boolean

Size of attributes

Where attributes are of the **text** type, it is normal to specify their size as the maximum possible characters for these attributes. This can be done in a **data dictionary** and shown as the data type followed by a number of characters, e.g. text (16). This would define the attribute as being text and a maximum of 16 characters.

2.1.5 Validation

Validation is used to check that the data entered for an attribute is what is expected. There are four types of validation that can be applied to an attribute.

1. **Presence check:** this validation rule ensures that there is a value allocated in the attribute. When an attribute has no value assigned to it, it will have a value of Null. Null is a technical term meaning no value is available. It is not the same as zero or empty text "" (a string with no characters in it). Often the word "Required" is entered in a data dictionary to show that a presence check is needed.
2. **Restricted choice:** validation of this type limits the values for an attribute to a selection of predefined options. For example: Jackets are available in sizes S, M, L, XL, XXL. These are the only allowed values for the attribute. Validation rules for this can be written as "Limit to: S, M, L, XL, XXL" or may be written as "Look up: S, M, X, XL, XXL" in a data dictionary.
3. **Field length:** Attributes of type text can be limited so that they must be a given number of characters. There could be a minimum number of acceptable characters (e.g. your PIN must be at least 4 characters), a maximum number of characters (e.g. the maximum length of your message is 280 characters) or a mix of both (e.g. you username must be more than 8 and less than 16 characters long).
4. **Range:** the value of an attribute can be limited to values between a maximum and a minimum, giving a range of possible values. For example, an attribute could be validated to ensure that it was between 0 and 10 or between 0.01 and 0.99 or between -25 and 25.

This isn't limited to just attributes of the type number but can also be applied to attributes of type text. For example, limiting values from $\geq A$ and $\leq E$ would allow the values A, B, C, D and E to be used.

Activity: Validation

Go online



Choose the correct type of presence check for the attributes in each of the following questions.

Q2: UK children's shoes are available in sizes 1 to 11. Assume that integers are required for data input.

Attribute: Shoesizes

Example: 10

Validation: ?

- a) Presence check
- b) Restricted Choice
- c) Field Length
- d) Range

.....

Q3: A password for a web site must be 10 or more characters long.

Attribute: Password

Example: "Secretpassword10"

Validation: ?

- a) Presence check
 - b) Restricted Choice
 - c) Field Length
 - d) Range
-

Q4: Users must accept the terms and conditions.

Attribute: TermsCheckBox

Example:

Validation: ?

- a) Presence check
 - b) Restricted Choice
 - c) Field Length
 - d) Range
-

Q5: A voting response allows users to return "Yes", "No" or "Don't Know".

Attribute: Vote

Example: "Yes"

Validation: ?

- a) Presence check
- b) Restricted Choice
- c) Field Length
- d) Range

2.1.6 Data dictionary

A data dictionary is data about data. It is the detailed design for one or more entities in a system. A data dictionary is laid out as a table containing the definition of each attribute (and entity). Consider this data dictionary for a system for booking a swimming pool.

Table: Customer

CustomerID	CustomerName	CustomerAddress	CustomerPhone
1001	Raymond Boyce	13 Western St, Dunfermline, KY11 7TQ	01929 928311
1002	Edgar Codd	17 Queen St, Dunfermline, KY12 3EA	01929 827191
1003	Larry Ellison	The Lake House, Carnegie, KY12 8YH	01929 883919
1004	Safra Catz	92a Viewfield, Dunfermline, KY11 9UA	01929 667818

Table: Booking

BookingID	BookingTime	BookingDate	CustomerID
2910	11:00	27/07/2018	1001
2912	12:30	27/07/2018	1002
2913	14:00	28/07/2018	1003
2918	12:00	28/07/2018	1004

A data dictionary for this system:

Entity Name: Customer

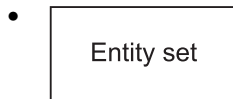
Attribute	Primary/Foreign Key	Data	Validation
CustomerID	Primary Key	Number	
CustomerName		Text(70)	
CustomerAddress		Text(475)	
CustomerPhone		Text(22)	

Entity Name: Booking

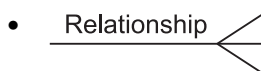
Attribute	Primary/Foreign Key	Data	Validation
BookingID	Primary Key	Number	
BookingTime		Time	
BookingDate		Date	
CustomerID	Foreign Key	Number	Lookup from Customer

2.1.7 Entity relationship diagrams

The **Entity relationship diagram** (ERD) illustrates the structure of any database. It provides a method of illustrating entity sets, relationships and attributes. Entity relationship diagrams make use of the following symbols.



A rectangle is used to represent each entity set. Remember, an entity set is a collection of entities. The name of the entity set is entered inside the rectangle to identify it. The name must never be a plural, it must always be a singular e.g. person rather than people.



The relationship between the two entity sets. Relationships illustrate how two entities sets share information. A short phrase is written above the line to describe the relationship. The 'crows feet' show the many side of the relationship.

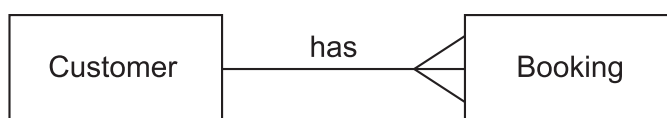


Attributes can be added to the ERD as ovals. The name of the attribute is entered inside the oval.

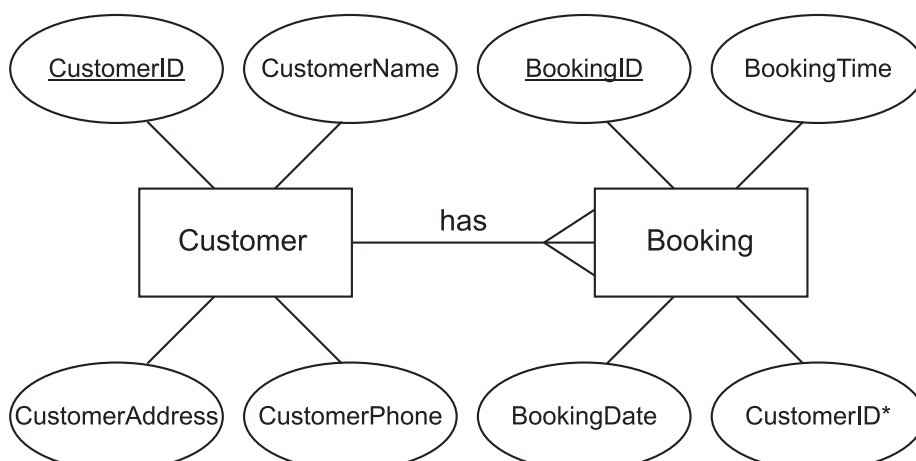


If an attribute is the primary key or part of the primary key for the entities in the entity set then its name is underlined inside the oval.

A simple entity relationship diagram for the swimming pool system shown in the previous data dictionary would be:



With the attributes added it would be:



2.1.8 Design a solution to a query

Previously you will have designed solutions to simple queries that access one or two tables. The design of a **query** requires that you make use of:

- multiple tables;
- fields;
- search criteria;
- sort order.

A **query**, when it runs, produces an **answer table**. The answer table is a collection of the data that matches the requirements of the query.

Multiple tables

An entity set is implemented as a table in a database. Each row of the table represents an instance of an entity.

Example

Table: artist

artist_id	firstname	lastname
0001	gary	barlow
0002	robbie	williams
0003	cheryl	cole
0004	dolly	parton

Table: album

album_id	title	artist_id*
01	Since I saw you last	0001
02	Open road	0001
03	Escapology	0002
04	Better day	0004
05	Swing when you are winning	0002

Fields

The attributes of an entity are implemented as fields in the database. These are the columns of the tables. The table *artist* in the example has the fields *artist_id*, *firstname* and *lastname* and the table *album* has the fields *album_id*, *title* and *artist_id*.

Search criteria

When constructing a query, we often need to select specific results based on the values of fields. Search criteria are applied to select the values that are required for the solution.

For example: `firstname="gary"`, `title="Better day"`, `title = "*you*"` (where `*` is a wildcard which means any value).

Sort order

The answer table produced by a query can be sorted based on values held in the columns. For example, the *album* table is sorted on *album_id*, ascending.

Answer tables can be sorted on one, two or more fields in ascending or descending orders.

For example, this answer table is sorted on:

year descending, comic_id ascending, hero_name ascending

Year	comic_id	hero_name
2017	6171	Aqua matrix
2017	6171	Super mole
2017	8109	Jules Volt
2017	8109	Rocket
2016	5261	Bear Meteor
2016	5261	No Capes
2016	5278	X-field
2015	9188	Superflea
2015	9188	Transfirma

Representing query design

To represent the design of a query the following layout can be used. Each element of the query is presented as follows.

Tables	
Fields	
Search criteria	
Sort order	

A database contains the following tables (as you saw earlier).

Table: Customer

CustomerID	CustomerName	CustomerAddress	CustomerPhone
1001	Raymond Boyce	13 Western St, Dunfermline, KY11 7TQ	01929 928311
1002	Edgar Codd	17 Queen St, Dunfermline, KY12 3EA	01929 827191
1003	Larry Ellison	The Lake House, Carnegie, KY12 8YH	01929 883919
1004	Safra Catz	92a Viewfield, Dunfermline, KY11 9UA	01929 667818

Table: Booking

BookingID	BookingTime	BookingDate	CustomerID
2910	11:00	27/07/2018	1001
2912	12:30	27/07/2018	1002
2913	14:00	28/07/2018	1002
2918	12:00	28/07/2018	1004

A query is required to display the name, booking time, booking date and phone number for all the people from post codes starting with 'KY11' who have a booking. The results (the answer table) should be sorted on Booking Date and Booking Time, both ascending.

The query design would be:

Tables	Customer, Booking
Fields	CustomerName, BookingTime, BookingDate, CustomerPhone
Search criteria	CustomerAddress = "*KY11*", Customer.CustomerID = Booking.CustomerID
Sort order	BookingDate ASC, BookingTime ASC

2.2 The relational model

A **relational database** is a database which contains more than one table. The tables are linked together by using primary and foreign keys.

Advantages of a relational database

Relational databases were developed to avoid unnecessary duplication of data in the database.

Let's look at a Flat-file database to see what we mean by unnecessary duplication:

Library loans:

Borrower No	Borrower Name	Borrower Address	Borrower Phone	Book No	Title	Author	Loaned Out	Due Back
1001	Edgar Codd	12 High Street, Scholar	01234 985443	F1301	Wuthering Heights	Bronte, C	12 Jan	11 Feb
1002	Raymond Boyce	6 Castle View, Scholar	01234 983458	F1301	Wuthering Heights	Bronte, C	12 Feb	11 Mar
1001	Edgar Codd	12 High Street, Scholar	01234 985443	F1109	To Kill a Mocking-bird	Lee, H	14 Jan	13 Feb

As you can see, all the fields about the borrower (Borrower No, Name, Address...) have to be inserted for each book the lender loans, and the same is also true of the book details each time it is loaned out. This is unnecessary duplication, and can cause problems:

- it is time consuming for the database operator (this is a simplified example - a real borrower may need many fields);
- mistakes can be made (can you see any in the table above?);
- what if the borrower leaves the library? The General Data Protection Regulation would mean the library would have to remove the borrower's details from the database - how would this work if they have loaned hundreds of books!
- what if you want to add book details to the library database - ideally you would need someone to borrow the book, so the record is complete.

Let's take a look at a relational version of this database now:

Borrower:

BorrowerNo	BorrowerName	BorrowerAddress	BorrowerPhone
1001	Edgar Codd	12 High Street, Scholarville	01234 985443
1002	Raymond Boyce	6 Castle View, Scholarville	01234 983458

Book:

BookNo	Title	Author
F1301	Wuthering Heights	Bronte, C
F1109	To Kill a Mockingbird	Lee, H

Loan:

LoanID	BorrowerNo*	BookNo*	LoanedOut	LoanedOut
10292	1001	F1301	12 Jan	11 Feb
10293	1002	F1301	12 Feb	11 Mar
10294	1001	F1109	14 Jan	13 Feb

As you can see now:

- all the information about books is stored only once;
- all the information about borrowers is stored only once;
- all the information about loans is stored only once.

Each loan is linked (related) by the *BookNo* and *BorrowerNo* to the record in each table. This solves all of the above problems (bar the mistake problem; however, at least if an operator enters the wrong value for some book or borrower data, it only needs to be changed once and all loan records will be correct).

In the relational model, tables of data are valid as long as they meet the following rules.

1. Every table in a database must have a unique name.
2. Every column in a table must have a unique name within that table.
3. All entries in a column must be of the same kind.
4. The ordering of columns in a table is not significant.
5. Each row in a table must be unique. Duplicate rows are not allowed in a table.
6. The order of the rows is not important in the table.
7. Each **cell** in the table (the intersection of a column and a row) must contain only one value. Multiple entries are not allowed in a cell.

Columns and column characteristics

Each column in a table must have a unique name. Two or more tables within the same database may have columns with the same name.

When the same column name appears in more than one table and tables that contain that column are referred to at the same time then we use the table name to distinguish between the two columns. For example, we would refer to the *BorrowerNo* column in the above two relations as:

Borrower.BorrowerNo and *Loan.BorrowerNo*

Each column in a table must draw the data that it contains from the same **domain**. This means that each column must contain data of the same nature. This is easier to understand if we look at this incorrect snapshot of part of the borrower table from the library example.

Table 2.4: Column from student table

BorrowName
Sally Burton
John Low
0821 928192
Sandy Ogston
Brian Wilson
Sally Smith

In this example, the column is intended to store the borrower name. However, the data entered for the third borrower is a phone number. Clearly, the data in this column is not entirely from same domain (i.e. not all borrower names) and, therefore, is invalid.

Rows and row characteristics

Each row in a table must be unique. This means that the combination of values in the columns of one row cannot be found elsewhere in the table. Again, this is much easier to understand if we look at an example from the library database:

Table 2.5: Duplicate rows from the Borrower table

Borrower No	Borrower Name	Borrower Address	Borrower Phone
1001	Edgar Codd	12 High Street, Scholar	01234 985443
1002	Raymond Boyce	6 Castle View, Scholar	01234 983458
1003	Safra Catz	92a Viewfield, Scholar	01929 667818
1001	Edgar Codd	12 High Street, Scholar	01234 985443

Here we see four rows from an incorrect snapshot of the Borrower table. Each cell (row/column intersection) of the first and fourth rows contains the same data. In fact, the fourth row is an exact duplicate of the first. The fourth row is not required because the data it stores is already held in the first row. If the fourth row was held in the table then the table would be invalid because each row would not then be unique. Now look at this other example:

Table 2.6: Correct rows from the Borrower table

Borrower No	Borrower Name	Borrower Address	Borrower Phone
1001	Edgar Codd	12 High Street, Scholar	01234 985443
1002	Raymond Boyce	6 Castle View, Scholar	01234 983458
1003	Safra Catz	92a Viewfield, Scholar	01929 667818
1004	Edgar Codd	12 High Street, Scholar	01234 985443

Your first reaction when you look at this sample of rows is that it is invalid. However, the values for *BorrowerNo* for rows one and four are different and, therefore, each row is unique. This example tells us there are two different people called 'Edgar Codd' at the same address with the same phone number!

Cells

Each cell in a table must contain only one piece of data. A cell is the point where a column and a row meet.

Figure 2.2: Example of a cell in a table

LoanID	BorrowerNo	BookNo	LoanedOut	DueBack
10292	1001	F1301	12 Jan	11 Feb
10293	1002	F1301	12 Feb	11 Mar
10294	1001	F1109	14 Jan	13 Feb

The value of the cell in Figure 2.6 is 'Susan'. Every cell in the above table contains only one value. Multi-value cells are not allowed so the following table would be invalid.

Table 2.7: A table with invalid rows

Borrower No	Borrower Name	Borrower Address	Borrower Phone	Book No	Title	Author	Loaned Out	Due Back
1001	Edgar Codd	12 High Street, Scholar	01234 985443	F1301 F1109	Wuthering Heights To Kill a Mocking-bird	Bronte, C Lee, H	12 Jan 14 Jan	11 Feb 13 Feb
1002	Raymond Boyce	6 Castle View, Scholar	01234 983458	F1301 F1109	Wuthering Heights To Kill a Mocking-bird	Bronte, C Lee, H	12 Feb 14 Feb	11 Mar 14 Mar

The preceding example has multiple values in cells in the *BookNo*, *Title*, *Author*, *LoanedOut* and *DueBack* columns. These multi-valued cells are not permitted in a relational database. Only single-valued cells are allowed.

Quiz: Introduction

Go online



Q6: Describe two features of a column in a relational database.

.....

Q7: Explain the difference between a flat file database and a relational database.

.....

Q8: Are multi-valued cells allowed in a relational database?

- a) Yes
- b) No

2.3 Relationships

In addition to the one-to-many relationship you are familiar with from National 5, there are two other types of relationship which can exist between entities.

The type of a relationship is called the **cardinality** of the relationship. In an examination, you may be asked to identify the cardinality of a relationship.

There are three cardinalities.

1. One-to-many
2. One-to-one
3. Many-to-many

You should already be able to describe and use one-to-many.

2.3.1 One-to-one

A one-to-one relationship specifies that for one entity there is only one other corresponding entity.

Example Students in Lochussie High School are each allocated a locker to use during their time at school. This locker is not shared and can only be used by the pupil to whom it has been allocated.

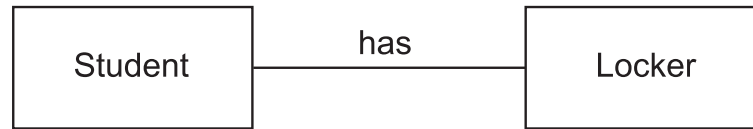
StudentID	StudentName	RegisterClass		LockerNum	LockerType
43781	Jennifer Brown	4A	_____	232C	Vanguard
57842	Amir Anderson	3E	_____	234C	Guardian
23423	Scott Andrew	6F	_____	728D	Athletic
23426	Wendy Alexander	2F	_____	483B	Vanguard

One-to-one relationship

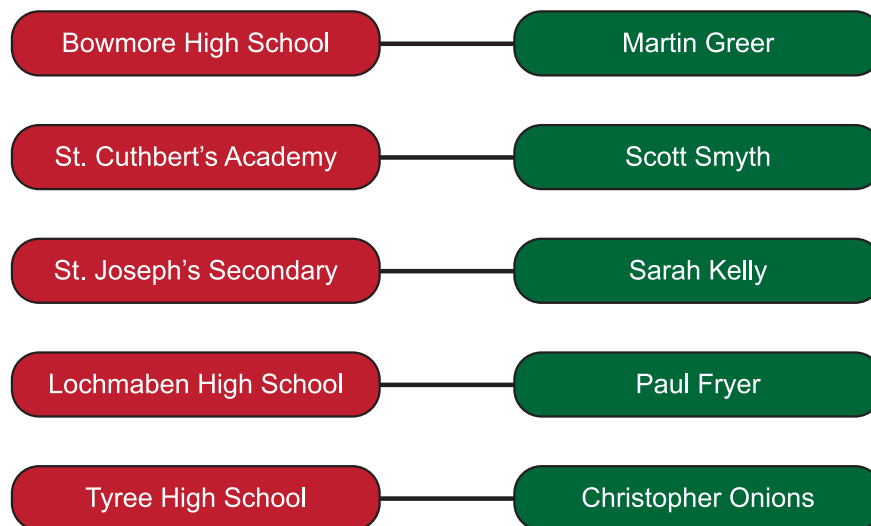
For every student there is one locker and for every locker there is one student. It is also acceptable for a student to not have a locker or for a locker to not be allocated.

In an ERD (entity relationship diagram) this is represented as 1:1.

Figure 2.3: One-to-one ERD



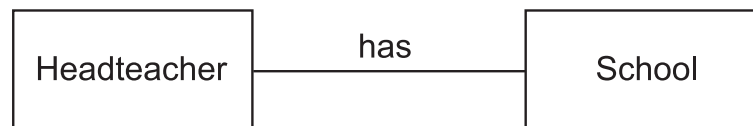
Example Each local secondary school has a head teacher, and each head teacher is head teacher at only one school.



One-to-one relationship

In an ERD (entity relationship diagram) this is represented as 1:1.

Figure 2.4: One-to-one relationship



2.3.2 Many-to-many

With many-to-many relationships, either side of the relationship can have none, one or many links.

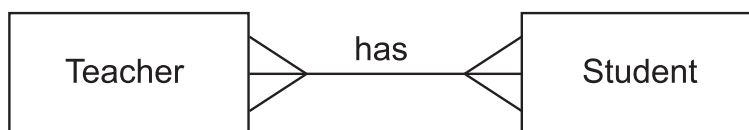
Example Teachers and students in a school are being recorded using an ERD. A teacher can have many students and a student can have many teachers.

TeacherID	TeacherName	StudentID	StudentName
92/71628	Carol Keller	0021	Amber Rose
93/01891	Jo Andrews	0145	Megan Jones
78/92387	Kylie Scott	0212	Vanessa Heart
87/23818	Carl West	0895	Adele Brooker
89/20832	Chris Hamilton	0992	Iona Barr
93/09230	Star West	2616	Louise White
93/10931	Adele Booker	4281	Ken White
05/89102	Iona McCrae	5298	Rohan Welsh
07/98219	Louise Ludd	7291	Kevin Welsh
07/98229	Kim White	8102	Mary Jones
08/98239	Madia Smith	9182	Isla MacRae

An example of a many-to-many relationship

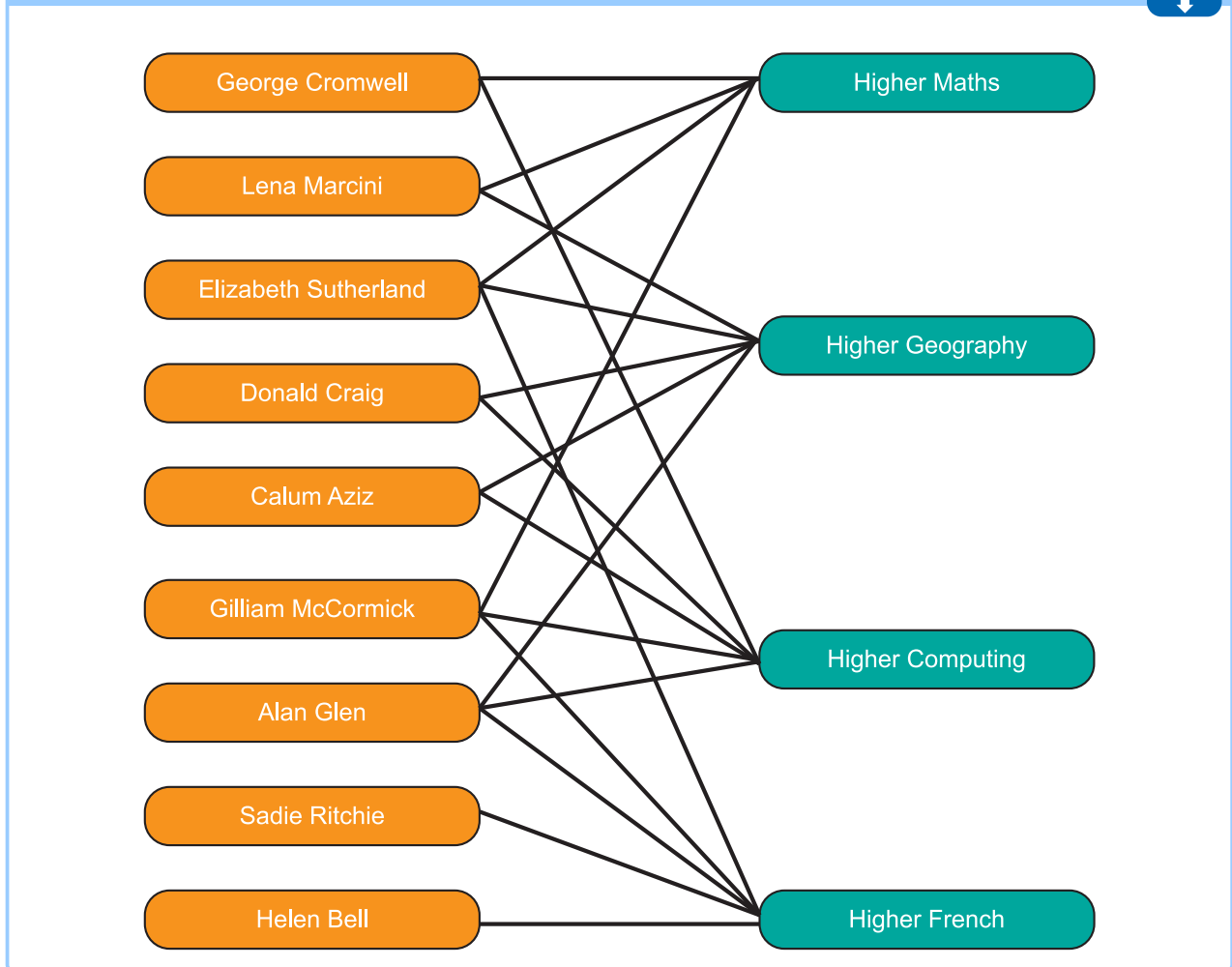
In an ERD (entity relationship diagram) this is represented as M:N.

Figure 2.5: Many-to-many relationship

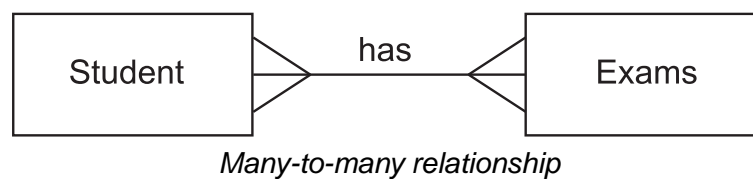


Activity: Many-to-many (1)

Go online



Here a student sits as many as four exams and each exam is sat by as many as nine students. This can be represented in an ERD as a many-to-many relationship.



Activity: Many-to-many (2)

Go online

**Q9:** Select the relationship between each pair of entities from the options provided.

Entity 1	Entity 2	Relationship
School	Head Teacher	
Teacher	Course	
Classroom	Desk	
School	Swimming pool	
Pupil	Exam	

Options: 1-to-1, 1-to-many, many-to-many.**Quiz: Relationships**

Go online

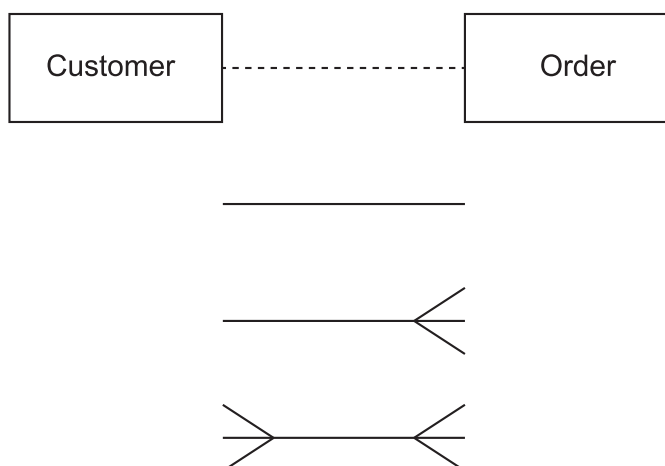
**Q10:** Explain each of these terms.

1. Entity
2. Attribute
3. Relationship

.....

Q11: An online ordering system needs to be designed. It will always have at least one customer and each customer may place multiple orders.

Complete the following ERD to describe such a system by selecting the correct relationship symbol between the entities:



2.4 Entity-occurrence diagrams

The first step in constructing a model of any system is to examine the entities and 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.

Creating an E/R occurrence diagram

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

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 BorrowerNo, 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 E/R occurrence diagram can be constructed.

Table 2.8: Primary key values

BorrowerID	LoanID
1001	10292
1002	10293
	10294

In the preceding table, each primary value from the two sample Borrowers from the Borrower entity and the Loan table have been listed.

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 borrower '1001' in the Borrower data in the sample, and look at the value for LoanID. On the E/R occurrence diagram we draw a line between the BorrowerID value and the LoanID value.

Table 2.9: Starting to create an entity occurrence diagram

BorrowerID	LoanID
1001	10292
1002	10293
	10294

This line indicates one 'link' between the Borrower entity set and the Loan entity set. To complete

the E/R 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.

Table 2.10: Entity occurrence diagram

BorrowerID		LoanID
1001	—————	10292
1002	————— \	10293
		10294

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 E/R 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 and many-to-many. Each of the following examples demonstrates the pattern that appears for each of the relationships.

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 and each employee has a unique payroll number. Following entity-relationship occurrence diagram was created:

Table 2.11: Entity Occurrence Diagram, one-to-one relationship

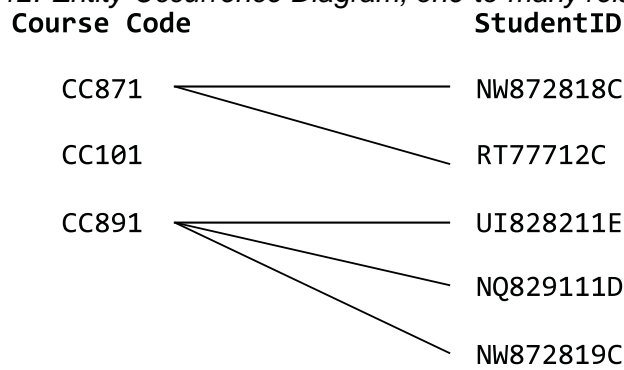
Computer Name		Payroll Number
Paris	—————	030/829112
Washington	—————	040/881929
London	—————	040/546781
Geneva	—————	030/112000
Edinburgh	—————	050/444878

One Computer Name has one Payroll Number and one Payroll Number has one Computer Name. This is, therefore, an example of a one-to-one relationship.

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 following entity-relationship occurrence diagram was created:

Table 2.12: Entity Occurrence Diagram, one-to-many relationship



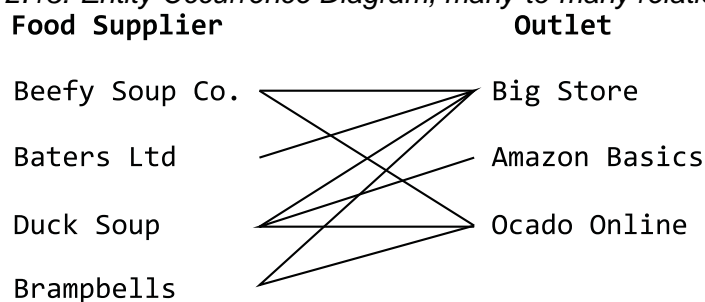
Each course has more than one student and each student has one course. This is, therefore, a one-to-many relationship.

Entity Occurrence Diagram - Many-To-Many Relationship

Food suppliers have numerous outlets and outlets have numerous food suppliers. So, for example, Beefy Soup Co. sell soup to Big Store, AmazonBasics and Ocado Online. Big Store buys soup from Beefy Soup Co., Baters Ltd., Duck Soup and Brampbells.

Following entity-relationship occurrence diagram was created:

Table 2.13: Entity Occurrence Diagram, many-to-many relationship



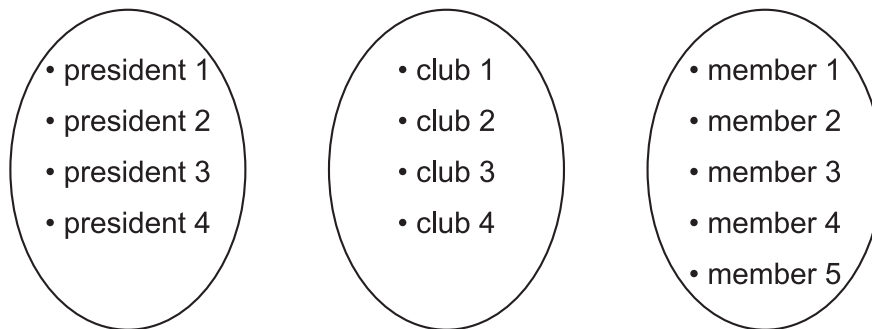
Each food supplier has one or more outlets and each outlet has one or more food suppliers. This is, therefore, a many-to-many relationship.

More on 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.

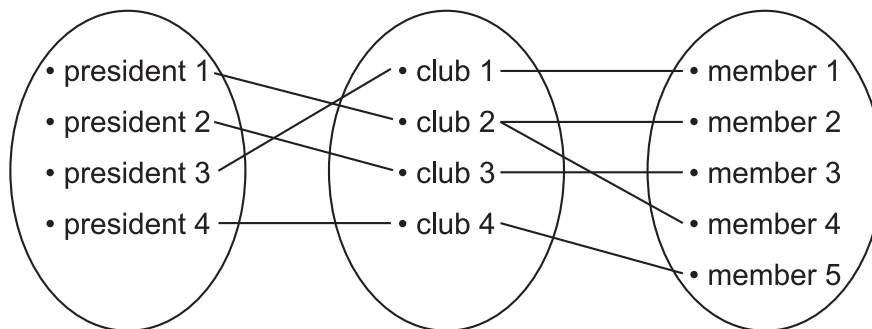
For example: You are told that each rugby club has one president and that each president belongs to only one rugby club. Each rugby club has many members. A member can only belong to one club.

Figure 2.6: Entity Occurrence Diagram - dummy data



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.

Figure 2.7: Entity Occurrence Diagram - dummy data - solution



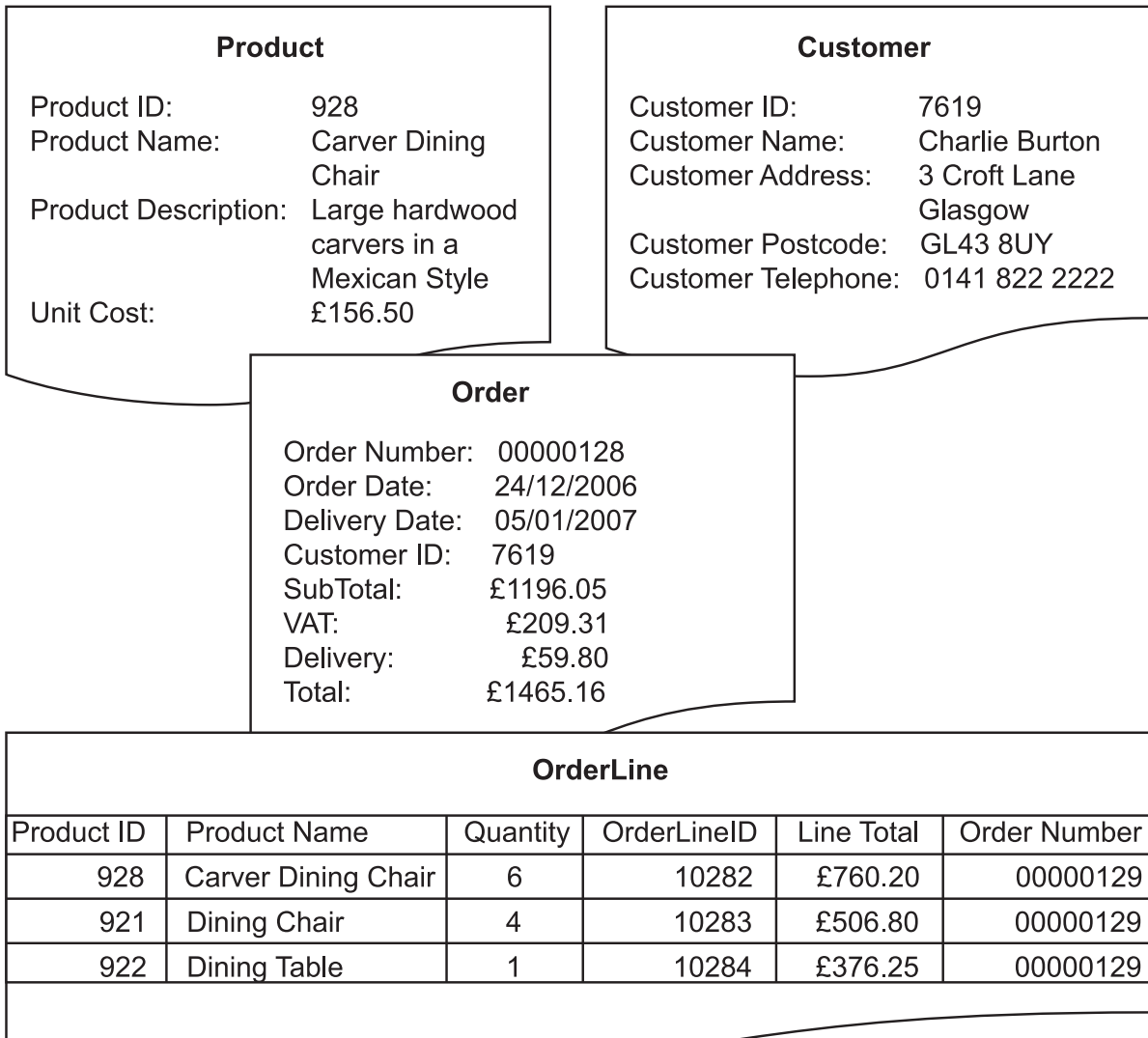
This shows that the relationship between President and Club is 1:1 (one-to-one) and that the relationship between club and member is 1:M (one-to-many).



Activity: Entity Occurrence Diagrams

An online ordering system makes use of four entities. These are *Customer*, *Order*, *OrderLine* and *Product*.

Three example orders with the associated data are shown.



Order 1

Product

Product ID: 921
 Product Name: Dining Chair
 Product Description: Large pinewood chair
 Unit Cost: £126.70

Customer

Customer ID: 7900
 Customer Name: David Proudfoot
 Customer Address: 10 The Mews
 Glasgow
 Customer Postcode: GL82 7YY
 Customer Telephone: 0141 822 3333

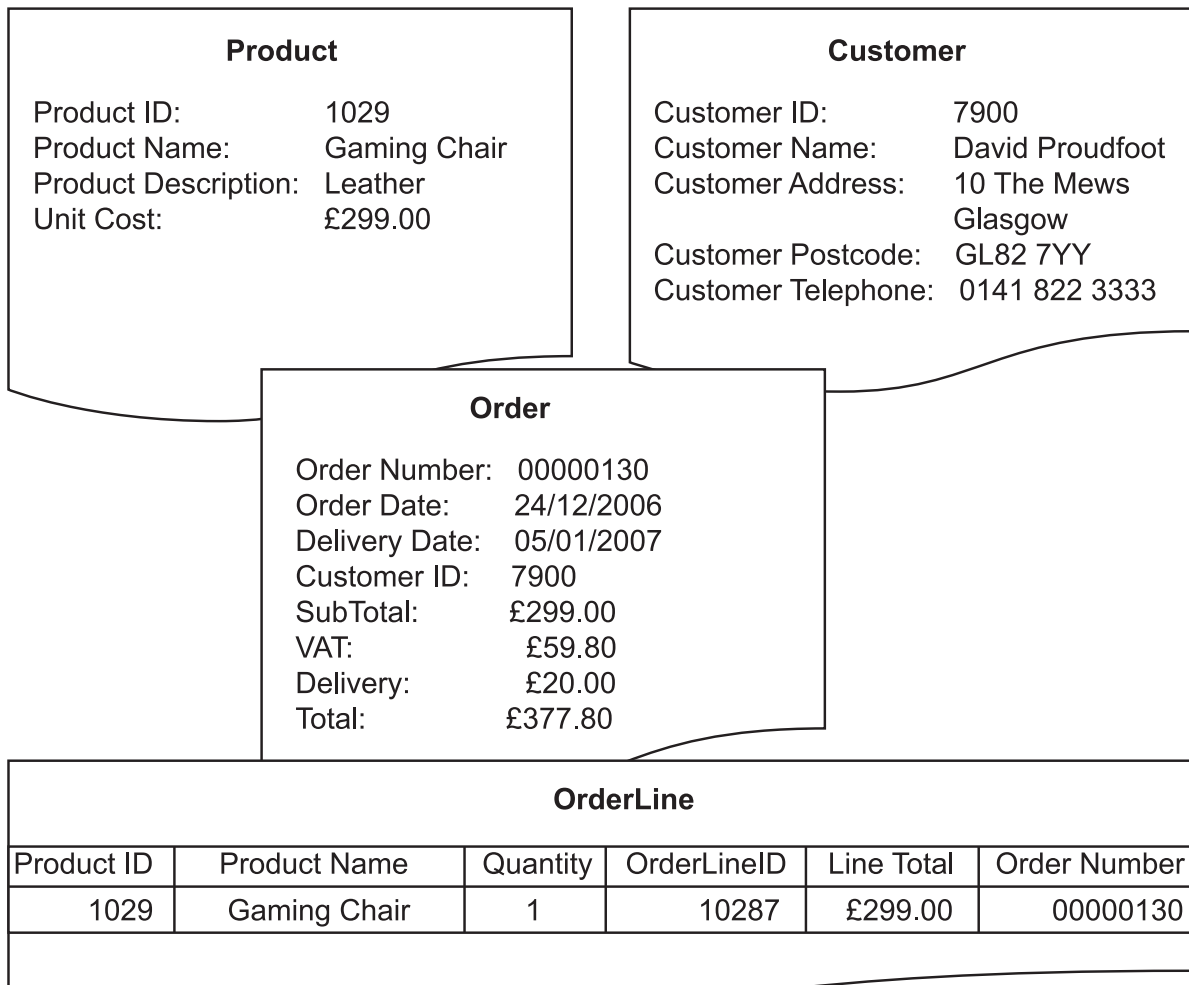
Order

Order Number: 00000129
 Order Date: 24/12/2006
 Delivery Date: 05/01/2007
 Customer ID: 7900
 SubTotal: £1136.45
 VAT: £227.29
 Delivery: £59.80
 Total: £1423.54

OrderLine

Product ID	Product Name	Quantity	OrderLineID	Line Total	Order Number
928	Carver Dining Chair	6	10285	£760.20	00000129
922	Dining Table	1	10286	£376.25	00000129

Order 2



Order 3

Q12: Create an entity occurrence diagram from the information above. You will be required to identify suitable primary keys and their values and link these to understand the relationships between the four entities.

Create your diagram and then list the relationships between each of the entities.

2.5 Compound keys

In some relations there is no one column that can act as a primary key. However, each row is unique because of the data values in the columns.

Example

Suit	Value	Number of times played
Hearts	Ace	5
Diamonds	2	4
Spades	3	3
Clubs	5	8
Clubs	Jack	5

In the example, no single column contains a set of unique values.

- Cards can be any one of four suits so the suit is not unique.
- There are four cards in each deck with the same value (four aces, four kings etc.).
- The *Number of times played* attribute won't be unique as it is a count of the number of times the card is played and cards will quite likely be played the same number of times.

This means that none of the columns can serve as a primary key on its own, but what if we used two columns together? Using the *Suit* and *Value* columns together as a key would give a unique value for each card.

This combination of columns is our primary key. This is an example of a compound key.

When writing the attributes, both would be underlined:

Suit

Value

Number of times played

More than one attribute is underlined; therefore you know that this is a compound key.

It is often the case that several columns will have to be used to create a compound primary key. Let's look at this example of people coming to visit properties to buy.

<u>property_id</u>	<u>client_name</u>	<u>potential_buyer_no</u>	<u>potential_buyer_name</u>	<u>date_of_viewing</u>
P221	Smith	282	Jones	17.07.2018
P221	Smith	982	Perkins	19.07.2018
P221	Smith	983	Patel	29.07.2018
P221	Smith	282	Jones	29.07.2018
P226	Parker	282	Jones	29.07.2018
P226	Parker	225	Mitchell	23.08.2018

There are no unique columns:

- because a property can be viewed more than once, *property_id* and *client_name* are repeated;
- because one potential buyer can view one or more properties on several different occasions, *potential_buyer_no* and *potential_buyer_name* can appear more than once;
- because any house can be visited more than once on the same day, the *date_of_viewing* is repeated. Note that the same person cannot view the same house twice on the same day.

This means that none of the columns can serve as a primary key on its own. However, a combination of *property_id*, *potential_buyer_no* and *date_of_viewing* is unique. This is because the combination of values in each of these three columns is never the same.

Quiz: Compound keys

Go online



Q13: Examine the following table and select a suitable compound primary key.

competitionID	position	points	gamerID
1	1	1500	9023
1	2	7000	1873
1	3	1000	1009
2	1	12000	0293
2	2	6000	3742
2	3	1500	1645
3	1	30000	1873
3	2	22000	9834
3	3	15000	1873
3	4	5000	1842

2.6 Entity relationship diagrams

From your studies at National 5, you will be familiar with creating ERDs with two entities and a single relationship. The problems that you will encounter at Higher level are more complex and will typically involve three or more entities and multiple relationships of different cardinalities (i.e. one-to-one, one-to-many, and many-to-many).

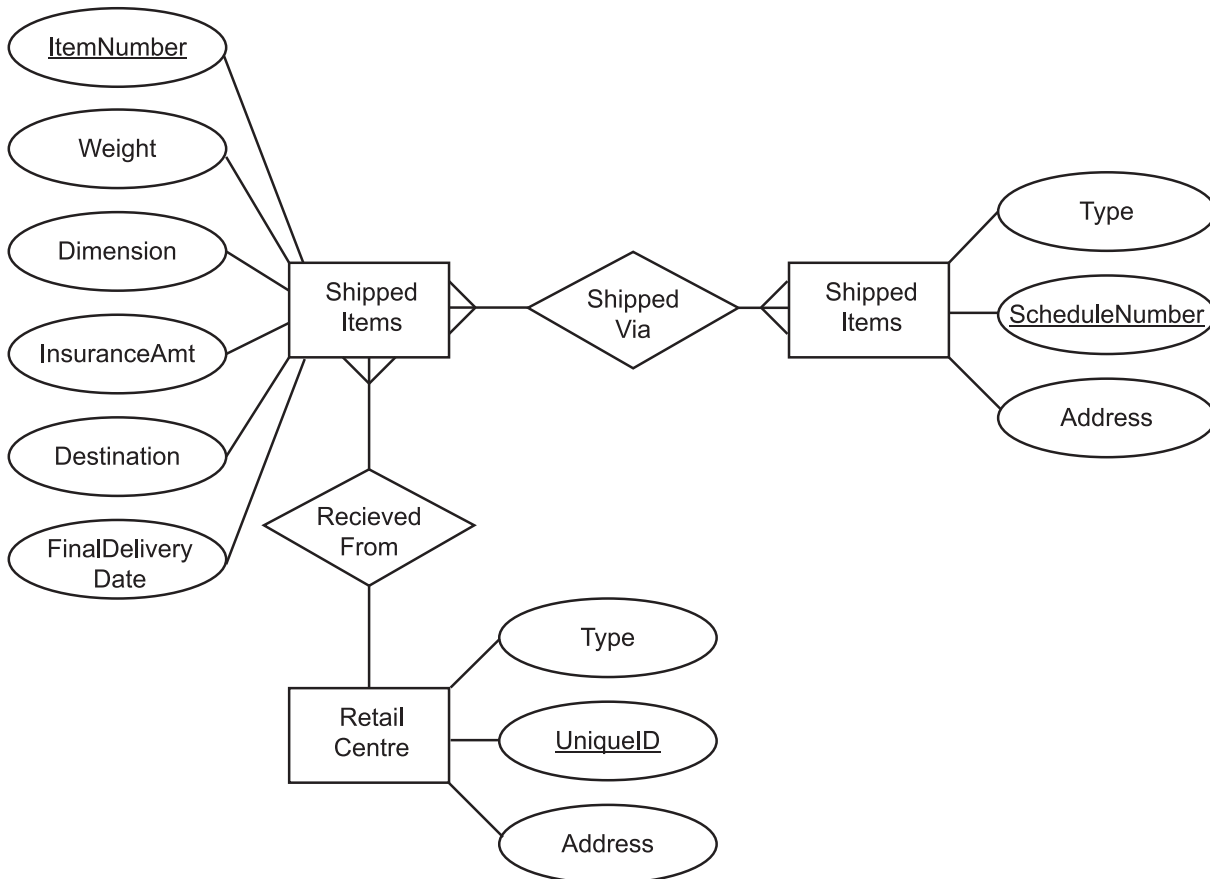
Example

ParcelMe is a delivery company with a company-wide database system. Shipped items are the heart of the ParcelMe product tracking system. Shipped items can be characterised by item number (unique), weight, dimensions, insurance amount, destination, and final delivery


date. Shipped items are received into the ParcelMe system at a Retail Center. Retail Centers have a type, uniqueID, and address. Shipped items make their way to their destination via one or more standard ParcelMe transportation events (i.e., flights, vehicle deliveries). These transportation events consist of a unique scheduleNumber, a type (e.g, flight, vehicle), and a deliveryRoute.

The ERD for the example system would be:

Figure 2.8: ParcelMe Entity Relationship Diagram



In the diagram, many *Shipped Items* are shipped through many *Transportation Events* and one *Retail Centre* receives many *Shipped Items*.

Quiz: Entity relationship diagrams 

Create ERDs for the following scenarios.

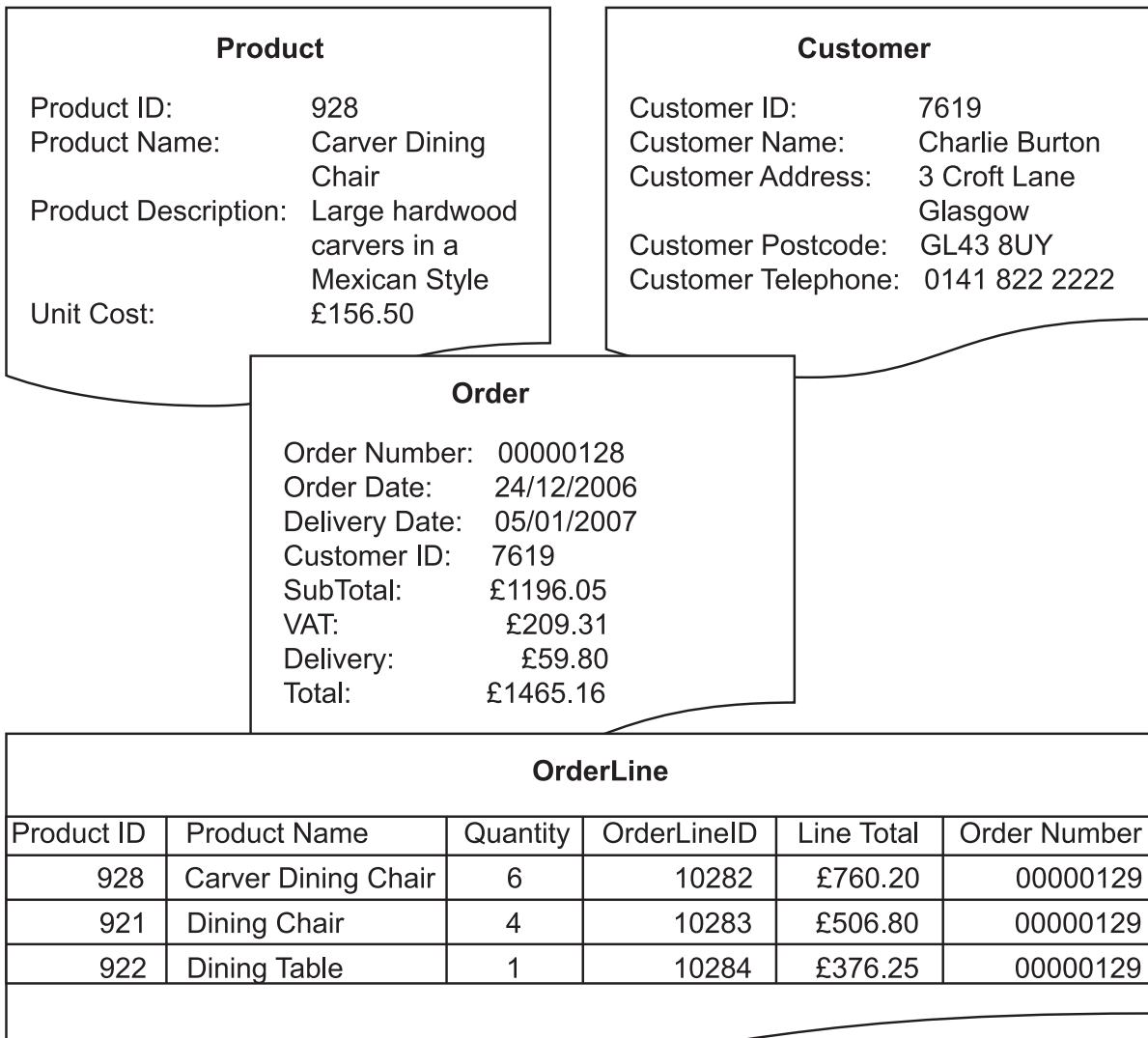
Q14: A company stores information about employees, departments and the children of employees. The employees details held are employeeID, salary and phone number. Each department has a department number and a budget. Children of employees uses a ChildID, Child Name and Age. Each employee works in only one department and assume that only one parent works at the company.

.....

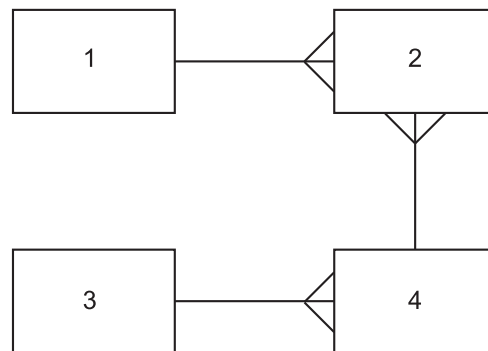
Q15: A computer game has teams of players. Each player has a playerID, name and a skill level. Each player has access to an inventory. This inventory includes items to use in the game. Each item has a unique itemID, a name and a cost. Teams play matches against the computer and, for each match, there is a unique MatchID, a score, a match level and a match map. Teams have a team name and teamID. The inventory has a compound primary key of playerID and itemID.

.....

Q16: Here is an example of information from a database system.



Complete this entity relationship diagram using the labels provided.



- a) customer
- b) order
- c) orderline
- d) product

2.7 Solutions to queries

At Higher level you will be expected to design solutions to a query making use of the following:

- tables and queries;
- fields;
- search criteria;
- sort order;
- calculations;
- **grouping.**

Tables and queries

From National 5 you will be aware of using tables as the source of the data for your queries. When a query runs on tables it will produce an answer table. Answer tables can also be used as the basis for a query. You can perform a query on the result of a query!

Example The Federation of Esports has a relational database that consists of three linked tables, storing data on solo players, tournaments and tournament results.

Extracts from these tables are shown as follows.

tournament:

<u>tournamentID</u>	country	place	eventDate
1	UK	Liverpool	13/05/2017
2	France	Paris	29/08/2017
3	USA	New York	08/09/2017
4	Germany	Berlin	12/03/2018
...

player:

<u>playerID</u>	forename	surname	rating	playercountry
1645	Barry	Jones	1756	USA
1873	Jenny	McKay	1260	Australia
2093	Ahmed	Ali	1934	UK
...

result:

<u>tournamentID</u>	position	prizemoney	playerID*
1	1	15000	1645
1	2	7000	1873
1	3	1000	2093
2	1	12000	1873
2	2	6000	2093
2	3	1500	2093
3	1	30000	1873
3	2	22000	2093
3	3	15000	0293
...

A query is designed to show all the results for "Jenny Mckay"

Query1

Tables(s) and/or query	player, result
Fields and/or calculations	position, prizemoney
Search criteria	forename = "Jenny", surname = "Mckay", player.playerid = result.playerid
Grouping	
Sort order	

The answer table generated by this query would then be used to find the largest sum of prize money that Jenny Mckay won when she finished in first in a tournament (position 1).

Query2

Tables(s) and/or query	[All results for Jenny Mckay]
Fields and/or calculations	MAX(prizemoney)
Search criteria	position = 1
Grouping	
Sort order	

This query acts on the results of Query 1 to produce a second answer table where the position = 1.

Fields

Fields in a database table are the columns of data. Each column has a unique name within the table. When designing a query you need to consider the fields that will appear in the answer table. There may be other fields that you use for search criteria, grouping and sort order but these are the fields (the columns) which are displayed in the answer table generated by the query.

Search criteria

Search criteria are field values that are used to reduce the scope of a query and select the required data. In the example above the search criteria for Query 1 are forename = "Jenny", surname = "Mckay", player.playerid = result.playerid. In an SQL query these would be joined together using an AND operator (or some other operator depending on the data that the query should generate). The criteria 'player.playerid = result.playerid' is the criteria that is used to link the two tables in the query.

We will cover more on the detail of Search Criteria in the next topic.

Sort order

The answer table produced by a query can be sorted based on values held in the columns. Answer tables can be sorted on one, two or more fields in ascending or descending orders.

Example

The following answer table is sorted on: *year descending, comic_id ascending, hero_name ascending*

Year	comic_id	hero_name
2017	6171	Aqua matrix
2017	6171	Super mole
2017	8109	Jules Volt
2017	8109	Rocket
2016	5261	Bear Meteor
2016	5261	No Capes
2016	5278	X-field
2015	9188	Superflea
2015	9188	Transfirma

Calculations

Calculations can be carried out within a query. These calculations can use field values to generate new values that can be displayed in the answer table.

Example

A query uses the data from the following table:

OrderID	ItemID	Qty	BasePrice
00928	1726	5	9.99
00928	1727	2	17.50
00929	1726	1	29.00
00930	2817	20	5.75

We can design a calculation that can be carried out on this table as part of the query. We can calculate the *Qty* multiple by the *BasePrice*. We can also give this calculation a name which will be used in the answer table.

Tables(s) and/or calculations	(Qty * Base Price) as LineTotal
--------------------------------------	---------------------------------

The answer table would then be:

LineTotal
49.95
35
29

It is also possible to use functions to carry out calculations. You need to know the following functions for higher level.

- **Minimum**
Find the minimum of the values available. Use the notation MIN(fieldname) in your design.
- **Maximum**
Find the maximum of the values available. Use the notation MAX(fieldname) in your design.
- **Average**
Calculates the average of all the values available. Use the notation AVG(fieldname) in your design.
- **Sum**
Adds together all the values available. Use the notation SUM(fieldname).
- **Count**
Counts the number of occurrences of a field. Use the notation COUNT(fieldname).

Grouping

Grouping gathers together the rows of a table or answer table based on a column or columns with the same value or values.

Examples

1.

The result table from an earlier example is given here.

result:

<u>tournamentID</u>	position	prizemoney	playerID*
1	1	15000	1645
1	2	7000	1873
1	3	1000	2093
2	1	12000	1873
2	2	6000	2093
2	3	1500	2093
3	1	30000	1873
3	2	22000	2093
3	3	15000	0293
...

A query design with grouping is shown.

Tables(s) and/or query	tournament
Fields and/or calculations	tournamentid, SUM(prizemoney) as PrizeFund
Search criteria	
Grouping	BY tournamentid
Sort order	

This will total all the *prizemoney* values for each tournament value. The answer table generated will be:

tournamentID	PrizeFund
1	23000
2	19500
3	67000

.....

2.

This a table of customers for a company.

CustomerID	CustomerName	Address	City	PostCode	Country
12	Island Trading	Garden House Crowther Way	Cowes	PO31 7PJ	UK
13	Königlich Essen	Maubelstr. 90	Brandenburg	14776	Germany
14	La corne d'abondance	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 Store	12 Orchestra Terrace	Walla Walla	99362	USA
18	Lehmanns Marktstand	Magazinweg 7	Frankfurt a.M.	60528	Germany
19	Let's Stop N Shop	87 Polk St. Suite 5	San Francisco	94117	USA

CustomerID	CustomerName	Address	City	PostCode	Country
20	LILA-Supermercado	Carrera 52 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	3508	Venezuela
21	LINO-Delicatesses	Ave. 5 de Mayo Porlamar	I. de Margarita	4980	Venezuela
22	Lonesome Pine Restaurant	89 Chiaroscuro Rd.	Portland	97219	USA
23	Magazzini Alimentari Riuniti	Via Ludovico il Moro 22	Bergamo	24100	Italy
24	Maison Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
25	Mère Paillarde	43 rue St. Laurent	Montréal	H1J 1C3	Canada
26	Morgenstern Gesundkost	Heerstr. 22	Leipzig	04179	Germany
27	North/South	South House 300 Queensbridge	London	SW7 1RZ	UK
28	Océano Atlántico Ltda.	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
29	Old World Delicatessen	2743 Bering St.	Anchorage	99508	USA

A query design with grouping as shown here:

Tables(s) and/or query	Customer
Fields and/or calculations	Count(CustomerID), Country
Search criteria	
Grouping	BY country
Sort order	

Will produce the answer table:

COUNT(CustomerID)	Country
1	Argentina
1	Belgium
2	Canada
2	France
3	Germany
1	Italy
2	UK
4	USA
2	Venezuela

2.8 Learning points

Summary

You should now be able to:

- create and describe entity-relationship diagrams using three or more entities;
- exemplify relationships and their cardinality as one-to-one, one-to-many or many-to-many;
- use entity-occurrence diagrams to identify and present the relationship between two entities;
- describe and explain through examples, a compound key;
- describe and explain through examples, a data dictionary with three or more entities including attribute names, types, size and validation;
- design, create and explain the design of a solution to a query, making use of tables, queries, fields, search criteria, sort order, calculations and grouping.

2.9 End of topic test

End of Topic 2 test

Go online



Q17: Which of the statements is the correct definition of a primary key?

- a) An attribute that is the same for all the entities.
 - b) The longest attribute within an entity.
 - c) The shortest attribute within an entity.
 - d) An attribute that has a unique value for each entity.
-

Q18: Which of the statements is the correct definition of a foreign key?

- a) An attribute that is the primary key of another entity set.
 - b) The attribute field within an entity.
 - c) The most important attribute in an entity.
 - d) An attribute in a foreign language.
-

Q19: Which of these is **not** a type of relationship?

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

Q20: Grouping in a query design:

- a) display values from the same column grouped with equivalent values in another column.
 - b) calculates a summary value for each column and displays this.
 - c) gathers together the rows of a table or answer table based on a column or columns with the same value or values.
 - d) creates a relationship between two answer tables and display the result from the two tables.
-

Q21: A compound key is:

- a) a combination of a primary and foreign key to link to entity sets.
- b) a combination of more than one attribute to uniquely identify an entity.
- c) an attribute of the entity that is important and will be indexed to ensure fast access within search.
- d) is a unique identifier for an entity that consists on a single attribute.

Topic 3

Implementation

Contents

3.1	Application software	55
3.2	Introduction	59
3.2.1	Referential integrity	59
3.2.2	SQL operations: SELECT	61
3.2.3	SQL operations: INSERT	66
3.2.4	SQL operations: UPDATE	67
3.2.5	SQL operations: DELETE	70
3.3	Example database	72
3.4	SQL Wildcards	74
3.5	Table and column aliases	76
3.6	Using sub-queries	77
3.7	SQL aggregate functions (MIN, MAX, AVG, SUM, COUNT)	80
3.8	Computed values	82
3.9	GROUP BY	83
3.10	ORDER BY	85
3.11	Learning points	86
3.12	End of topic test	87

Prerequisites

From your studies at National 5 you should already know how to:

- implement relational databases with two linked tables to match a design with referential integrity;
- describe, exemplify and implement SQL operations for pre-populated relational databases, with a maximum of two linked tables using SELECT, INSERT, UPDATE, DELETE queries;
- use:
 - FROM, WHERE and ORDER BY clauses within queries;
 - WHERE clauses which make use of AND, OR, <, >, =;
 - ORDER BY clauses which make use of a maximum of two fields;
 - equi-joins to select data from two tables.

Learning objective

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

- describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables;
- use:
 - UPDATE, SELECT, DELETE and INSERT statement on more complex related tables;
 - wildcards in your queries;
 - aggregated functions MIN, MAX, AVG, SUM and COUNT to process data using queries;
 - GROUP BY clauses to gather data of similar values together;
 - ORDER BY clauses to sort answer tables on multiple fields;
- make more complex use of WHERE clauses to select data;
- use sub-queries so that the answer table generated by a previous query can be used as the basis for a further query;
- read and explain SQL statements which makes use of the above points.

3.1 Application software

You will require access to a RDBMS which supports the use of SQL commands for this Topic. There are many technologies that support the use of SQL commands.

MySQL / MariaDB

MySQL is an open-source relational database management system (RDBMS) and one of the most popular applications used to build databases on the web and other systems. MariaDB is a different version (fork) of MySQL which developers have chosen to develop in a different direction. Both these applications are high-performance RDBMSs that can be used for this course.

You can use these applications from the command line however a common method is to use an application such as phpMyAdmin or MySQLWorkBench

Figure 3.1: Design View within phpMyAdmin

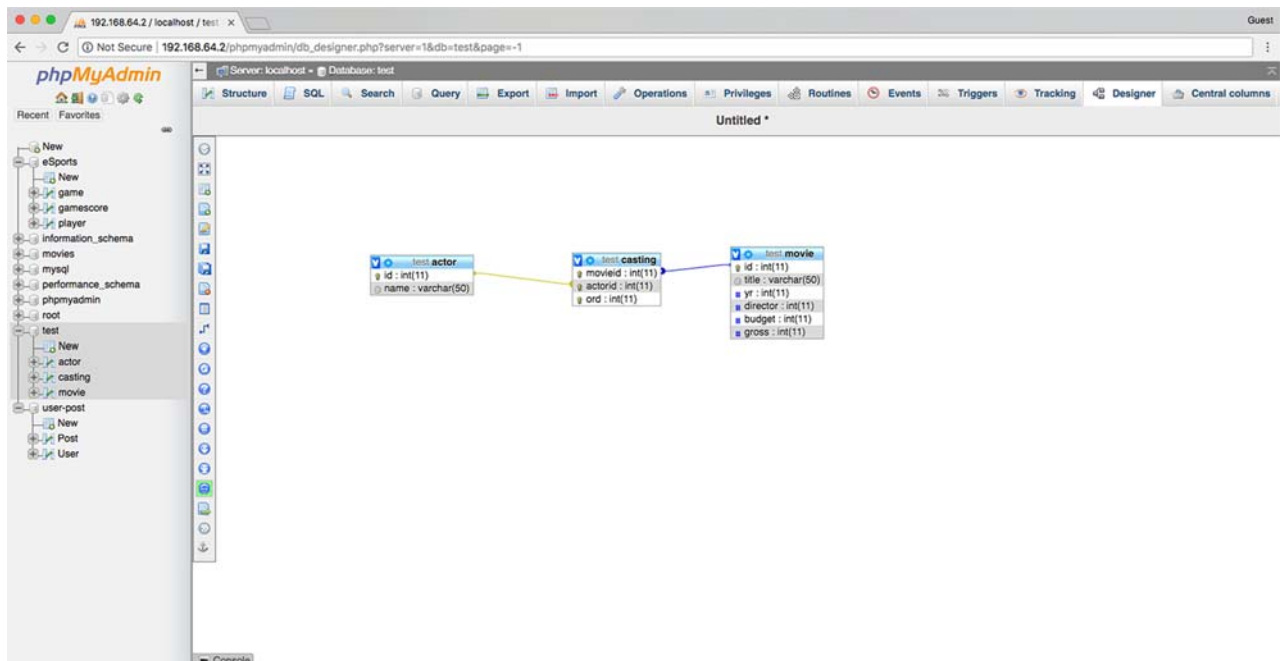


Figure 3.2: Browsing a table in phpMyAdmin

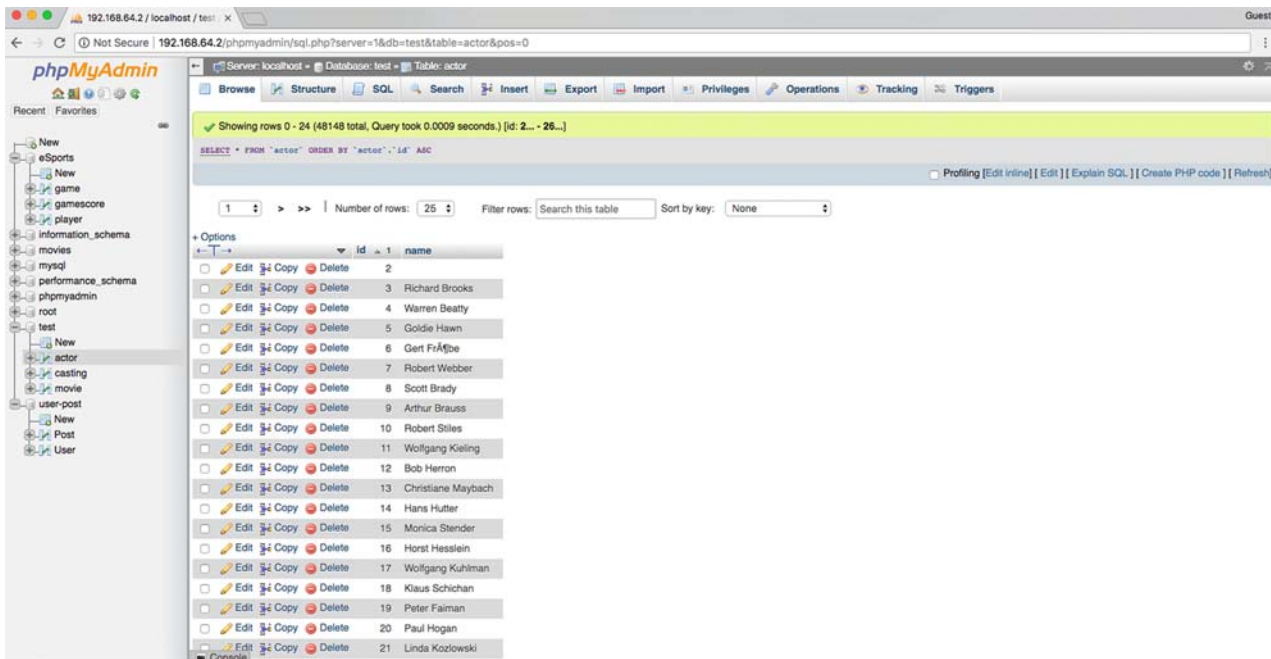


Figure 3.3: Screen from MySQL WorkBench

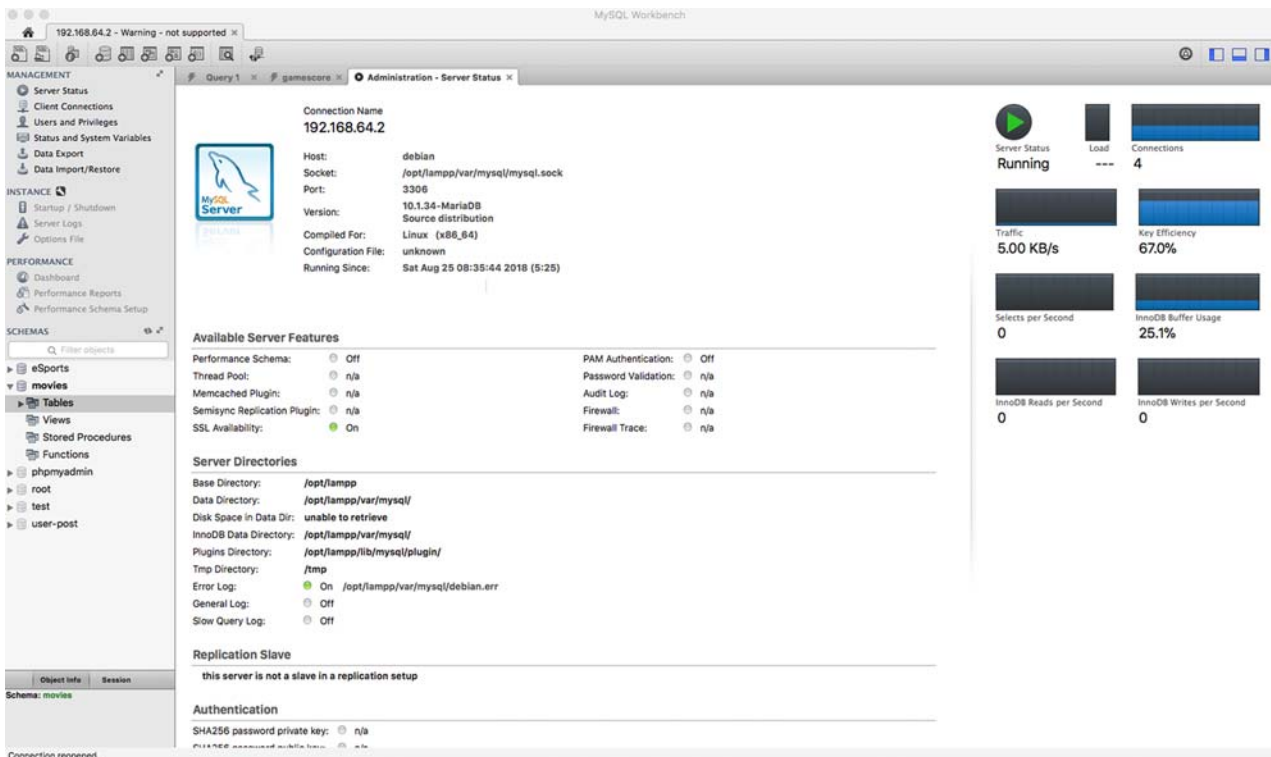
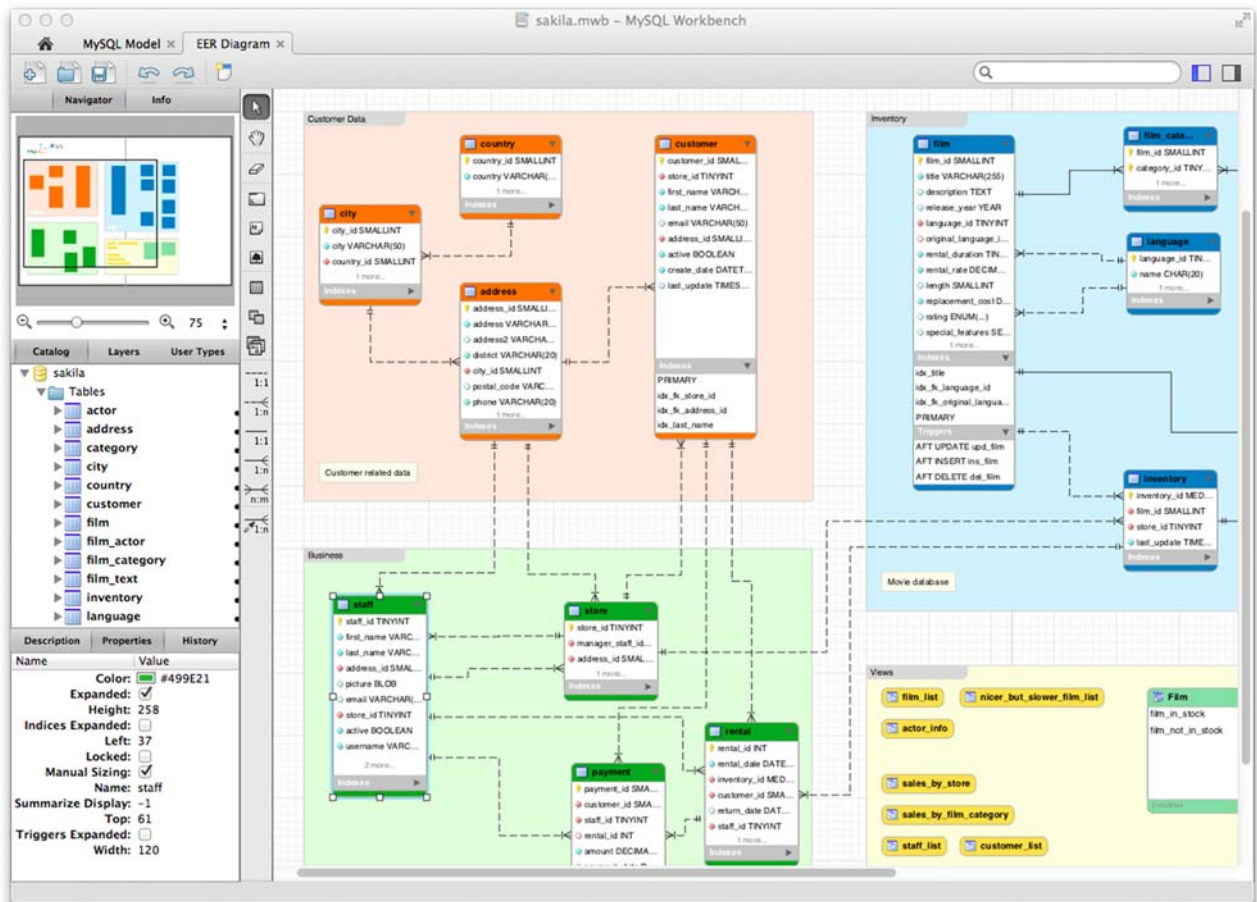


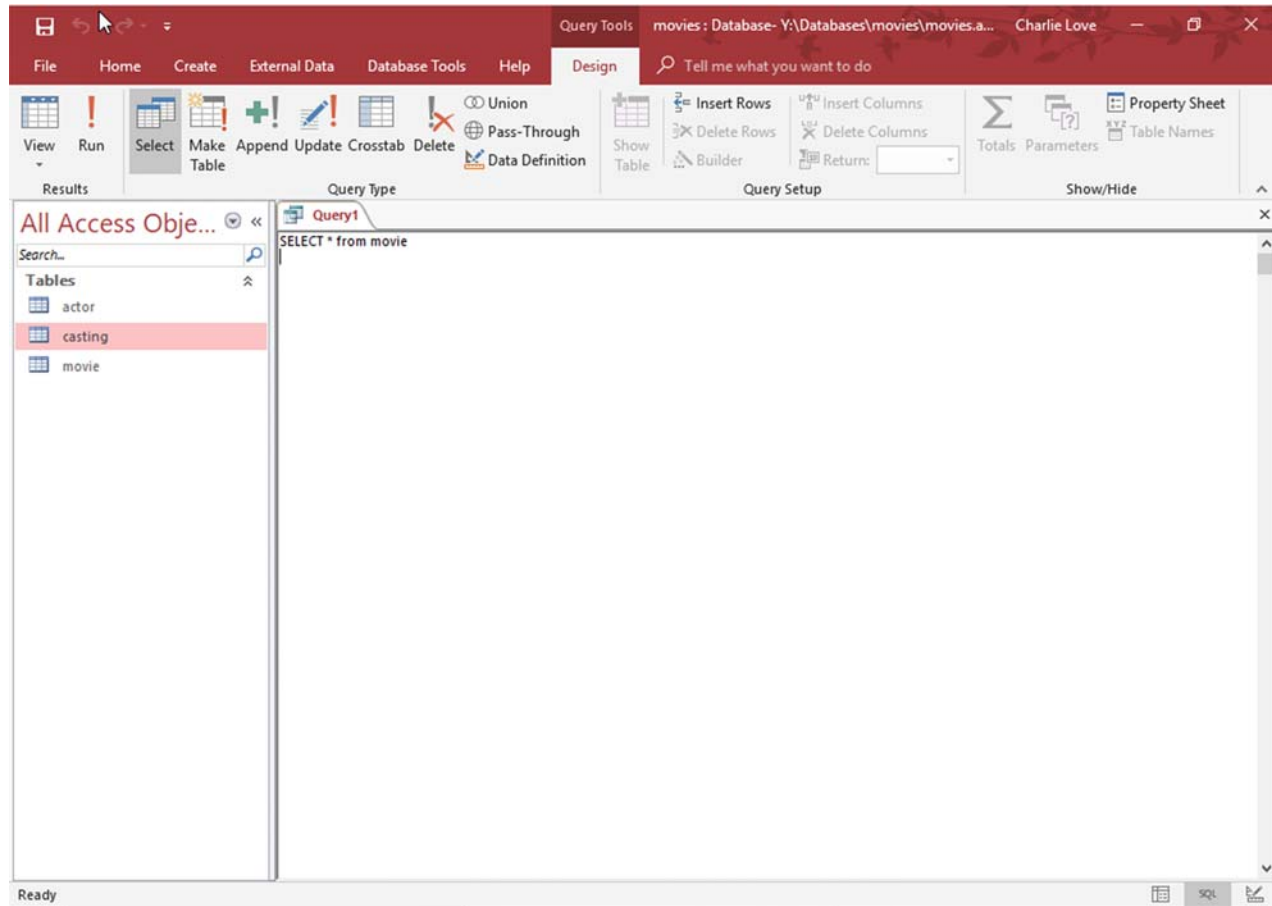
Figure 3.4: Relationships Screen from MySQL WorkBench



SQL view within Microsoft Access

Within Microsoft Access is the Query View. This view allows access to a SQL editor for you enter commands. Queries can be entered here and then run.

Figure 3.5: SQL Window within Query View in Microsoft Access



*AMP Stack

*AMP is a term used to define a combination of operating system, Apache Web Server, MySQL or MariaDB database server and PHP. The popularity of *AMP across the World Wide Web is driven by the low-cost of deployment; the components of *AMP (other than the operating system) are Open-Source applications, free software that can be used without purchasing a license.

Versions of the *AMP stack can be downloaded for Windows, MacOS X and Linux at no cost from:

- XAMMP: <https://www.apachefriends.org/>
- MAMP: <https://www.mamp.info/>
- WampServer: <http://www.wampserver.com/en/>

MySQL or MariaDB is the key database component of *AMP and dominates online databases used to provide services. There are over 10 million active installations of MySQL/Maria DB and it is the database technology that drives popular web tools such as Wordpress, Tumblr and Twitter.

If you are using phpMyAdmin then you will likely have access to an *AMP stack.

3.2 Introduction

From National 5 Computing Science you should already be familiar with queries which manipulate data from two linked tables.

These linked tables are connected by a primary / foreign key and adhere to referential integrity.

3.2.1 Referential integrity

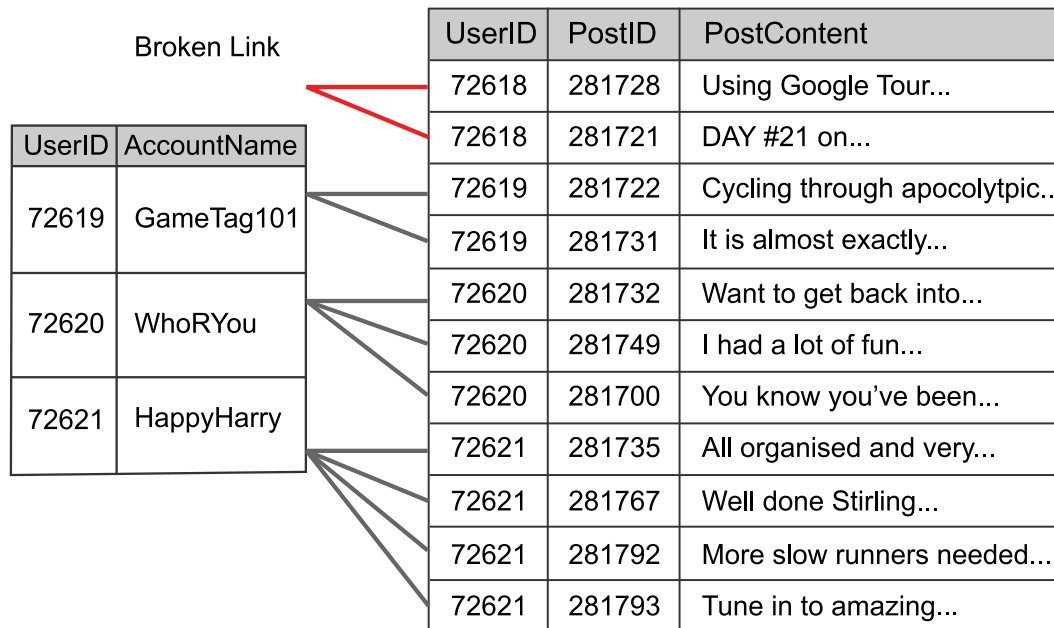
Referential integrity means that the relationship between two tables should always be consistent. A **foreign key** is a primary key value from another table.

Figure 3.6: One-to-many relationship with primary and foreign keys

User		Post		
UserID	AccountName	UserID	PostID	PostContent
72618	NKSL	72618	281728	Using Google Tour...
		72618	281721	DAY #21 on...
72619	GameTag101	72619	281722	Cycling through apocolytpic...
		72619	281731	It is almost exactly...
72620	WhoRYou	72620	281732	Want to get back into...
		72620	281749	I had a lot of fun...
72621	HappyHarry	72620	281700	You know you've been...
		72621	281735	All organised and very...
		72621	281767	Well done Stirling...
		72621	281792	More slow runners needed...
		72621	281793	Tune in to amazing...

In the example above, the UserID is the **primary key** of the *User* table and is a foreign key in the *Post* table. If the row for NKSL was removed, then the table would be as follows.

Figure 3.7: One-to-many relationship with primary and foreign keys without referential integrity

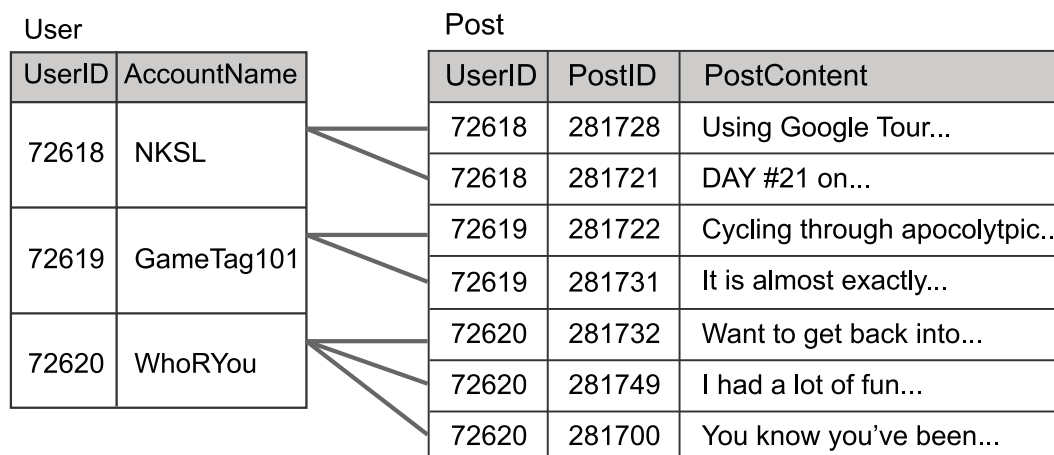


In this example, there is no primary key value for the foreign key value 72618. Deleting the row for NKSL creates inconsistent rows in the Post table. It breaks the link between the two tables.

Enforcing referential integrity means that there cannot be foreign key values that do not have a corresponding primary key value in the linked table. It prevents breaking the link between primary and foreign keys in related tables. It also prevents new rows being added in a table where the foreign key does not have a matching primary key value in the linked table.

If referential integrity is enforced then removing a row that has the primary key value, should also delete all the rows in the linked table with the related foreign key. This process is called "cascade deletion". So, deleting the row for HappyHarry in the User table will also delete all the related rows in the Post table e.g.

Figure 3.8: Cascaded deletions



In the rare occasion that there is a need to update the primary key value, then it is important that all the linked foreign keys are also updated. This process is called "cascade update". Any changes to a primary key value are also copied to all the related foreign key values. If the primary key for GamerTag101 is changed from 72619 to 89212 then all the related foreign key values are also

updated e.g.

Figure 3.9: Cascaded updates

User		Post		
UserID	AccountName	UserID	PostID	PostContent
72618	NKSL	72618	281728	Using Google Tour...
		72618	281721	DAY #21 on...
89212	GameTag101	89212	281722	Cycling through apocalyptic...
		89212	281731	It is almost exactly...
72620	WhoRYou	72620	281732	Want to get back into...
		72620	281749	I had a lot of fun...
		72620	281700	You know you've been...

3.2.2 SQL operations: SELECT

At National 5 you used SQL operations to query pre-populated databases with two linked tables.

SQL Statements

A single line of an SQL command is called a statement. These statements will consist of one or more clauses. Each clause contains a specific SQL keyword and some data that it acts upon. The following is an example of a statement using the SELECT command which locates and displays data according to the details entered.

```

1 SELECT userID, AccountName
2 FROM User
3 WHERE UserID > 70000
4 ORDER BY AccountName DESC;
```

In this example, each clause has been placed on a separate line and the SQL keywords have been capitalised. Notice that the semi-colon (;) is used to terminate the statement. There is no requirement to place a statement across multiple lines or to capitalise commands but for clarity in the following activities this is the approach that will be used.

Practical tasks using SELECT

The following activities make use of a database of Users and Posts.

Download <https://courses.scholar.hw.ac.uk/vle/asset/Downloads/H-CCMP/Course%20Downloads/C45F14CD-EA9B-6478-B678-E3ACFF7C1CE4/SELECT.zip>, extract the files and then do one of the following, depending on your database management system:

- import *user-post.sql* to an empty database to create the complete database;
- open *user-post.accdb* in Microsoft Access;
- CSV files *users.csv* and *posts.csv* are available for each database table for conversion to other platforms.

You will require an RDMS, such as MariaDB/MySQL (PhpMyAdmin) or Microsoft Access, to enter SQL commands and to run these to produce answer tables.

Activity: Using SELECT (1)



Enter the SQL: `SELECT * FROM User;`

This will display an answer table containing all columns and rows from the User table. The asterisk (*) is a wildcard that means EVERYTHING.

Q1: Write the SQL to display everything from the Post table.

Activity: Using SELECT (2)



Enter the SQL: `SELECT AccountName FROM User;`

This will display the AccountName column for all the rows from the User table.

Q2: Write the SQL to display just the UserID from the Post table.

Activity: Using SELECT (3)



Enter the SQL: `SELECT PostID, PostContent FROM Post;`

This will display the PostID and PostContent columns from the Post table.

Q3: Write the SQL to display just the UserID and PostContent from the Post table.

Activity: Using SELECT (4)



Enter the SQL: `SELECT * FROM User, Post;`

This shows the rows from each table with every row from the corresponding table. Without a JOIN the query doesn't work.

The type of join you know from National 5 is EQUI-JOIN which is an equal join between the tables. The equi-join is the "link" between the primary and foreign keys. In this case, the primary key `User.UserID` and foreign key `Post.UserID`.

Q4: Write the SQL to display all the columns from both tables where the value of the primary key (`User.UserID`) is the same as the foreign key (`Post.UserID`).

Activity: Using SELECT (5)

Enter the SQL:

```
1 SELECT * FROM User, Post
2 WHERE User.UserID = Post.UserID
3 AND AccountName = "WhoRYou";
```

AND is used to apply both parts of the WHERE clause. This query displays all the columns from both tables for a user with the AccountName "WhoRYou".

Q5: Write the SQL to display all the columns for the PostID with a value of 281756.

Activity: Using SELECT (6)

Enter the SQL:

```
1 SELECT * FROM User
2 WHERE UserID > 72631;
```

This will display all the columns where the User ID is greater than 72631.

Q6: Write the SQL to display all the columns from the Post table where the PostID is less than 281749.

**Activity: Using SELECT (7)**

Enter the SQL:

```
1 SELECT * FROM User
2 WHERE UserID < 72620 OR UserID > 72658;
```

OR is used to connect the two conditions. This query will find all rows for UserIDs that are less than 72620 or where they are greater than 72658.

Q7: Write the SQL to display the PostID and PostContent from the Post table where the PostID is either less than 281722 or greater than 281954.

**Activity: Using SELECT (8)**

Enter the SQL:

```
1 SELECT * FROM User, Post
2 WHERE User.UserID = Post.UserID
3 AND (Post.UserID < 72620 OR Post.UserID > 72658);
```

This is a little more complicated. The equi-join between the two tables is established using the `User.UserID = Post.UserID`. The conditions, `UserID < 72620 OR UserID > 72658`, are written inside curved brackets.

We want BOTH of the conditions to be true AND we want the join to be true as well. Because of the brackets, the query will find all the matching values for UserID first and then find the values that match for the relationship.

Q8: Write the SQL to display the AccountName and PostContent from the User and Post tables where the PostID is equal to either 281821 or 281829.

**Activity: Using SELECT (9)**

Enter the SQL:

```
1 SELECT *
2 FROM User
3 ORDER BY AccountName DESC;
```

Results in answer tables can be sorted on one or more columns. These columns can be placed in ascending or descending order. This query sorts the rows of the User table into descending order of AccountName.

Q9: Write the SQL to display sorted rows from the Post table in ascending order of UserID.

Activity: Using SELECT (10)

Enter the SQL:

```
1 SELECT *
2 FROM Post
3 ORDER BY UserID DESC, PostContent ASC;
```

This places all the rows into order by UserID descending (from biggest number to smallest) and, where the UserID values are the same, places the PostContent column into alphabetical order ascending.

Q10: Write the SQL to display sorted rows from the Post table in ascending order of UserID ascending and then PostID descending.

Activity: Using SELECT (11)

Enter the SQL:

```
1 SELECT *
2 FROM Post
3 WHERE PostID = 281763 OR PostID = 281765 OR PostID = 281768
4 ORDER BY UserID DESC, PostContent ASC;
```

This places all the rows into order by UserID descending (from biggest number to smallest) and, where the UserID values are the same, places the PostContent column into alphabetical order ascending.

Q11: Write the SQL to display sorted rows from the Post table in ascending order of UserID ascending and then PostContent descending where PostID is greater than 281952.

Activity: Using SELECT (12)

Enter the SQL:

```
1 SELECT AccountName, PostID, PostContent
2 FROM User, Post
3 WHERE User.UserID = Post.UserID
4       AND (PostID = 281763 OR PostID = 281765 OR PostID = 281768)
5 ORDER BY Post.UserID DESC, PostContent ASC;
```

This combines all the clauses that we used previously. It selects the columns AccountName from the User table and PostID, PostContent from the Post table.

It implements the equi-join between the two tables (User.UserID = Post.UserID) and uses OR to select three matching PostIDs.

The results are then sorted by UserID descending and PostContent ascending.

Q12: Write the SQL to display the UserID and the PostID where the UserID is greater than 72656. Sort the results by UserID ascending and PostID ascending.

3.2.3 SQL operations: INSERT

The INSERT command is used to add a row or rows to a table. The format of the command is:

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

If you are adding values for all the columns of the 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.

The statement to add a new row, with the value 72661 for UserID and "shadow99" for AccountName, to the User table would be:

```
1 INSERT INTO User
2   VALUES (72661, "shadow99");
```

The INSERT command can be used to add more than one row at the same time. For example:

```
1 INSERT INTO User
2   VALUES (72661, "shadow99"),
3           (72662, "Good101"),
4           (72663, "BlakeyBoy");
```

Key point

Multiple Insert Value clauses are not supported by Microsoft Access. If you are using Microsoft Access then you can create an INSERT statement for each row of data to be entered.

The INSERT command also change the order of the column values being added. For example:

```
1 INSERT INTO User (AccountName, UserID)
2   VALUES ("shadow99", 72661);
```

Practical tasks using INSERT

The following activities make use of a database of Users and Posts. You previously used this database for the tasks using the SELECT command. If you have not already downloaded the necessary files, download <https://courses.scholar.hw.ac.uk/vle/asset/Downloads/H-CCMP/Course%20Downloads/C45F14CD-EA9B-6478-B678-E3ACFF7C1CE4/SELECT.zip>, extract the files and then carry out one of the following actions with your RDMS:

- import *user-post.sql* to an empty database to create the complete database;
- open *user-post.accdb* in Microsoft Access;
- CSV files *users.csv* and *posts.csv* are available for each database table for conversion to other platforms.

This section will add new rows to both tables.

Activity: Using INSERT (1)



Enter the SQL:

```
1 INSERT INTO User
2 VALUES (72661, "shadow99");
```

This will insert a row with the value 72661 for UserID and "shadow99" for AccountName.

Q13: Write the SQL to insert a new user with a UserID of 72662 and the AccountName of "Good101".

Activity: Using INSERT (2)



Enter the SQL:

```
1 INSERT INTO Post
2 VALUES (72661, 281957, "Wow, So many newbies in Fortnite.");
```

This will insert a row in the Post table.

Q14: Write the SQL to insert a new post with the following values.

UserID: 72662

PostID: 281958

PostContent: "I so need a Durr Burger Skin!!"

3.2.4 SQL operations: 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
2 SET column1 = value1, column2 = value2, ...
3 WHERE condition;
```

Key point

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

The statement to update the AccountName for "shadow99" to "Shatterwind" would be:

```
1 UPDATE User
2   SET AccountName = "Shatterwind"
3   WHERE AccountName = "shadow99";
```

The INSERT command can update more than one record at a time. For example, all the posts for the AccountName "WhoRYou" (UserID 72620) should have been linked to "Good101". User "Good101" has a UserID of 72662.

```
1 UPDATE Post
2   SET UserID = 72662
3   WHERE UserID = 72620;
```

This will update nine rows with the new UserID value of 72662. Another way to do the same update would be to use linked tables. When an UPDATE query references two tables, it will only ever update the first table listed.

```
1 UPDATE Post, User
2   SET Post.UserID = 72662
3   WHERE Post.UserID = User.UserID AND AccountName = "WhoRYou";
```

Cascade updates

If your RDBMS does not automatically cascade updates to enforce referential integrity, then you will have to do this manually. So, changing the value for a primary key:

```
1 UPDATE User
2   SET UserID = 72617
3   WHERE UserID = 72618;
```

Would require the following query to also be run to update the related foreign key values in the Post table.

```
1 UPDATE Post
2   SET UserID = 72617
3   WHERE UserID = 72618;
```

Activity: Using UPDATE (1)



Enter the SQL:

```
1 UPDATE User
2   SET AccountName = "Shatterwind"
3   WHERE AccountName = "shadow99";
```

This will update the AccountName where the current value is "shadow99" and replace it with "Shatterwind".

Q15: Write the SQL to update the AccountName "QuietRam" to "RohanBear".

**Activity: Using UPDATE (2)**

Enter the SQL:

```

1 UPDATE Post
2   SET UserID = 72662
3   WHERE UserID = 72620;
```

This will update the UserID where the current value is "72620" and replace it with "72662".

Q16: Write the SQL to update the UserID in the posts table to 72645 where the current value is 72634.

**Activity: Using UPDATE (3)**

Enter the SQL:

```

1 UPDATE Post
2   SET PostContent = "Kind of like Lukewarm Water"
3   WHERE PostID = 281721;
```

This will update the PostContent column where the current value of PostID is 281721.

Q17: Write the SQL to update the PostContent, of the row with PostID of 281758, to "Best leave it unsolved.".

3.2.5 SQL operations: DELETE

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 command is:

```

1 DELETE FROM table
2   WHERE condition;
```

Key point

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

To delete the Post with PostID of 281767 would require this query statement.

```

1 DELETE FROM Post
2   WHERE PostId = 281767;
```

To delete the all the posts for UserID of 72622 would require this query statement.

```
1 DELETE FROM Post
2   WHERE UserID = 72622;
```

Cascade deletes

If referential integrity is not enforced by your RDBMS, then you will have to ensure that it is maintained manually. So, if you delete a primary key value from User, you need to delete the related foreign key rows of Post.

```
1 DELETE FROM User
2   WHERE UserID = 7264;
```

Removing this row from the User table would require that the following query is carried out on the Post table.

```
1 DELETE FROM Post
2   WHERE UserID = 72640;
```

You can delete data using linked tables. Again, this will only delete data from the first listed table.

```
1 DELETE FROM Post, User
2   WHERE Post.UserID = User.UserID AND AccountName = "ChainToffee";
```

Activity: Using DELETE (1)



Enter the SQL:

```
1 DELETE FROM Post
2   WHERE PostId = 281767;
```

Q18: Write the SQL to DELETE a row from Post where PostID = 281954.

Activity: Using DELETE (2)



Enter the SQL:

```
1 DELETE FROM User
2   WHERE UserID = 72640;
```

Now delete the related rows in Post:

```
1 DELETE FROM Post
2   WHERE UserID = 72640;
```

Q19: Write the SQL to DELETE the user and posts related to UserID 72660.

**Activity: Using DELETE (3)**

Enter the SQL:

```
1 DELETE FROM Post
2 WHERE PostID > 281774 AND PostID < 281778;
```

This will delete all the rows with a PostID of more than 281774 and less than 281778.

Q20: Write the SQL to DELETE the posts with a PostID of greater than 281795 and less than 281802.

**Activity: Using DELETE (4)**

Enter the SQL:

```
1 DELETE FROM Post
2 WHERE PostID = 281820 OR PostID = 281836;
```

This will delete the two rows with the PostID values given.

Q21: Write the SQL to DELETE the posts with the following PostIDs: 281838, 281841, 281843 from the Post Table.

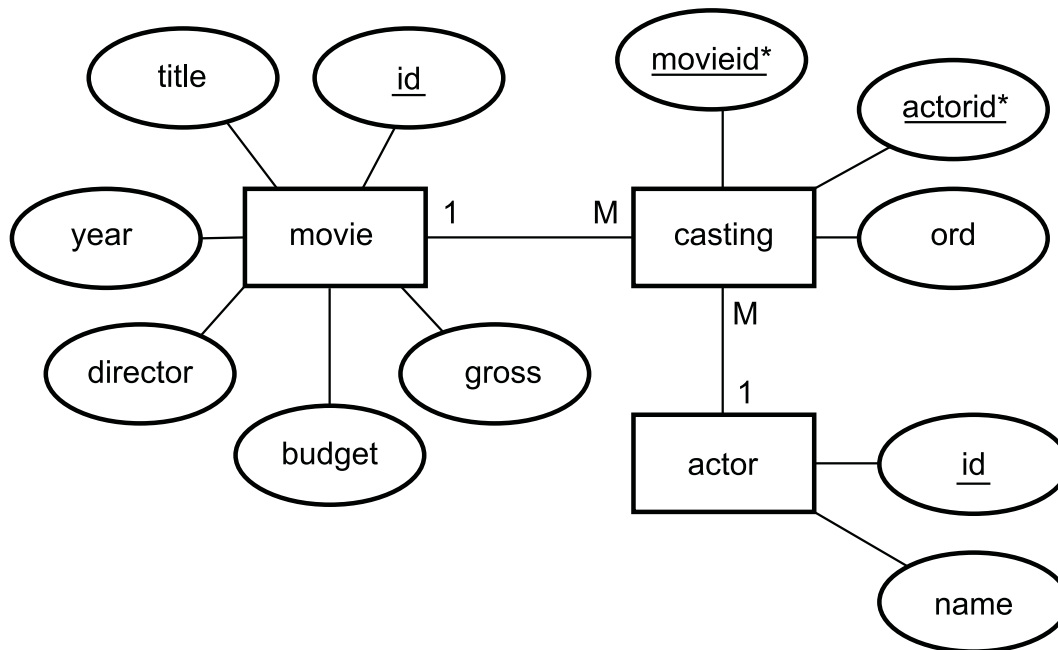
3.3 Example database

Exercises for the rest of this Topic will make use of the Movies database. The Movies database is provided with permission from <http://sqlzoo.net/>. The site contains many SQL exercises that you may wish to view.

Download <https://courses.scholar.hw.ac.uk/vle/asset/Downloads/H-CCMP/Course%20Downloads/4D0AE6A2-01F6-8137-D618-772AE364ECFC/movies.zip>, extract the files and then carry out the following actions using your RDMS:

- import *movies.sql* to an empty database to create the complete database;
- open *movies.accdb* in Microsoft Access;
- CSV files *actor.csv*, *casting.csv* and *movie.csv* are available for each database table for conversion to other platforms.

Figure 3.10: Entity Relational Diagram for the movies database



The movies database contains the details of over 48,000 actors, 12,000 movies and 120,000 casting roles. The actor table contains an id and name e.g. id: 6066, name: Mena Suvari.

The movie table has the following columns:

- id: Unique id for each row
- title: The title of the movie
- year: The year the movie was released
- director: The id (from the actor table) of the director
- budget: The budget for the movie (where it is known)
- gross: How much money the movie grossed (where known). The gross the total income for the movie before costs are deducted.

The casting table has movieid, actorid and ord columns. The movieid contains the foreign key linked to the movie table id. The actorid is linked to the actor table id column. The ord column is the position of the actor in the credits for the movie. The main star would have an ord value of 1, the co-star 2 and so on.

Table 3.1: Sample of data from the Movies database

actor	
id	name
3875	James Whitmore
4058	Bob Gunton
5353	Mark Rolston
6152	Morgan Freeman
6514	Tim Robbins
8029	William Sadler
9571	Clancy Brown
26722	Gil Bellows

casting

movieid	actorid	ord
20434	3875	8
20434	4058	3
20434	5353	7
20434	6152	2
20434	6514	1
20434	8029	4
20434	9571	5
20434	26722	6

movie

id	title	year	director	budget	gross
20434	The Shawshank Redemption	1994	41208	25000000	28341469

The above example shows that Tim Robbins (actor.id value of 6514) is cast in "The Shawshank Redemption" with an ord column value of 1. This means that he appears in the first position in the credits for the movie (because he is the main star of the movie).

3.4 SQL Wildcards

SQL queries can make use of wildcards in a number of ways. From your National 5 course you are already familiar with the * wildcard that can be used to select all columns from the underlying tables in a query.

Wildcards can also be used with the LIKE operator. The LIKE operator is used in a WHERE clause to match specific data in a column. The query `SELECT * FROM actor WHERE id LIKE 4058` functions in the same way as `SELECT * FROM actor WHERE id = 4058`. But using wildcards with the LIKE operator allows us more options.

The wildcards used with the LIKE operator are:

- % - the percent sign represents zero, one, or multiple characters;
- _ - the underscore represents a single character.

Key point

Microsoft Access uses a question mark (?) instead of the underscore (_). All these examples will use _. If using Microsoft Access please substitute a ? where a _ is shown.

Table 3.2: Wildcards

SELECT query with LIKE operator	Description
<pre> 1 SELECT * 2 FROM actor 3 WHERE name LIKE "Charl%"; </pre>	Show all rows from the actor table where the name starts with the letters "Charl".
<pre> 1 SELECT * 2 FROM actor 3 WHERE name LIKE "%love"; </pre>	Show all rows from the actor table where the name ends with the letters "love".
<pre> 1 SELECT * 2 FROM actor 3 WHERE name LIKE "%L.%"; </pre>	Show all rows from the actor table which contain "L." in any position.
<pre> 1 SELECT * 2 FROM actor 3 WHERE name LIKE "_ans%"; </pre>	Shows all rows from the actor table which have "ans" in the second position.
<pre> 1 SELECT * 2 FROM actor 3 WHERE name LIKE "Z_%_%"; </pre>	Shows all rows from the actor table which have at least three characters starting with "Z".

Activity: Using Wildcards



Q22: Write the SQL to display the id and title of the movies with a title of "Star Wars" followed by something e.g. "Star Wars 1: The Phantom Menace"

.....

Q23: Write the SQL to display the id and title of the movies with three letter titles.

.....

Q24: Write the SQL to display the id and title of the movies which end with the word "Part" followed by two characters. For example, "Atlas Shrugged: Part I".

3.5 Table and column aliases

Table alias

A table alias is a shortened version of the table name which can be used to refer to columns. This is particularly useful if the SELECT statement contains references to two columns with the same name in different tables.

For example:

```
1 SELECT Post.UserID, Post.PostContent, User.AccountName
2 FROM User, Post
3 WHERE User.UserID = Post.UserID;
```

This is a long SQL statement but it can be shortened by using table aliases. To create an alias an alternate name for the table is entered after the table name in the FROM clause. So the above SQL statement could be shortened to:

```
1 SELECT p.UserID, p.PostContent, u.AccountName
2 FROM User u, Post p
3 WHERE u.UserID = p.UserID;
```

The MySQL keyword AS can be used in the clause for clarity, but it is not a requirement i.e.

```
1 SELECT p.UserID, p.PostContent, u.AccountName
2 FROM User AS u, Post AS p
3 WHERE u.UserID = p.UserID;
```

Column alias

A column, calculation or aggregate function can be given an alias within a query.

For example, the query:

```
1 SELECT UserID, COUNT(*)
2 FROM Post
3 GROUP BY UserID;
```

Generates this answer table:

UserID	COUNT(*)
72618	4
72619	7
72621	4
72622	7
...	...

It would be better if the COUNT(*) column had a name that was more helpful. This can be done by using the AS operator.

```
1 SELECT UserID, COUNT(*) AS "Number of Posts"
2 FROM Post
3 GROUP BY UserID;
```

Now the answer table is:

UserID	Number of posts
72618	4
72619	7
72621	4
72622	7
...	...

The AS operator can be omitted.

```
1 SELECT UserID, COUNT(*) "Number of Posts"
2 FROM Post
3 GROUP BY UserID;
```

This generates the same answer table.

A column alias can be given to any calculated value, aggregated function or column.

3.6 Using sub-queries

Sub-queries are used to use one query as the basis for another query.

So, for example, a query to find the movie with the biggest budget would be:

```
1 SELECT MAX(budget) FROM movie;
```

This will display the budget column for the movie that has the largest budget. But if you wanted to know which movie had that budget you could use:

```
1 SELECT title, year, budget FROM movie
2 WHERE budget = (SELECT MAX(budget) FROM movie);
```

This uses the answer to one query as the criteria for another. In this case, it works, because `SELECT MAX(budget) FROM movie;` only returns a single value.

The query:

```
1 SELECT * FROM movie WHERE title LIKE "%Star Wars%";
```

... generates an answer table which contains all the movies related to "Star Wars".

id	title	year	director	budget	gross
17781	Star Wars: The Clone Wars	2008	37204	8500000	68284217
17782	Star Wars Episode I: The Phantom Menace	1999	5969	115000000	924317558
17783	Star Wars Episode II: Attack of the Clones	2002	5969	NULL	NULL
17784	Star Wars Episode III: Revenge of the Sith	2005	5969	NULL	848754768
17785	Star Wars Episode IV: A New Hope	1977	5969	11000000	NULL
17786	Star Wars Episode V: The Empire Strikes Back	1980	20166	32000000	538375067
17787	Star Wars Episode VI: Return of the Jedi	1983	10439	32500000	475106177
20542	The Star Wars Holiday Special	1978	22625	NULL	NULL

To query this answer table further, the original query could be used as a sub-query:

```

1 SELECT title
2 FROM actor,
3     (SELECT * FROM movie WHERE title LIKE "%Star Wars%") AS starwars
4 WHERE actor.id = starwars.director
5 AND actor.name = "George Lucas";

```

This uses the answer table generated by (SELECT * FROM movie WHERE title LIKE "%Star Wars%"). It gives this answer table the alias "starwars" and then it is treated like any other table.

So, the query (including the sub-query) would generate the answer table of all the Star Wars movies that were directed by George Lucas.

title
Star Wars Episode I: The Phantom Menace
Star Wars Episode II: Attack of the Clones
Star Wars Episode III: Revenge of the Sith
Star Wars Episode IV: A New Hope

Query design

When you have a query design, that uses one query as the basis for a second query e.g.

Examples

1. Query 1

Table(s) and / or query	player, gamescore
Fields and / or calculations	position, prizemoney
Search criteria	forename = "Jenny", surname = "McKay", player.playerid = result.playerid
Grouping	
Sort order	

.....

2. Query 2

Table(s) and / or query	[All results for Jenny McKay]
Fields and / or calculations	MAX(prizemoney)
Search criteria	position = 1
Grouping	
Sort order	

Implemented in SQL the first query would be:

```
1 SELECT result.position, result.prizemoney
2 FROM player, result
3 WHERE player.forename = "Jenny"
4 AND player.surname = "Mckay"
5 AND player.playerid = result.playerid;
```

This query can be assigned the alias 'All results for Jenny Mckay', in SQL to be used in the second query. Note the use of ' - this is because there are spaces in the alias so we have to use ' around it.

Implemented in SQL the second query would be:

```
1 SELECT Max('All results for Jenny Mckay'.prizemoney)
2 FROM (
3     SELECT result.position, result.prizemoney
4     FROM player, result
5     WHERE player.forename = "Jenny"
6     AND player.surname = "Mckay"
7     AND player.playerid = result.playerid)
8 AS 'All results for Jenny Mckay'
9 WHERE 'All results for Jenny Mckay'.position = 1;
```

This is how, in SQL, we use the result of one query as the basis for a second query.

3.7 SQL aggregate functions (MIN, MAX, AVG, SUM, COUNT)

Aggregate functions in SQL examine multiple values across a selected row and perform a calculation on these.

Table 3.3:

Function	Description	Example
MIN	Calculates the smallest value for the column or calculation.	SELECT MIN(budget) FROM movie;
MAX	Calculates the largest value for the column or calculation.	SELECT MAX(wages * hours) FROM timesheet;
AVG	Calculates the average for the column or calculation.	SELECT AVG(height) FROM students;
SUM	Calculates the total for the column or calculation.	SELECT SUM(salary) FROM staffing;
COUNT	Calculates the total number of rows / occurrences for values in a column. Can use * to count rows.	SELECT Count(*) FROM teachers;

Activity: Using MIN



Using the movies database, enter the following query:

```
1 SELECT * FROM movie
2 WHERE budget IS NOT Null AND gross IS NOT Null;
```

Sometimes some of the rows may contain empty values. For example, in the movie table the budget or the gross figure for a movie may not be known. In this case, the value is a NULL. NULL means that the cell has no value set - this is different from an empty value (e.g. "") and different from 0.

To find the smallest budget for a movie with a known **budget** and **gross**:

```
1 SELECT MIN(budget) FROM movie
2 WHERE budget IS NOT Null AND gross IS NOT Null;
```

Q25: Write the SQL to display the minimum gross for movies where both the budget and gross are not null.

Activity: Using MAX



We can find the maximum value for the budget in a movie using the query:

```
1 SELECT MAX(budget) FROM movie;
```

We can use this to find the row in the table which contains this value:

```
1 SELECT title, year, budget FROM movie
2 WHERE budget = (SELECT MAX(budget) FROM movie);
```

Q26: Write the SQL to display the title and year of the most recent movie.

Activity: Using AVG



We can find the average budget for a movie using:

```
1 SELECT AVG(budget) FROM movie;
```

If we want to ignore the rows where the budget is NULL we would use:

```
1 SELECT AVG(budget) FROM movie
2 WHERE budget IS NOT NULL;
```

Q27:

- Write the SQL to calculate the average gross for movies where the gross figure is not NULL.
- Write the SQL to calculate the average budget for all movies made in 1975.

Activity: Using SUM



We can find the total budget for movies made in 1920 using:

```
1 SELECT SUM(budget) FROM movie
2 WHERE year = 1920;
```

Q28: Write the SQL to calculate the total gross for movies made in 1999.

Activity: Using COUNT



We can count the number of movies made in 1969 using:

```
1 SELECT COUNT(*)
2 FROM movie
3 WHERE year = 1969;
```

Q29: Write the SQL to find out how many movies were made after 1974 and before 1980.

3.8 Computed values

Calculations can be carried out using SQL. For example, the query `SELECT 7+2` will display 9. Columns can be used to calculate computed values.

The star of a "Mission Impossible" movie will be paid 10% of the gross for the movie.

```
1 SELECT gross*10/100 AS payment FROM movie
2 WHERE title LIKE "Mission Impossible%";
```

This will calculate 10% of the gross for the movies with "Mission Impossible" at the start of the title. An alias is used to display the computed value with a column name of Payment.

Another example is the stars of the movie "Zoolander" will each be paid 60000. What is the total "Staff Pay" for the movie.

```
1 SELECT COUNT(*)*60000 as "Staff Pay"
2 FROM movie, casting
3 WHERE movieid = id and title = "Zoolander";
```

This query uses the `COUNT` function to total the number of actors cast in roles and then multiplies this by the payment to each actor. This computed value is given an alias of "Staff Pay".

Activity: Using computed values (1)



We can calculate the ratio of budget against gross for movies made in 2010 using the query:

```
1 SELECT title, year, gross/budget AS "Ratio"
2 FROM movie
3 WHERE year = 2010 AND budget IS NOT NULL AND gross IS NOT NULL;
```

Where this is greater than 1, the movie made a profit. Where it is less than 1 the movie made a loss.

To display only the profitable movies, we would use the query:

```
1 SELECT title, year, gross/budget AS "Ratio"
2 FROM movie
3 WHERE year = 2010 AND budget IS NOT NULL AND gross IS NOT NULL AND
   gross/budget > 1;
```

The biggest loss-making movie of 2010 has the minimum ratio.

```
1 SELECT MIN(gross/budget)
2 FROM movie
3 WHERE year = 2010 AND budget IS NOT NULL AND gross IS NOT NULL;
```


To find the details of the biggest loss-making movie, we can use the query:

```

1 SELECT title, year, gross/budget AS "Ratio"
2 FROM movie
3 WHERE year = 2010 AND budget IS NOT NULL AND gross IS NOT NULL AND
   gross/budget = (
4 SELECT MIN(gross/budget)
5 FROM movie
6 WHERE year = 2010 AND budget IS NOT NULL AND gross IS NOT NULL
7 );

```

Q30: Write the SQL to find the most profitable movie of 2001 based on its budget i.e. one with the highest ratio.

Activity: Using computed values (2)



A famous director has decided to give all the actors with "Smith" in their name £150.

This would be calculated as:

```

1 SELECT COUNT(*)
2 FROM actor
3 WHERE name LIKE "%Smith%";

```

Q31: The government is going to tax all movies with the word "Gold" in the title. These will all have to pay tax of 15% of their budget. Display the title and tax due in your query.

3.9 GROUP BY

The GROUP BY statement is often used with aggregate functions (MIN, MAX, AVG, SUM, COUNT) to group results by one or more columns.

```

1 SELECT column, column, ...
2 FROM table
3 WHERE condition
4 GROUP BY column, column, ...

```

We can show all the movies by the same director using the query:

```

1 SELECT name, director, Count(*)
2 FROM movie, actor
3 WHERE director=actor.id
4 GROUP BY director;

```

This will group the results by the director value and the count function adds up the grouped rows for each director e.g.

Table 3.4: Table grouped by director

name	director	Count(*)
	2	94
Richard Brooks	3	15
Warren Beatty	4	4
Peter Fairman	19	2
John Cornell	36	1
Charles S. Dutton	39	2
Scott Winant	49	1
Ken Olin	62	1
Edmund Goulding	65	14
Bo Widerberg	77	4
Karyn Kusama	82	3
Marc Webb	95	1
Jonathan Parker	107	1
Roland Emmerich	121	8
...

Another example would be a query used to count the number of times an actor has been the main star (ord = 1) in a movie:

```

1 SELECT name, count(*)
2 FROM actor, casting, movie
3   WHERE actor.id = casting.actorid AND
4         casting.movieid = movie.id AND
5         ord=1
6   GROUP BY actor.id;
```

This would generate the answer table:

name	COUNT(*)
	1
Warren Beatty	9
Goldie Hawn	12
Arthur Brauss	1
Paul Hogan	4
John Meillon	1
Anne Carlisle	1
Paul Hogan	1
Kenneth Welsh	1
Jeanne Tripplehorn	1
...	...

Activity: Using GROUP BY



Q32: Write the SQL to display each year with the total budget for movies made in that year. The column for total budget should be called "Movie Budget". The query should ignore NULL values for budget.

.....

Q33: Write the SQL to count the number of actors cast in each movie with a budget of more than 180000000. The count of actors should have the name "Total Cast". The query should display the movie title and the "Total Cast".

.....

Q34: Write the SQL a query to count the number of times an actor has been the main star (ord = 1) in a movie. Sort the results by the number of times the actor has been the main star.

3.10 ORDER BY

From your studies for National 5, you are aware that Order By is used to sort the answer table into order by one or more fields. These fields can be in ascending or descending order. In the National 5 course, you would have sorted your answer tables by a maximum of two fields however in Higher you can use more than two.

For example, a query is required to display the following details:

- Year of movie
- Total number of actors cast in movies in that year

This will be sorted by total actors descending and year ascending.

```

1 SELECT year, Count(*) AS "TotalActors"
2 FROM movie, casting
3 WHERE movie.id = casting.movieid
4     GROUP BY year
5     ORDER by TotalActors DESC, year ASC;
```

Activity: Using ORDER BY



Q35: Write the SQL to display the number of movies that each actor has been cast in for each year. Sort the answer table so that the actor with the most movies in any year is at the top of the table. Where the number of movies is the same sort the results by year ascending.

.....

Q36: Write the SQL to count the number of actors cast in each movie with a budget of more than 180000000.

The count of actors should have the name "Total Cast". The query should display the movie title and the "Total Cast". Sort the results by title ascending

.....

Q37: Write code to sort the movie table by year ascending, budget descending and then gross descending.

3.11 Learning points

Summary

You should now be able to:

- describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables;
- use:
 - UPDATE, SELECT, DELETE and INSERT statement on more complex related tables;
 - wildcards in your queries;
 - aggregated functions MIN, MAX, AVG, SUM and COUNT to process data using queries;
 - GROUP BY clauses to gather data of similar values together;
 - ORDER BY clauses to sort answer tables on multiple fields;
 - use sub-queries so that the answer table generated by a previous query can be used as the basis for a further query;
- make more complex use of WHERE clauses to select data;
- read and explain SQL statements which makes use of the above points.

3.12 End of topic test

End of topic 3 test

Go online



Questions 39 to X make use of these tables:

branch (branchno, street, city, postcode)

staff (staffno, fname, lname, position, sex, dob, annualsalary, branchno*)

propertyForRent (propertyno, street, city, postcode, type, rooms, rent, owner, staffno*, branchno*)

client (clientno, fname, lname, telno, preftype, maxrent)

viewing (clientno*, propertyno*, viewdate, comments)

Q38: Which of these queries produces a list of all branch offices in London or Glasgow?

- a) SELECT * FROM branch WHERE city = 'London' OR city='Glasgow';
- b) SELECT * WHERE branch.city = 'London' OR branch.city = 'Glasgow';
- c) SELECT * FROM branch WHERE city = 'London' OR 'Glasgow';
- d) SELECT * FROM staff, branch WHERE staff.branchno = branch.branchno;

.....

Q39: Which query produces a list of the staff number and last name of staff who work in the branch at "13 Main Road"?

a)

```
1 SELECT staffno, lname
2 FROM staff, branch
3 WHERE staff.branchno = branch.branchno
4 AND branch.street = '13 Main Road';
```

b)

```
1 SELECT *
2 FROM staff, branch
3 WHERE staff.branchno = branch.branchno
4 AND branch.street = '13 Main Road';
```

c)

```
1 SELECT staffno, lname
2 FROM staff, branch
3 WHERE staff.branchno = branch.branchno;
```

d)

```
1 SELECT staffno, branch.street='13 Main Road', lname
2 FROM staff, branch
3 WHERE staff.branchno = branch.branchno;
```

.....

Q40: Which of these queries produces an answer table which shows the details for all properties being sold by the Edinburgh branch?

a)

```
1 SELECT * FROM propertyForRent
2 WHERE city = 'Edinburgh';
```

b)

```
1 SELECT * FROM branch
2 WHERE city = 'Edinburgh';
```

c)

```
1 SELECT * FROM branch, propertyForRent
2 WHERE branchno.city = 'Edinburgh'
3 AND propertyForRent.branchno = branch.branchno;
```

d)

```
1 SELECT * FROM branch, propertyForRent
2 WHERE branchno.city = 'Edinburgh';
```

.....

Q41: Which query produces an answer table which shows the fname, lname and annualsalary of the staff member in Glasgow with the highest annualsalary?

a)

```
1 SELECT fname, lname, MAX(annualsalary) FROM staff
2 WHERE city = 'Glasgow';
```

b)

```
1 SELECT fname, MAX(annualsalary) FROM staff
2 WHERE staff.branchno = branch.branchno AND city = 'Glasgow';
```

c)

```
1 SELECT staff.*, MAX(annualsalary) FROM staff, branch
2 WHERE staff.branchno = branch.branchno AND city = 'Glasgow';
```

d)

```
1 SELECT fname, lname, MAX(annualsalary) FROM staff, branch
2 WHERE staff.branchno = branch.branchno AND city = 'Glasgow';
```

.....

Q42: Which query produces an answer table which shows all the clients with a maxrent of less than 800?

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

.....

Q43: With SQL, how do 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';

.....

Q44: With SQL, how can you return all the records from a table named "developer" sorted descending by "firstname"?

- a) SELECT * FROM developer ORDER firstname DESC;
- b) SELECT * FROM developer SORT 'firstname' DESC;
- c) SELECT * FROM developer ORDER BY firstname DESC;
- d) SELECT * FROM developer SORT BY firstname DESC;

.....

Q45: With SQL, how can you delete the records where the "firstname" is "Peter" in the developer table?

- 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;

.....

Q46: 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:

```

1 SELECT type, MAX(listPrice) AS "Most Expensive"
2 FROM Car
3 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

.....

Q47: A query is run against following two tables:

Table: competition

competitionID	country	venue	eventDate
1	UK	Southport	13/05/2018
2	Switzerland	Bern	29/08/2018
3	Canada	Montreal	08/09/2018
...

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:

```

1 SELECT country, SUM(PrizeValue)
2 FROM competition, entryticket
3 WHERE competition.competitionID = entryticket.competitionID
4 GROUP BY country;
```

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

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

Topic 4

Testing and evaluation

Contents

4.1 Testing SQL queries	95
4.2 Evaluating SQL queries	100
4.2.1 Fitness for purpose	101
4.2.2 Accuracy of output	101
4.3 Learning points	101
4.4 End of topic test	102

Prerequisites

From your studies at National 5 you should already know how to:

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

4.1 Testing SQL queries

From your studies at National 5 level you will be aware of testing queries which make use of a maximum of two linked tables. You will have learnt that SQL queries are tested to ensure that they function correctly and produce the required result - either an updated answer table (for SELECT queries) or a resulting change to the data (INSERT, UPDATE, DELETE queries).

When testing you are effectively testing that the require results are present.

Testing SQL queries are correct

The following eSports examples will use this database:

player (username, realname, password, email, message, terms_and_conditions)

gamescore (score_id, username*, game, score*)

game (game, platform)

player

username	realname	password	email	message	terms_and_conditions
shocker	Paul White	pink10red	paul@gamers.org	I want to play competitively	on
peach	Sally McDonald	trustme1	sally@scott.org	I'm part of an eSports team	on
destroyer	Chloe Davidson	shadow99	chloe@coders.org	I love games	on

gamescore

score_id	username	game	score
625	shocker	Massive RPG	726122
626	peach	SuperJoe	102928
627	peach	Massive RPG	625100
628	shocker	Terra 1999	821200
629	destroyer	Terra 1999	120001
630	peach	SuperJoe	283102
630	destroyer	SuperJoe	299000

game

game	platform
Massive RPG	X-station
SuperJoe	S-box
Terra 1999	PC

To effectively test queries, you should be able to read and explain the SQL for a given query. A query is created to show the maximum score for each platform.

```
1 SELECT platform, MAX(score)
2 FROM gamescore gs, game g
3 WHERE gs.game = g.game GROUP BY platform;
```

This query will display only the platform and the maximum score from "gamescore". The results will be grouped by platform so there will only be one row for each different value of platform. The two tables are joined using the primary key / foreign key pair of game.game and gamescore.game.

The resulting answer table would be:

platform	MAX(score)
X-station	821200
S-box	299000
PC	726122

The following query is designed to produce the list of maximum scores for each player.

```
1 SELECT player, MAX(score)
2 FROM player p, gamescore gs
3 GROUP by player;
```

There are several problems with this query.

The query refers to a column called "player" but no column of that name exists. The column used should be "username". This would make the query:

```
1 SELECT p.username, MAX(score)
2 FROM player p , gamescore gs
3 GROUP by p.username;
```

There is no equi-join to link the two tables, so the query will show the maximum score (821200) with each unique value for username. This would produce an incorrect answer table of:

username	MAX(score)
destroyer	821200
peach	821200
shocker	821200

To correct the query the equi-join is required. This would give the following SQL:

```
1 SELECT p.username, MAX(score)
2 FROM player p , gamescore gs
3 WHERE p.username = gs.username
4 GROUP by p.username;
```

This query now correctly produced an answer table containing two columns, username and MAX(score), and is grouped on the username so that it will only display one row for each unique value of username.

username	MAX(score)
destroyer	299000
peach	625100
shocker	821200

Testing SQL operations

Ultimately, the SQL queries that you create should generate the required answer tables or output.

Do your INSERT, UPDATE and DELETE queries produce the intended results?

Do your SELECT queries generate answer tables which include the required data?

Activity: SQL operations



player

username	realname	password	email	message	terms_ and_ conditions
shocker	Paul White	pink10red	paul@gamers.org	I want to play competitively	on
peach	Sally McDonald	trustme1	sally@scott.org	I'm part of an eSports team	on
destroyer	Chloe Davidson	shadow99	chloe@coders.org	I love games	on

gamescore

score_id	username	game	score
625	shocker	Massive RPG	726122
626	peach	SuperJoe	102928
627	peach	Massive RPG	625100
628	shocker	Terra 1999	821200
629	destroyer	Terra 1999	120001
630	peach	SuperJoe	283102
630	destroyer	SuperJoe	299000

game

game	platform
Massive RPG	X-station
SuperJoe	S-box
Terra 1999	PC

Q1: Using the eSports database tables, identify which answer table would be created using the SQL:

```

1 SELECT p.username, platform
2 FROM gamescore gs, game g, player p
3 WHERE gs.username=p.username AND gs.game = g.game;
```

a)

username	platform
shocker	PC
peach	S-box
shocker	X-station

b)

username	platform
destroyer	S-box
peach	X-station
shocker	X-station

c)

username	platform
shocker	X-station
peach	S-box
peach	X-station
shocker	PC
destroyer	PC
peach	S-box
destroyer	S-box

d)

username	platform
destroyer	X-station
peach	X-station
shocker	X-station

.....

Q2: Using the eSports database tables above, identify which answer table would be created using the SQL:

```

1 SELECT g.game, gs.username, email, message, MAX(Score)
2 FROM player p, gamescore gs, game g
3 WHERE terms_and_conditions = "on"
4     AND g.game =gs.game
5     AND gs.username = p.username
6 GROUP BY gs.username
7 ORDER BY gs.username DESC;
    
```

a)

game	username	email	message	MAX(score)
Massive RPG	shocker	paul@gamers.org	I want to play competitively	821200
SuperJoe	peach	sally@scott.com	I'm part of an eSports team	625100
Terra 1999	destroyer	chloe@coders.org	I love games	299000

b)

game	username	email	MAX(score)
Massive RPG	shocker	paul@gamers.org	821200
SuperJoe	peach	sally@scott.com	625100
Terra 1999	destroyer	chloe@coders.org	299000

c)

game	username	email	message	MAX(score)
Terra 1999	destroyer	chloe@coders.org	I love games	299000
SuperJoe	peach	sally@scott.com	I'm part of an eSports team	625100
Massive RPG	shocker	paul@gamers.org	I want to play competitively	821200

d)

game	username	email	message	MAX(score)
Massive RPG	shocker	paul@gamers.org	I want to play competitively	821200
Massive RPG	peach	sally@scott.com	I'm part of an eSports team	625100
Massive RPG	destroyer	chloe@coders.org	I love games	299000

.....

Q3: Using the eSports database tables above, identify which answer table would be created using the SQL:

```
1 SELECT * FROM player
2 WHERE username LIKE "d%" OR username LIKE "p%";
```

a)

username	realname	password	email	message	terms_and_conditions
destroyer	Chloe Davidson	shadow99	chloe@coders.org	I love games	on
peach	Sally McDonald	trustme1	sally@scott.org	I'm part of an eSports team	on
shocker	Paul White	pink10red	paul@gamers.org	I want to play competitively	on

b)

username
destroyer
peach

c)

realname	password	email
Chloe Davidson	shadow99	chloe@coders.org
Sally McDonald	trustme1	sally@scott.org

d)

username	realname	password	email	message	terms_and_conditions
destroyer	Chloe Davidson	shadow99	chloe@coders.org	I love games	on
peach	Sally McDonald	trustme1	sally@scott.org	I'm part of an eSports team	on

4.2 Evaluating SQL queries

From National 5 Computing Science you should already be familiar with queries which manipulate data from two linked tables.

At Higher level you are asked to review the database solution you have developed and determine if it is "fit for purpose" and produces the required "accuracy of output".

4.2.1 Fitness for purpose

When evaluating **fitness for purpose**, you are evaluating if the solution is "good enough to do the task required".

For example, if the queries you have written present additional columns of data that were not required, they would still likely be "fit for purpose" but the "accuracy of the output" may not be suitable.

4.2.2 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?

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.

4.3 Learning points

Summary

You should now be able to:

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

4.4 End of topic test

End of topic 4 test

Go online



Q4: "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.

.....

Q5: Identify the query which will have produced this table:

Name	Owner	Hull Type	Crew	Max Speed
Lucky Lady	H. Owen	double	3	29
Gallant	S. Scott	double	5	28
Gretel	W. Robertson	double	4	28
Fulmar II	J. Low	single	3	14
Skylark	J. Unwin	single	2	14
Jasmin	H. Owen	triple	5	24
Ocean Flyer	J. Low	triple	6	22

a)

```

1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC, Sailboat.Crew DESC;
```

b)

```

1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC;
```

c)

```

1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.Crew > 2 AND Sailboat.Crew < 7
4 AND Sailboat.'Max speed' > 13
5 ORDER BY Sailboat.'Hull Type' ASC;
```

d)

```

1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, MAX(Sailboat.'Max speed')
2 FROM Sailboat, Owner
3 WHERE Sailboat.Crew > 2 AND Sailboat.Crew < 7
4 AND Sailboat.'Max speed' > 13
5           GROUP BY MAX(Sailboat.'Max speed')
6 ORDER BY Sailboat.'Hull Type' ASC;
    
```

.....

Q6: Identify the query which has been applied to this table of data...

Date of sale	Item Purchased	Customer Name	Price of item
03/03/07	Lounge suite	Mrs W Jackson	2395
04/03/07	Dining table set	Mrs W Jackson	2100
01/03/07	Dining table set	Mr D Macey	1595
02/03/07	Leather suite	Miss L Mackie	945
05/03/07	King-size bed	Mrs W Jackson	769
05/03/07	Cane swivel seat	Mrs J Hetherington	375
02/03/07	Occasional tables	Mrs J Hetherington	299
02/03/07	Coffee table	Mrs J Hetherington	230
03/03/07	Standard lamp	Ms R al-Jamali	89

... to produce this answer table.

Miss L Mackie	945
Mr D Macey	1595
Mrs J Hetherington	904
Mrs W Jackson	5264
Ms R al-Jamali	89

a)

```

1 SELECT 'Customer Name', SUM('Price of item')
2 FROM Sales
3 GROUP BY 'Customer Name';
    
```

b)

```

1 SELECT * SUM('Price of item')
2 FROM Sales
3 GROUP BY SUM('Price of item');
    
```

c)

```

1 SELECT 'Customer Name', 'Price of item'
2 FROM Sales
3 WHERE SUM('Price of item')
4 GROUP BY 'Customer Name';
    
```

d)

```
1 SELECT 'Customer Name', SUM('Price of item')
2 FROM Sales
3 GROUP BY 'Customer Name', SUM('Price of item');
```

.....

Q7: "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) query results are close enough so that required data can be visually identified.
- d) the output generated exactly matches the requirements.

Topic 5

End of unit test

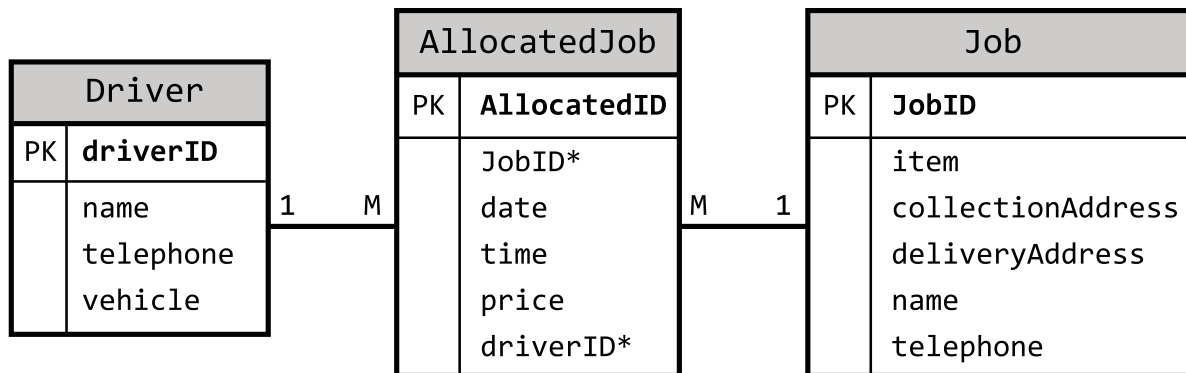
End of unit 3 test

Go online



Use this diagram to answer Questions 1 and 2.

A database is designed as shown.



Q1: Which SQL statement will display the JobID of jobs allocated to a driver called 'Colin Shaw'?

a)

```

1 SELECT JobID
2 FROM Job J, AllocatedJob AJ, Driver D
3 WHERE J.JobID = AJ.DriverID
4     AND AJ.JobID=D.DriverID
5     AND J.name = 'Colin Shaw';

```

b)

```

1 SELECT JobID
2 FROM Job
3 WHERE Driver.name = "Colin Shaw"
4 JOIN driverID = JobID;

```

c)

```

1 SELECT JobID
2 FROM Job J, AllocatedJob AJ, Driver D
3 WHERE J.JobID = AJ.JobID
4     AND AJ.driverID=D.DriverID
5     AND D.name = 'Colin Shaw';

```

d)

```

1 SELECT JobID
2 FROM Job J, AllocatedJob AJ, Driver D
3 WHERE J.JobID = AJ.DriverID
4     AND AJ.JobID=D.DriverID
5     AND D.name = 'Colin Shaw';

```

.....

Q2: Which SQL statement will display all the details about Jobs which are allocated to driverID 1921 where the collectionAddress includes the "Aberdeen". The list should show the driver name, vehicle, date, JobID and collectionAddress. The results should be displayed so that the earliest job is listed first.

a)

```

1 SELECT Driver.name, Driver.vehicle, 'date', Job.JobID,
   collectionAddress
2 FROM Driver, AllocatedJob, Job
3 WHERE Driver.driverID=AllocatedJob.driverID
4     AND AllocatedJob.JobID = Job.JobID
5     OR Driver.driverID = 1921
6     OR collectionAddress LIKE "Aberdeen"
7 ORDER BY 'date' ASC;
    
```

b)

```

1 SELECT Driver.name, Driver.vehicle, 'date', Job.JobID,
   collectionAddress
2 FROM Driver, AllocatedJob, Job
3 WHERE Driver.driverID=AllocatedJob.driverID
4     AND AllocatedJob.JobID = Job.JobID
5     AND Driver.driverID = 1921
6     AND collectionAddress LIKE "%Aberdeen%"
7 ORDER BY 'date' ASC, 'time' ASC;
    
```

c)

```

1 SELECT Driver.name, Driver.vehicle, 'date', Job.JobID,
   collectionAddress
2 FROM Driver, AllocatedJob, Job
3 WHERE Driver.driverID=AllocatedJob.driverID
4     AND AllocatedJob.JobID = Job.JobID
5     AND Driver.driverID LIKE "%1921%"
6     AND collectionAddress LIKE "%Aberdeen%"
7 ORDER BY 'date' ASC, 'time' ASC;
    
```

d)

```

1 SELECT Job.name, Driver.vehicle, date, Job.JobID,
   collectionAddress
2 FROM Driver, AllocatedJob, Job
3 WHERE Driver.driverID=AllocatedJob.driverID
4     AND AllocatedJob.JobID = Job.JobID
5     AND Driver.driverID = 1921
6     AND collectionAddress LIKE "%Aberdeen%"
7 ORDER BY 'date' ASC, 'time' ASC;
    
```

.....

Q3: Define the term "End Users" when considering the development of a system?

- a) End users are the actual users who will operationally make use of the system.
 - b) End users are the client who will be paying for the system.
 - c) End users are the general public who would be interacting with the system.
 - d) End users are the system owners who have an interest in the system development.
-

Q4: Read this scenario and then identify the End User(s).

Paul Jackson operates a skateboard design service. He custom builds skateboards for his customer. He wants a system developed which will allow customers to select the parts, supplied by various manufacturers, for their skateboards, pay for them and the build process online. The system will then produce work jobs for Paul, so that he can build the boards with the requested parts.

Identify who the end user(s) would be.

- a) Customers
 - b) Paul Jackson
 - c) Manufacturers and Paul Jackson
 - d) Customers and Paul Jackson
-

Q5: Define the term Functional Requirements.

Functional requirements...

- a) are specific actions that the system must perform.
 - b) are how the system must work.
 - c) define the security of the system.
 - d) define the reliability of the system.
-

Q6: In Agile development, the functional requirements are captured in the...

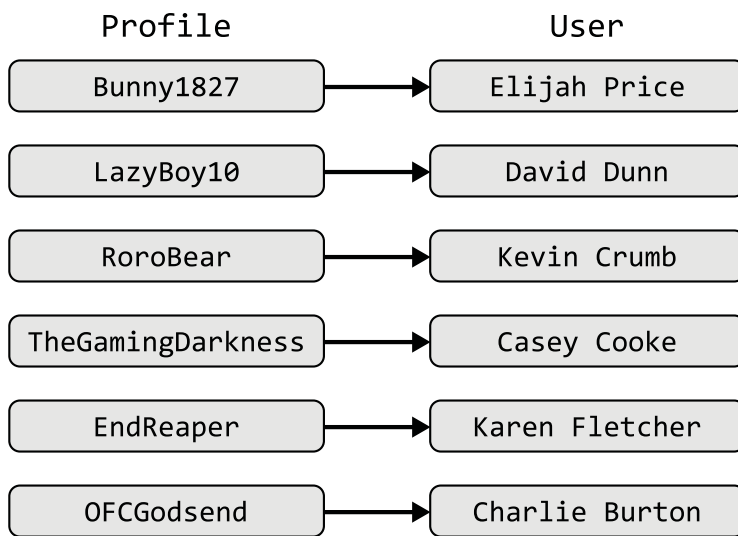
- a) system design.
 - b) requirements specification.
 - c) product backlog.
 - d) outline hardware specification.
-

Q7: How are functional requirements different from non-functional requirements?

- a) Functional and non-functional requirements explain the quality requirements of system. Non-functional requirements state which parts of the system should be measured and the functional requirement detail how each part should be measured.
- b) Functional requirements define what the system must do. Non-functional requirements define how the system should behave.
- c) Functional requirements detail how the modules of code in the system should be linked to each other. Non-functional requirements detail how the documentation will be written.
- d) Functional requirements are defined by the system owner only whereas non-functional requirements can be suggested by anyone associated with the system.

.....

Q8: This is an entity-occurrence diagram that shows the links between instances in two entities. What is the relationship between the two entites?



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

.....

Q9: Examine this data model and then identify the correct Entity Relationship Diagram for this data model.

Table: Customer

cust_id
 cust_name
 cust_address
 cust_tel_no
 credit_limit

Table: Order

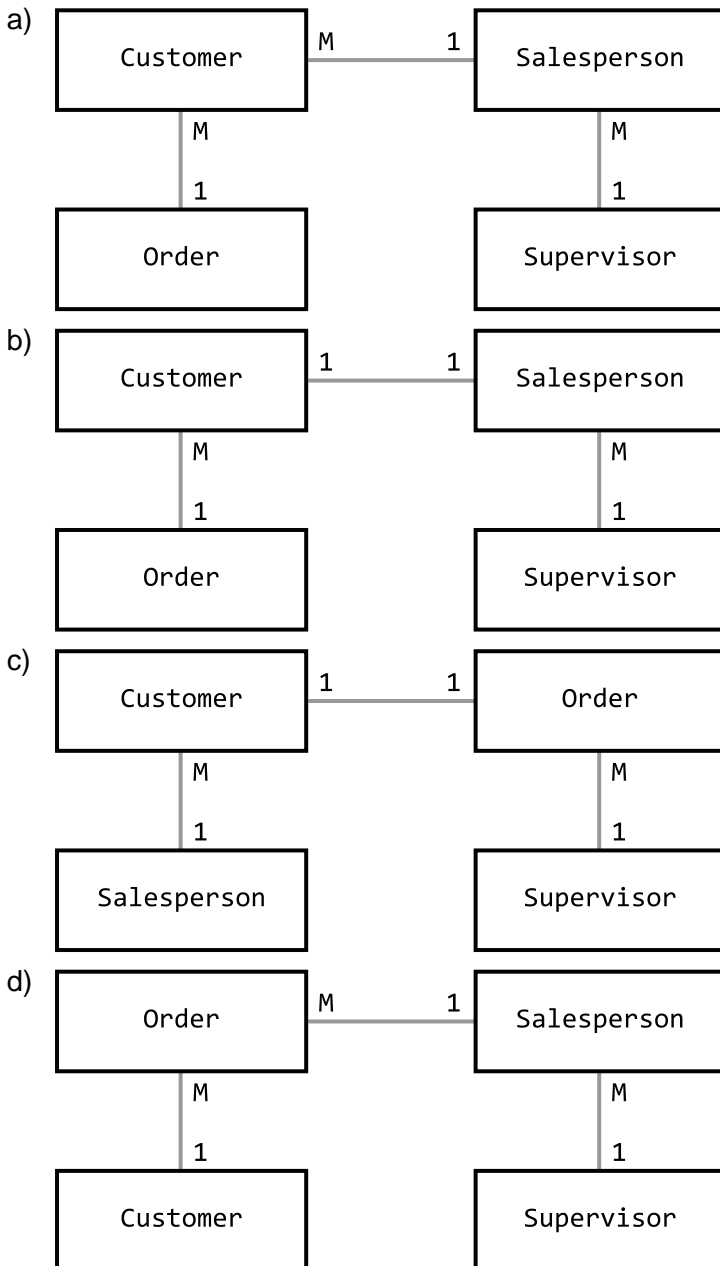
order_code
 item_name
 Price
 cust_id*
 sales_id*
 saledate

Table: Salesperson

sales_id
 grade
 date_employed
 sup_id*

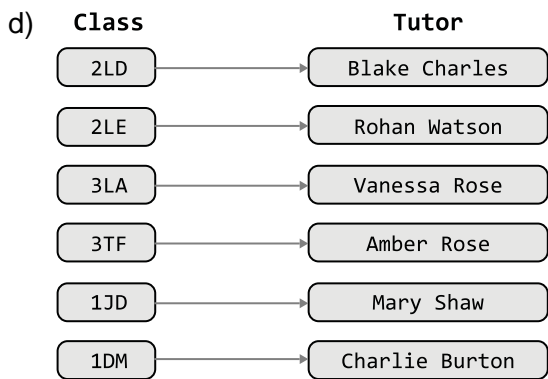
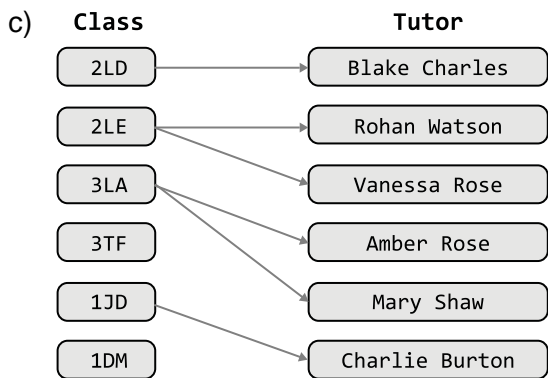
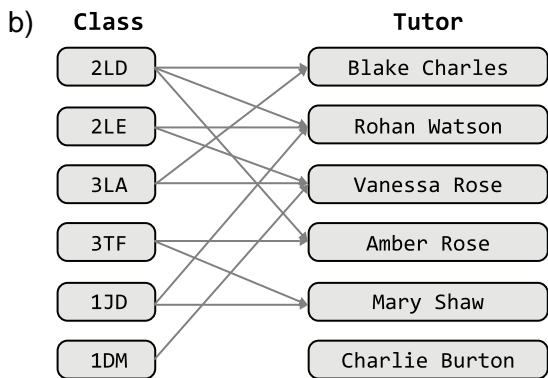
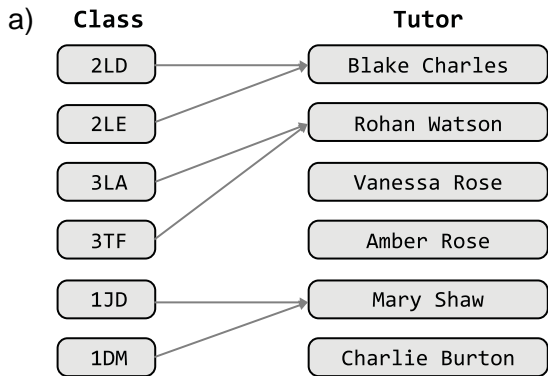
Table: Supervisor

sup_id
 dept



Q10: Senior Pupil Tutors at Someplace Academy attend classes to provide extra support for teachers. Any class in the school may have one, two or more tutors allocated to it and tutors can go to more than one class.

Which Entity Occurrence Diagram is correctly identifies the relationship between Tutors and Classes?



.....

Q11: MIN is a functional of relational database. Identify the four other aggregate functions of database that you should be familiar with.

- a) MOST, MEAN, TOTAL, NUMBEROF
- b) MAX, MEDIAN, SUM, COUNT
- c) MAX, AVG, SUM, COUNT
- d) LEVEL, AVG, TOTAL, NUMBEROF

.....

Q12: Which of these statements is the correct definition of a primary key?

- a) An attribute that is the same for all the entities.
- b) The longest attribute within an entity.
- c) The shortest attribute within an entity.
- d) An attribute that has a unique value for each entity.

.....

Q13: Which of these statements is the correct definition of a foreign key?

- a) An attribute that is the primary key of another entity set.
- b) The attribute field within an entity.
- c) The most important attribute in an entity.
- d) An attribute in a foreign language.

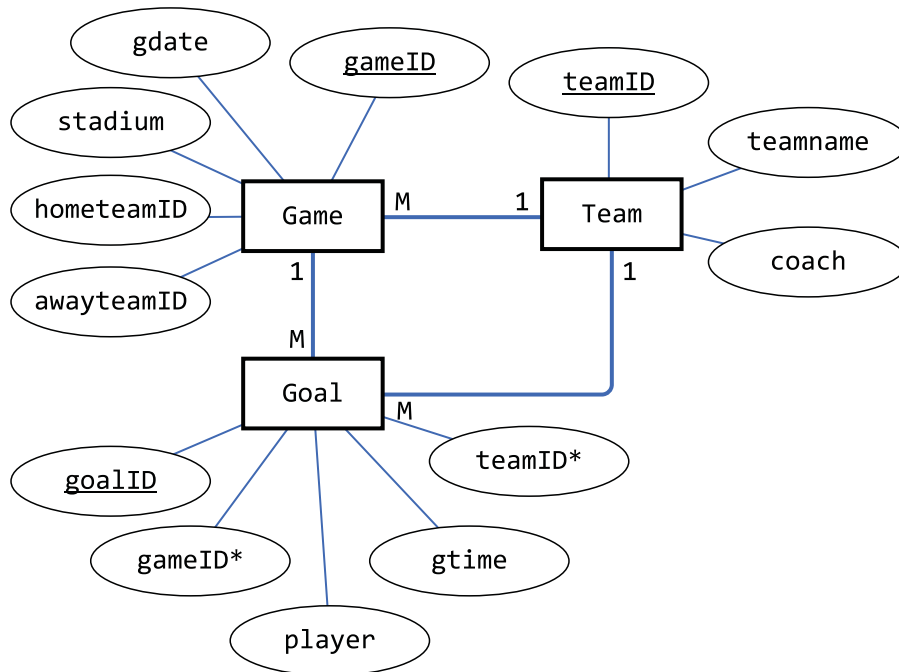
.....

Q14: A compound key is...

- a) a combination of a primary and foreign key to link to entity sets.
- b) a combination of more than one attribute to uniquely identify an entity.
- c) an attribute of the entity that is important and will be indexed to ensure fast access within search.
- d) an unique identifier for an entity that consists on a single attribute.

Use this diagram to answer Questions 15, 16, 17 and 18.

Examine this ERD. It records each goal stored in a football game. It also holds the stadium where the game was played and who the home and away teams were. The details of the team; their name and coach, is also held.



Q15: An SQL query is required to show the player, team, coach and gtime for all goals scored in the first 10 minutes of a game. Identify the correct SQL query.

a)

```
1 SELECT player, teamname, coach, gtime
2 FROM goal, game, team
3 WHERE game.gameID = goal.gameID
4     AND game.hometeamID = team.teamID
5     AND game.awayteamID = team.teamID
6 AND gtime <= 10;
```

b)

```
1 SELECT player, teamname, coach, gtime
2 FROM goal, game, team
3 WHERE game.gameID = goal.gameID
4     AND goal.teamID = team.teamID
5 AND gtime <= 10;
```

c)

```
1 SELECT player, teamname, coach, gtime
2 FROM goal, game, team
3 WHERE game.gameID = goal.gameID
4     AND game.hometeamID = team.teamID
5 AND gtime <= 10;
```

d)

```
1 SELECT player, teamname, coach, gtime
2 FROM goal, game, team
3 WHERE game.gameID = goal.gameID
4     AND game.awayteamID = team.teamID
5 AND gtime<=10;
```

.....

Q16: An SQL query is required to show all the players and the total goals they scored in each stadium. Identify the correct SQL query.

a)

```
1 SELECT player, stadium, SUM(*) as Goals
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4 GROUP BY player, stadium;
```

b)

```
1 SELECT player, stadium, SUM(*) as Goals
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4 GROUP BY stadium, player;
```

c)

```
1 SELECT *, COUNT(*) as Goals
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4 GROUP BY player, stadium;
```

d)

```
1 SELECT player, stadium, COUNT(*) as Goals
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4 GROUP BY player, stadium;
```

.....

Q17: An SQL query is required to show the gameID and player for all goals scored by the team with a teamID value of "IRL". Identify the correct SQL query.

a)

```
1 SELECT gameID, player
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4     AND teamID = "IRL";
```

b)

```
1 SELECT *
2 FROM Game, Goal
3 WHERE hometeamID = Team.teamID
4     AND teamID = "IRL";
```

c)

```
1 SELECT gameID, player
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4     FIND teamID = "IRL";
```

d)

```
1 SELECT gameID, player
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4     AND teamname = "IRL";
```

.....

Q18: An SQL query is required to show the "gdate" of games and name of the teams in which "Fernando Santos" is was the hometeam coach. Identify the correct SQL query.

a)

```
1 SELECT gdate, teamname
2 FROM Game, Team
3 WHERE Game.hometeamID = Team.teamID
4     AND coach = "Fernando Santos";
```

b)

```
1 SELECT *
2 FROM Game, Team
3 WHERE Game.awayteamID = Team.teamID
4     AND coach = "Fernando Santos";
```

c)

```
1 SELECT gdate, teamname
2 FROM Game, Team
3 WHERE Game.hometeamID = Team.teamID
4     AND coach LIKE "%Santos%";
```

d)

```
1 SELECT gdate, teamname
2 FROM Game, Team
3 WHERE Game.hometeamID = Goal.teamID
4     AND coach = "Fernando Santos";
```

.....

Q19: A query will calculate subtotal for an online order. It will use the columns "quantity" and "itemPrice" from a table called "ShoppingCart". It will also calculate the VAT to be paid as 20% of the amount to pay from this subtotal. Select the correct SQL to perform this calculation.

a)

```
1 SELECT (quantity*itemPrice) AS subtotal, subtotal*20/100 AS VAT
2 FROM cart;
```

b)

```
1 SELECT subtotal, (subtotal*20/100) AS VAT FROM
2 (
3 SELECT (quantity*itemPrice) AS subtotal FROM cart
4 )
5 as calculatedCart;
```

c)

```
1 SELECT (quantity*itemPrice) FROM
2 (
3 SELECT (subtotal*20/100) AS VAT FROM cart
4 )
5 as calculatedCart;
```

d)

```
1 SELECT (quantity*itemPrice) AS subtotal,
2     (quantity*itemPrice)*20/100 AS VAT
3 FROM cart;
```

.....

Q20: A query is run against the two tables shown.

competition

competitionID	country	venue	eventDate
1	UK	Southport	13/05/2018
2	Switzerland	Bern	29/08/2018
3	Canada	Montreal	08/09/2018
...

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:

```

1 SELECT country, SUM(PrizeValue)
2 FROM competition, entryticket
3 WHERE competition.competitionID = entryticket.competitionID
4 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.

.....

Q21: Identify the query which will have produced this table:

Name	Owner	Hull Type	Crew	Max Speed
Lucky Lady	H. Owen	double	3	29
Gallant	S. Scott	double	5	28
Gretel	W. Robertson	double	4	28
Fulmar II	J. Low	single	3	14
Skylark	J. Unwin	single	2	14
Jasmin	H. Owen	triple	5	24
Ocean Flyer	J. Low	triple	6	22

a)

```
1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC, Sailboat.Crew DESC;
```

b)

```
1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.Crew > 2 AND Sailboat.Crew < 7
4 AND Sailboat.'Max speed' > 13
5 ORDER BY Sailboat.'Hull Type' ASC;
```

c)

```
1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC;
```

d)

```
1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type',
   Sailboat.Crew, MAX(Sailboat.'Max speed')
2 FROM Sailboat, Owner
3 WHERE Sailboat.Crew > 2 AND Sailboat.Crew < 7
4 AND Sailboat.'Max speed' > 13
5 GROUP BY MAX(Sailboat.'Max speed')
6 ORDER BY Sailboat.'Hull Type' ASC;
```

Glossary

*AMP

A combination operating system, Apache web sever, MySQL/MariaDB and PHP commonly used to provide web services.

accuracy of output

the output generated by a solution, matches the specification exactly, based on the data available.

Answer table

this is produced when a query runs

Boolean

a value which can only be true or false

Cardinality

the type of relationship between tables in a relational database: 1:1, 1:many or many:many

Cell

the intersection of a column and a row in database table

Data

any information in a form that can be processed by a computer system

Database

a collection of information stored in a structured format that can be quickly searched or sorted

Data dictionary

a detailed list of data structures and relationships in a programming project

Date

a representation of dates as a data type

Domain

the permitted values of an attribute, for example a type of data (text, numeric, Boolean) and/or a range of numbers, dates, times

End user

the person who makes use of a system by interacting with it directly

Entity

something that exists as a particular and discrete form

Entity occurrence diagram

a diagramming tool which can be used to identify the relationships between different entity sets

Entity relationship diagram

(ERD) a data modelling technique that graphically illustrates an information system's entities and the relationships between those entities

Entity set

a collection of entities of similar form represented in a database as a table

Field

one piece of information in a record

fitness for purpose

the solution is "good enough to do the task required".

Foreign key

the primary key of a separate table that is included in a table

Functional requirements

a description of what a system must do

Grouping

gathering together the rows of a table or answer table based on a column or columns with the same value or values

Number

any representation of numbers: integers or real numbers

Primary key

an attribute of an entity that can be used to unique identify it. Within a database, a primary key is a column or columns with a unique value for each row of the database table

Product backlog

in Agile development methodologies, a product backlog details the features of the software which are to be developed

Query

performing a complex search or sort on a database

referential integrity

Defines that the relationship between two tables should always be consistent. A foreign key is a primary key value from another table and must have a related value or the database is inconsistent.

Relational database

a database that contains more than one linked table

Relational database management system

(RDBMS) a system for creating and managing relational databases

Relationship

the association between two entity sets - these are either one-to-one, one-to-many or many-to-many

Requirements specification

in interactive development approaches, the requirements specification details what the system is required to do

System owner

the organisation or individual who has commissioned a system to be developed or who has ownership of a system already in place

Text

refers to any data stored as strings of characters using ASCII, UNICODE or another text representation

Time

time represented as a data type

Validation

a rule or rules used to ensure that data entered is what was expected

Answers to questions and activities

Topic 1: Analysis

Activity: End user scenarios (page 3)

Q1: d) The showroom staff.

Q2: a) Both Amir and the holidaymakers.

Q3: c) Visitors to the library

End of Topic 1 test (page 8)

Q4: c) The system will be protected by an authentication system to prevent unauthorised access.

Q5: c) Systems analyst

Q6: b) Functional requirements define what the system must do. Non-functional requirements define how the system should behave.

Topic 2: Design**Activity: Attribute types and size (page 16)****Q1:**

Attribute	Example	Data type
Surname	Toner	Text
First name	Geraldo	Text
Postcode	TD7 0EG	Text
SQA candidate number	2311447853	Text
Enrollment fee paid	£27.50	Number
All documents received	True	Boolean
Enrolment date	23/03/14	Date
Course begins at	19:30	Time
Qualification credit value	24	Number

Activity: Validation (page 17)**Q2:** d) Range**Q3:** c) Field Length**Q4:** a) Presence check**Q5:** b) Restricted Choice**Quiz: Introduction (page 27)****Q6:** A column must have a unique name.

All the data in a column must be of the same type.

The order of columns in the table isn't important.

Q7: A flat database will only contain one table whereas a relational database will contain more than one.**Q8:** b) No

Activity: Many-to-many (2) (page 32)

Q9:

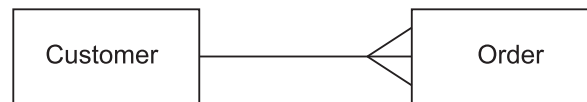
Entity 1	Entity 2	Relationship
School	Head Teacher	1-to-1
Teacher	Course	many-to-many
Classroom	Desk	1-to-many
School	Swimming pool	1-to-1
Pupil	Exam	many-to-many

Quiz: Relationships (page 32)

Q10:

1. An entity is something about which we want to store information.
2. An attribute is a characteristic of an entity, it is a piece of information about that entity.
3. A relationship defines how two entities are related. Entities can be related as one-to-one, one-to-many or many-to-many.

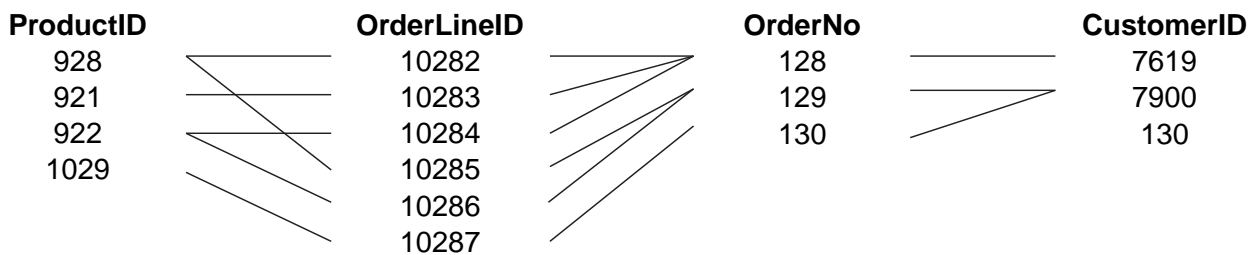
Q11:



Activity: Entity Occurrence Diagrams (page 37)

Q12:

Product 1:M OrderLine
 OrderLine M:1 Order
 Order M:1 Customer

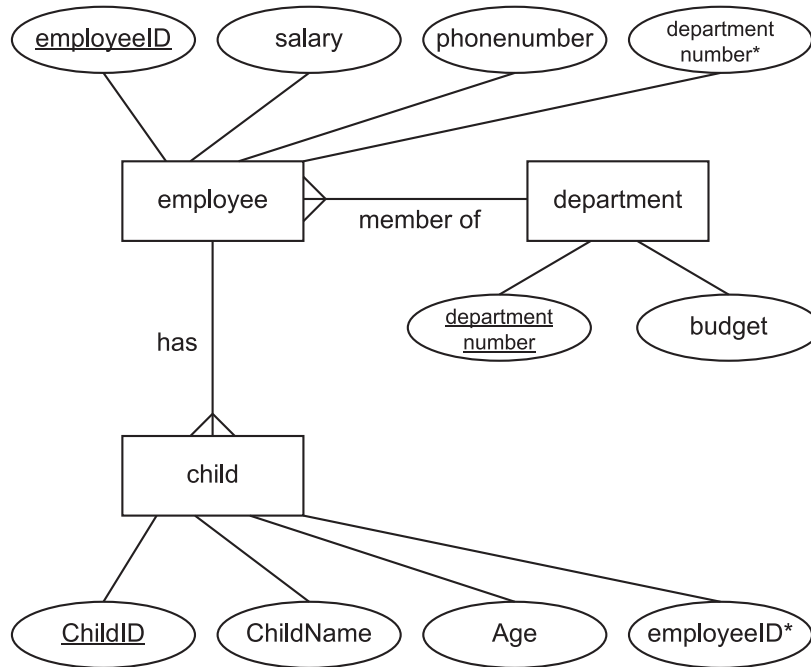


Quiz: Compound keys (page 41)

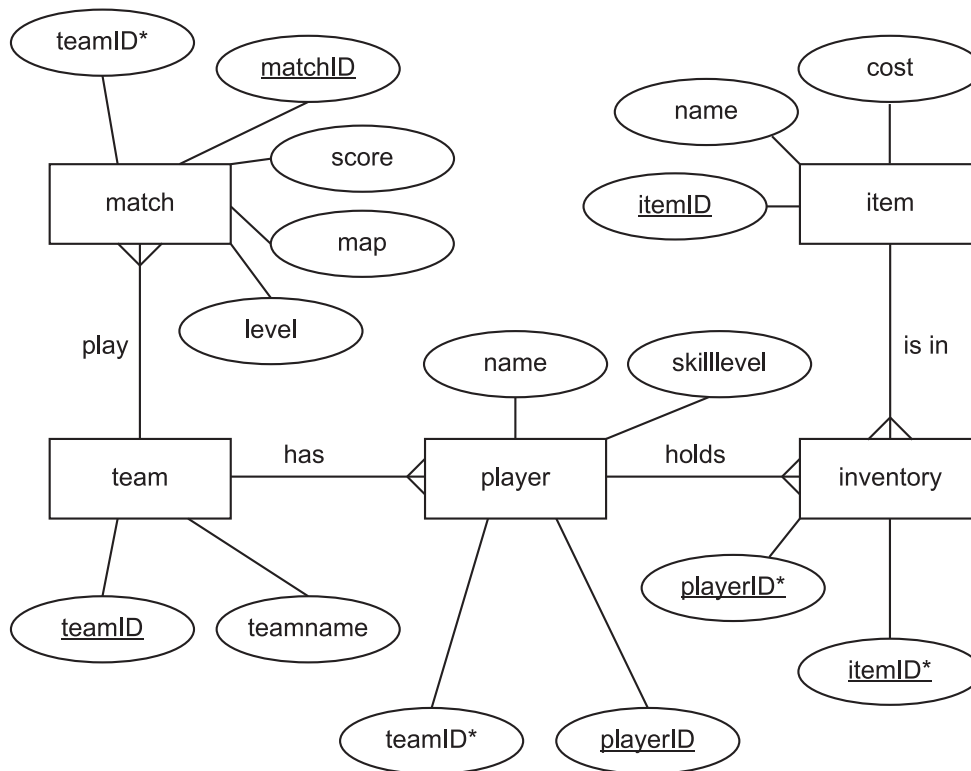
Q13: *competitionID* and *position*

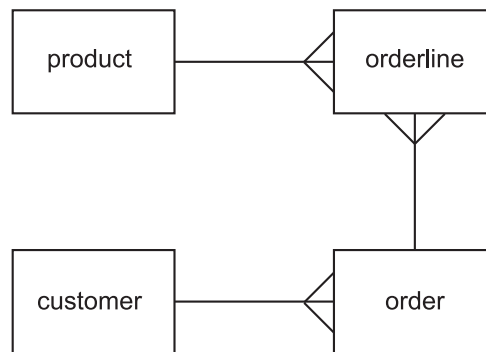
Quiz: Entity relationship diagrams (page 42)

Q14:



Q15:



Q16:**End of Topic 2 test (page 52)****Q17:** d) An attribute that has a unique value for each entity.**Q18:** a) An attribute that is the primary key of another entity set.**Q19:** b) One-to-several**Q20:** c) gathers together the rows of a table or answer table based on a column or columns with the same value or values.**Q21:** b) a combination of more than one attribute to uniquely identify an entity.

Topic 3: Implementation**Activity: Using SELECT (1) (page 62)**

Q1: SELECT * FROM Post;

Activity: Using SELECT (2) (page 62)

Q2: SELECT UserID FROM Post;

Activity: Using SELECT (3) (page 62)

Q3: SELECT UserID, PostContent FROM Post;

Activity: Using SELECT (4) (page 62)

Q4: SELECT * FROM User, Post WHERE User.UserID = Post.UserID;

Activity: Using SELECT (5) (page 63)

Q5:

```
1 SELECT * FROM User, Post
2 WHERE User.UserID = Post.UserID
3 AND PostID = 281756;
```

Activity: Using SELECT (6) (page 63)

Q6:

```
1 SELECT * FROM Post
2 WHERE PostID < 281749;
```

Activity: Using SELECT (7) (page 64)

Q7:

```
1 SELECT PostID, PostContent FROM Post
2 WHERE PostID < 281722 OR PostID > 281954;
```

Activity: Using SELECT (8) (page 64)

Q8:

```
1 SELECT AccountName, PostContent FROM User, Post
2 WHERE User.UserID = Post.UserID
3 AND (PostID = 281821 OR PostID = 281829);
```

Activity: Using SELECT (9) (page 64)**Q9:**

```
1 SELECT *
2 FROM Post
3 ORDER BY UserID ASC;
```

Activity: Using SELECT (10) (page 65)**Q10:**

```
1 SELECT *
2 FROM Post
3 ORDER BY UserID DESC, PostContent ASC;
```

Activity: Using SELECT (11) (page 65)**Q11:**

```
1 SELECT *
2 FROM Post
3 WHERE PostID > 281952
4 ORDER BY UserID ASC, PostContent DESC;
```

Activity: Using SELECT (12) (page 65)**Q12:**

```
1 SELECT User.UserID, PostID
2 FROM User, Post
3 WHERE User.UserID = Post.UserID AND User.UserID > 72656
4 ORDER BY User.UserID ASC, PostID ASC;
```

Alternative answer:

```
1 SELECT Post.UserID, PostID
2 FROM User, Post
3 WHERE User.UserID = Post.UserID AND Post.UserID > 72656
4 ORDER BY User.UserID ASC, PostID ASC;
```

Activity: Using INSERT (1) (page 67)**Q13:**

```
1 INSERT INTO User
2 VALUES (72662, "Good101");
```

Activity: Using INSERT (2) (page 67)**Q14:**

```
1 INSERT INTO Post
2 VALUES (72662, 281958, "I so need a Durr Burger Skin!!");
```

Activity: Using UPDATE (1) (page 69)**Q15:**

```
1 UPDATE User
2 SET AccountName = "RohanBear"
3 WHERE AccountName = "QuietRam";
```

Activity: Using UPDATE (2) (page 70)**Q16:**

```
1 UPDATE Post
2 SET UserID = 72645
3 WHERE UserID = 72634;
```

Activity: Using UPDATE (3) (page 70)**Q17:**

```
1 UPDATE Post
2   SET PostContent = "Best leave it unsolved."
3   WHERE PostID = 281758;
```

Activity: Using DELETE (1) (page 71)**Q18:**

```
1 DELETE FROM Post
2   WHERE PostId = 281954;
```

Activity: Using DELETE (2) (page 71)**Q19:**

```
1 DELETE FROM User
2   WHERE UserID = 72660;
3
4 DELETE FROM Post
5   WHERE UserID = 72660;
```

Activity: Using DELETE (3) (page 72)**Q20:**

```
1 DELETE FROM Post
2   WHERE PostID > 281795 AND PostID < 281802;
```

Activity: Using DELETE (4) (page 72)**Q21:**

```
1 DELETE FROM Post
2   WHERE PostID = 281838 OR PostID = 281841 OR PostID = 281843;
```

Activity: Using Wildcards (page 75)**Q22:**

```
1 SELECT id, title FROM movie
2   WHERE title LIKE "Star Wars%";
```


Q23:

```
1 SELECT id, title FROM movie
2 WHERE title LIKE "___";
```

Q24:

```
1 SELECT id, title FROM movie
2 WHERE title LIKE "%Part__";
```

Activity: Using MIN (page 80)**Q25:**

```
1 SELECT MIN(gross) FROM movie
2 WHERE budget IS NOT NULL AND gross IS NOT NULL;
```

Activity: Using MAX (page 80)**Q26:**

```
1 SELECT title, year
2 FROM movie
3 WHERE year = (SELECT MAX(year) from movie);
```

Activity: Using AVG (page 81)**Q27:**

a)

```
1 SELECT AVG(gross) FROM movie
2 WHERE gross IS NOT NULL;
```

b)

```
1 SELECT AVG(budget) FROM movie
2 WHERE year = 1975;
```

Activity: Using SUM (page 81)**Q28:**

```
1 SELECT SUM(gross) FROM movie
2 WHERE year = 1999;
```

Activity: Using COUNT (page 81)**Q29:**

```
1 SELECT COUNT(*)
2 FROM movie
3 WHERE year > 1974 AND year < 1980;
```

Activity: Using computed values (1) (page 82)**Q30:**

```
1 SELECT title, year, gross/budget AS "Ratio"
2 FROM movie
3 WHERE year = 2001 AND budget IS NOT NULL AND gross IS NOT NULL AND
   gross/budget = (
4 SELECT MAX(gross/budget)
5 FROM movie
6 WHERE year = 2001 AND budget IS NOT NULL AND gross IS NOT NULL
7 );
```

Activity: Using computed values (2) (page 83)**Q31:**

```
1 SELECT title, budget*15/100 AS "Tax due"
2 FROM movie
3 WHERE title LIKE "%Gold%" AND budget IS NOT NULL
4 AND gross IS NOT NULL;
```

Activity: Using GROUP BY (page 85)**Q32:**

```
1 SELECT year, SUM(budget) AS "Movie Budget"
2 FROM movie
3 WHERE budget IS NOT NULL
4 GROUP BY year;
```

Q33:

```
1 SELECT title, COUNT(*) AS "Total Cast"
2 FROM movie, casting
3 WHERE movie.id = casting.movieid AND budget > 180000000
4 GROUP BY title;
```

Q34:

```
1 SELECT name, count(*) AS "Cast"
2 FROM actor, casting, movie
3     WHERE actor.id = casting.actorid AND
4           casting.movieid = movie.id AND
5           ord=1
6 GROUP BY actor.id
7 ORDER BY Cast DESC;
```

Activity: Using ORDER BY (page 85)**Q35:**

```
1 SELECT name, Count(*) as "NumberOfMovies", year
2 FROM actor, casting, movie
3 WHERE actor.id = casting.actorid AND movie.id = casting.movieid
4 GROUP BY name, year
5 ORDER by NumberOfMovies DESC, year ASC;
```

Q36:

```
1 SELECT title, COUNT(*) AS "Total Cast"
2 FROM movie, casting
3 WHERE movie.id = casting.movieid AND budget > 180000000
4     GROUP BY title
5     ORDER BY title ASC;
```

Q37:

```
1 SELECT *
2 FROM movie
3     ORDER BY year ASC, budget DESC, gross DESC;
```

End of topic 3 test (page 87)

Q38: a) SELECT * FROM branch WHERE city = 'London' OR city='Glasgow';

Q39: a)

```
1 SELECT staffno, lname
2 FROM staff, branch
3 WHERE staff.branchno = branch.branchno
4 AND branch.street = '13 Main Road';
```

Q40: c)

```
1 SELECT * FROM branch, propertyForRent
2 WHERE branchno.city = 'Edinburgh'
3 AND propertyForRent.branchno = branch.branchno;
```

Q41: d)

```
1 SELECT fname, lname, MAX(annualsalary) FROM staff, branch
2 WHERE staff.branchno = branch.branchno AND city = 'Glasgow';
```

Q42: b) SELECT * FROM client WHERE maxrent < 800;

Q43: a) SELECT * FROM developer WHERE firstname LIKE 'a%';

Q44: c) SELECT * FROM developer ORDER BY firstname DESC;

Q45: b) DELETE FROM developer WHERE firstname = 'Peter';

Q46: c)

type	most expensive
SUV	73000
Coupe	75000

Q47: c) To group results by country so that each country only appears once.

Topic 4: Testing and evaluation**Activity: SQL operations (page 97)**

Q1: c)

username	platform
shocker	X-station
peach	S-box
peach	X-station
shocker	PC
destroyer	PC
peach	S-box
destroyer	S-box

Q2: a)

game	username	email	message	MAX(score)
Massive RPG	shocker	paul@gamers.org	I want to play competitively	821200
SuperJoe	peach	sally@scott.com	I'm part of an eSports team	625100
Terra 1999	destroyer	chloe@coders.org	I love games	299000

Q3: d)

username	realname	password	email	message	terms_and_conditions
destroyer	Chloe Davidson	shadow99	chloe@coders.org	I love games	on
peach	Sally McDonald	trustme1	sally@scott.org	I'm part of an eSports team	on

End of topic 4 test (page 102)

Q4: c) the solution is good enough to meet the required need.

Q5: b)

```

1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type', Sailboat.Crew,
   Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC;
```

Q6: a)

```
1 SELECT 'Customer Name', SUM('Price of item')
2 FROM Sales
3 GROUP BY 'Customer Name';
```

Q7: d) the output generated exactly matches the requirements.

Topic 5: End of unit test

End of unit 3 test (page 106)

Q1: c)

```

1 SELECT JobID
2 FROM Job J, AllocatedJob AJ, Driver D
3 WHERE J.JobID = AJ.JobID
4     AND AJ.driverID=D.DriverID
5     AND D.name = 'Colin Shaw';
    
```

Q2: b)

```

1 SELECT Driver.name, Driver.vehicle, 'date', Job.JobID, collectionAddress
2 FROM Driver, AllocatedJob, Job
3 WHERE Driver.driverID=AllocatedJob.driverID
4     AND AllocatedJob.JobID = Job.JobID
5     AND Driver.driverID = 1921
6     AND collectionAddress LIKE "%Aberdeen%"
7 ORDER BY 'date' ASC, 'time' ASC;
    
```

Q3: a) End users are the actual users who will operationally make use of the system.

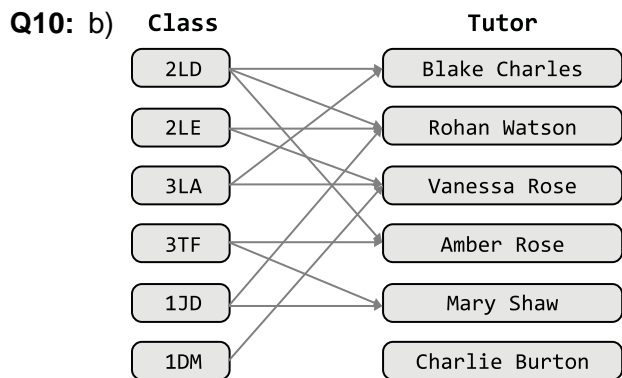
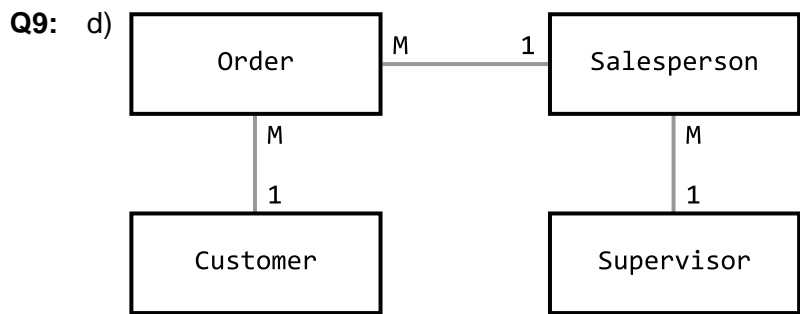
Q4: d) Customers and Paul Jackson

Q5: a) are specific actions that the system must perform.

Q6: c) product backlog.

Q7: b) Functional requirements define what the system must do. Non-functional requirements define how the system should behave.

Q8: c) One-to-one



Q11: c) MAX, AVG, SUM, COUNT

Q12: d) An attribute that has a unique value for each entity.

Q13: a) An attribute that is the primary key of another entity set.

Q14: b) a combination of more than one attribute to uniquely identify an entity.

Q15: b)

```
1 SELECT player, teamname, coach, gtime
2 FROM goal, game, team
3 WHERE game.gameID = goal.gameID
4     AND goal.teamID = team.teamID
5 AND gtime <= 10;
```

Q16: d)

```
1 SELECT player, stadium, COUNT(*) as Goals
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4 GROUP BY player, stadium;
```

Q17: a)

```
1 SELECT gameID, player
2 FROM Game, Goal
3 WHERE game.gameID = goal.gameID
4     AND teamID = "IRL";
```

Q18: a)

```
1 SELECT gdate, teamname
2 FROM Game, Team
3 WHERE Game.hometeamID = Team.teamID
4     AND coach = "Fernando Santos";
```


Q19: b)

```
1 SELECT subtotal, (subtotal*20/100) AS VAT FROM
2 (
3 SELECT (quantity*itemPrice) AS subtotal FROM cart
4 )
5 as calculatedCart;
```

Q20: c) To group results by country so that each country only appears once.

Q21: c)

```
1 SELECT Sailboat.name, Owner.Owner, Sailboat.'Hull Type', Sailboat.Crew,
   Sailboat.'Max speed'
2 FROM Sailboat, Owner
3 WHERE Sailboat.OwnerID = Owner.OwnerID
4 AND Sailboat.Crew > 2 AND Sailboat.Crew < 7
5 AND Sailboat.'Max speed' > 13
6 ORDER BY Sailboat.'Hull Type' ASC;
```