Advanced Higher
Course
Specification

SQA

# Advanced Higher Computing Science

| Course code: | C816 77 |
|---|---|
| Course assessment code: | X816 77 |
| SCQF: | level 7 (32 SCQF credit points) |
| Valid from: | session 2024-25 |

This document provides detailed information about the course and course assessment to ensure consistent and transparent assessment year on year. It describes the structure of the course and the course assessment in terms of the skills, knowledge and understanding that are assessed.

This document is for teachers and lecturers and contains all the mandatory information required to deliver the course.

This edition: September 2024 (version 3.2)

# Contents

# Course overview

This course consists of 32 SCQF credit points, which includes time for preparation for course assessment. The notional length of time for candidates to complete the course is 160 hours.

The course assessment has two components.

| Component | Marks | Duration |
|---|---|---|
| Question paper | 55 | 2 hours |
| Project | 80 | see 'Course assessment' section |

| Recommended entry | Progression |
|---|---|
| Entry to this course is at the discretion of the centre.<br><br>Candidates should have achieved the Higher Computing Science course or equivalent qualifications and/or experience prior to starting this course. | ♦ a range of computing-related Higher National Diplomas (HNDs)<br>♦ degrees in computing science or related disciplines<br>♦ careers in computing, IT and/or related areas<br>♦ further study, employment and/or training |

## Conditions of award

The grade awarded is based on the total marks achieved across both course assessment components.

# Course rationale

National Courses reflect Curriculum for Excellence values, purposes and principles. They offer flexibility, provide time for learning, focus on skills and applying learning, and provide scope for personalisation and choice.

Every course provides opportunities for candidates to develop breadth, challenge and application. The focus and balance of assessment is tailored to each subject area.

This course highlights the central role of computing professionals as creative problem-solvers and designers, able to conceive, design, implement, and operate complex systems. It provides candidates with an understanding of contemporary computing technologies, and develops a wide range of practical skills that underpin our modern, digital world.

The course also builds awareness of the importance of computing in meeting our needs today and for the future, in many fields including science, education, business, and industry. Many organisations regard computing skills as vital to their growth and sustainability, while a growing number of individuals use computing technologies as a way to create entrepreneurial, social and enterprise-building opportunities.

# Purpose and aims

The course provides a broad and challenging exploration of computing technologies, focusing on developing advanced programming and research skills. Candidates learn to apply a rigorous approach to the design and development process.

The course enables candidates to:

♦ understand and apply computational-thinking skills across a range of computing contexts
♦ extend and apply knowledge and understanding of advanced concepts and processes in computing science
♦ apply skills and knowledge in analysis, design, development, implementation, testing, and evaluation to a range of digital solutions with increasingly complex aspects
♦ apply creative problem-solving skills across a range of contexts
♦ develop autonomous learning, investigative, and research skills
♦ communicate advanced computing concepts clearly and concisely, using appropriate terminology
♦ develop an informed understanding of the role and impact of computing technologies in influencing our environment and society

# Who is this course for?

The course is suitable for candidates interested in exploring the role and impact of contemporary computing technologies. It provides a pathway for those who want to progress to more specialised training, further education, or entry into a diverse range of occupations

and careers, such as software programming and/or engineering, databases, and web design and development.

The skills in the course are transferable to all areas of computing-related study including robotics, artificial intelligence, e-commerce, networking, cyber security, and systems analysis and testing.

# Course content

The course has three areas of study:

### Software design and development

Candidates develop knowledge, understanding, and advanced practical problem-solving skills in software design and development. They do this by using appropriate software development environments. Candidates develop object-oriented programming and computational-thinking skills by analysing, designing, implementing, testing, and evaluating practical solutions and explaining how these modular programs work. They use their knowledge of data types and constructs to create efficient programs to solve advanced problems.

### Database design and development

Candidates develop knowledge, understanding, and advanced practical problem-solving skills in database design and development. They do this through a range of practical tasks, using SQL to create and query relational databases. Candidates apply computational-thinking skills to analyse, design, implement, test, and evaluate practical solutions, using a range of development tools. Candidates apply interpretation skills to tasks involving some complex features in both familiar and new contexts.

### Web design and development

Candidates develop knowledge, understanding, and advanced practical problem-solving skills in web design and development. They do this through a range of practical and investigative tasks. Candidates apply computational-thinking skills to analyse, design, implement, test, and evaluate practical solutions to web-based problems, using a range of development tools including HTML, Cascading Style Sheets (CSS) and PHP. Candidates apply interpretation skills to tasks involving some complex features in both familiar and new contexts.

### Integration

The integration of technologies is central to the course. Teachers and lecturers should consider candidates' previous experience in 'Database design and development' and 'Web design and development' when planning delivery. This will ensure candidates are prepared for the integration that is required for the question paper and project assessment components. These requirements are set out in 'Course assessment structure: question paper' on pages 14–17 and in the Coursework Assessment Task.

# Skills, knowledge and understanding

## Skills, knowledge and understanding for the course

The following provides a broad overview of the subject skills, knowledge and understanding developed in the course:

♦ applying computational thinking to solve complex computing problems

♦ analysing complex problems within computing science, across a range of contemporary contexts

♦ designing, developing, implementing, testing, and evaluating digital solutions (including computer programs) to complex problems across a range of contexts

♦ developing advanced skills in computer programming and the ability to communicate how a program works

♦ communicating an understanding of complex concepts related to computing science design and development, clearly and concisely, using appropriate terminology

♦ knowledge and understanding of the role and impact of contemporary computing technologies on the environment and society

## Skills, knowledge and understanding for the course assessment

The following provides details of skills, knowledge and understanding sampled in the course assessment.

| | Software design and development | Database design and development | Web design and development |
|---|---|---|---|
| Analysis | Identify the purpose and functional requirements of a problem that relates to the design and implementation at this level in terms of:<br><br>♦ inputs<br>♦ processes<br>♦ outputs<br><br>Describe, exemplify and implement research for:<br><br>♦ feasibility studies:<br>— economic<br>— time<br>— legal<br>— technical<br>— user surveys<br><br>Describe, exemplify and implement planning in terms of:<br><br>♦ scheduling<br>♦ resources<br>♦ Gantt charts | | |

| | Software design and development | Database design and development | Web design and development |
|---|---|---|---|
| Analysis (continued) | Produce requirement specifications for end users and develop:<br><br>♦ end-user requirements<br>♦ scope, boundaries and constraints<br>♦ functional requirements<br><br>Describe, exemplify and implement Unified Modelling Language (UML):<br><br>♦ use case diagrams:<br>  — actors<br>  — use cases<br>  — relationships | | |
| Design | Identify the data types and structures required for a problem that relates to the implementation at this level.<br><br>Read and understand designs of solutions to problems at this level using the following design techniques:<br><br>♦ structure diagrams<br>♦ pseudocode<br>♦ UML | Describe, exemplify and implement entity-relationship diagrams with three or more entities indicating:<br><br>♦ entity name<br>♦ entity type (strong, weak)<br>♦ attributes<br>♦ relationship participation (mandatory, optional)<br>♦ name of relationship<br>♦ cardinality<br><br>Identify relationship participation from an entity-occurrence diagram. | Describe, exemplify and implement wireframe designs showing:<br><br>♦ visual layout<br>♦ navigation<br>♦ consistency<br>♦ underlying processes<br><br>Describe, exemplify and implement low-fidelity prototype from wireframe design. |

|  | **Software design and development** | **Database design and development** | **Web design and development** |
|---|---|---|---|
| Design (continued) | Exemplify and implement efficient design solutions to a problem at this level, using pseudocode, showing:<br><br>♦  top-level design<br>♦  the data flow<br>♦  refinements<br><br>Describe, exemplify and implement UML for the following:<br><br>♦  class diagrams:<br>— class name<br>— instance variables and data types<br>— methods<br>— public and private<br>— inheritance<br>— constructor<br>— array of objects | Describe, exemplify and implement surrogate keys.<br><br>Describe and exemplify a data dictionary, in relation to SQL, with three or more entities for the following:<br><br>♦  entity name<br>♦  attribute name<br>♦  primary and foreign key<br>♦  attribute type:<br>— varchar<br>— integer<br>— float<br>— date<br>— tie<br>♦  attribute size<br>♦  validation:<br>— presence check<br>— restricted choice<br>— field length<br>— range | Read and understand designs of server-side processes at this level, using the following techniques:<br><br>♦  structure diagrams<br>♦  pseudocode<br><br>Exemplify and implement the design of server-side processes using pseudocode. |

| | Software design and development | Database design and development | Web design and development |
|---|---|---|---|
| Design (continued) | Describe, exemplify and implement user-interface design using a wireframe, indicating:<br><br>♦ visual layout<br>♦ inputs<br>♦ validation<br>♦ underlying processes<br>♦ outputs | Exemplify a design of a solution query using:<br><br>♦ tables and queries<br>♦ fields<br>♦ search criteria<br>♦ sort order<br>♦ calculations<br>♦ grouping<br>♦ having | |
| Implementation | **Data types and structures**<br>Describe, exemplify, and implement the following structures in solutions to problems at this level:<br><br>♦ parallel 1-D arrays<br>♦ records<br>♦ arrays of records<br>♦ 2-D arrays<br>♦ array of objects<br><br>Describe and exemplify the operation of linked lists (double and single). | **SQL**<br>Implement relational database using SQL Data Definition Language (DDL) and Data Manipulation Language (DML) to match the design. | **CSS**<br>Describe, exemplify, and implement responsive pages using the following media queries:<br><br>♦ media type:<br>— print<br>— screen<br>♦ media feature:<br>— max-width |

| | Software design and development | Database design and development | Web design and development |
|---|---|---|---|
| Implementation (continued) | **Computational constructs**<br><br>Describe, exemplify, and implement the following object-oriented constructs:<br><br>♦ object<br>♦ property<br>♦ method<br>♦ class<br>♦ sub-class<br>♦ encapsulation<br>♦ inheritance<br>♦ instantiation<br>♦ polymorphism<br><br>Describe, exemplify, and implement code to:<br><br>♦ open and close connection to database server<br>♦ execute SQL query<br>♦ format query results | Describe, exemplify, and implement the following SQL operations:<br><br>♦ CREATE statement:<br>— CREATE DATABASE<br>— CREATE TABLE<br>constraints:<br>   o primary key<br>   o foreign key<br>   o not null<br>   o check<br>   o auto increment<br>♦ DROP statement:<br>— DROP DATABASE<br>— DROP TABLE<br>♦ HAVING clause of the SELECT statement<br>♦ subqueries used with the WHERE clause of SELECT statements<br>♦ data types:<br>— varchar<br>— integer<br>— float<br>— date<br>— time | **HTML**<br><br>Describe, exemplify, and implement form elements including:<br><br>♦ FORM element:<br>— action<br>— method (get and post)<br>♦ INPUT, SELECT and TEXTAREA elements:<br>— name<br>— value<br>♦ TABLE element:<br>— th, tr, td<br><br><br>**PHP**<br><br>Describe, exemplify, and implement coding of server-side processing to:<br><br>♦ assign form data to server-side variables:<br>— $_get()<br>— $_post() |

| | **Software design and development** | **Database design and development** | **Web design and development** |
|---|---|---|---|
| Implementation (continued) | **Algorithm specification**<br><br>Describe, exemplify, and implement standard algorithms including:<br><br>♦ binary search<br>♦ insertion sort<br>♦ bubble sort<br><br>Read and explain code that uses constructs appropriate to this level. | ♦ logical operators:<br>— IN<br>— NOT<br>— BETWEEN<br>— ANY<br>— EXISTS<br><br>Read and explain code that uses the SQL at this level. | ♦ open and close connection to database server:<br>— die()<br>— mysqli_connect()<br>— mysqli_close()<br>♦ execute SQL query:<br>— mysqli_query()<br>♦ format query results:<br>— echo<br>— mysqli_fetch_array()<br>— mysqli_num_row()<br><br>and:<br><br>♦ assignment, repetition and selection using server-side local and global variables<br>♦ sessions:<br>— session_start()<br>— session_destroy()<br><br>Read and explain code that uses constructs appropriate to this level. |

|  | **Software design and development** | **Database design and development** | **Web design and development** |
|---|---|---|---|
| Testing | Describe, exemplify and implement the following:<br><br>♦ integrative testing<br>♦ usability testing based on prototypes<br>♦ final testing<br>♦ end-user testing | | |
| | and:<br><br>♦ component testing during the development of the solution | and:<br><br>♦ SQL-implemented tables match design<br>♦ SQL operations work correctly at this level | |
| Evaluation | Evaluate solution in terms of:<br><br>♦ fitness for purpose<br>♦ maintainability<br>  — perfective<br>  — corrective<br>  — adaptive<br>♦ robustness | | |
| | and:<br><br>♦ efficiency<br>♦ usability | and:<br><br>♦ accuracy of output | and:<br><br>♦ usability |

Skills, knowledge and understanding included in the course are appropriate to the SCQF level of the course. The SCQF level descriptors give further information on characteristics and expected performance at each SCQF level, and are available on the SCQF website.

# Skills for learning, skills for life and skills for work

This course helps candidates to develop broad, generic skills. These skills are based on [SQA's Skills Framework: Skills for Learning, Skills for Life and Skills for Work](#) and draw from the following main skills areas:

**2      Numeracy**

2.3    Information handling

**3      Health and wellbeing**

3.1    Personal learning

**4      Employability, enterprise and citizenship**

4.2    Information and communication technology (ICT)

**5      Thinking skills**

5.3    Applying
5.4    Analysing and evaluating

Teachers and lecturers must build these skills into the course at an appropriate level, where there are suitable opportunities.

# Course assessment

Course assessment is based on the information in this course specification.

The course assessment meets the purposes and aims of the course by addressing:

♦ breadth — drawing on knowledge and skills from across the course
♦ challenge — requiring greater depth or extension of knowledge and/or skills
♦ application — requiring application of knowledge and/or skills in practical or theoretical contexts as appropriate

This enables candidates to apply:

♦ knowledge and skills from across the course to plan, analyse, design, implement, test and evaluate a solution to solve an appropriately challenging practical computing science problem
♦ breadth of knowledge from across the course, and depth of understanding, to answer appropriately challenging questions in computing science contexts

## Course assessment structure: question paper

### Question paper                                                              55 marks

The question paper gives candidates the opportunity to:

♦ apply computational thinking to solve complex computing problems
♦ analyse complex problems within computing science, across a range of contemporary contexts
♦ design, develop, implement, test, and evaluate digital solutions (including computer programs) to complex problems across a range of contexts
♦ communicate how a well-structured, complex, modular program works
♦ demonstrate understanding of complex concepts relating to computing science design and development by communicating clearly and concisely, using appropriate terminology
♦ demonstrate knowledge and understanding of key aspects of contemporary project planning and management
♦ demonstrate knowledge and understanding of object-oriented programming

The question paper has 55 marks, which is approximately 40% of the overall marks for the course assessment (135 marks).

The question paper has three sections. Section 1 is mandatory, and candidates have the option to complete wither section 2 or section 3.

♦ Section 1: Software design and development — 35 marks
♦ Section 2: Database design and development — 20 marks
♦ Section 3: Web design and development — 20 marks

Each section begins with a number of short, stand-alone questions. These are predominantly 'C' mark questions, based on Advanced Higher concepts, presented in a clear and concise way, in a simple and/or familiar context.

This is followed by more challenging, context-based questions with multiple subparts. These require a range of responses including restricted and extended response, designing solutions and writing code, and feature both 'C' mark and 'A' mark questions. Some questions are designed to be more challenging and feature higher-order Advanced Higher concepts, such as the integration of technologies or understanding and/or designing solutions to complex, unfamiliar problems.

The questions will:

♦ require candidates to understand and design solutions to complex, unfamiliar problems
♦ be set in meaningful contexts that require candidates to provide some descriptions and explanations
♦ provide integration by drawing on understanding from other areas of the course
♦ sample across the course in a balanced way

**Integration**
The 'Database design and development' and 'Web design and development' sections will each contain a question set in the context of a database-driven website. Part of this question will require some integration with the other option. The tables below detail what could be asked in the question paper.

For 'Database design and development', candidates will need to be familiar with the following 'Web design and development' skills, knowledge and understanding so they can design and implement HTML forms.

**Design**
Describe, exemplify, and implement wireframe designs showing:

♦ visual layout
♦ navigation
♦ consistency
♦ underlying processes

**Implementation**

Describe, exemplify, and implement form elements including:

♦ FORM element:
— action
— method (get and post)
♦ INPUT, SELECT and TEXTAREA elements:
— name
— value

Describe, exemplify and implement form elements:

♦ form element: input
— text
— number
— textarea
— radio
— submit
♦ form element: select

Describe, exemplify and implement form data validation:

♦ length
♦ presence
♦ range

Read and explain code that makes use of the above HTML.

For 'Web design and development', candidates will need to be familiar with the following 'Database design and development' skills, knowledge and understanding in order that they can implement SQL queries:

♦ select:
— from
— where:
o AND, OR, <, >, =
o order by with a single field
— use of MAX, MIN, AVG, COUNT and SUM to return a single value
♦ insert
♦ update
♦ delete

Read and explain code that makes use of the above SQL.

### SQA's standardised reference language

Questions assessing understanding and application of programming skills are expressed using SQA's standardised reference language. Further information can be found in the document *Reference language for Computing Science question papers,* which can be downloaded from the Advanced Higher Computing Science subject page on SQA's website.

Where candidates need to answer by writing code, answers may be expressed using any programming language. Candidates are not expected to write code in SQA's standardised reference language. Marks are awarded for demonstrating understanding, not for the correct use of syntax.

### Setting, conducting and marking the question paper

SQA sets and marks the question paper. It is conducted in centres under conditions specified for external examinations by SQA.

Candidates have 2 hours to complete the question paper.

Specimen question papers for Advanced Higher courses are published on SQA's website. These illustrate the standard, structure and requirements of the question papers. The specimen papers also include marking instructions.

# Course assessment structure: project

## Project                                                                          80 marks

The project gives candidates the opportunity to:

♦ apply computational thinking to solve a complex computing problem
♦ analyse a complex problem within a computing science context
♦ design, develop, implement, test, and evaluate a digital solution to a complex problem
♦ demonstrate advanced skills in computer programming
♦ communicate understanding of complex concepts related to computing science, clearly and concisely, using appropriate terminology

The project is designed to allow candidates to demonstrate their ability to work independently.

The project must:

♦ be based on one of the following study areas of the course:
  — software design and development
  — database design and development
  — web design and development
♦ include at least two concepts from this area of the course
♦ integrate with one of the other two areas of the course

It is important for teachers and lecturers to discuss potential project ideas with candidates to ensure that they meet the criteria for the Advanced Higher project, and are achievable within the constraints of time, expertise and resources available.

The project has 80 marks, which is approximately 60% of the overall marks for the course assessment (135 marks).

Candidates gain marks for the following stages of the project:

| | |
|---|---|
| ♦ analysis of the problem | 10 marks |
| ♦ design of the solution | 20 marks |
| ♦ implementation | 30 marks |
| ♦ testing the solution | 15 marks |
| ♦ evaluation of the solution | 5 marks |

## Setting, conducting and marking the project

The project is:

♦ an open brief — candidates choose the topic for their project in discussion with their teacher or lecturer
♦ conducted under some supervision and control
♦ submitted to SQA for external marking

## Assessment conditions

### Time

There is no time limit for the project. It is recommended that the project is completed within 40 hours. This can be broken down for each section as follows:

♦ Analysis — 5 hours
♦ Design — 10 hours
♦ Implementation — 15 hours
♦ Testing — 8 hours
♦ Evaluation — 2 hours

Candidates should start at an appropriate point in the course.

### Supervision, control and authentication

The project is conducted under some supervision and control.

Candidates can complete part of the work outwith the learning and teaching setting; therefore, teachers and lecturers must exercise professional responsibility to ensure that evidence submitted by a candidate is their own work.

### Resources

This is an open-book assessment. Candidates can access any appropriate resources.

**Reasonable assistance**

Candidates must carry out the assessment independently. However, teachers and lecturers can provide reasonable assistance prior to, and during, the formal assessment process.

Teachers and lecturers should advise candidates on their choice of problem. This is to ensure that their chosen problem meets the criteria for the Advanced Higher project and is achievable.

Candidates must work independently once the formal assessment process has started, with teacher and lecturer input limited to constructive comment and/or questioning.

Once projects are completed and submitted, they must not be returned to candidates for further work.

## Evidence to be gathered

Candidate evidence includes program listings, screenshots, web page source files, data files or similar, as appropriate.

**Volume**

There is no word count. The project should have no more than 24 functional requirements. This will ensure that the volume of evidence is not excessive.

# Grading

Candidates' overall grades are determined by their performance across the course assessment. The course assessment is graded A–D based on the total mark for both course assessment components.

**Grade description for C**

For the award of grade C, candidates will typically have demonstrated successful performance in relation to the skills, knowledge and understanding for the course.

**Grade description for A**

For the award of grade A, candidates will typically have demonstrated a consistently high level of performance in relation to the skills, knowledge and understanding for the course.

# Equality and inclusion

This course is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

Guidance on assessment arrangements for disabled candidates and/or those with additional support needs is available on the assessment arrangements web page: [www.sqa.org.uk/assessmentarrangements](www.sqa.org.uk/assessmentarrangements).

# Further information

- [Advanced Higher Computing Science subject page](#)
- [Assessment arrangements web page](#)
- [Building the Curriculum 3–5](#)
- [Guidance on conditions of assessment for coursework](#)
- [SQA Skills Framework: Skills for Learning, Skills for Life and Skills for Work](#)
- [Educational Research Reports](#)
- [SQA e-assessment web page](#)
- [SCQF website: framework, level descriptors and SCQF Handbook](#)

# Appendix: course support notes

## Introduction

These support notes are not mandatory. They provide advice and guidance to teachers and lecturers on approaches to delivering the course. Please read these course support notes in conjunction with the course specification and the specimen question paper and coursework.

## Approaches to learning and teaching

At Advanced Higher, a significant amount of learning may be self-directed and require candidates to demonstrate initiative and work on their own.

Some candidates may find this challenging, so it is important that you have strategies in place to support them, for example planning time for regular feedback sessions and/or discussions on a one-to-one or group basis.

You should encourage candidates to use an enquiring, critical and problem-solving approach to their learning. Give them the opportunity to practise and develop research and investigation skills, and higher-order evaluation and analytical skills.

Where possible, provide opportunities to personalise learning to enable candidates to have choices in approaches to learning and teaching. The flexibility in the Advanced Higher course and the independence with which candidates carry out the work lends itself to this.

Encourage candidates to participate fully in active learning and practical activities by working together, analysing, investigating, debating and evaluating topics, problems and solutions, while you act increasingly as a facilitator.

You should use an appropriate balance of teaching methodologies when delivering the course. A variety of active learning approaches is encouraged, including the following:

### Activity-based learning

You should balance whole-class, direct teaching opportunities with activity-based learning using practical tasks. An investigatory approach is encouraged, with candidates actively involved in developing their skills, knowledge and understanding by investigating a range of real-life and relevant problems and solutions related to areas of study. You should support learning with appropriate practical activities, so that skills are developed simultaneously with knowledge and understanding.

### Group work

Practical activities and investigations lend themselves to group work, and you should encourage this. Candidates engaged in collaborative group working strategies can capitalise on one another's knowledge, resources and skills by questioning, investigating, evaluating and presenting ideas to the group. Working as a team is a fundamental aspect of working in the IT and related industries, and so should be encouraged and developed.

## Problem-based learning

Problem-based learning (PBL) is another approach that can support candidates to progress through the course. This method may be best utilised at the end of a topic, where additional challenge is required to ensure candidates are secure in their knowledge and understanding, and to develop the ability to apply knowledge and skills in less familiar contexts. Learning through PBL develops skills in problem solving, decision making, investigation, creative thinking, team working and evaluation.

## Computational thinking

Computational thinking is recognised as a key skill set for all 21st century candidates — whether they intend to continue with computing science or not. It involves a set of problem-solving skills and techniques used by software developers to write programs.

There are various ways of defining computational thinking. One useful structure is to group these problem-solving skills and techniques under five broad headings (concepts):

♦ **Abstraction**: seeing a problem and its solution at many levels of detail and generalising the necessary information. Abstraction allows us to represent an idea or a process in general terms (for example variables) and use it to solve other problems that are similar in nature.

♦ **Algorithms**: the ability to develop a step-by-step strategy for solving a problem. Algorithm design is often based on the decomposition of a problem and the identification of patterns that help to solve the problem. In computing science as well as in mathematics, algorithms are often written abstractly, utilising variables in place of specific numbers.

♦ **Decomposition**: breaking down a task so that we can clearly explain a process to another person — or to a computer. Decomposing a problem frequently leads to pattern recognition and generalisation/abstraction, and ultimately the ability to design an algorithm.

♦ **Pattern recognition**: the ability to notice similarities or common differences that help us make predictions or lead us to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.

♦ **Generalisation**: realising that we can use a solution to one problem to solve a whole range of related problems.

Underpinning all of these concepts is the idea that computers are **deterministic**: they do exactly what we tell them to do and so can be understood.

Computational thinking can be a component of many subjects; computing science delivers this particularly well. You are encouraged to emphasise, exemplify and make these aspects of computational thinking explicit, wherever there are opportunities to do so throughout the teaching and learning of this course.

## Using online and outside resources

Stimulating interest and curiosity should be a prime objective when teaching this course. Engaging with outside agencies or industry professionals can greatly enhance the learning process. Online resources can provide a valuable addition to teaching and learning activities,

encouraging research, collation and storage of information and evaluation of these materials. Using interactive multimedia learning resources, online quizzes, and web-based software can also support teacher-led approaches.

Blending assessment activities with learning activities throughout the course can support learning, for example:

♦ sharing learning intentions and/or success criteria
♦ using assessment information to set learning targets and next steps
♦ adapting teaching and learning activities based on assessment information
♦ boosting confidence by providing supportive feedback

If appropriate, you should encourage self-assessment and peer-assessment techniques.

**Meeting the needs of all candidates**

Within any class, each candidate has individual strengths and areas for improvement. If there are candidates capable of achieving a higher level in some aspects of the course, you should give them the opportunity to do so, where possible. Advanced Higher is particularly suited to candidates researching knowledge and developing skills beyond the course requirements.

Where Advanced Higher candidates have studied National 5 and Higher in previous years, it is important that you provide them with new and different contexts for learning to avoid demotivation. For example, candidates could work in a different type of development environment or language at Advanced Higher. You should also consider candidates' previous experience in 'Database design and development' and 'Web design and development' when planning delivery of integration across the different areas of the course.

## Suggested learning activities

The course is structured around three areas of study.

Some aspects of analysis, testing and evaluation apply to all three practical areas of the course (SDD, DDD and WDD), as well as solutions to problems that integrate these technologies.

You are encouraged to use an investigatory approach, with candidates actively involved in developing their skills, knowledge and understanding of a range of development problems and solutions.

**Development methodologies**

♦ Working in groups, candidates could discuss using an agile methodology, compared to an iterative development process. This is not assessed in the course but candidates have to decide which approach they will follow for their project.

**Analysis (SDD, DDD and WDD)**

♦ Working individually or in groups, candidates could analyse a number of problems by creating a use case diagram, and deciding on purpose and functional requirements.

These could be SDD, DDD and WDD problems, as well as problems that integrate these areas (which is a requirement of the project).

♦ Working individually or in groups, candidates could prepare requirements specifications for end users.

## Software design and development

♦ **Design:**
  — You could present candidates with a variety of completed requirement specifications, and ask them to complete the top level algorithm, data flow, and Unified Modelling Language (UML) class diagram for each problem.
  — Candidates could then design user interfaces using wireframes annotated with underlying processes, inputs (including any necessary validation) and outputs.

♦ **Implementation:**
  — You could provide candidates with working programs that demonstrate the use of object-oriented programming techniques, including classes and methods.
  — Ask candidates to identify and explain sections of code from within these programs.
  — Using the pre-defined functions stated in the course content, candidates could tackle a number of problems.
  — Using appropriate programs created in Higher, candidates could think about how they could use their knowledge of 2-D arrays and arrays of objects to implement them using the new data structures.
  — Working in groups, candidates could write code from designs provided in pseudocode, structure diagrams or UML class diagrams. This would help them implement object-oriented code.
  — Using a range of working programs that use a variety of standard algorithms, candidates could interpret and explain what is happening in the code. This would help them develop their own modular programs that use these constructs and standard algorithms.
  — You could demonstrate how a program language is used to create a link to a database and execute an SQL statement. You could then give candidates a sequence of problems that requires them to update and query the database.

♦ **Testing**:
  — Using a variety of modular programs, candidates could carry out component testing.
  — You could demonstrate debugging techniques, for example dry runs, trace tables, breakpoints and watchpoints, to show how they can help programmers find errors within their code.

♦ **Evaluation**:
  — In groups, candidates could evaluate completed programs in terms of efficient use of coding constructs and usability.

## Database design and development

♦ **Design**:
— You could explain the differences between a data dictionary at National 5 and Higher, and the same dictionary at Advanced Higher (which uses SQL data types and validation).
— Candidates could complete different types of exercises to create a data dictionary from given data.
— Using supplied scenarios with completed analysis, candidates could complete entity-relationship diagrams, including notation of weak or strong entities and mandatory or optional relationships.
— Using sample database tables, candidates could design queries to produce a required output.

♦ **Implementation**:
— You could demonstrate the SQL operations required to create a database and subsequently create or drop tables.
— You could demonstrate SQL operations using HAVING and Advanced Higher logical operators. Candidates could then complete a number of exercises to solve problems relating to using the appropriate SQL operations.
— Using SQL code and databases, candidates could explain what the output of the code would be.

♦ **Testing and evaluation**:
— Using SQL code, candidates could test it and evaluate its fitness for purpose, and accuracy of output.
— Using an incorrect SQL operation along with the correct expected output, candidates could identify how to correct the SQL statement in order to produce the expected output.


## Web design and development

♦ **Design**:
— Candidates could use wire-framing design techniques to design website structures and pages relating to multi-level websites. These could involve multiple screen views, for example mobile and desktop.
— Using the completed website designs, candidates could create low-fidelity prototypes to test their effectiveness.
— Candidates could complete pseudocode design for server-side processes.

♦ **Implementation**:
— Using HTML, Cascading Style Sheets (CSS) and PHP code from sample web pages, candidates could explain which parts of the code relate to the web page.
— Using HTML, CSS and PHP, candidates could implement a design that requires data to be retrieved from a database and displayed as a table.
— Using HTML, CSS and PHP, candidates could implement a design that requires form data to be processed and stored in a database.

♦ **Evaluation**:
— Working in groups or individually, candidates could evaluate previous solutions for usability.

**Testing (SDD, DDD and WDD)**

These learning and teaching activities could be in the context of SDD, DDD or WDD problems, as well as problems that integrate these areas (which is a requirement of the project):

- Working in groups, candidates could discuss how to carry out testing of integrated components.
- Working in groups, candidates could create prototypes and test each other's solutions to a problem. Following implementation of each prototype, the same end-user testing could be carried out.
- Working individually or in groups, candidates could discuss or plan a final testing solution for a given problem.

**Evaluation (SDD, DDD and WDD)**

These learning and teaching activities could be in the context of SDD, DDD or WDD problems, as well as problems that integrate these areas (which is a requirement of the project):

- Working in groups or individually, candidates could compare a solution to functional requirements and discuss its fitness for purpose.
- Working in groups, candidates could discuss the maintainability of a solution in terms of correcting, adapting or expanding a solution.
- Working in groups, candidates could perform destructive testing on each other's solutions to evaluate the robustness of the solution.

# Resources

You need access to an SQL server and a web server to implement the following:

- PHP code to process form data
- PHP code to connect to a database
- server-side SQL execution
- create and maintain a database using SQL statements

You may wish to use prebuilt solutions installed locally, such as XAMPP or arrange access to online resources.

You should ensure that the programming language used for the Advanced Higher course is object-oriented (OO) capable and has the capacity to connect to a database file.

You also need:

- internet-enabled computers and a digital projector
- access to software development tools
- access to application development software and tools
- web development tools (for example HTML5 script enabled browsers and wire-framing software)

## Some suggested software development environments

For this course, you can use any software development environment. You should base your decision on the suitability of the chosen environment to support the delivery of the mandatory content of the course.

Possible examples include:

♦ Python
♦ Visual Basic
♦ Java
♦ Live Code


## Teaching and learning materials

A number of online resources are available.

♦ **Software design and development**

www.java.com
www.python.org
www.codeacademy.com
www.programiz.com/python-programming
www.livecode.com
www.draw.io

♦ **Database design and development**

www.w3schools.com
www.codeacademy.com
www.tutorialspoint.com/sql
www.sqlcourse.com
Apex.oracle.com/en

♦ **Web design and development**

www.w3schools.com
www.codeacademy.com
html.net/tutorials
www.khanacademy.org
pencil.evolus.vn
balsamiq.com
resources.infosecinstitute.com/prototyping
Goggles.mozilla.org
http://hackasaurus.toolness.org

[date accessed August 2019]

# Comparison of skills, knowledge and understanding for Higher and Advanced Higher

The following table shows the relationship between the mandatory Higher and Advanced Higher skills, knowledge and understanding.

You can use this to:

♦ ensure seamless progression between levels
♦ identify important prior learning for candidates at Advanced Higher

**Analysis**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | Identify the:<br><br>♦ purpose<br>♦ scope<br>♦ boundaries<br>♦ functional requirements<br><br>of a problem that relates to the design and implementation at this level, in terms of:<br><br>♦ inputs<br>♦ processes<br>♦ outputs | Identify the purpose and functional requirements of a problem that relates to the design and implementation at this level in terms of:<br><br>♦ inputs<br>♦ processes<br>♦ outputs<br><br>Describe, exemplify, and implement research for:<br><br>♦ feasibility studies:<br>— economic<br>— time |

**Analysis (continued)**

| Area | Higher | Advanced Higher |
|---|---|---|
| **DDD** | Identify the end-user and functional requirements of a database problem that relates to the implementation at this level. | — legal<br>— technical<br>♦ user surveys |
| **WDD** | Identify the end-user and functional requirements of a website problem that relates to the design and implementation at this level. | Describe, exemplify, and implement planning in terms of:<br><br>♦ scheduling<br>♦ resources<br>♦ Gantt charts<br><br>Produce requirement specifications for end users and develop:<br><br>♦ end-user requirements<br>♦ scope, boundaries and constraints<br>♦ functional requirements<br><br>Describe, exemplify, and implement Unified Modelling Language (UML):<br><br>♦ use case diagrams:<br>  — actors<br>  — use cases<br>  — relationships |

**Design**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | Identify the data types and structures required for a problem that relates to the implementation at this level.<br><br>Read and understand designs of solutions to problems at this level, using the following design techniques:<br><br>♦ structure diagrams<br>♦ pseudocode<br><br><br><br>Exemplify and implement efficient design solutions to a problem, using a recognised design technique, showing:<br><br>♦ top level design<br>♦ the data flow<br>♦ refinements<br><br><br>Describe, exemplify and implement user-interface design, in terms of input and output, using a wireframe. | Identify the data types and structures required for a problem that relates to the implementation at this level.<br><br>Read and understand designs of solutions to problems at this level using the following design techniques:<br><br>♦ structure diagrams<br>♦ pseudocode<br>♦ UML<br><br>Exemplify and implement efficient design solutions to a problem at this level, using pseudocode, showing:<br><br>♦ top level design<br>♦ the data flow<br>♦ refinements<br><br><br>Describe, exemplify, and implement UML for the following:<br><br>♦ class diagrams:<br>   — class name<br>   — instance variables and data types<br>   — methods<br>   — public and private |

**Design (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | | — inheritance |
| | | — constructor |
| | | — array of objects |
| | | |
| | | Describe, exemplify, and implement user-interface design using a wireframe, indicating: |
| | | |
| | | ♦ visual layout |
| | | ♦ inputs |
| | | ♦ validation |
| | | ♦ underlying processes |
| | | ♦ outputs |
| **DDD** | Describe and exemplify entity-relationship diagrams with three or more entities, indicating: | Describe, exemplify, and implement entity-relationship diagrams with three or more entities indicating: |
| | | |
| | ♦ entity name | ♦ entity name |
| | ♦ attributes | ♦ entity type (strong, weak) |
| | ♦ name of relationship | ♦ attributes |
| | ♦ cardinality of relationship (one-to-one, one-to-many, many-to-many) | ♦ relationship participation (mandatory, optional) |
| | | ♦ name of relationship |
| | | ♦ cardinality |
| | Describe and exemplify an instance using an entity-occurrence diagram. | |
| | | Identify relationship participation from an entity-occurrence diagram. |

**Design (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **DDD** | Describe and exemplify a compound key.<br><br>Describe and exemplify a data dictionary with three or more entities:<br><br>♦ entity name<br>♦ attribute name<br>♦ primary and foreign key<br>♦ attribute type:<br>　— text<br>　— number<br>　— date<br>　— time<br>　— Boolean<br>♦ attribute size<br>♦ validation:<br>　— presence check<br>　— restricted choice<br>　— field length<br>　— range<br><br>Exemplify a design of a solution to a query:<br><br>♦ tables and queries | Describe, exemplify, and implement surrogate keys.<br><br>Describe and exemplify a data dictionary, in relation to SQL, with three or more entities for the following:<br><br>♦ entity name<br>♦ attribute name<br>♦ primary and foreign key<br>♦ attribute type:<br>　— varchar<br>　— integer<br>　— float<br>　— date<br>　— time<br>♦ attribute size<br>♦ validation:<br>　— presence check<br>　— restricted choice<br>　— field length<br>　— range<br><br>Exemplify a design of a solution to a query using:<br><br>♦ tables and queries |

**Design (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **DDD** | ♦ fields<br>♦ search criteria<br>♦ sort order<br>♦ calculations<br>♦ grouping | ♦ fields<br>♦ search criteria<br>♦ sort order<br>♦ calculations<br>♦ grouping<br>♦ having |
| **WDD** | Describe and exemplify the website structure of a multi-level website with a home page and two additional levels, with no more than four pages per level.<br><br>Describe, exemplify and implement, taking into account end-user requirements and device type, an effective user-interface design (visual layout and readability) using wire-framing:<br><br>♦ horizontal navigational bar<br>♦ relative horizontal and vertical positioning of the media<br>♦ form inputs<br>♦ file formats of the media (text, graphics, video, and audio)<br><br>Describe, exemplify and implement prototyping (low fidelity) from wireframe design at this level. | Describe, exemplify, and implement wireframe designs showing:<br><br>♦ visual layout<br>♦ navigation<br>♦ consistency<br>♦ underlying processes<br><br>Describe, exemplify, and implement low-fidelity prototype from wireframe design.<br><br>Read and understand designs of server-side processes at this level, using the following design techniques:<br><br>♦ structure diagrams<br>♦ pseudocode<br><br>Exemplify and implement the design of server-side processes using pseudocode. |

**Implementation**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | **Data types and structures**<br>Describe, exemplify and implement appropriately the following structures:<br><br>♦ parallel 1-D arrays<br>♦ records<br>♦ arrays of records | **Data types and structures**<br>Describe, exemplify, and implement the following structures in solutions to problems at this level:<br><br>♦ parallel 1-D arrays<br>♦ records<br>♦ arrays of records<br>♦ 2-D arrays<br>♦ array of objects<br><br>Describe and exemplify the operation of linked lists (double and single). |
| | **Computational constructs**<br>Describe, exemplify and implement the appropriate constructs in a procedural high-level (textual) language:<br><br>♦ parameter passing (formal and actual)<br>♦ the scope of local and global variables<br>♦ sub-programs/routines, defined by their name and arguments (inputs and outputs):<br>  — functions<br>  — procedures | **Computational constructs**<br>Describe, exemplify, and implement the following object-oriented (OO) constructs:<br><br>♦ object<br>♦ property<br>♦ method<br>♦ class<br>♦ sub-class<br>♦ encapsulation<br>♦ inheritance |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | ♦ pre-defined functions (with parameters):<br>— to create substrings<br>— to convert from character to ASCII and vice versa<br>— to convert floating-point numbers to integers<br>— modulus<br>♦ file handling:<br>— sequential CSV and txt files (open, create, read, write, close)<br><br>Read and explain code that makes use of the above constructs.<br><br>**Algorithm specification**<br>Describe, exemplify and implement standard algorithms using 1-D arrays or arrays of records:<br><br>♦ linear search<br>♦ find minimum and maximum<br>♦ count occurrences | ♦ instantiation<br>♦ polymorphism<br><br>Describe, exemplify, and implement code to:<br><br>♦ open and close connection to database server<br>♦ execute SQL query<br>♦ format query results<br><br>**Algorithm specification**<br>Describe, exemplify, and implement standard algorithms including:<br><br>♦ binary search<br>♦ insertion sort<br>♦ bubble sort<br><br>Read and explain code that uses constructs appropriate to this level. |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **DDD** | Describe, exemplify and use SQL operations for pre-populated relational databases, with three or more linked tables:<br><br>♦ UPDATE, SELECT, DELETE, INSERT statements making use of:<br>   — wildcards<br>   — aggregate functions (MIN, MAX, AVG, SUM, COUNT)<br>   — computed values, alias<br>   — GROUP BY<br>   — ORDER BY<br>   — WHERE<br><br>Read and explain code that makes use of the above SQL. | Implement relational database using SQL Data Definition Language (DDL) and Data Manipulation Language (DML) to match the design.<br><br>Describe, exemplify, and implement the following SQL operations:<br><br>♦ CREATE statement:<br>   — CREATE DATABASE<br>   — CREATE TABLE<br>   — constraints:<br>      o primary key<br>      o foreign key<br>      o not null<br>      o check<br>      o auto increment<br>♦ DROP statement:<br>   — DROP DATABASE<br>   — DROP TABLE<br>♦ HAVING clause of the SELECT statement<br>♦ subqueries used with the WHERE clause of SELECT statements<br>♦ data types:<br>   — varchar<br>   — integer |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|---|---|---|
| **DDD** | | — float |
| | | — date |
| | | — time |
| | | ♦ logical operators: |
| | | — IN |
| | | — NOT |
| | | — BETWEEN |
| | | — ANY |
| | | — EXISTS |
| | | |
| | | Read and explain code that uses the SQL at this level. |
| **WDD** | **CSS**<br>Describe, exemplify and implement efficient inline, internal and external Cascading Style Sheets (CSS) using grouping and descendant selectors to:<br><br>♦ control appearance and positioning:<br>— display (block, inline, none)<br>— float (left, right)<br>— clear (both)<br>— margins/padding<br>— sizes (height, width)<br>♦ create horizontal navigation bars:<br>— list-style-type:none<br>— hover | **CSS**<br>Describe, exemplify, and implement responsive pages using the following media queries:<br><br>♦ media type:<br>— print<br>— screen<br>♦ media feature:<br>— max-width |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **WDD** | Read and explain code that makes use of the above CSS. **HTML** Describe, exemplify and implement HTML code: ♦ nav ♦ header ♦ footer ♦ section ♦ main ♦ form ♦ id attribute Describe, exemplify and implement form elements: ♦ form element: input — text — number — textarea — radio — submit ♦ form element: select | **HTML** Describe, exemplify, and implement form elements including: ♦ FORM element: — action — method (get and post) ♦ INPUT, SELECT and TEXTAREA elements: — name — value ♦ TABLE element: — th, tr, td |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **WDD** | Describe, exemplify and implement form data validation:<br><br>♦ length<br>♦ presence<br>♦ range<br>Read and explain code that makes use of the above HTML.<br><br>**JavaScript**<br>Describe, exemplify and implement coding of JavaScript functions related to mouse events:<br><br>♦ onmouseover<br>♦ onmouseout<br>♦ onclick<br><br>**PHP**<br>No content at Higher | **JavaScript**<br>No content at Advanced Higher<br><br><br><br><br><br><br><br><br>**PHP**<br>Describe, exemplify, and implement coding of server-side processing to:<br><br>♦ assign form data to server-side variables:<br>— $_get()<br>— $_post()<br>♦ open and close connection to database server:<br>— die() |

**Implementation (continued)**

| Area | Higher | Advanced Higher |
|---|---|---|
| **WDD** | | — mysqli_connect()<br>— mysqli_close()<br>♦ execute SQL query:<br>— mysqli_query()<br><br>♦ format query results:<br>— echo<br>— mysqli_fetch_array()<br>— mysqli_num_row()<br><br>and:<br><br>♦ assignment, repetition and selection using server-side local and global variables<br>♦ sessions:<br>— session_start()<br>— session_destroy()<br><br>Read and explain code that uses constructs appropriate to this level. |

**Testing**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | Describe, exemplify and implement a comprehensive final test plan to show that the functional requirements are met.<br><br>Identify syntax, execution, and logic errors at this level.<br><br>Describe and exemplify debugging techniques:<br><br>♦ dry runs<br>♦ trace tables/tools<br>♦ breakpoints<br>♦ watchpoints | Describe, exemplify, and implement the following for SDD, DDD and WDD:<br><br>♦ integrative testing<br>♦ usability testing based on prototypes<br>♦ final testing<br>♦ end-user testing<br><br>and for SDD only:<br><br>♦ component testing during the development of the solution<br><br>and for DDD only:<br><br>♦ SQL implemented tables match design<br>♦ SQL operations work correctly at this level |
| **DDD** | Describe and exemplify testing:<br><br>♦ SQL operations work correctly at this level | |
| **WDD** | Describe, exemplify and implement usability testing using personas, test cases and scenarios based on low-fidelity prototypes.<br><br>Describe and exemplify testing:<br><br>♦ input validation<br>♦ navigational bar works<br>♦ media content displays correctly | |

**Testing (continued)**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | Describe and exemplify compatibility testing:<br><br>♦ device type:<br>— tablet, smartphone, desktop<br>♦ browser | |

**Evaluation**

| Area | Higher | Advanced Higher |
|------|--------|-----------------|
| **SDD** | Describe, identify and exemplify the evaluation of a solution in terms of:<br><br>♦ fitness for purpose<br>♦ efficient use of coding constructs<br>♦ usability<br>♦ maintainability<br>♦ robustness | Evaluate solution for SDD, DDD and WDD in terms of:<br><br>♦ fitness for purpose<br>♦ maintainability<br>  — perfective<br>  — corrective<br>  — adaptive<br>♦ robustness |
| **DDD** | Evaluate solution at this level in terms of:<br><br>♦ fitness for purpose<br>♦ accuracy of output | and for SDD only:<br><br>♦ efficiency<br>♦ usability |
| **WDD** | Evaluate solution at this level in terms of:<br><br>♦ fitness for purpose<br>♦ usability | and for DDD only:<br><br>♦ accuracy of output<br><br><br>and for WDD only:<br><br>♦ usability |

# Preparing for course assessment

The course assessment focuses on breadth, challenge and application. Candidates should apply the skills, knowledge and understanding they have gained during the course.

In preparation, you should give candidates the opportunity to practise activities similar to those expected in the course assessment. For example, you could develop questions and tasks similar to those in the specimen question paper and coursework.

You may find the following information useful:

♦ course assessment overview
♦ question paper brief

**Course assessment overview**

Marks: 135

The course assessment has two components:

♦ question paper: 55 marks
♦ project: 80 marks

Proportion of 'A' and 'C' type questions:

♦ approximately 30% of marks 'A' type
♦ approximately 50% of marks 'C' type

The course assessment (question paper and project) is designed using the following breakdown of marks for each skill assessed.

| Skill | Course assessment | | Project | Question paper |
|---|---|---|---|---|
| | % marks | Total marks (approximate) | Marks | Marks |
| Analysis | 10% | 14 | 10 | 3–8 |
| Design | 30% | 41 | 20 | 16-24 |
| Implementation | 40% | 54 | 30 | 20–28 |
| Testing | 15% | 21 | 15 | 2–8 |
| Evaluation | 5% | 7 | 5 | 0–5 |

**Question paper brief**

Marks: 55

Duration: 2 hours

The question paper has three sections. Section 1 is mandatory, and candidates have the option to complete either Section 2 or Section 3.

♦ Section 1: Software design and development — 35 marks
♦ Section 2: Database design and development — 20 marks
♦ Section 3: Web design and development — 20 marks

Proportion of 'A' and 'C' type marks:

♦ approximately 30% of marks 'A' type (primarily in context-based questions)
♦ approximately 50% of marks 'C' type

The question paper is designed using the following range of marks, against each area of content and skills.

| Skill | Range |
|---|---|
| Analysis | 3–8 |
| Design | 16-24 |
| Implementation | 20–28 |
| Testing | 2-8 |
| Evaluation | 0–5 |

The skills, knowledge and understanding across the 'Database design and development' and 'Web design and development' areas of study are not directly comparable, for example, there is more assessable content in design for DDD than WDD, but more for implementation in WDD than DDD.

As a result, the mark breakdown across analysis, design, implementation, testing and evaluation will not be identical across the options, however, there will be a balance of 'A' type and 'C' type marks across the options.

# Developing skills for learning, skills for life and skills for work

You should identify opportunities throughout the course for candidates to develop skills for learning, skills for life and skills for work.

Candidates should be aware of the skills they are developing and you can provide advice on opportunities to practise and improve them.

SQA does not formally assess skills for learning, skills for life and skills for work.

There may also be opportunities to develop additional skills depending on the approach centres use to deliver the course. This is for individual teachers and lecturers to manage.

Some examples of potential opportunities to practise or improve these skills are as follows:

| Skill | How to develop |
|---|---|
| **Numeracy**<br>2.3  Information handling | Develop skills by setting problem-solving contexts where candidates use data set out in tables or a graphical format as the basis for input to their programs, processing the data to produce the required output. |
| **Health and Wellbeing**<br>3.1  Personal learning | Candidates work autonomously on their project, taking responsibility for completing it within the time available to them. They plan for this and have opportunities to follow up on curiosity, think constructively and learn from experience. |
| **Employability, enterprise and citizenship**<br>4.2  Information and communication technology (ICT) | Throughout the course, candidates continually interact with the technology around them. This should provide plenty of opportunities to extend their ICT skills. |
| **Thinking skills**<br>5.3  Applying | Give candidates opportunities to analyse a wide range of problems, apply the knowledge and skills they have acquired, and then test and review their solutions. |
| 5.4  Analysing and evaluating | Develop skills through the process of creating computer programs to solve problems and testing them. |

# Resources to support the Advanced Higher Computing Science course

These resources provide clarification and exemplification of some of the skills, knowledge and understanding developed in the Advanced Higher course.

Note 1: appendix 10 uses a relational database that can be found on the Advanced Higher page on SQA's website.

Note 2: appendices 12-14 use the Advanced Higher example website to exemplify the course requirements for teachers and lecturers. You can download the example website from SQA's secure site, but you must not distribute it to candidates, as it would provide a framework for a web-based Advanced Higher project.

# Appendix 1: problem analysis (SDD, DDD and WDD)

## Requirements specification

In addition to the purpose, scope, boundaries, and requirements exemplified at Higher, analysis of any development should identify the constraints of a problem.

### Constraints

Constraints are restrictions that apply to the development. These restrict the changes made to design decisions during the development. Time, scope and cost are the main constraints of project management; however, depending on the type of development, other constraints may apply, for example:

### Technical constraints

♦ knowledge and/or availability of development tools and programming language

♦ the operating system or platforms that will be used to deliver the working solution

♦ hardware considerations such as capacity

♦ non-functional requirements such as performance considerations

### Business constraints

♦ schedule and timescales that must be met

♦ available budget

♦ composition and makeup of the development team

♦ software licensing restrictions or requirements

### Further constraints

♦ economic considerations

♦ political issues

Note: the requirements specification document is often the basis of a legal contract between the client (customer) and the software company writing the software.

## Worked example of a requirements specification (SDD)

### Analysis

The purpose of a program is to allow the end user to search for an item on an unsorted list of data. If a match is found, the program will display the row of data for the item.

### Scope

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

♦ a detailed design of the program structure

♦ a test plan with a completed test data table

- ♦ a working program
- ♦ the results of testing
- ♦ an evaluation report

## Boundaries

- ♦ the program will read the data (itemID, price, and number in stock) from a sequential file
- ♦ if the data is accurate, there is no need to implement input validation

## End-user requirements

End users will expect:

- ♦ to enter an itemID while the program is running
- ♦ the data corresponding to the itemID to be displayed
- ♦ a user interface that is clearly labelled and easy to use for all user types

## Functional requirements

Functional requirements are defined in terms of the inputs, processes, and outputs listed below. All inputs are imported from a sequential file and all outputs displayed on the screen. The program is activated by double clicking on the file icon and then selecting "Run" from the menu. Each process should be a separate procedure or function that is called from the main program.

### Inputs

- ♦ itemID
- ♦ price
- ♦ number in stock

### Processes

- ♦ read in data from an external file to a 2D array
- ♦ sort the data in order of itemID from low to high
- ♦ search the 2D array for the required itemID, based on the end-user input

### Output

- ♦ if a match is found, the data (itemID, price, and number in stock) will correspond to the end-user input
- ♦ if no match is found, a suitable message will inform the end user

## Constraints

The constraints that apply to this development are:

- ♦ Live Code, Python, or Visual Basic must be used to develop the program.
- ♦ The working program will run on the Windows operating system.
- ♦ The work must be completed within 8 hours.

# Worked example of a requirements specification (DDD)

## Analysis

GoGoGadgets.com is a company specialising in quirky and unusual gadgets that are available for purchase through its online catalogue.

Before customers can make a purchase, they must first register with the GoGoGadgets website and be allocated a unique customerID.

Customers can browse the product range through an online catalogue. Each item is categorised as one of the following: Toys, Gizmos, Office Distractions, Personal Grooming, and Computer Accessories. All items cost less than £50.

A database is required to store details of customers, items, and orders.

## Scope

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

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

## Boundaries

♦ the database will contain a maximum of 10 000 items
♦ each item will cost £50 or less
♦ all items should be categorised as one of the following: Toys, Gizmos, Office Distractions, Personal Grooming, and Computer Accessories
♦ users must enter a valid email address to register

## End-user requirements

End users (customers) will expect queries that enable them to:

♦ register as a user and store their details in the database
♦ search for items based on the category of the item
♦ search for items based on the name of an item
♦ sort items by price (low to high), price (high to low) or rating

End users (administrators) will expect queries that enable them to:

♦ edit the price of items
♦ edit customer contact details
♦ add and remove details of individual items
♦ remove details of customers from the database
♦ view details of all orders placed each month

## Functional requirements

Functional requirements are defined in terms of the inputs, processes and outputs listed below.

### Inputs (customers)

♦ register: user email, password, password re-entered, firstName, lastName, address, and postcode:
 — search details: category
 — search details: itemName
♦ sort details: field (price or rating) and order required (ascending or descending)

### Inputs (administrators)

♦ edit item details: itemID and price
♦ edit customer details: customerID, address, postcode, and email
♦ add item details: itemID, itemName, description, category, and price
♦ delete item details: itemID
♦ delete customer details: customerID
♦ monthly orders: month

### Processes

♦ auto generate customerID whenever a new customer registers
♦ queries to:
 — insert records into the Customer and Item tables
 — sort item details in order of price and rating
 — delete a specific customer and an item record from the database
 — edit records in the Customer and Item tables
 — search Item table
 — display details of all orders placed in a particular month

### Output

♦ confirmation of successful:
 — insertions
 — deletions
 — edits
♦ answer tables showing details of:
 — sorted items (sorts)
 — required items (searches)

**Constraints**

The constraints that apply to this development are:

♦ The Oracle MySQL server must be used to develop the database.

♦ The working database will run on the Windows operating system.

♦ The work must be completed within 15 hours.


# Unified Modelling Language (UML)

Unified Modelling Language (UML) provides a standard way to visualise, specify, construct, and document the analysis and design of a software system.

UML is a pictorial language used to make software blueprints that can be used to model software and non-software systems.


## UML use case diagram

To model a system, it is important to capture the dynamic behaviour of the system. Dynamic behaviour is when the system is running or operating.

The purpose of a use case diagram is to capture the dynamic aspect of the system. Use case diagrams:

♦ are used to gather the requirements of the system

♦ are used to get an outside view of the system

♦ identify the internal and external factors that influence the system

♦ show the interaction among the requirements as 'actors'

♦ aid communication between the client and the developer


## Drawing a use case diagram

Use case diagrams consist of four components:

♦ a system boundary

♦ actors

♦ use cases

♦ relationships


## System boundary

In a UML case diagram, a system boundary is shown as a rectangle. All components of the use case diagram are shown inside the system boundary.

The system boundary represents the limits of the system being developed: only those actors and processes to be considered are illustrated within the system boundary.

**Actors**

An actor interacts with the system being developed. The actor may be a human or an entity that interacts with the system, for example another system or server, and is external to the system being developed.

An actor performs a role in a system and may be a primary or secondary actor.

A primary actor is one that uses the system to achieve a goal, for example a customer buying an item.
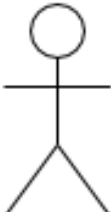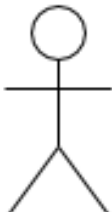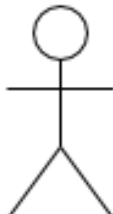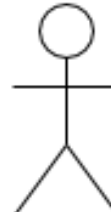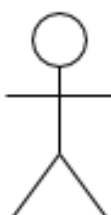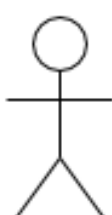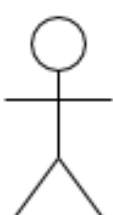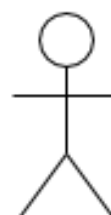
A secondary actor is one that supports the system in delivering the goal, for example a bank used to pay for the item.

A UML case diagram shows an actor by using the symbol:



Actor

The following are examples of actors, depending on the problem being solved.

Actor

| Human | Systems software | Hardware | Timer (clock) |
|---|---|---|---|
| Customer | Payroll | Phone network | Scheduled backup |
| Passenger | Library | Server | Scheduled anti-virus |

**Use cases**

A use case describes an action (process) or a sequence of actions (processes) that must be in the system being developed.

A UML case diagram shows a use case using an ellipse:

Use Case

Use cases help to determine the requirements of the system under consideration, by describing the functionality that the system will provide.

Use case functionality (process) may be initiated by an actor or may be started by the system itself, providing a useful result to an actor.

**Naming use cases**

Each use case must have the name written within the ellipse. The name describes some observable or useful result to an actor.

Examples of naming are Update Subscription, Manage Account, and Place Order.

Update Subscription

**Relationships**

A use case diagram can have five types of relationship:

♦   association between an actor and a use case
♦   generalisation of an actor
♦   extend between two use cases
♦   include between two use cases
♦   generalisation of a use case

## Association between actor and use case

Each actor must be associated with at least one use case, although it can be associated with many use cases.

Customer ──────── ( Make Purchase )

Customer
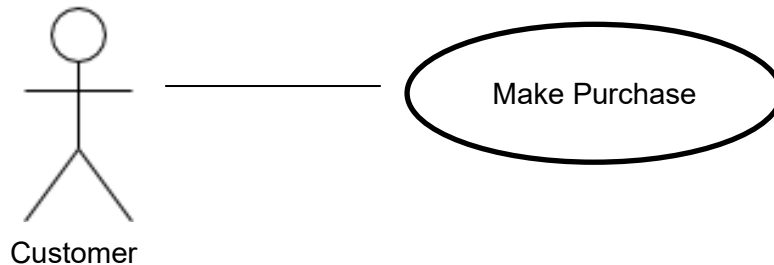
A line with no arrowheads connects an actor to a use case.

## Generalisation of an actor

Generalisation of an actor means one actor can inherit the role of another actor. The descendant actor inherits all the use cases of the ancestor.

User ◄──────── Registered user

User                    Registered user
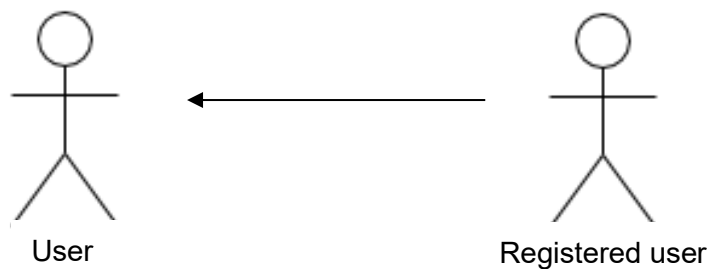
A line, with a single solid arrowhead pointing at the ancestor actor, connects a descendant actor to the ancestor actor.

## Extend between two use cases

Extending a basic use case provides additional functionality to the system.

( Credit Card Payment ) ◄---- <<extend>> ----( Calculate Interest )

An extended use case is connected to a basic use case using a dashed line, with a single solid arrowhead pointing at the basic use case. The label <<extend>> is placed on the line.

## Include between two use cases

An included use case is part of the basic use case. It is a mandatory process, as the basic use case is incomplete without it.
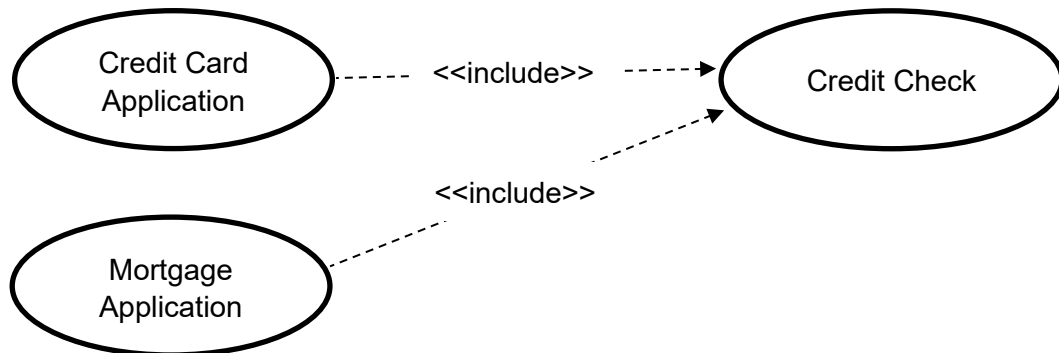


An included use case is connected to the basic use case using a dashed line with a single solid arrowhead, pointing at the common basic use case. The label <<include>> is placed on the line.

## Generalisation of a use case

This is similar to generalisation of an actor.



A line, with a single solid arrowhead pointing at the ancestor use case, connects a descendant use case to the ancestor use case.

**Creating a use case diagram**
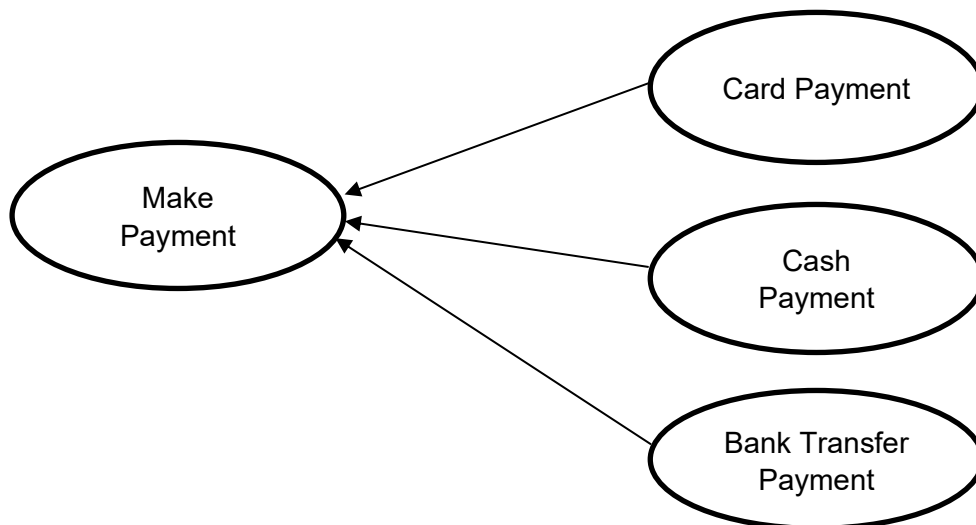
The following is an example of a use case diagram.

**Example**

This example appeared in the 2016 question paper for Advanced Higher Computing Science:

The owners of a monthly magazine decide to update the company website. The current website allows users to access online versions of articles printed in the monthly magazines.

Requirements for the updated website are listed below.

The updated website will allow **all** users to:

♦ access a maximum of five free articles every month
♦ search for articles over 12 months old
♦ subscribe to the full service using a secure payment system

The updated website will allow **subscribed** users to:

♦ login to gain access to the full service
♦ access any number of articles
♦ search for articles without restriction
♦ renew their subscription at a reduced rate using a secure payment system

Draw a use case diagram to represent these requirements.

The following is a sample use case diagram for this scenario.
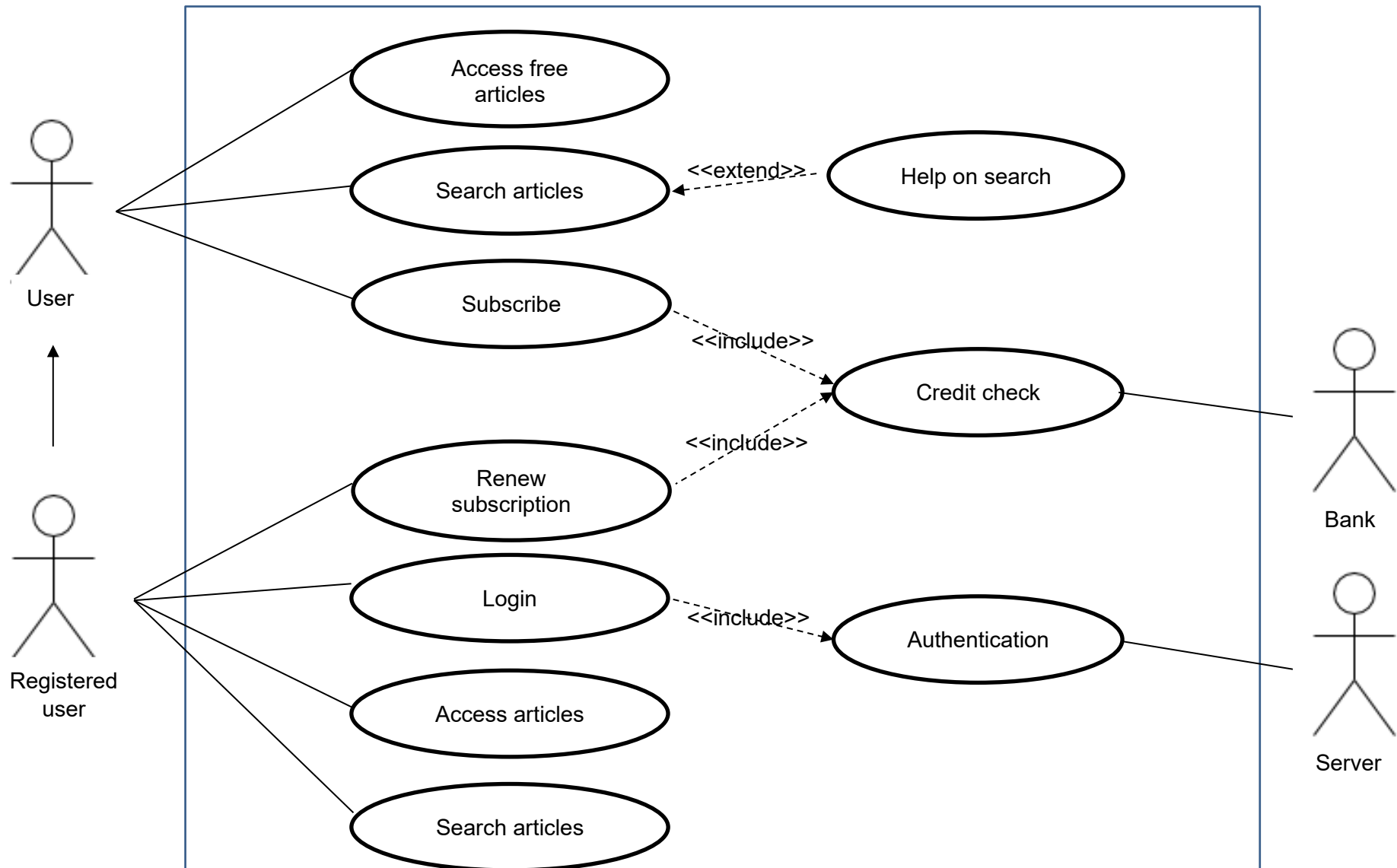
# Appendix 2: Unified Modelling Language (UML) — class diagrams (SDD)

To model a system, it is important to capture the static behaviour of the system.

A class diagram is used for a quick overview of the system. It describes the structure of a system by showing its:

♦ classes
♦ variables, structures and types
♦ methods of the class
♦ relationships between the classes

The purpose of a class diagram is to model the static aspect of the system.

## Drawing a class diagram

A class is a blueprint for an object. A class diagram describes each class and the relationships between the classes.

### UML class notation

A class diagram consists of:

♦ a class name
♦ instance variables and data types:
  — public
  — private
♦ methods:
  — public
  — private
  — constructor
♦ inheritance between classes

### Example
A program is being written for an estate agency to store the details of houses for sale or available to rent.

**Class diagram for House**

Part of the class diagram for the House class is shown below.

| House |
| --- |
| address: String |
| town: String |
| bedrooms: Integer |
| description: String |
| houseValue: Integer |
| house() |
| setAddress() |
| getAddress() |
| updateBedrooms() |
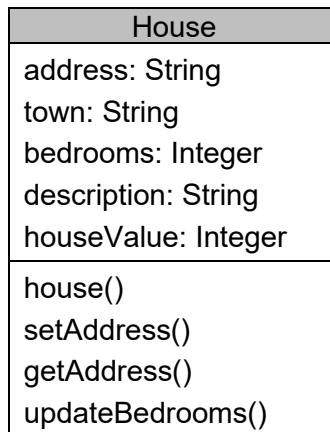
**Explanation**

The class diagram indicates the:

♦ class name

♦ instance variables with data types in the class (instantiation variables)

♦ methods associated with the class (including the constructor method)
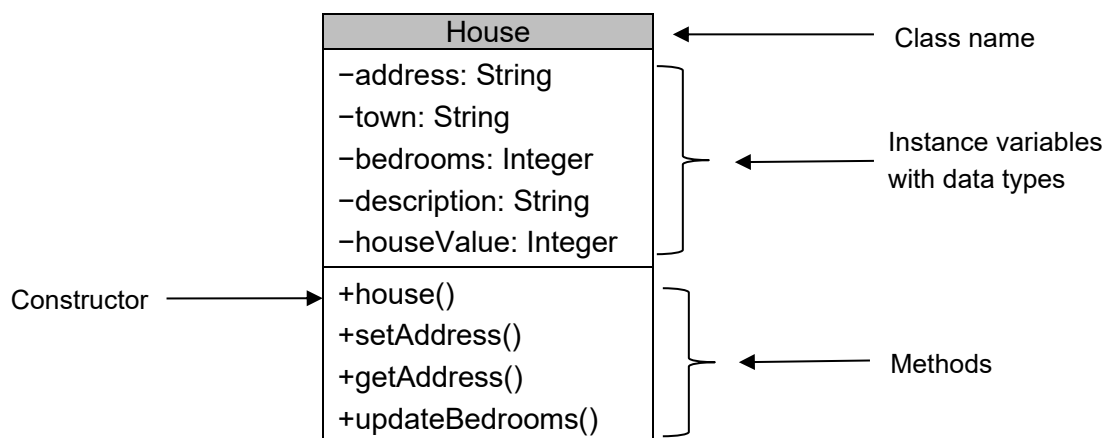


**Constructor**

A constructor is shown on a UML class diagram in the methods section. The constructor will have the same name as the class name. The constructor method is used to create an individual object that belongs to the class.

**Public and private**

The instance variables and methods within a class can be public or private elements.

Public elements can be used by any class; however, private elements can only be used by the owning class.

UML allows any variable or method to be shown as public or private.

In a class diagram:

♦ public elements are preceded with a + sign
♦ private elements are preceded with a − sign

The House class, with public and private elements, will look as follows.

| House |
| --- |
| −address: String |
| −town: String |
| −bedrooms: Integer |
| −description: String |
| −houseValue: Integer |
| +house() |
| +setAddress() |
| +getAddress() |
| +updateBedrooms() |
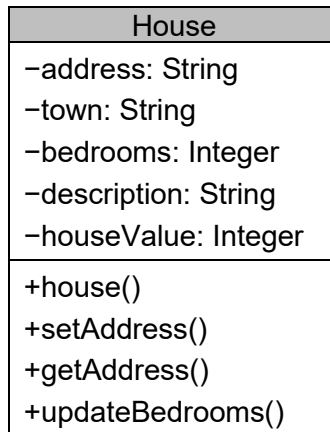
The set and get methods (sometimes called mutators and accessors) are needed to retrieve (get) or edit (set) the values held in private variables.

**Example code: `setAddress()`**

Used to edit the value stored in the private instance variable `address`.

```
PROCEDURE setAddress(STRING newAddress)
     SET THIS.address TO newAddress
END FUNCTION
```

**Example code: `getAddress()`**

Used to retrieve the value stored in the private instance variable `address`.

```
FUNCTION getAddress() RETURNS STRING
     RETURN THIS.address
END FUNCTION
```

**Inheritance**

UML allows the object-oriented construct of inheritance to be exemplified.

A sub-class can inherit all of the properties and methods of a superclass.

On a UML class diagram, this type of inheritance is indicated by an arrow from the sub-class to the superclass.

## Array of objects

The instance variables of a class or sub-class can include an array data structure. This can be used to store instances of another class.

An array of objects is written as:

```
scores: Array of Score[ ]
```

where Score is another class. On a UML class diagram, the connection between the array of objects and the object (class) is also indicated by an arrow.

## Example

The program below is for an estate agency to store the details of houses available for sale or to rent.

**House**

−address: String
−town: String
−bedrooms: Integer
−description: String
−houseValue: Integer

+house()
+setAddress()
+getAddress()
+updateBedrooms()

**ForSale**

−askingPrice: Real
−closingDate: String
−underOffer: Boolean
−offersReceived: Array of Offer[ ]
−sold: Boolean

+forSale()
+updateAskingPrice()
+updateSoldStatus()

**ForRent**

−rentalCost: Real
−deposit: Real
−rentalLength: Integer
−rented: Boolean

+forRent()
+updateRentStatus

**Offer**

−dateOfOffer: String
−amountOfOffer: Integer

+offer()
+setOfferDate()
+getOfferDate()
+setOfferAmount()
+getOfferAmount()

# Appendix 3: entity-relationship diagrams (DDD)

The Advanced Higher course requires candidates to describe, exemplify and implement entity-relationship diagrams with three or more entities, indicating:

♦ entity name
♦ entity type (strong, weak)
♦ attributes
♦ relationship participation (mandatory, optional)
♦ name of relationship
♦ cardinality

Candidates also need to be able to identify relationship participation from an entity-occurrence diagram.

## Entity type

A **strong entity** is one whose existence does not depend on the existence of any other entity in the same database. The primary key of a strong entity uniquely identifies each occurrence within the entity.

A **weak entity** is one that depends on one or more strong entities for its existence. For this reason, strong entities are sometimes referred to as owner entities. A weak entity cannot be used independently because its existence depends on one or more owner entities.

The primary key of a weak entity is formed, in part, using the primary key of its owner entity(ies). The presence of a weak entity is indicated by using a double line. The weak entity itself is indicated by using optionality.

Consider the (incomplete) entity-relationship diagram shown below. This illustrates three of the entities that form part of an online ordering system.
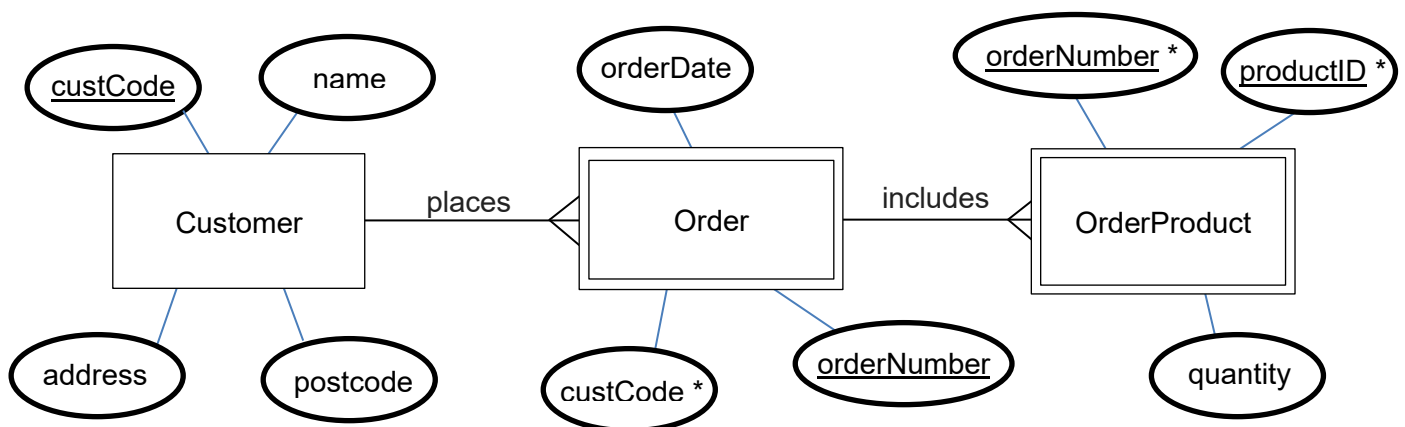


In this situation, the Customer and Order entities both have a primary key that uniquely identifies individual occurrences in each entity.

However, only the Customer entity is a strong entity. Order is a weak entity. Since an order occurrence can only be added if the customer details are known, the Order entity relies on the existence of the Customer entity.

The primary key of the OrderProduct entity is a compound key that is formed using the primary key of the Order entity. This means that OrderProduct is a **weak entity**. The double line is used to represent the weak entity.

## Relationship participation

Participation refers to the nature of the relationship between entities. Participation can be either mandatory or optional.

**Mandatory** participation describes a relationship where at least one occurrence of an entity must exist before any occurrences can be added to its associated entity. The mandatory side of any relationship is indicated by using a vertical line.

**Optional** participation describes a relationship between two entities where it is possible to add occurrences of one entity without the need to have existing occurrences in the associated entity. The optional side of a relationship is indicated by using a bold circle.

Participation has been added to the entity-relationship diagram introduced earlier and is shown below. For completeness, the Product entity has also been added to show all four entities that form the online ordering system.
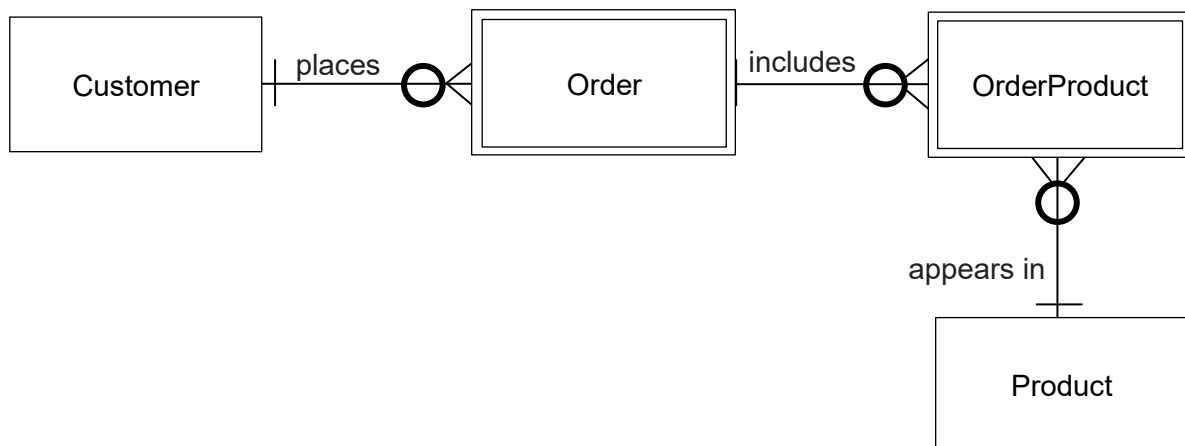


Customer is a strong entity, as it has its own uniquely identifying primary key and is not dependent on any other entity.

The Customer entity is linked to the Order entity using the **places** relationship. Each customer in the Customer entity can place many orders but it is also possible for details of a customer to be stored without them placing any orders.

As each order in the Order entity must always have one set of corresponding customer details in the Customer entity, it is not possible to add a new set of details to the Order entity without first having added details of the relevant customer to the Customer entity.

Order is a weak entity. Although it does have its own identifying primary key, its entity occurrence relies on the existence of a matching occurrence in the Customer entity.

The Order entity is linked to the weak OrderProduct entity using the **includes** relationship. The entity-relationship diagram shows that a new order can be created without a pre-existing, corresponding occurrence in the OrderProduct entity. Once it has been added to the Order entity, the order can be linked to several occurrences within the OrderProduct entity; it is also possible for an order to have no corresponding OrderProduct occurrences.

As OrderProduct is a weak entity, it is not possible to add an OrderProduct occurrence without first having an existing, corresponding occurrence in the Order entity.

The Product entity is linked to OrderProduct entity using the **appears in** relationship. As Product is a strong entity with its own uniquely identifying primary key, new product details can be added without the need to have any corresponding occurrences in the OrderProduct entity. The entity-relationship diagram shows that each product can appear in many individual OrderProduct occurrences, but it is possible that a product is never ordered.

As OrderProduct is a weak entity, it is not possible to add an OrderProduct occurrence without first having an existing, corresponding occurrence in the Product entity.


## Example

A travel agency uses a relational database to store details on a booking system.

It stores details of Scottish holiday resorts, hotels in each resort, customers and their bookings. These details are arranged in four separate entities.

The attributes stored in each entity are shown below.

| Resort | Hotel | Customer | Booking |
|---|---|---|---|
| resortID<br>resortName<br>resortType | hotelRef<br>hotelName<br>resortID *<br>starRating<br>seasonStartDate<br>mealPlan<br>checkInTime<br>pricePersonNight | customerNo<br>firstname<br>surname<br>address<br>town<br>postcode | hotelRef *<br>customerNo *<br>startDate<br>numberOfNights<br>numberInParty |

**Strong and weak entities**

From the list of attributes, we can see that Resort, and Customer are all **strong entities** because they have primary keys that uniquely identify each occurrence within the entities. Booking is a **weak entity** because its primary key relies on attributes from the Hotel and Customer entities. Hotel is a **weak entity** because its existence relies on the resortID attribute from the Resort entity.

## Relationship participation

An entity-occurrence diagram indicating the relationships between the entities is shown below.
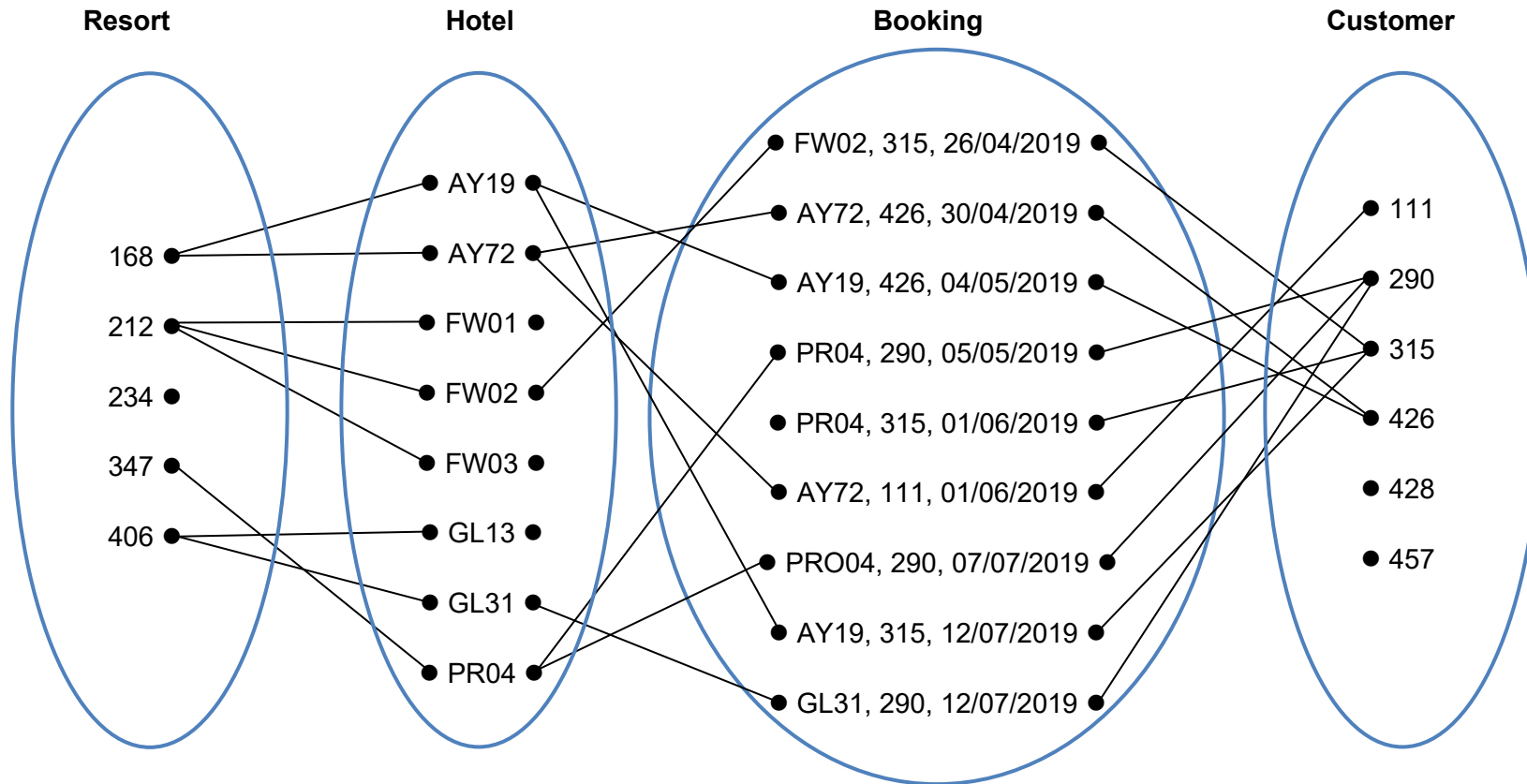


Using an entity-occurrence diagram helps to clarify the nature of each relationship.

The entity-occurrence diagram for the travel agency booking system makes it clear that:

♦ Resort: Hotel is a 1: M relationship:
— Resort has **mandatory** participation in this relationship:
  o every hotel must be located in exactly one resort
— Hotel has **optional** participation in this relationship:
  o a resort may or may not have a hotel

♦ Hotel: Booking is a 1: M relationship:
— Hotel has **mandatory** participation in this relationship:
  o each booking must be associated with exactly one hotel
— Booking has **optional** participation in this relationship:
  o a hotel may exist without any bookings

♦ Customer: Booking is a 1: M relationship:
– Customer has **mandatory** participation:
  o every booking must be associated with a customer
— Booking has **optional** participation in this relationship:
  o it is possible that some customers never make a booking (for example, details of customers on the mailing list will be stored in the database, even though they have never made any bookings)

## Entity-relationship diagram

The complete entity-relationship diagram that represents the relationships between the entities and relationships in the travel agency booking system is shown below.
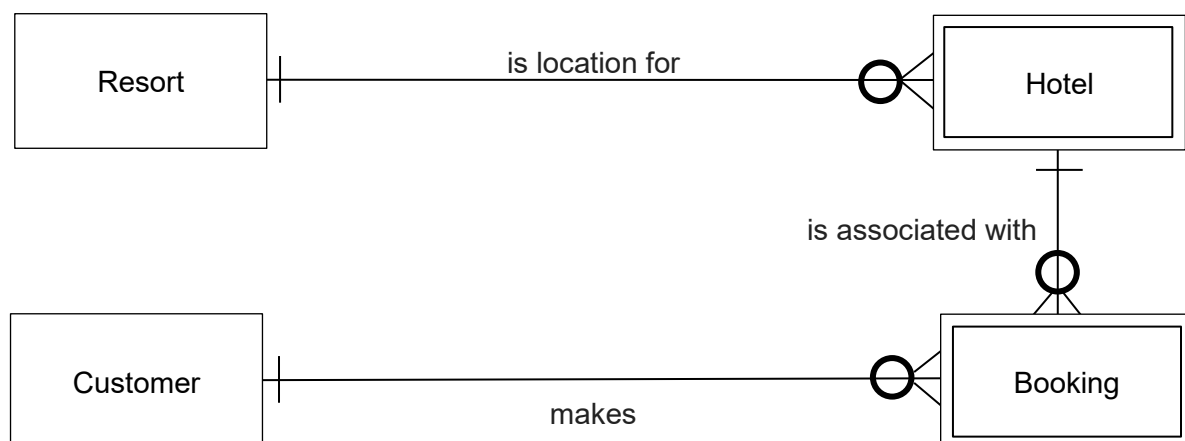
# Appendix 4: data dictionary (DDD)

A travel agency uses a relational database to store details on a booking system.

It stores details of Scottish holiday resorts, hotels in each resort, customers and their bookings. These details are arranged in four separate entities.

A data dictionary is used to indicate the properties of each attribute needed to define the entities.

Sample data stored in each table of the database are shown below.

**Sample data for resort**

| Resort ID | Resort name | Resort type |
|---|---|---|
| 168 | Ayr | coastal |
| 347 | Portree | island |

**Sample data for hotel**

| Hotel ref | Hotel name | Resort ID | Star rating | Season start date | Meal plan | Check-in time | Price/person/night (£) |
|---|---|---|---|---|---|---|---|
| AY72 | Cliff Top | 168 | 3 | 2019/04/29 | Half Board | 14:30:00 | 85.50 |
| PR04 | Sea View | 347 | 5 | 2019/05/01 | Bed and Breakfast | 16:00:00 | 58.99 |
| AY19 | Glee | 168 | 2 | | Full Board | 15:00:00 | 179.00 |

**Sample data for customer**

| Customer No | Firstname | Surname | Address | Town | Postcode |
|---|---|---|---|---|---|
| 315 | Edwina | Jones | 121 Main Street | Greenock | PA16 1JK |
| 426 | Omar | Shakir | 26a High Bridge | Perth | PH42 6QW |

**Sample data for booking**

| Hotel ref | Customer No | Start date | Number of nights | Number in party |
|---|---|---|---|---|
| PR04 | 315 | 2018/06/01 | 3 | 2 |
| AY19 | 315 | 2018/07/12 | 4 | 4 |
| PR04 | 315 | 2019/06/02 | 2 | 2 |

In the Advanced Higher course, data dictionary attribute types are expressed as SQL data types.

The completed data dictionary for the travel agency database is shown below.

**Entity: Resort**

| Attribute name | Key | Type | Size | Required | Validation |
|---|---|---|---|---|---|
| resortID | PK | integer | | yes | |
| resortName | | varchar | 20 | yes | |
| resortType | | varchar | 20 | yes | Restricted choice: coastal, city, island, country |

**Entity: Hotel**

| Attribute name | Key | Type | Size | Required | Validation |
|---|---|---|---|---|---|
| hotelRef | PK | varchar | 4 | yes | Length=4 |
| hotelName | | varchar | 20 | yes | |
| resortID | FK | integer | | Yes | Existing resortID from Resort table |
| starRating | | integer | | yes | Range: >=1 and <=5 |
| seasonStartDate | | date | | no | |
| mealPlan | | varchar | 17 | yes | Restricted choice: see list below* |
| checkInTime | | time | | yes | |
| pricePersonNight | | float | | yes | Range: >=50 and <=250 |

* Restricted choice for mealPlan: Room Only, Bed and Breakfast, Half Board, Full Board

**Entity: Customer**

| Attribute name | Key | Type | Size | Required | Validation |
|---|---|---|---|---|---|
| customerNo | PK | integer | | yes | Auto increment |
| firstname | | varchar | 20 | yes | |
| surname | | varchar | 20 | yes | |
| address | | varchar | 40 | yes | |
| town | | varchar | 20 | yes | |
| postcode | | varchar | 8 | yes | Length<=8 |

**Entity: Booking**

| Attribute name | Key | Type | Size | Required | Validation |
|---|---|---|---|---|---|
| hotelRef | PK FK | varchar | 4 | yes | Existing hotelRef from Hotel table |
| customerNo | PK FK | integer | | yes | Existing customer# from Customer table |
| startDate | PK | date | | yes | |
| numberNights | | integer | | yes | Range: >=1 |
| numberInParty | | integer | | yes | Range: >=1 |

# Appendix 5: query design (DDD)

A travel agency uses a relational database to store details on a booking system.

It stores details of Scottish holiday resorts, hotels in each resort, customers and their bookings. These details are stored in four separate entities.

The attributes stored in each entity are shown below.

| Resort | Hotel | Customer | Booking |
|--------|-------|----------|---------|
| resortID | hotelRef | customerNo | hotelRef* |
| resortName | hotelName | firstname | customerNo* |
| resortType | resortID * | surname | startDate |
| | starRating | address | numberOfNights |
| | seasonStartDate | town | numberInParty |
| | mealPlan | postcode | |
| | checkInTime | | |
| | pricePersonNight | | |

The design of an SQL query should indicate:

♦ the fields and/or calculations required
♦ the table(s) or query(-ies) needed to provide the details required
♦ any search criteria to be applied
♦ what grouping is needed (if appropriate)
♦ the criteria to be applied to the grouping (if appropriate)
♦ the field(s) used to sort the data and the type(s) of sort required

Encourage candidates to plan — this helps to reduce the amount of frustration they may otherwise encounter when working with the SQL code.

Candidates can use a simple table template to indicate the planned design of the SQL query, see the following examples.

## Example 1: HAVING with GROUPING and row COUNT

Display the resort name and number of hotels in any resort that has at least two hotels.

| Field(s)/ calculation(s) | resortName, Number of Hotels = COUNT(*) |
|---|---|
| Table(s) query(-ies) | Resort, Hotel |
| Search criteria | |
| Grouping | resortName |
| Having | COUNT(*) >= 2 |
| Sort order | |

## Example 2: HAVING with GROUPING and sort

Display the full name and the total cost of all bookings for each customer. The query should only list details of customers whose total cost exceeds £2000 and should list the details of the biggest spending customer first.

| Field(s)/ calculation(s) | firstName, surname, Total cost of all Bookings = SUM(pricePersonNight * numberNights * numberInParty) |
|---|---|
| Table(s) query(-ies) | Customer, Booking, Hotel |
| Search criteria | |
| Grouping | firstName, surname |
| Having | SUM(pricePersonNight * numberNights * numberInParty) >= 2000 |
| Sort order | SUM(pricePersonNight * numberNights * numberInParty) DESC |

## Example 3: HAVING with conditional statement

Display the average price per person, per night for each holiday resort. Display only those resorts with an average price per person, per night that exceeds £100.

| Field(s)/ calculation(s) | resortName, Average Price = AVG(pricePersonNight) |
|---|---|
| Table(s) query(-ies) | Resort, Hotel |
| Search criteria | |
| Grouping | resortName |
| Having | AVG(pricePersonNight) > 100 |
| Sort order | |

## Example 4: NOT operator

Display the name and type of non-coastal resort, together with the name and meal plan for each hotel that meets these criteria.

| Field(s)/ calculation(s) | resortName, resortType, hotelName, mealPlan |
|---|---|
| Table(s) query(-ies) | Resort, Hotel |
| Search criteria | resortType NOT "coastal" |
| Grouping | |
| Having | |
| Sort order | |

## Example 5: BETWEEN operator with numeric values

Display the full name and total number of bookings made by each customer who has made between two and four bookings.

| Field(s)/ calculation(s) | firstName, surname, Total Bookings = COUNT(*) |
|---|---|
| Table(s) query(-ies) | Customer, Booking |
| Search criteria | |
| Grouping | surname, firstName |
| Having | COUNT(*) BETWEEN 2 and 4; |
| Sort order | |

## Example 6: BETWEEN operator with text

Display the surname, postcode, and town of customers who live in towns that begin with the letters 'E' through to 'M'. The query should list customers in alphabetical order of town.

| Field(s)/ calculation(s) | surname, postcode, town |
|---|---|
| Table(s) query(-ies) | Customer |
| Search criteria | town BETWEEN "E" and "M" |
| Grouping | |
| Having | |
| Sort order | town ASC |

## Example 7: IN operator

Display the hotel name and meal plan for hotels that offer room only, half board or full board.

| Field(s)/ calculation(s) | hotelName, mealPlan |
|---|---|
| Table(s) query(-ies) | Hotel |
| Search criteria | mealPlan IN the list ("Room Only", "Half Board", "Full Board") |
| Grouping | |
| Having | |
| Sort order | |

## Example 8: NOT with the IN operator

Display the name and type of resorts that are neither city nor country resorts.

| Field(s)/ calculation(s) | resortName, resortType |
|---|---|
| Table(s) query(-ies) | Resort |
| Search criteria | resortType NOT IN the list ("city", "country"); |
| Grouping | |
| Having | |
| Sort order | |

## Example 9: subquery in the where clause

Display the hotel name, star rating, and price per person for the most expensive hotel.

| Field(s)/ calculation(s) | hotelName, starRating, pricePersonNight | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Hotel | | | |
| | | | | |
| Search criteria | pricePersonNight = | Inner query | Field(s)/ calculation(s) | MAX(pricePersonNight) |
| | | | Table(s) | Hotel |
| | | | Search criteria | |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 10: subquery in the where clause

Display the resort name, hotel name, and star rating of all hotels that have a below-average star rating.

| Field(s)/ calculation(s) | resortName, hotelName, starRating | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Resort, Hotel | | | |
| | | | | |
| Search criteria | starRating < | Inner query | Field(s)/ calculation(s) | AVG(starRating) |
| | | | Table(s) | Hotel |
| | | | Search criteria | |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 11: subquery using the NOT operator

Display the full name and postcode of the customer who booked the same hotel as the customer with ID 111.

| Field(s)/ calculation(s) | resortName, hotelName, starRating | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Resort, Hotel | | | |
| | | | | |
| Search criteria | customerNo NOT 111 | | | |
| | AND hotelRef = | Inner query | Field(s)/ calculation(s) | hotelRef |
| | | | Table(s) | Booking |
| | | | Search criteria | customerNo = 111 |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 12: subquery using the IN operator

Display the hotel name and star rating of all hotels booked by the customer with ID 315.

| Field(s)/ calculation(s) | hotelName, starRating | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Hotel | | | |
| | | | | |
| Search criteria | hotelName IN | Inner query | Field(s)/ calculation(s) | hotelName |
| | | | Table(s) | Hotel, Booking |
| | | | Search criteria | customerNo = 315 |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 13: subquery using the NOT and IN operators

Display the names and types of resort **not** booked by the customer with ID 315.

| Field(s)/ calculation(s) | resortName, resortType | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Resort | | | |
| | | | | |
| Search criteria | resortName NOT IN | Inner query | Field(s)/ calculation(s) | resortName |
| | | | Table(s) | Resort, Hotel, Booking |
| | | | Search criteria | customerNo = 315 |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 14: subquery using the ANY operator

Display the customer number, hotel reference, and booking cost for any booking that costs more than any bookings made by customers with surnames Lowden, Shawfair or Sheriffhall.

| Field(s)/ calculation(s) | customerNo, hotelRef, Booking Cost = pricePersonNight * numberNights * numberInParty | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Booking. Hotel | | | |
| | | | | |
| Search criteria | pricePersonNight * numberNights * numberInParty > ANY | Inner query | Field(s)/ calculation(s) | pricePersonNight * numberNights * numberInParty |
| | | | Table(s) | Booking, Hotel, Customer |
| | | | Search criteria | surname in ("Sheriffhall", "Lowden", "Shawfair") |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

## Example 15: subquery using the EXISTS operator

Display the details (hotel name, star rating, meal plan and resort name) of all 3-star hotel bookings. The query should list the hotels in alphabetical order of meal plan.

| Field(s)/ calculation(s) | hotelName, mealPlan, starRating, resortName | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Hotel, Resort | | | |
| | | | | |
| Search criteria | starRating = 3 | | | |
| | AND EXISTS | Inner query | Field(s)/ calculation(s) | * |
| | | | Table(s) | Booking |
| | | | Search criteria | |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | mealPlan ASC | | | |

## Example 16: subquery using the NOT and EXISTS operators

Display the full name and address of customers who have never made a booking.

| Field(s)/ calculation(s) | firstName, surname, address | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Customer | | | |
| | | | | |
| Search criteria | NOT EXISTS | Inner query | Field(s)/ calculation(s) | |
| | | | Table(s) | Booking |
| | | | Search criteria | |
| | | | | |
| Grouping | | | | |
| Having | | | | |
| Sort order | | | | |

# Example 17: query requiring two subqueries

Display the name, star rating, and total number of customer nights booked for hotels that have:

♦ a total number of customer nights booked that is more than the total number of nights booked by the customer with ID 290 (number of nights booked multiplied by number in party)

**and**

♦ a star rating which is less than that of the hotel with the highest star rating

The query should list the hotels from lowest star rating to the highest.

| Field(s)/ calculation(s) | hotelName, starRating, Nights x Number in Party = SUM(numberNights*numberInParty) | | | |
|---|---|---|---|---|
| Table(s) query(-ies) | Hotel, Booking | | | |
| | | | | |
| Search criteria | numberNights * numberInParty > | Inner query | Field(s)/ calculation(s) | SUM(numberNights * numberInParty) |
| | | | Table(s) | Booking |
| | | | Search criteria | customerNo = 290 |
| | | | | |
| Search criteria | AND starRating < | Inner query | Field(s)/ calculation(s) | MAX(starRating) |
| | | | Table(s) | Hotel |
| | | | Search criteria | |
| | | | | |
| Grouping | hotelName, starRating | | | |
| Sort order | starRating ASC | | | |

# Appendix 6: server-side process design (WDD)

In this course, candidates are required to read and understand pseudocode, and structure diagram designs for server-side processes. They are also required to write pseudocode for design server-side processes.

Processes can include:

♦ opening and closing a database connection
♦ initialising and assigning session variables
♦ selection using conditions
♦ executing SQL statements
♦ displaying the results of SQL queries

Examples of these processes are in the structure diagrams and pseudocode below.

## Example 1: executing an SQL query and displaying results

In this example, the user enters search criteria into a web form to find contact details for companies in a 'Suppliers' database. The results of the query are displayed in an HTML table.

**Pseudocode**

1    assign server connection variables
2    open connection to Suppliers database on database server
3    receive search criteria from HTML 'find suppliers' form
4    assign search criteria to PHP variables
5    execute SQL query to find company names and phone numbers of selected suppliers
6    display the results of the query in an HTML table
7    close connection to Suppliers database server

6.1 if number of rows = 0
6.2     display 'no companies found'
6.3     else
6.4     display opening HTML table element
6.5     display field names (companyName, phoneNo) in header row of the HTML table
6.6     display names and phone numbers results in individual HTML table rows
6.7     display closing HTML table element
6.8 end if

**Structure diagram**
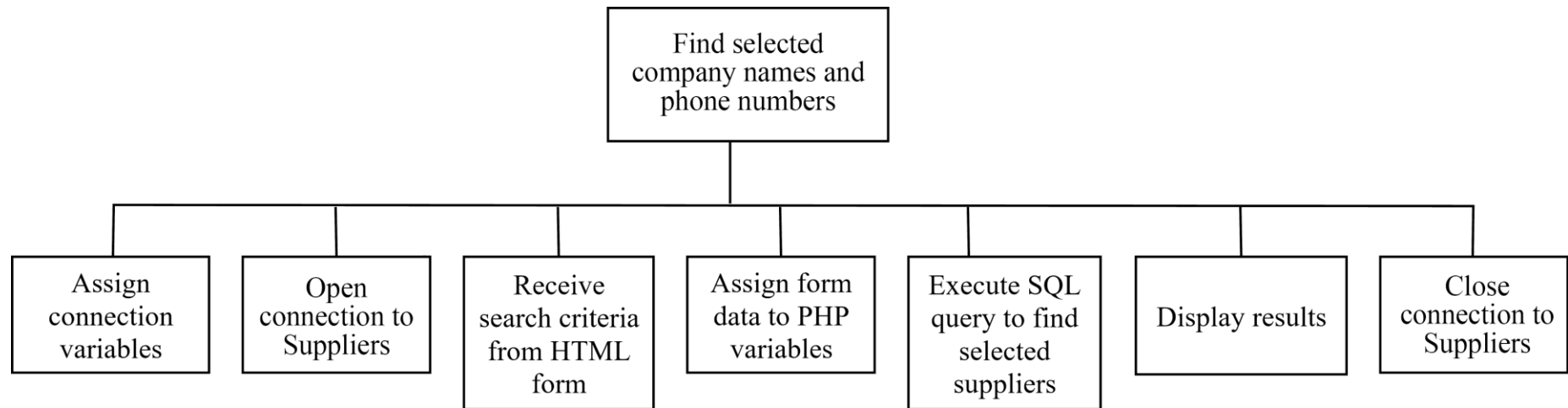
Refinement of 'Display results'
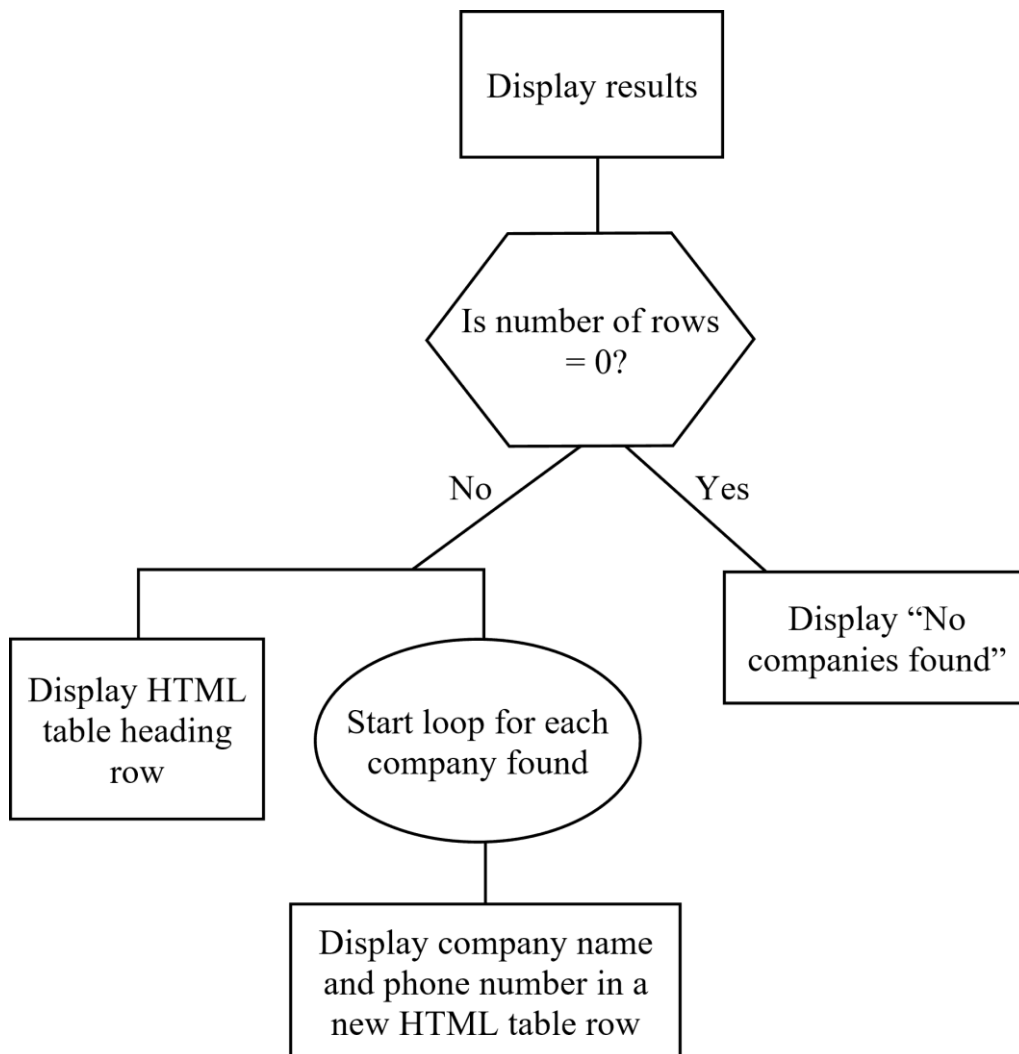
# Example 2: authenticating a user login

In this example, the user (customer) needs to log into a website. The customer username and password are authenticated by checking that the values exist within a 'Customers' database. Once authenticated, the customer login details are stored in PHP session variables.

## Pseudocode

1    start PHP session
2    use HTML to display login form
3    authenticate username and password submitted by the customer
4    if authenticated, assign contents of login variables to session variables

3.1 assign server connection variables
3.2 open connection to Customers database on database server
3.3 assign customer login details to PHP variables
3.4 execute SQL query to confirm customer login details
3.5 close connection to Customers database server

**Structure diagram**
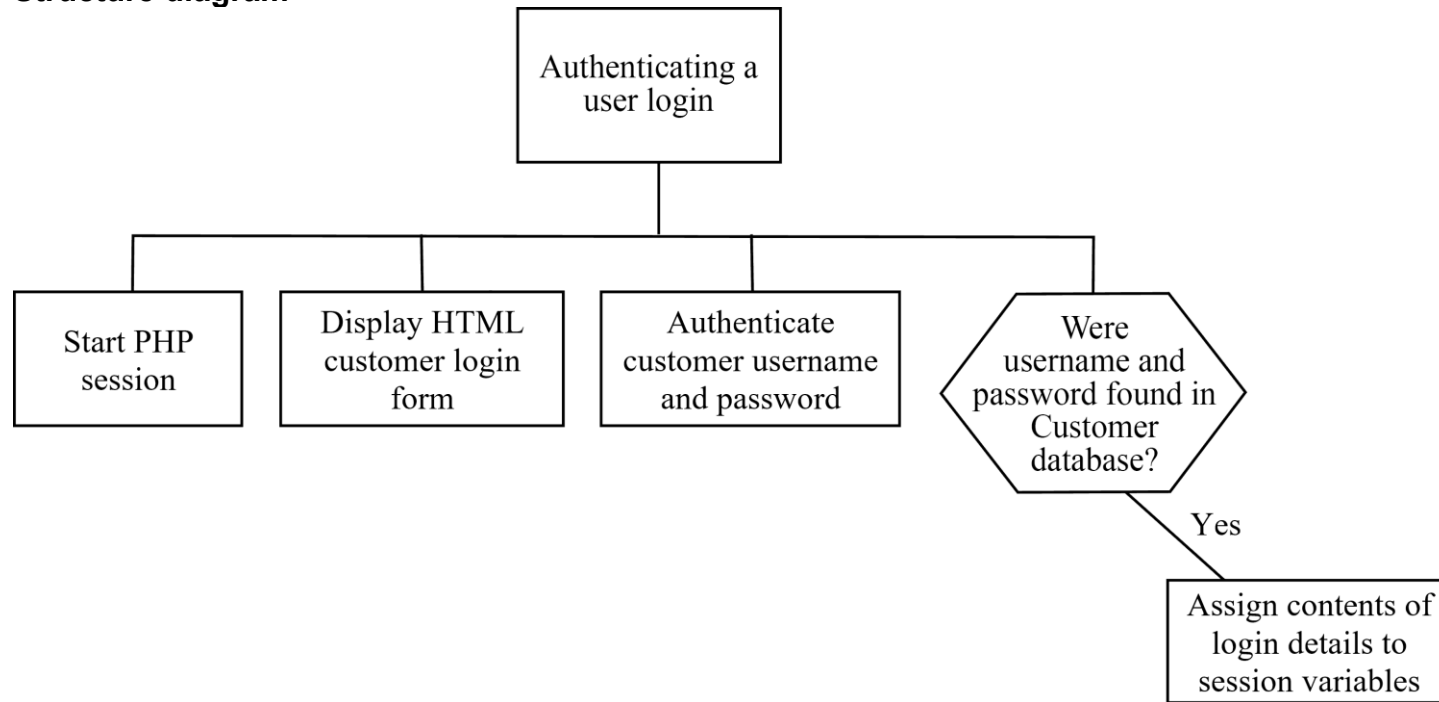


Refinement of 'Authenticate customer username and password'
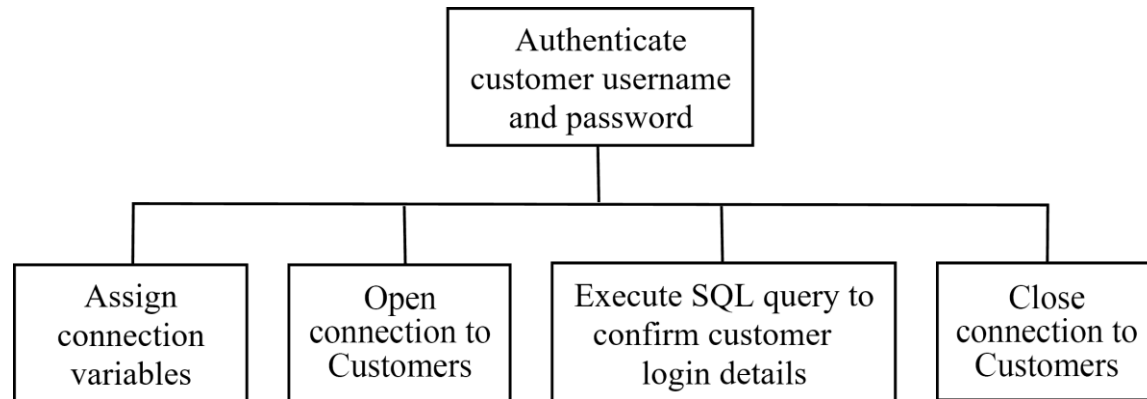
# Appendix 7: linked lists (SDD)

## Single linked list

A linked list is a dynamic data structure. Unlike a 1-D array (which stores each piece of data sequentially in memory), a linked list stores each data item and a pointer (address) to the next data item.

A linked list is a dynamic data structure, as it has no fixed size — it grows and shrinks as required; whereas a 1-D array typically has a set size based on its declaration.

Each element of a linked list is called a NODE. The start of a linked list is called the HEAD and the last element points to the NULL. Each node has its own address in memory, and stores the data item and a pointer to the next node.

The following diagram represents a node.
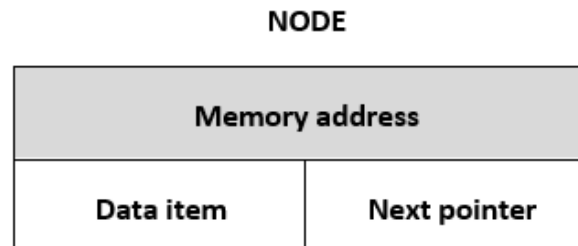


A simple example of a single linked list with four nodes is shown below. The four node linked list stores the words 'Computing', 'Science', 'is', and 'fun'.
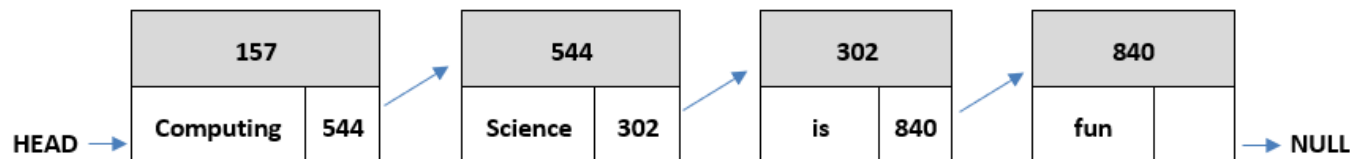
Points to note:

♦ A linked list can store data of multiple data types; a 1-D array is usually limited to one.
♦ A linked list is a linear data structure; to get to a specific data item, it must always start at the HEAD and work through each node until the data is found.
♦ A single linked list can only be traversed in one direction — from HEAD to NULL.
♦ Inserting data into a linked list is more efficient than a 1-D array, as only a pointer is changed rather than shifting the contents of the list (array) into different memory locations.
♦ Deleting data from a linked list is more efficient than a 1-D array, as only a pointer is changed rather than shifting the contents of the list (array) into different memory locations.

## Inserting new data

To insert new data into the list, for example inserting the word 'really' between 'is' and 'fun', a new node is created somewhere in memory and the pointers updated accordingly. To then update the pointer, the list is traversed until 'is' is found.

**Original**
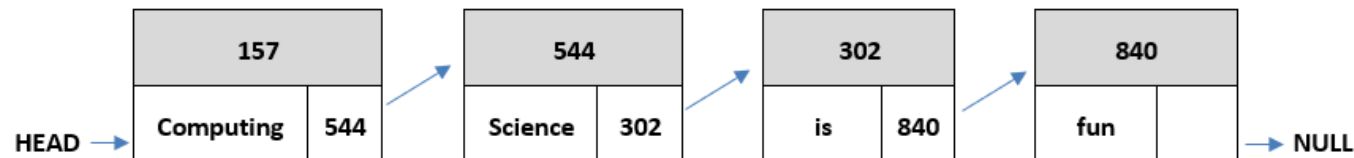


The following is an updated diagram with the word 'really' inserted at memory location 555.
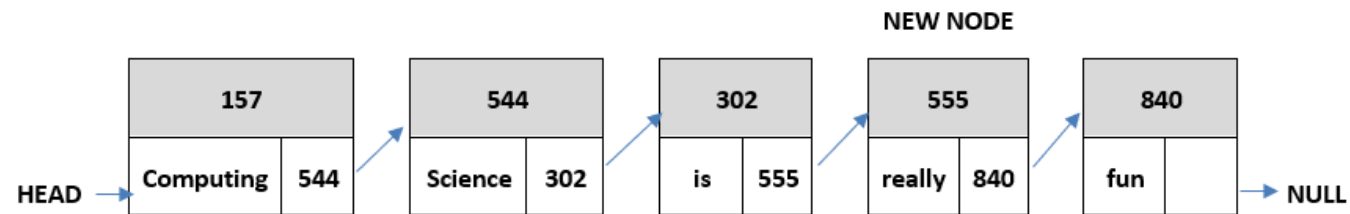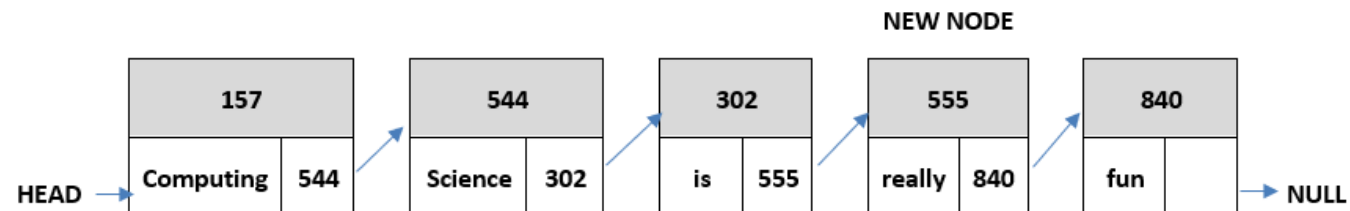
**Updated**



Using a 1-D array data structure and inserting data at a given index means that all data beyond that point is shifted along one location in memory. A linked list is more efficient than a 1-D array, as no data is moved and just one pointer is updated.
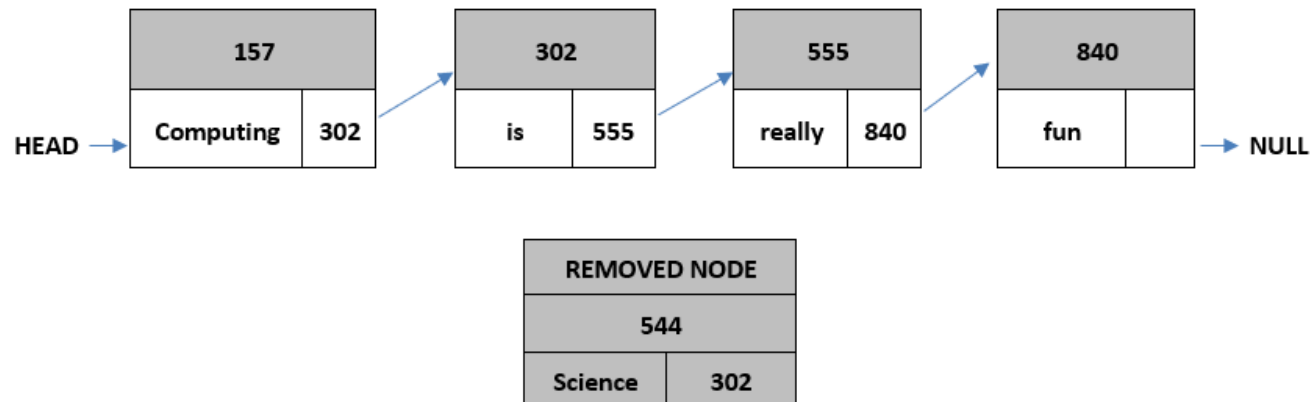
## Removing data

To remove data from the list, for example removing the word 'Science', the memory location where the node is stored is freed up and the pointer on the node removed before it is updated. To do this, the list is traversed until the node before 'Science' is found.

**Original**



The following is an updated diagram with the word 'Science' removed.

**Updated**



Using a 1-D array data structure and removing data at a given index means that all data beyond that point is shifted along one location in memory. A linked list is more efficient than a 1-D array, as no data is moved and just one pointer is updated.
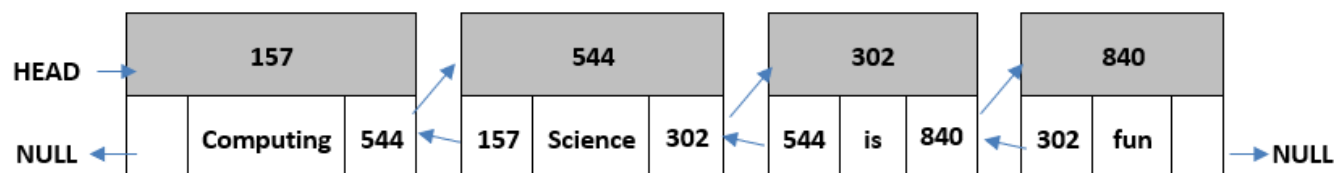
## Double linked list

A double linked list is very similar to a single linked list, but has an additional pointer in each node that stores the address of the previous node.

The following diagram represents a node.
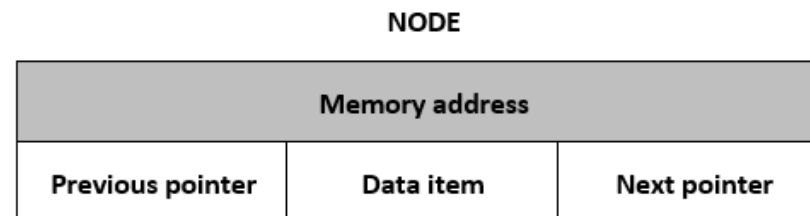
Using the same example as for the single linked list, a sample of a double linked list with four nodes is shown below. The four node linked list stores the words 'Computing', 'Science', 'is', and 'fun'.



Points to note:

♦ A double linked list can be traversed in both directions.

♦ A double linked list requires additional memory, as an extra pointer is being stored on each node.

♦ If the pointer to the node to be removed is known, then removing a node in a double linked list is more efficient than in a single linked list:
  — In a single linked list, to remove a node, the pointer from the previous node is required — to find the pointer, the list is traversed.
  — In a double linked list, the previous node is determined using the previous pointer.

♦ To insert a node into a single linked list, the list is traversed until the position is found.

♦ To insert a node into a double linked list, the list is not traversed if the node is being inserted:
  — at the start of the list
  — at the end of the list
  — after a given node

    **or**

  — before a given node
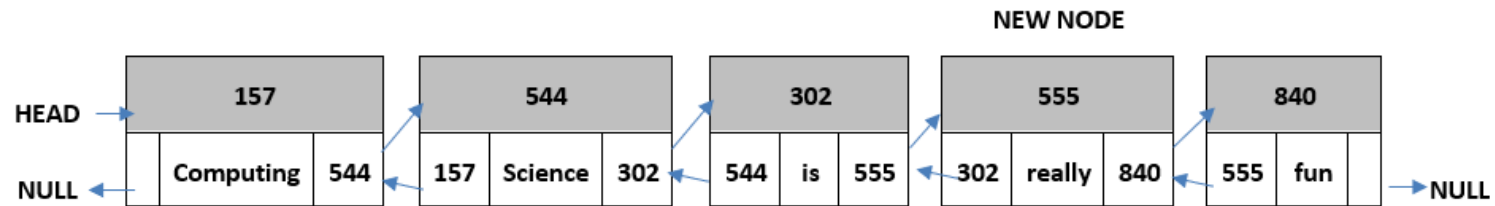
## Inserting new data

To insert new data into the list, for example the word 'really' to go after the node at address 302, a new node is created somewhere in memory and the pointers before it and after it are updated accordingly.

**Original**



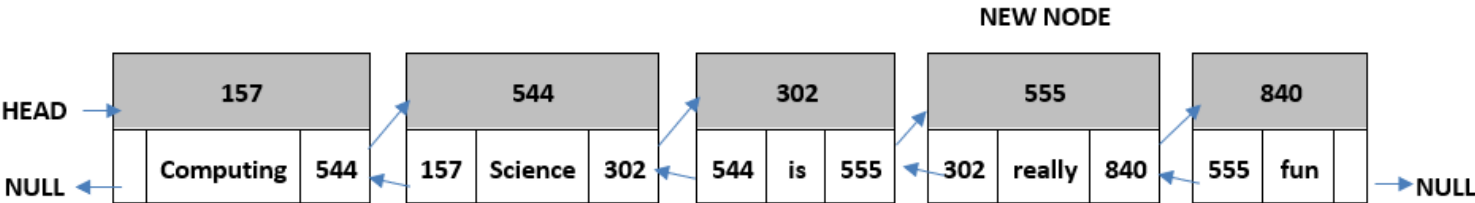The following is an updated diagram with the word 'really' inserted at memory location 555.
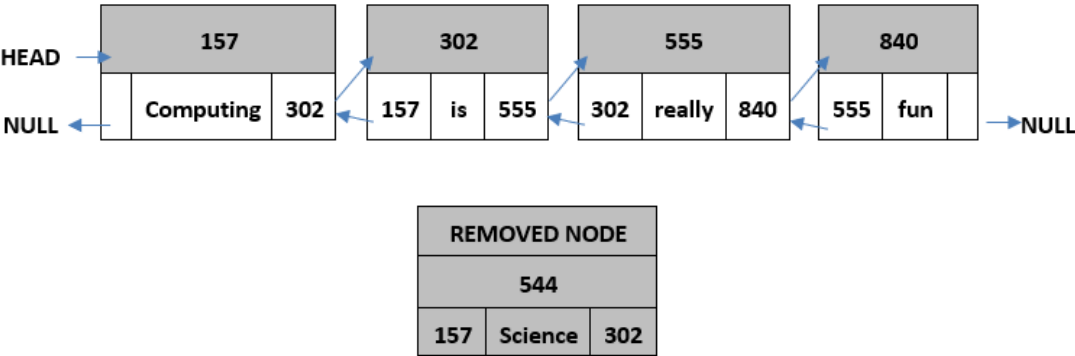
**Updated**

## Removing data

To remove data from the list, for example the word 'Science', the memory location where the node is stored is freed up and the pointer on the node before and after it is updated.

**Original**



The following is an updated diagram with the word 'Science' removed.

**Updated**

# Appendix 8: connecting to a database using a programming language (SDD)

The Advanced Higher Computing Science course specifies that candidates use a programming language to read from, and write data to, database files using SQL. Python, Visual Basic and Java are all popular languages used by many centres to deliver the course content. Note: all of these languages can create a database connection and execute an SQL statement.

The question paper will only contain SQA's standardised reference language, so the code included in this appendix does not appear in the question paper. This appendix focuses on supporting teachers and lecturers to deliver the content, and helping candidates develop their projects.

For each of the three languages above, the following is included:

♦ advice on set-up requirements
♦ examples of instructions and syntax required to create a database connection
♦ examples of SQL execution
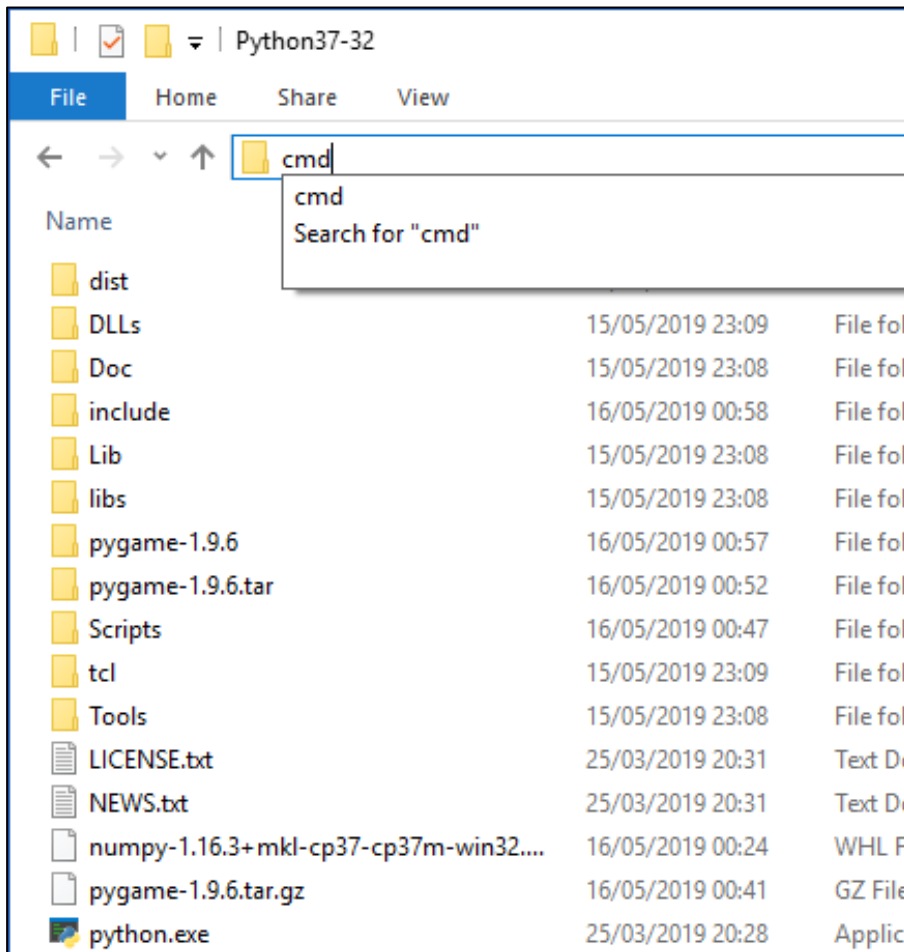
## Python

### Set-up requirements

To connect to a MySQL database using Python, the database driver 'MySQL Connectors' must be installed. In a school or college, IT technicians will probably install this, as teachers and lecturers are unlikely to have the required administration rights.

If candidates want to install Python at home, they can use the following instructions for Windows 10. Similar instructions for Linux or Apple OS are available online — these set-up instructions assume that Python is already installed.

### Step 1 — checking the system path to Python is set up

Before installing Python, check that a system path is set up. This ensures that the operating system knows where the python.exe application is located.

Open the folder containing the python.exe program. Click on the address bar at the top of the window, type 'cmd' and press enter to open the command window.



Type 'python' in the command window. If the system path is already set up, a message stating the version of Python installed is displayed — move on to step 2.
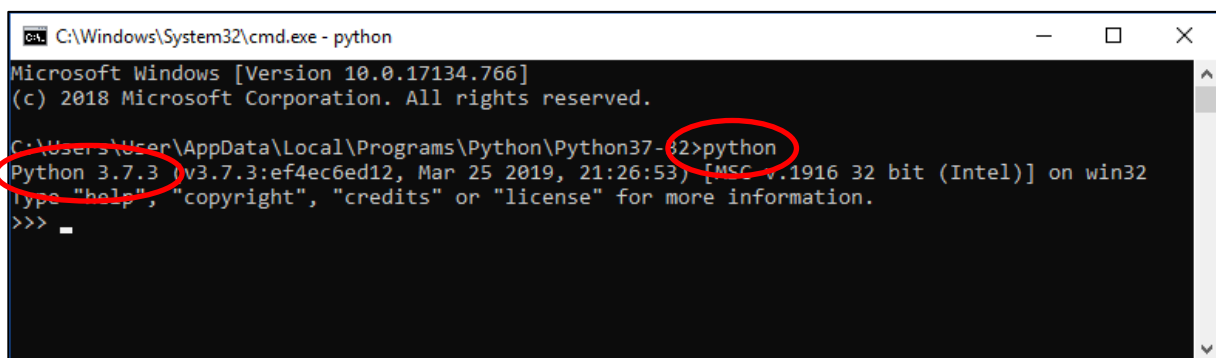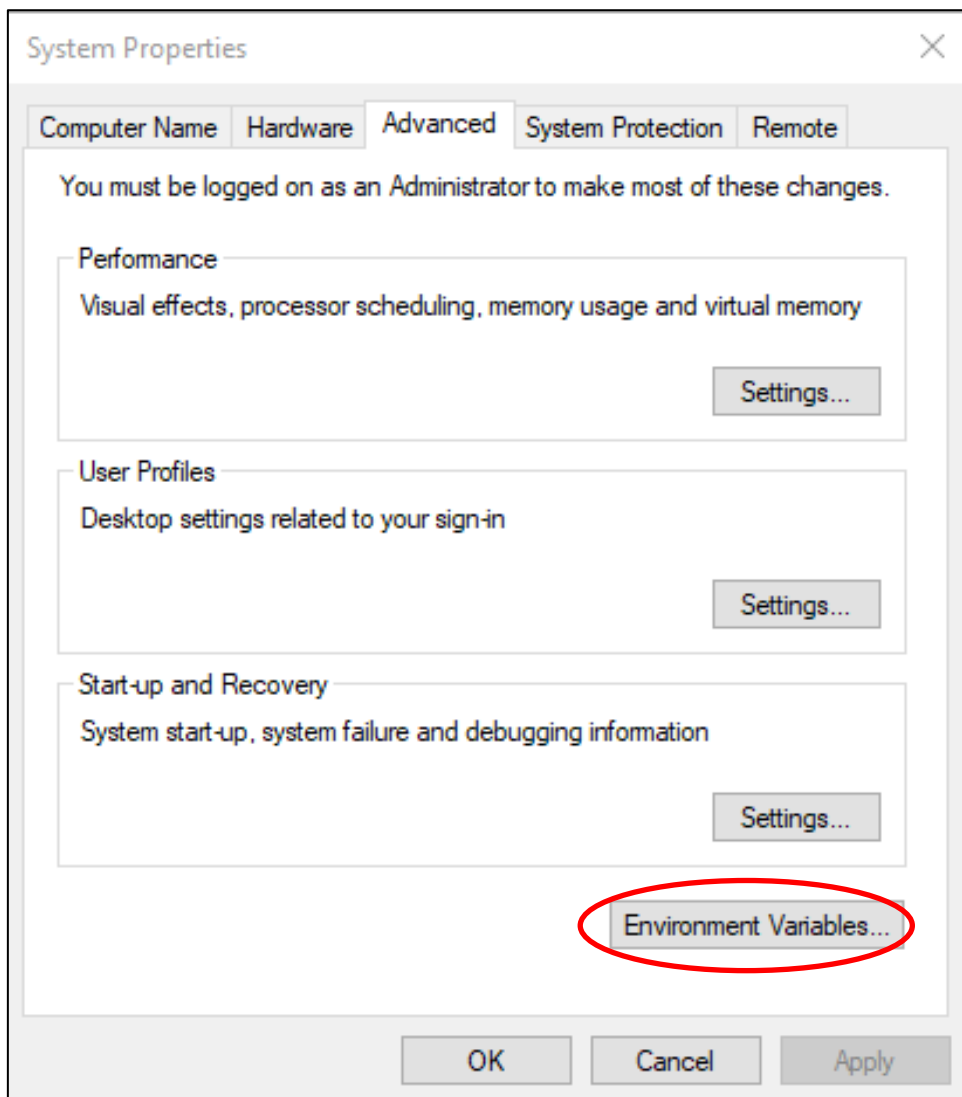


If an error is displayed, close the command window, click on the address at the top of the window again and copy the address. The address is required later.

Open the 'Window Settings' folder and type 'advanced' into the search bar at the top.
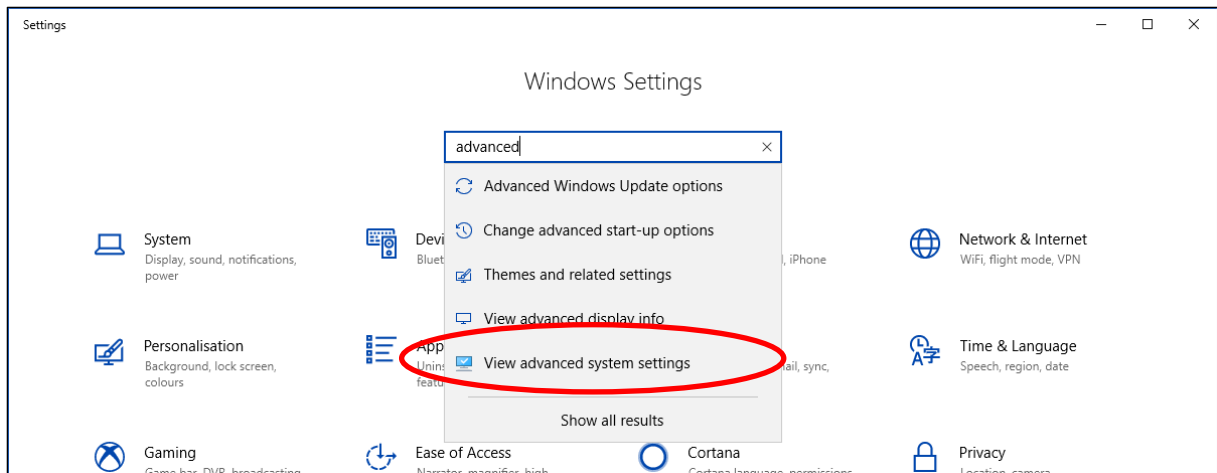


Select 'View advanced system settings' followed by Environment Variables.

The system path is set up using this window. Click 'Path' followed by the 'Edit' button, as shown below.
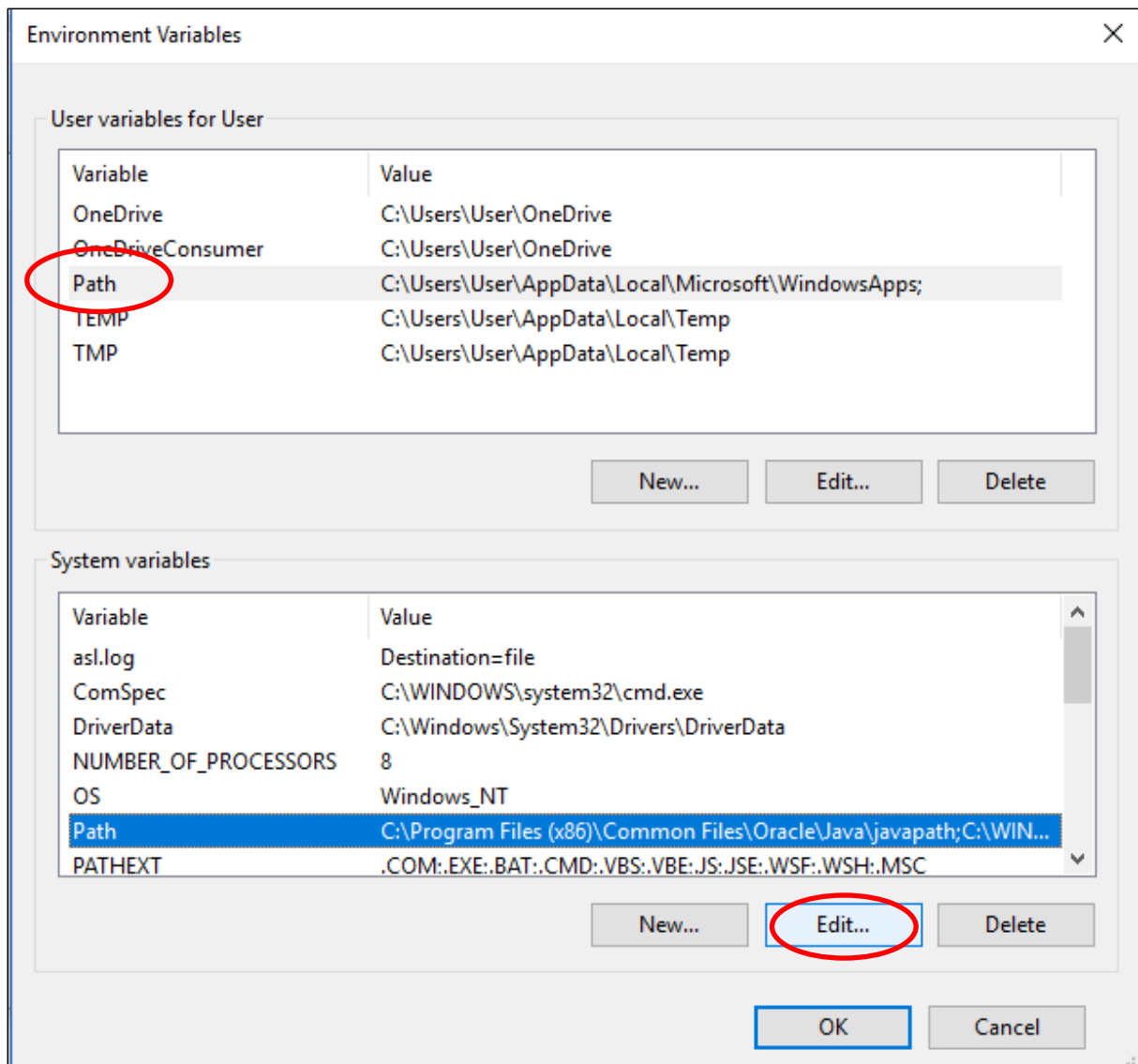


In the 'Edit environment variable' window, click 'New' and paste in the location of Python copied earlier from the address bar of the Python window.

**Step 2 — installing the 'mysql-connector' library using pip**

Python maintains a list of online installable libraries. Any of these libraries can be installed using the cmd prompt from within the Python folder (see step 1).

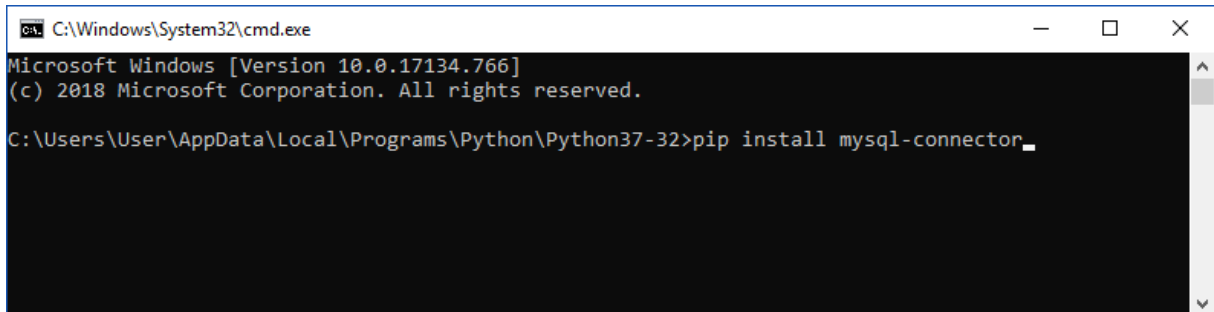Open the window containing the python.exe file, and type 'cmd' into the address bar to open the command window.

Enter the following instruction: `pip install mysql-connector`



Note: if when using 'pip install' it generates the error "`'pip' is not recognized as an internal or external command, operable program or batch file.`", then pip also requires a system path set up. Repeat the system path instructions using the address of the pip.exe file. You can find this file inside the Python Scripts folder. Once the path is added, close and reopen the cmd window from the Python folder. Re-enter the `pip install mysql-connector` instruction.

## Creating a connection

The code below creates a connection to a MySQL database.

```
 1   import mysql.connector
 2
 3   try:
 4       conn = mysql.connector.connect(
 5       host="localhost",
 6       user="root",
 7       passwd="",
 8       database="StudentData"
 9       )
10   except:
11       print("Database connection error")
12   else:
13       ## Further code goes here
14
```

Line 1 imports the mysql-connector library.

Lines 3 to 8 assign the database connection parameters for a chosen database and make a connection, '`conn`'.

To ensure connection errors do not crash the program, place the connection code inside a Python try structure. The '`try`' statement prints an error if the connection fails. If the connection is successful, any code placed within the '`else`' statement will be executed.

## SQL execution

The following examples use a single table. Set this up using MySQL before any code is executed, ensuring that:

♦  the database name is 'StudentData'
♦  the table name is 'Student'

A data dictionary for the Student table is shown below.

| Entity | Attributes | Type | Size |
|---|---|---|---|
| Student | studentid | int | 4 |
| | firstname | VARCHAR | 25 |
| | lastname | VARCHAR | 25 |
| | address | VARCHAR | 40 |

Insert example data for the table before executing the examples.

| | studentid | firstname | lastname | address |
|---|---|---|---|---|
| ▶ | 1001 | Jane | White | 12 Holburn Crescent |
| | 1002 | Mary | Cromwell | 4 Fraser Street |
| | 1003 | Tessa | Bolden | 10 Fraserboo St |
| * | NULL | NULL | NULL | NULL |

### Example 1 — SELECT and display results

The code below displays every row from the Student table.

```
12  else:
13      mycursor = conn.cursor()
14      mycursor.execute("SELECT * FROM Student")
15      myresult = mycursor.fetchall()
16
17      for x in myresult:
18          print(x)
```

Line 13 associates the database connection with a new instance of a `cursor` object. Cursor objects contain a variety of methods used to manipulate databases and data.

Line 14 uses the cursor's `execute()` method to execute an SQL statement.

Line 15 uses the `fetchall()` method to return the result as a list of tuples as shown below.

```
[(1001, 'Jane', 'White', '12 Holburn Crescent'), (1002, 'Mary',
'Cromwell', '4 Fraser Street'), (1003, 'Tessa', 'Bolden', '10
Fraserboo St')]
```

Lines 17 and 18 display each of the tuples, on a single line, generating the output shown below.

```
(1001, 'Jane', 'White', '12 Holburn Crescent')
(1002, 'Mary', 'Cromwell', '4 Fraser Street')
(1003, 'Tessa', 'Bolden', '10 Fraserboo St')
```

Rather than displaying the whole tuple unformatted, edit line 18 to separate out each of the four values within each tuple (`x[0], x[1], x[2]` and `x[3]`) to display a concatenated string.

```
17    for x in myresult:
18        print(str(x[0])+"- "+x[1]+" "+x[2]+", "+x[3])
```

This produces the formatted output shown below.

```
1001- Jane White, 12 Holburn Crescent
1002- Mary Cromwell, 4 Fraser Street
1003- Tessa Bolden, 10 Fraserboo St
```

### Example 2 — INSERT using user inputted values

The code below uses input boxes to input and store the details of a new student.

```
20    studentid = int(input("Please enter student id"))
21    fname = str(input("Please enter student's forename"))
22    sname = str(input("Please enter student's surname"))
23    address = str(input("Please enter student's address"))
24
25    insert_Student = "INSERT INTO Student () VALUES (%s, %s, %s, %s)"
26    data = (studentid, fname, sname, address)
27    mycursor.execute(insert_Student, data)
28    conn.commit()
```

Lines 20 to 23 ask the user to input data for a new student.

Lines 25 and 26 build an `INSERT` statement. The placeholders, used in place of values, are replaced by the variables specified in line 26, when line 27 is executed.

Line 28 is required to confirm the change to the database.

**Example 3 — counting the number of rows returned by a query**

The code below asks the user to enter a name. The number of times that name appears in the Student table is displayed.

```
30      findName = """SELECT * FROM Student WHERE firstname = %s"""
31      nameData = str(input("Please enter forename to find"))
32      mycursor.execute(findName, (nameData,))
33      rows = mycursor.fetchall()
34      print("The number of " + nameData + "s is: " + str(mycursor.rowcount))
```

Lines 30 to 32 build and execute a SELECT statement. A placeholder is replaced by the user's input.

Line 34 displays concatenated output, including the number of rows returned by the SELECT statement. Note: the rows must be fetched before the method rowcount can be used (line 33).
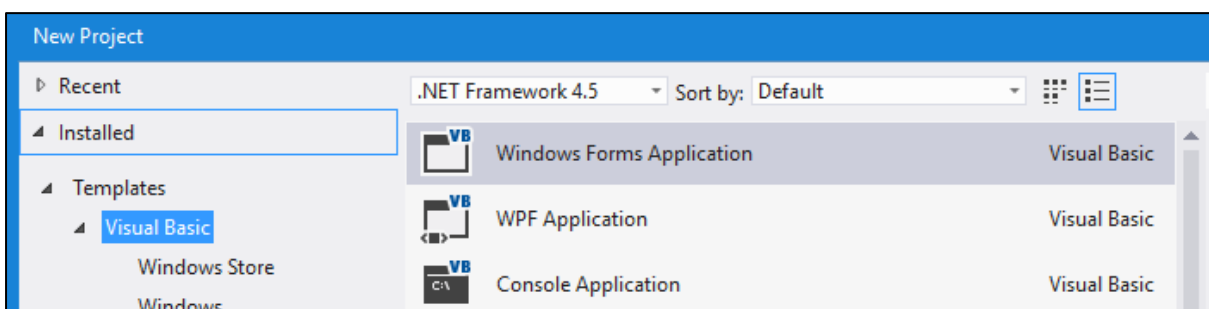
# Visual Basic

## Set-up requirements

The following instructions are for:

♦ Microsoft Visual Studio 2012 or later

♦ Microsoft Access 2016

This code should still be compatible with newer editions of the software.

## Creating a connection

Load Visual Basic and create a new Windows Forms Application.

Add the following to the blank SQL Connection Form:

- one list box named 'lstoutput'
- one command button named 'cmd_read'
- one text box named 'txterror'
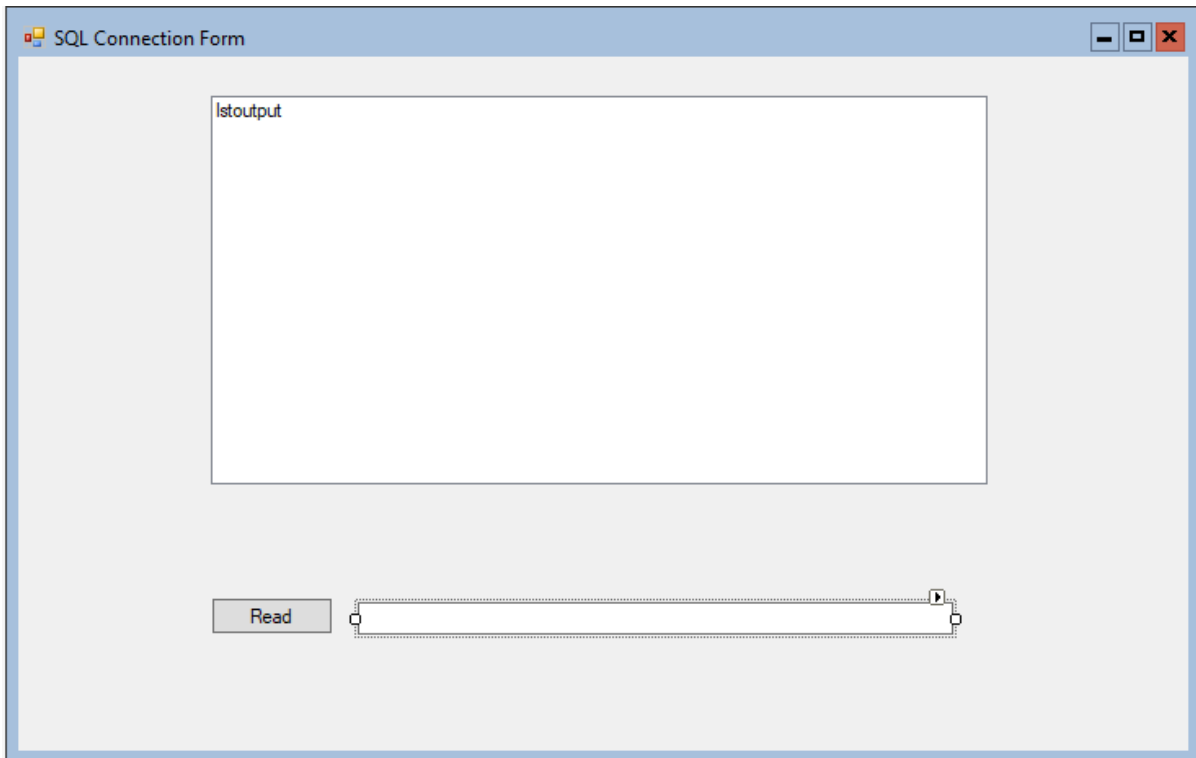


Double click on the command button to bring up the coding window.

Add the highlighted line of code to the very top of the code, ensuring it is above the Form Class Code — as shown below.
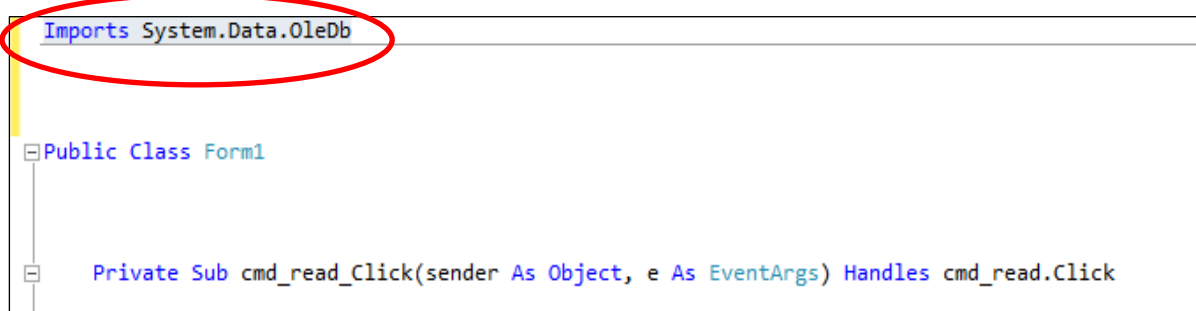


This adds the required additional libraries.

A 'Try Catch' block is used to connect to the database. If any code within the try block returns an error, the catch block is called to display the returned error message.

Add the following code to the 'cmd_read' button.

```
Try


Catch ex As Exception

      lstoutput.Items.Add(ex.Message)

End Try
```

Add all subsequent code between the 'Try' and 'Catch' statements.

Use the following code to create a connection to the example database.

```
Dim SQLReader As OleDbDataReader
Dim connection_type As String = "Provider=Microsoft.ACE.OLEDB.12.0;"
Dim file_location As String = "Data Source=c:\desktop\test.accdb"
Dim conn As OleDbConnection
conn = New OleDbConnection(connection_type & file_location)
conn.Open()
```

The first line creates an object called SQLReader that is used to read data from the database.

Next, the connection type and the location of the database file are stored as strings.

A new object called conn is used to create the connection to the database.

The conn object is set as a new OleDBConnection, with the parameters stored earlier. Note: a single string is passed into this procedure, as the parameters have been concatenated.

The final line opens the connection to the database.

## SQL execution

The following examples use a simple one-table Access database:

♦ The database file is called **test.accdb**
♦ The table is called **Customers**

Screen shots of the table design and contents are shown below.





## Example 1 — SELECT and display results

The following code reads and displays all the data in the example database.

```
Dim query As String = "Select * FROM [Customers]"
Dim command As New OleDbCommand(query, conn)
SQLReader = command.ExecuteReader()

If SQLReader.HasRows Then
        While SQLReader.Read
           lstoutput.Items.Add(SQLReader("ID") & " " &
                SQLReader("Firstname") & " " & SQLReader("Surname"))
        End While
Else
        lstoutput.Items.Add("No Results Returned")
End If
```

A simple string object is created to store the SQL query. Note: table names require square brackets.

A new OleDbCommand object called command is created. This object contains the query and the connection data.

The SQLReader object stores the results of the executed query.

**Example 2 — INSERT using user inputted values**

The code below uses input boxes to input and store the details of a new customer.

Ask the user to enter the details of a new customer. Note: all the data entered has to be stored as string, regardless of the datatype in the database.
Add the following code to a new button.

```
Dim id As String = InputBox("Please enter customer ID")
Dim firstname As String = InputBox("Please enter customer's firstname")
Dim surname As String = InputBox("Please enter customer's surname")
```

Convert the stored data into an SQL query, as shown below.

```
Dim query As String = "INSERT INTO [customers]  VALUES ( " & id & " , ' " & firstname & " ', ' " & surname & " ');"
```

When inserting partial data, field names are required. ID information is not necessary, because ID is an auto number and the database uses the next available number.

```
Dim query As String = "INSERT INTO [customers] (firstname, surname) VALUES ( ' " & firstname & " ', ' " & surname & "');"
```

Note: the above example now specifies the two fields that data is entered into.

Execute the built query, as shown below.

```
Dim command As New OleDbCommand(query, conn)
SQLReader = command.ExecuteReader()
```

**Example 3 — counting the number of rows returned by a query**

The code counts the number of times that name appears in the Customer table.

Ask the user to enter the customer's name.

```
Dim firstname As String = InputBox("Please enter firstname of person(s) you would like to count")
```

A counter is required later to count the number of rows returned by the query.

```
Dim counter As Integer = 0
```

A SELECT statement is built using the user's input.

```
Dim query As String = "SELECT * FROM [customers] WHERE firstname = '" & firstname & "';"
```

To count the rows returned, add two additional lines to the output code used in example 1:

♦ one line to store the result of a running total for each row

♦ one line to display this result

```
Dim command As New OleDbCommand(query, conn)
SQLReader = command.ExecuteReader()

If SQLReader.HasRows Then
        While SQLReader.Read
           lstoutput.Items.Add(SQLReader("ID") & " " &
                SQLReader("Firstname") & " " & SQLReader("Surname"))
        counter = counter + 1
        End While

lstoutput.Items.Add(counter & " Results Returned")

Else
        lstoutput.Items.Add("No Results Returned")
End If
```

## Java

### Set-up requirements

The following Java database connection examples require two installations:

♦ Java SE Development Kit (often referred to as JDK) — this can be downloaded from the Oracle website.
https://www.oracle.com/technetwork/java/javase/downloads/index.html

♦ NetBeans — a popular integrated development environment (IDE) used to develop Java applications.
https://netbeans.org/features/

If candidates wish to code in Java at home, they can download and install both examples at no cost.

To connect to a database, Java Database Connectivity (JDBC) is required. JDBC drivers are software libraries that communicate between a Java application and a database. JDBC is already included in NetBeans, so requires no further installation.

If candidates use a different IDE for Java development, they must ensure that it includes the JDBC library, as this is required to create a database connection.

Follow the instructions below to download the JDBC library, if required:

1   Open the webpage
    https://dev.mysql.com/downloads/

2   Scroll down to MySQL Connectors and click the download link shown below.
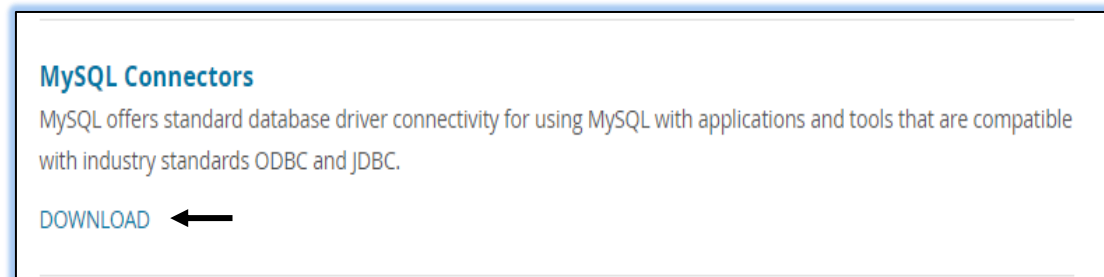


3   Select 'Connector/J' from the list.
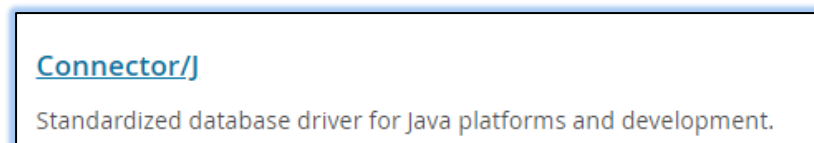


4   Choose 'Select Operating System: Select platform independent'

5   Download the ZIP or TAR file.
    Note: it is not necessary to login or sign up — click 'No thanks, just start my download'.

6   It does not matter where the JDBC library is saved.

**Creating a connection**

Before any coding can be implemented, a new project must be created.

Open NetBeans and create a new project using the following steps:

1   File
2   New
3   Java
4   Java Application
5   Name the application — the following example is named 'School Application'

To create a database connection, the JDBC library must be included in your project.

Right click on Libraries and select Add Library.
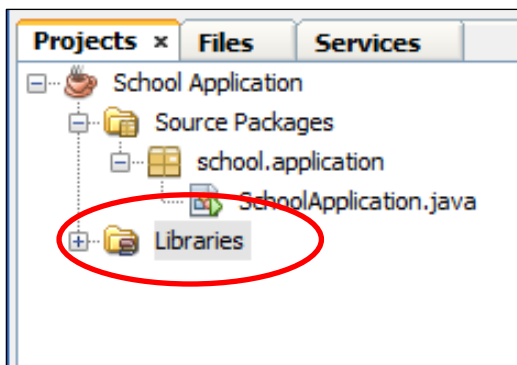


Select MySQL JDBC Driver from the list as shown.
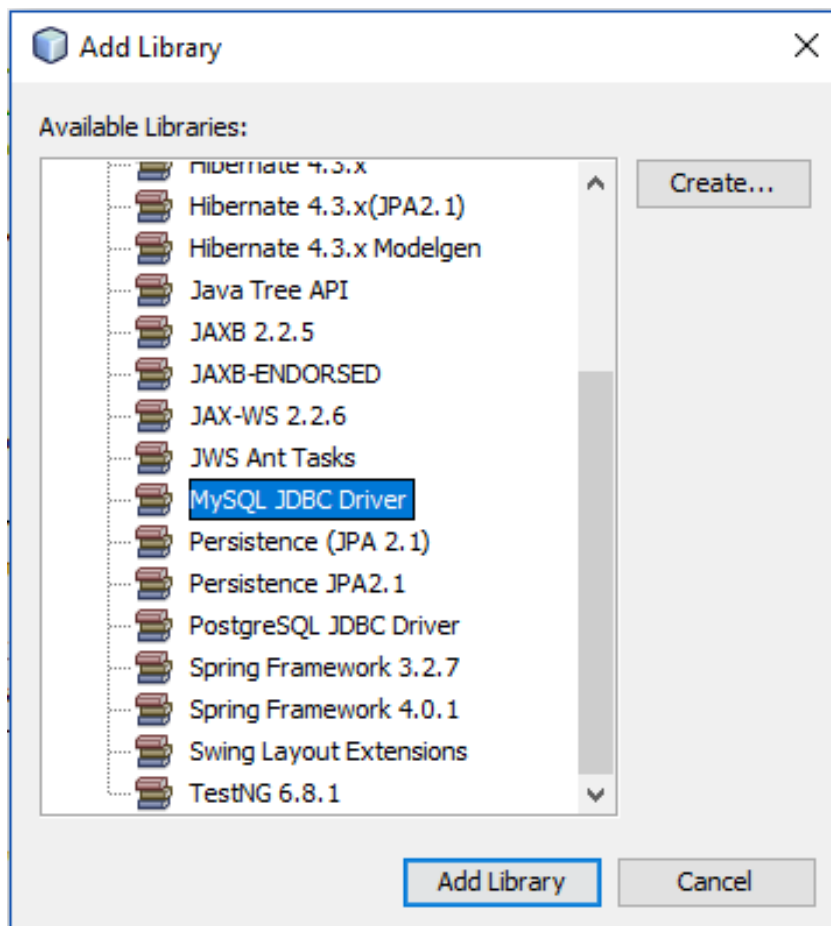
Use the code shown below create a database connection.

```
1    package school.application;
2
3    import java.sql.Connection;
4    import java.sql.DriverManager;
5    import java.sql.ResultSet;
6    import java.sql.SQLException;
7    import java.sql.Statement;
8    import javax.swing.JOptionPane;
9    import java.sql.PreparedStatement;
10
11   public class SchoolApplication {
12
13       private static final String USERNAME = "root";
14       private static final String PASSWORD = "";
15       private static final String CONNECTION_URL = "jdbc:mysql://localhost/StudentData";
16
17       public static void main(String[] args) throws SQLException {
18           try (
19               Connection conn = DriverManager.getConnection(CONNECTION_URL, USERNAME, PASSWORD))
20           {
21               System.out.println("Successfully connected to database");
22               //Additional Code Goes Here
23
24           } catch (SQLException e) {
25               System.err.println(e);
26           }
27
28       }
29
30   }
```

Lines 3 to 9 list several libraries that must be included at the top of the code. These contain methods that are called when creating the connection or executing SQL.

Lines 13 to 15 initialise three variables to the database connection parameters. This is similar to PHP but in Java, the server and database names are contained in a single URL.

Line 18 creates the connection using the parameters assigned above.

Place the connection within a '`try`' statement (lines 18 to 26), to ensure the program does not crash if the connection fails. If the connection is successful, a message stating this is displayed (this message can be used during testing and removed when the successful connection is confirmed).

```
es    Output - School Application (run)  ×
 run:
 Successfully connected to database
 BUILD SUCCESSFUL (total time: 0 seconds)
```

If the connection fails, a system error is printed stating the issue that occurred.

```
s     Output - School Application (run)  ×
 run:
 java.sql.SQLException: Access denied for user 'root'@'localhost' (using password: YES)
 BUILD SUCCESSFUL (total time: 0 seconds)
```

Further code should be contained within the success section of the '`try`' statement at line 22.

### SQL execution

The following examples use a single table. Set this up using MySQL before any code is executed, ensuring that:

♦ the database name is 'StudentData'
♦ the table name is 'Student'

A data dictionary for the Student table is shown below.

| Entity | Attributes | Type | Size |
|--------|-----------|------|------|
| Student | studentid | int | 4 |
| | firstname | VARCHAR | 25 |
| | lastname | VARCHAR | 25 |
| | address | VARCHAR | 40 |

Insert example data for the table before executing the examples.

| | studentid | firstname | lastname | address |
|---|---|---|---|---|
| ▶ | 1001 | Jane | White | 12 Holburn Crescent |
| | 1002 | Mary | Cromwell | 4 Fraser Street |
| | 1003 | Tessa | Bolden | 10 Fraserboo St |
| * | NULL | NULL | NULL | NULL |

### Example 1 — SELECT and display results

The code below displays every row from the Student table.
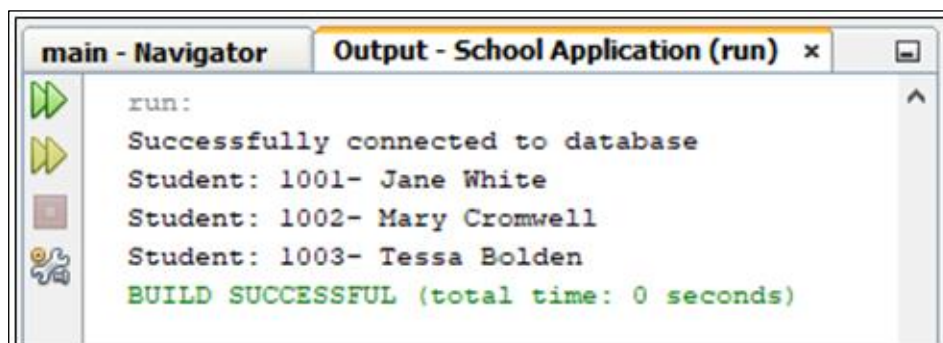
```
23            // SELECT statement example
24            Statement stmt = conn.createStatement();
25            ResultSet rs = stmt.executeQuery("SELECT * FROM Student");
26
27            while(rs.next()){
28                StringBuilder build = new StringBuilder();
29                build.append("Student" + ": ").append(rs.getInt("studentid"))
30                        .append("- ").append(rs.getString("firstname"))
31                        .append(" ").append(rs.getString("lastname"));
32                System.out.println(build.toString());
33            }
```

Line 24 creates a statement object that allows basic SQL queries to be executed.

Line 25 executes a SELECT query by calling the executeQuery(String) method with the SQL to be used. The results of a query are retrieved through the ResultSet class.

Lines 27 to 33 exemplify one of many solutions that could be used to display the result of the query, now stored in 'rs'. This solution uses StringBuilder to concatenate each row of the results.

The output from this code is shown below.

```
main - Navigator    Output - School Application (run)  ×     ⬛
run:
Successfully connected to database
Student: 1001- Jane White
Student: 1002- Mary Cromwell
Student: 1003- Tessa Bolden
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Example 2 — INSERT using user inputted values

The code below uses input boxes to input and store the details of a new student.

```
35                    // INSERT INTO Example with User Input from Dialog Box
36              String studID = JOptionPane.showInputDialog("Enter the Students ID");
37              String first = JOptionPane.showInputDialog("Enter First Name");
38              String last = JOptionPane.showInputDialog("Enter Surname");
39              String address = JOptionPane.showInputDialog("Enter Address");
40              // input dialog returns always a string so convert id to integer
41              double id = Double.parseDouble(studID);
42
43              String sql = "INSERT INTO Student VALUES (?,?,?,?)";
44              PreparedStatement addStudent = conn.prepareStatement(sql);
45              addStudent.setDouble(1, id);
46              addStudent.setString(2, first);
47              addStudent.setString(3, last);
48              addStudent.setString(4, address);
49              addStudent.executeUpdate();
```

Lines 36 to 41 use the JOptionPane library to create four input boxes, one for each item of student data. Note: as each input is stored as a string, it is necessary to convert the student id to an integer (line 41).

Line 43 creates an INSERT statement that contains four markers (?). PreparedStatement then replaces each marker with the student data stored earlier in the `id`, `first`, `last` and `address` variables.

Line 49 uses the `executeUpdate()` method to execute the now complete SQL statement.

To directly INSERT values, use the following code.

```
stmt.executeUpdate("INSERT INTO Student VALUES
(1010,'Cameron','Stott','17 Dover Heights')");
```

## Example 3 — counting the number of rows returned by a query

The code below displays the number of rows returned by an SQL `SELECT` query.

```
51                    // DISPLAY Number of Rows in SELECT
52              rs = stmt.executeQuery("SELECT * FROM Student WHERE firstname='Jane'");
53              rs.last();
54              System.out.println("Number of rows returned: " + rs.getRow());
```

Line 52 executes an SQL query to return all the students called Jane. Note: the ResultSet, used at the beginning of the same code earlier is not required, as the 'rs' object has already been initialised.

Line 53 uses the `last()` method to move the cursor to the last row of the result set.

Line 54 displays the current row number using the `getRow()` method.

# Appendix 9: standard algorithms (SDD)

The following Advanced Higher standard algorithms are exemplified below in pseudocode and SQA reference language:

♦ bubble sort
♦ insertion sort
♦ binary search

The two sort algorithms presented both sort into ascending order. With small changes, they are easily adapted to descending order.

## Bubble sort

A bubble sort continually swaps values in adjacent array elements until the entire list is in the correct order.

## Pseudocode

Consider an array that stores the following values:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 45 | 23 | 99 | 7 | 3 | 64 | 37 | 63 | 34 |

After one pass through the array, the largest value will always 'bubble' up to the end of the array.

| 23 | 45 | 7 | 3 | 64 | 37 | 63 | 34 | 99 |
|----|----|---|---|----|----|----|----|----|

After a second pass, the second-largest number is also sorted.

| 23 | 7 | 3 | 45 | 37 | 63 | 34 | 64 | 99 |
|----|---|---|----|----|----|----|----|----|

When bubble sorting a list of values, the number of iterations carried out by each nested loop can be reduced by one each pass. This improves the efficiency of the bubble sort algorithm.

| Design | Commentary |
|--------|------------|
| `n equals the length of an array called list` | The length of the array is stored in a variable |
| `set swapped to true` | |
| `start conditional loop while swapped = true` | |
| `   set swapped to false` | |
| `   fixed loop i = 0 to n - 2` | Loop from the first element to the penultimate array element |
| `      if list[i] > list[i+1] then` | |
| `         swap the two values` | |
| `         set swapped to true` | |
| `      end if` | |
| `   end fixed loop` | |
| `   n = n - 1` | Each fixed loop reduces the iterations by 1, as one more element is sorted correctly at the end of the array |
| `end conditional loop` | |

**SQA reference language: bubble sort implementation**

```
PROCEDURE bubble_sort(list)
     DECLARE n INITIALLY length(list)
     DECLARE swapped INITIALLY TRUE
     WHILE swapped
          SET swapped TO False
          FOR i = 0 to n-2 DO
               IF list[i] > list[i+1] THEN
                    SET temp TO list[i]
                    SET list[i] TO list[i+1]
                    SET list[i+1] TO temp
                    SET swapped TO TRUE
               END IF
          END FOR
          SET n TO n - 1
     END WHILE
END PROCEDURE
```

## Insertion sort

An insertion sort traverses an array from the second element to the last. Each element is compared to the elements before in turn, working backwards down the list. Values are swapped until the element being compared is placed in order.

The following is a worked example of an insertion sort.

### Iteration 1

Start with element 1 of the list to be sorted. This value is temporarily stored.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----|----|----|----|----|----|----|----|----|
| 45 | 23 | 99 | 7 | 8 | 64 | 37 | 63 | 34 |

temp

| 23 |
|----|

If the temporary value (23) is smaller than the value before it (45), then the value before it is copied to the right.

| 45 | 45 | 99 | 7 | 8 | 64 | 37 | 63 | 34 |
|----|----|----|----|----|----|----|----|----|

| 23 |
|----|

Each value, to the left of the element where the temporary value was originally stored, is compared in turn until:

♦ the value being compared is smaller than the stored temporary value

or

♦ the start of the list has been reached

When either of the previous bullets is true, the temporary value is copied back into the list.

| 23 | 45 | 99 | 7 | 8 | 64 | 37 | 63 | 34 |

| 23 |

## Iteration 2

When the next element (99) is examined, the element before it (45) is smaller, so no further action is required.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 23 | 45 | 99 | 7 | 8 | 64 | 37 | 63 | 34 |

temp

| 99 |

## Iteration 3

When the value in element 3 is compared to every element before it, the result is that the values in indexes 0, 1 and 2 are all copied one element to the right (as 7 is smaller than 23, 45 and 99).

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 23 | 45 | 99 | 7 | 8 | 64 | 37 | 63 | 34 |

temp

| 7 |

The temporary value is copied into element index 0.

| 7 | 23 | 45 | 99 | 8 | 64 | 37 | 63 | 34 |

| 7 |

## Iteration 4

When the value in element 4 is compared to the values in indexes 0 to 3, it is smaller than every value, except element 0 (7).

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 7 | 23 | 45 | 99 | 8 | 64 | 37 | 63 | 34 |

temp

| 8 |

The values 23, 45 and 99 all move right. The temporary value this time is copied into index 1.

| 7 | 8 | 23 | 45 | 99 | 64 | 37 | 63 | 34 |

| 8 |

By this stage, the algorithm of an insertion sort should be apparent, as follows:

♦ Each element from 1 to the length of the array is copied into temporary storage and dealt with in turn.

♦ Every larger value to the left is moved up one element.

♦ The temporary value is copied back into the list when the next value is smaller, or when the end of the array is reached.

**Pseudocode**

| Design | Commentary |
|---|---|
| `fixed loop i = 1 to length(list)-1` | Loop from the second element to the last |
| `   store the value at array index i` | Store the current temporary value |
| `   store the starting position of the inner loop` | Store the current position in the array — this will be used as a starting point to count backwards during the comparisons |
| `   while index > 0 and value < list[index-1]` | Continue comparing previous values in the list with the temporary value until the start of the array is reached or the two values are in the correct order |
| `      copy the value at index i into index i+1` | The compared value is copied into the element to the right |
| `      reduce the index by 1` | Decrement the element being compared next |
| `   end while` | |
| `   copy the stored value into index i` | The temporarily stored value is copied into the correct place |
| `end fixed loop` | |

**SQA reference language: insertion sort  implementation**

```
PROCEDURE insertion_sort(list)
     DECLARE value INITIALLY 0
     DECLARE index INITIALLY 0
     FOR i = 1 to length(list)-1 DO
          SET value TO list[i]
          SET index TO i
          WHILE (index > 0) AND (value < list[index-1]) DO
               SET list[index+1] TO list[index]
               SET index TO index - 1
          END WHILE
          SET list[index] TO value
     END FOR
END PROCEDURE
```

# Binary search

A binary search finds a value by continually halving a sorted list until a target is, or is not, found.

The code begins by designating a start (S) point and an end (E) point in the list. These are initially the first and last elements of the array.

From these, the target value positioned in the middle of the sorted list is identified ( $M=(E-S)/2$ ).

Target = 8

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| 2 | 5 | 8 | 10 | 11 | 14 | 17 | 25 | 30 | 37 | 38 | 39 | 50 | 51 | 60 | 65 | 77 |
| S | | | | | | | | M | | | | | | | | E |

The algorithm compares the target to the value stored at M and makes one of three decisions:

1. If the middle value is larger than the target, then the target must be in the half of the list that contains smaller values.
2. If the middle value is smaller, the target must be in the larger half of the list.
3. If the middle value is equal to the target, then the target has been found and the search ends.

If either bullet points 1 or 2 are true, then the start or end are reassigned as required. The middle point is then calculated for the remaining list and the same decision is made again.

Target = 8

| 2 | 5 | 8 | 10 | 11 | 14 | 17 | 25 | 30 | 37 | 38 | 39 | 50 | 51 | 60 | 65 | 77 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| S | | | | M | | | E | | | | | | | | | |

This is carried out again, until a match is found at M.

Target = 8

| 2 | 5 | 8 | 10 | 11 | 14 | 17 | 25 | 30 | 37 | 38 | 39 | 50 | 51 | 60 | 65 | 77 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| S | | M | E | | | | | | | | | | | | | |

## Pseudocode

Note: pseudocode is not a fixed design notation, and candidates may prefer to use more 'code-like' pseudocode when designing algorithms. An example of this approach is shown below.

| Design | Commentary |
|---|---|
| `low = 0` | The lowest index point (S) is stored |
| `high = length(list)-1` | The highest index point (E) is stored |
| `found = false` | Set a flag variable to show that a match has not yet been found |
| `while not found and low <= high` | Conditional loop until the target is found or there are no elements left to examine |
| `  set mid = (low + high) / 2` | Find the midpoint (M) as halfway between the lowest and highest index |
| `  if target = list[mid] then` | If a match with the target is found… |
| `    display "found"` | …display found to user… |
| `    set found to true` | ...and end the conditional loop using the flag variable |
| `  else if target > list[mid]`<br>`    set low = mid + 1`<br>`  else`<br>`    set high = mid − 1`<br>`  end if` | Reset the lowest or highest index depending on whether the target is greater or smaller than the value in the middle index |
| `end while` | |
| `If not found then`<br>`  display "not found"`<br>`end if` | An optional 'not found' may be added to the end of the algorithm, if required |

## SQA reference language: binary search implementation (procedure)

The procedure below displays the position of the target, if it is found within the passed list.

```
PROCEDURE binary_search(list,target)
      DECLARE low INITIALLY 0
      DECLARE high INITIALLY length(list)-1
      DECLARE mid INITIALLY 0
      DECLARE found INITIALLY FALSE

      WHILE NOT found AND low <= high
          SET mid TO (low+high)/2

          IF target = list[mid] THEN
              SEND "Found" TO DISPLAY
              SET found TO TRUE
          ELSE IF target > list[mid] THEN
              SET low TO mid+1
          ELSE
              SET high TO mid-1
          END IF
      END WHILE

      IF NOT found THEN
          SEND "Not found" TO DISPLAY
      END IF

END PROCEDURE
```

**SQA reference language: binary search implementation (function)**

The function below returns a Boolean value used to store whether the target value is found, or not, in the array. The main program can then use the returned value.

```
FUNCTION binary_search(list,target) RETURNS BOOLEAN

        DECLARE low INITIALLY 0
        DECLARE high INITIALLY length(list)-1
        DECLARE mid INITIALLY 0
        DECLARE found INITIALLY FALSE

        WHILE NOT found AND low <= high
            SET mid TO (low+high)/2

            IF target = list[mid] THEN

                SET found TO TRUE
            ELSE IF target > list[mid] THEN
                SET low TO mid+1
            ELSE
                SET high TO mid-1
            END IF
        END WHILE

        RETURN found
END FUNCTION

#main program
DECLARE numList AS ARRAY OF INTEGER INITIALLY [3,4,7,10,15,21,36]
RECEIVE find FROM KEYBOARD
foundIt = binary_search[numList, find]
IF foundIt THEN
        SEND "Target found" TO DISPLAY
ELSE
        SEND "Target not found" TO DISPLAY
```

# Appendix 10: SQL operations (DDD)

Candidates need to implement relational databases using SQL Data Definition Language (DDL) and Data Manipulation Language (DML) in the Advanced Higher course.

### DDL

♦ CREATE statement — used to create a database and the structure of each table in the database

♦ DROP statement — used to remove individual tables from a database or even the entire database

### DML

♦ INSERT statement — used to populate a table by adding records (this was introduced at National 5)

♦ UPDATE statement — used to edit values stored in database records (this was introduced at National 5 and extended at Higher)

♦ DELETE statement — used to remove records from a database table (this was introduced at National 5)

In addition, Advanced Higher candidates should be able to describe, exemplify and implement SQL SELECT statements that make use of:

♦ the HAVING clause

♦ logical operators IN, NOT, ANY, BETWEEN, EXISTS in the WHERE or HAVING clause

♦ a subquery in the WHERE clause

## SQL data types

When using the SQL CREATE statement, SQL data types must be used.

| Data type | Sample | SQL implementation | Comment |
|---|---|---|---|
| integer | 32, -846 | int | using a size parameter is optional; it is used to restrict the maximum display width |
| float | 3.14 | float(size, d) | the size parameter specifies the total number of digits displayed, while d specifies the number of digits after the decimal point |
| varchar | ABC123D | varchar(size) | the size parameter is mandatory, to restrict number of characters possible between 0 and 65535 |
| date | 2019-05-23 | date | format is YYYY-MM-DD |
| time | 09:12:47 | time | format is hh:mm:ss |

Information about each of these data types and examples of SQL statements are on the following pages.

# CREATE statement

A database is defined as being a structured set of data. The first step in building an SQL database is to create the database structure using `CREATE DATABASE`.

```
CREATE DATABASE databaseName;
```

Once a database has been created, the structure for each table in the database needs to be built using `CREATE TABLE`.

```
CREATE TABLE tableName (
      fieldName1 dataType,
      fieldName2 dataType,
      ......
);
```

### Validation constraints

The following can be specified for individual fields:

**PRIMARY KEY:** uniquely identifies each record in the table

```
fieldName dataType PRIMARY KEY
                                                                         or
PRIMARY KEY (fieldName1)
                                                                         or
PRIMARY KEY (fieldName1, fieldName2, ...)
```

**FOREIGN KEY:** links two tables together by referencing the primary key of another table

```
fieldName dataType FOREIGN KEY REFERENCES tableName (fieldName)
                                                                         or
FOREIGN KEY(fieldName) REFERENCES tableName (fieldName)
```

**NOT NULL:** ensures that a field always contains a value and is not left empty

```
fieldName dataType NOT NULL
```

**CHECK:** ensures that all values in a field satisfy a specific condition

```
fieldName dataType CHECK(fieldName condition)
```

**AUTO INCREMENT:** automatically generates a unique number when a new record is inserted

```
fieldName dataType AUTO_INCREMENT
```

## Additional notes on constraints

### PRIMARY and FOREIGN KEY constraints

♦ Some dialects of SQL allow the `PRIMARY` or `FOREIGN KEY` constraint to be applied in the clause used to identify the data type for the field; other dialects require the `PRIMARY` or `FOREIGN KEY` constraint to be applied in a separate clause.

♦ Users should refer to the relevant documentation or reference guide to check the syntax for the version of SQL they are using.

♦ If the primary or foreign key consists of multiple columns, users **must** specify them in a separate clause at the end of the `CREATE TABLE` statement.

### CHECK constraint

♦ Standard SQL provides the `CHECK` constraint, as described and exemplified in this appendix. However, the `CHECK` constraint is not provided in all dialects of SQL (for example, MS Access and MySQL do not support the use of `CHECK`).

♦ In the case of MySQL, the `CHECK` constraint is ignored and the intended data validation is not carried out. To implement the `CHECK` constraint in MySQL, triggers or views must be used.

Note: candidates should implement triggers or views within their project solution, as required; however, these constraints are not assessed in the Advanced Higher Computing Science course.

♦ Users should refer to the relevant documentation or reference guide to check the syntax for the version of SQL they are using.

### Applying multiple constraints

It is possible to apply several constraints to one field, for example:

```
fieldName dataType NOT NULL PRIMARY KEY
```

## DROP statement

The `DROP` statement is used to drop or delete a whole database. Be careful when using this statement, as all the tables and data stored in them are removed and cannot be restored. This statement is often exploited by cyber criminals in SQL injections.

The `DROP` statement can be used to permanently remove an entire database.

```
DROP DATABASE databaseName;
```

It can also be used to delete individual tables from a database. Used in this format, the statement results in the complete loss of all data stored in the named table.

```
DROP TABLE tableName;
```

**Note**: The `DROP` statement is not supported in MS Access.


## HAVING clause of a SELECT statement

The SQL `HAVING` clause is used in combination with the `GROUP BY` clause or an aggregate function, to restrict the returned rows to only those where the `HAVING` condition is true.

`HAVING` is used to filter records that work on summarised `GROUP BY` results. It was added to the SQL language because the `WHERE` clause cannot be used with aggregate functions. The `HAVING` clause is applied to grouped records, but `WHERE` is applied to individual records. Only groups that meet the `HAVING` criteria will be returned.

`HAVING` can also be used in combination with `WHERE` and `ORDER BY` clauses, for example:

♦ the `WHERE` clause is used to restrict the rows that are returned from the tables(s)
♦ the `ORDER BY` clause is used to sequence the rows in the answer table
♦ the `HAVING` clause is used to filter summarised and/or aggregated data or grouped data

Note: using `HAVING` requires a `GROUP BY` clause to be present.

```
SELECT list of field names
FROM list of table names
WHERE condition
GROUP BY list of field names
HAVING condition
ORDER BY list of field names;
```


## Logical operators

Logical operators are used, together with the comparison operators =, <, >, <=, >= and `LIKE,` in the `WHERE` clause of a `SELECT` query to form a condition that restricts the rows

returned from the tables. At National 5, logical operators `AND` and `OR` were introduced. At Advanced Higher, five specialist operators are introduced.

**NOT**  This returns a record from the underlying tables when the specified condition is **not** true.

```
SELECT list of field names
FROM list of table names
WHERE NOT condition;
```

**BETWEEN**  This selects values that fall within a specified range of (inclusive) values.

```
SELECT list of field names
FROM list of table names
WHERE fieldname BETWEEN value1 AND value2;
```

**IN**  This allows multiple values to be specified as an alternative to multiple OR conditions.

```
SELECT list of field names
FROM list of table names
WHERE fieldName IN (value1, value2, ......);
```

subquery
```
SELECT list of field names
FROM list of table names
WHERE fieldName IN (SELECT statement);
```

**ANY**  This returns true if any of the subquery values meet the condition specified in the main query.

subquery
```
SELECT list of field names
FROM list of table names
WHERE fieldName operator ANY (SELECT statement);
```

**EXISTS**  This tests for the existence of records within the subquery and returns true when the subquery returns one or more records (this is very useful to obtain records that do not meet a certain condition).

subquery
```
SELECT list of field names
FROM list of table names
WHERE EXISTS (SELECT statement);
```

subquery
```
SELECT list of field names
FROM list of table names
WHERE NOT EXISTS (SELECT statement);
```

### Additional notes on operators

**ANY operator**

The images below provide pictorial explanations of the SQL `ANY` operator.

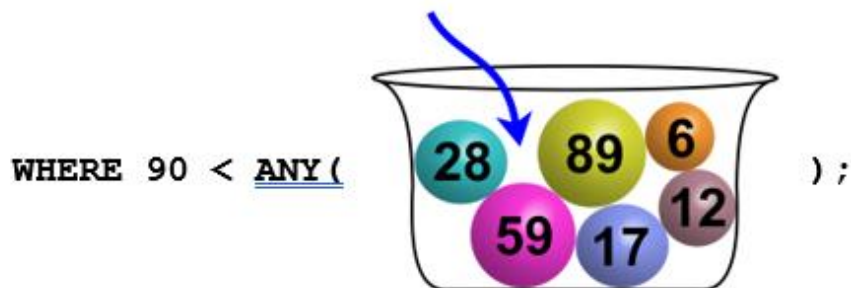**Query 1:** using the `ANY` operator, generates TRUE and so returns data to the main query.



>ANY means greater than at least one value, that is, greater than the minimum.

WHERE 90 > ANY ( 28 89 6 59 17 12 );
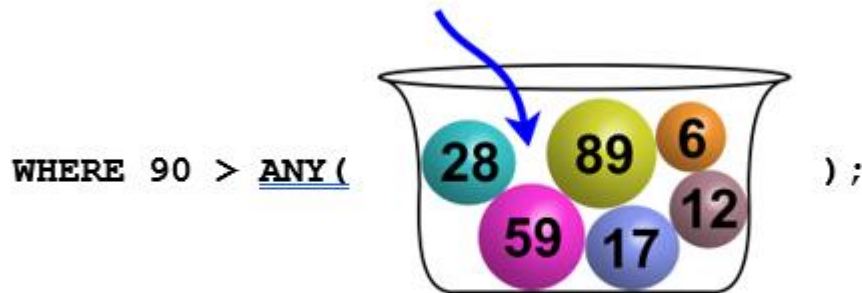
So >ANY(28,59,89,17,6,12) means greater than 6.
As 90>6 is true, data is returned.

**Query 2**: using the `ANY` operator, generates FALSE and so does not return data to the main query.



<ANY means less than at least one value, that is, less than the maximum.

WHERE 90 < ANY ( 28 89 6 59 17 12 );

So <ANY(28,59,89,17,6,12) means less than 89.
As 90<89 is false, no data is returned.

**Query 3**: using the `ANY` operator, generates TRUE and so returns data to the main query.

<ANY means less than at least one value, that is, less than the maximum.



```
WHERE  4  <  ANY (                    );
```
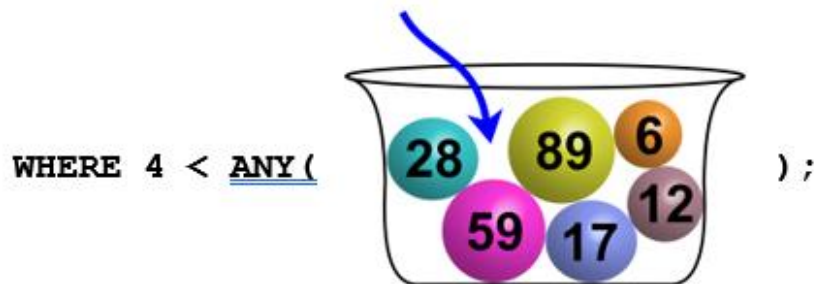
So <ANY(28,59,89,17,6,12) means less than 89.
As 4<89 is false, data is returned.


**EXISTS operator**

The images below provide pictorial explanations of the SQL EXISTS operator.

**Query 4:** general format of an SQL query that uses the EXISTS operator.

```
WHERE EXISTS (       subquery        );
```

```
EXISTS...
```
- is a comparison operator
- is used in the `WHERE` clause to validate an 'it exists' condition
- will tell whether a query returned results
- returns a Boolean, (TRUE or FALSE)
- returns TRUE if a subquery contains any rows


**Query 5:** using the EXISTS operator, returns TRUE.
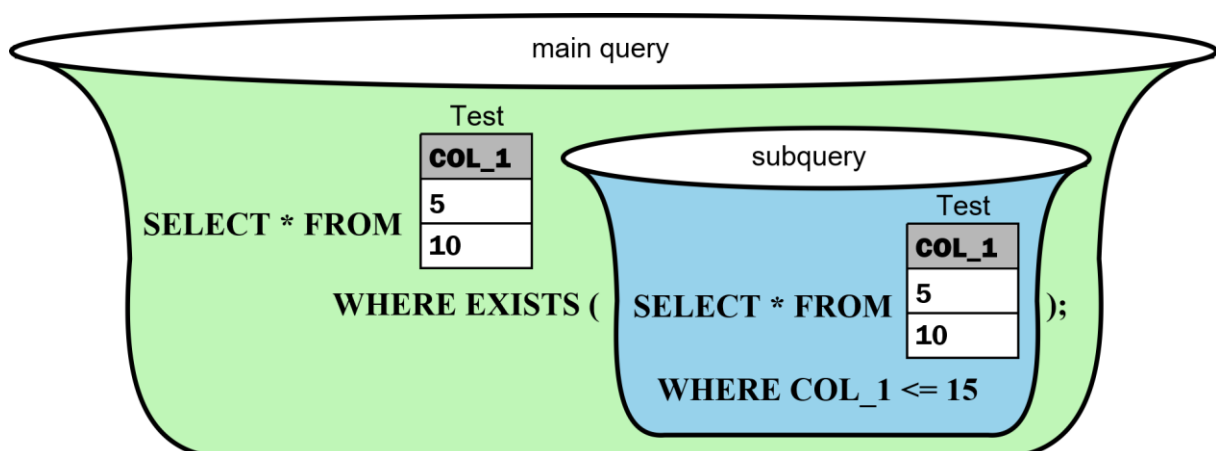


The subquery contains more than one row, so it returns TRUE. Data is therefore returned from the main query.

**Query 6: using** the EXISTS operator, returns FALSE.
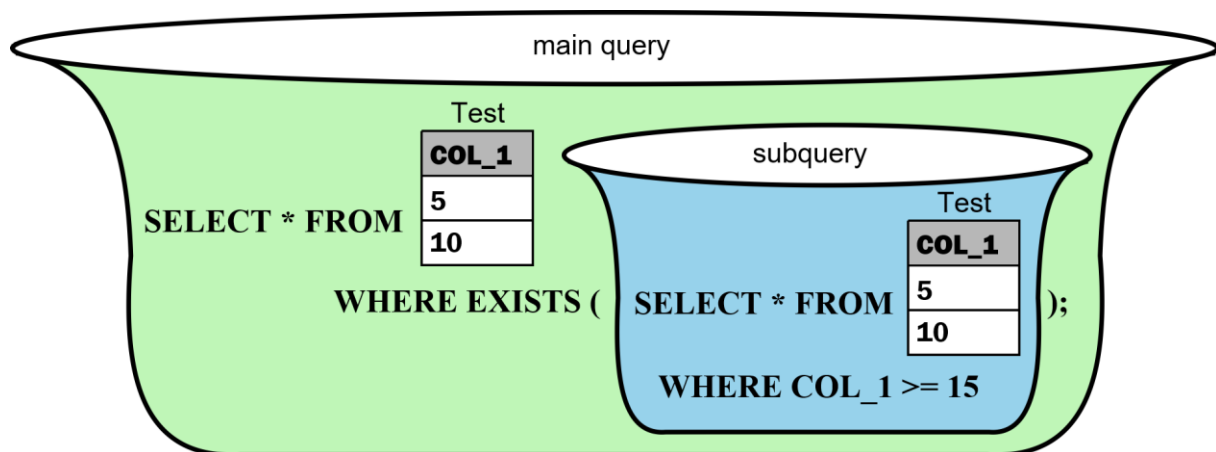


The subquery contains no rows, so it returns FALSE. No data is therefore returned from the main query.

## Subquery in the WHERE clause of a SELECT query

A subquery is a query embedded within the WHERE clause of another SQL query. A subquery is sometimes referred to as an inner query or a nested query, and an SQL query is sometimes referred to as the outer query or the parent query.

The subquery executes before the main query, so the results can be passed to the main query as a condition to further restrict the data to be retrieved.

There are a few rules that subqueries must follow:

♦ Subqueries must be enclosed within brackets.

♦ Unless the main query has multiple fields in its SELECT clause, a subquery can have only one field in its SELECT clause.

♦ The BETWEEN operator can be used within a subquery but cannot be applied to the results of a subquery returned to the main query.

♦ Although an ORDER BY clause can be used with the main query, an ORDER BY clause cannot be used in a subquery; if it is needed, the GROUP BY clause can be used to perform the same function as the ORDER BY within a subquery.

♦ Many subqueries return exactly one record (called single-value subqueries); the developer must check that this is the case, because an error will be generated if a subquery returns more results than expected.

♦ Subqueries that return more than one row (called multiple-value subqueries), can only be used with multiple-value operators such as EXISTS, IN and ANY.

```
SELECT list of field names
FROM list of table names
WHERE fieldName OPERATOR
     (SELECT list of field names
      FROM list of table names
      WHERE condition)
ORDER BY list of field names;
```

# Example queries: travel agency database

A travel agency uses a relational database to store details on a booking system.

It stores details of Scottish holiday resorts, hotels in each resort, customers and their bookings. These details are stored in four separate entities.

The attributes stored in each entity are shown below.

| Resort | Hotel | Customer | Booking |
|--------|-------|----------|---------|
| resortID<br>resortName<br>resortType | hotelRef<br>hotelName<br>resortID *<br>starRating<br>seasonStartDate<br>mealPlan<br>checkInTime<br>pricePersonNight | customerNo<br>firstname<br>surname<br>address<br>town<br>postcode | hotelRef *<br>customerNo *<br>startDate<br>numberOfNights<br>numberInParty |

## SQL CREATE statement

The SQL statements below can be used to build the structure of the travel agency database. The full data dictionary for this database is in appendix 4: data dictionary.

```
CREATE DATABASE TravelAgency;


CREATE TABLE Resort (
        resortID int NOT NULL PRIMARY KEY,
        resortName varchar(20) NOT NULL,
    resortType varchar(20) NOT NULL CHECK (resortType IN('coastal',
    'city', 'island', 'country'))
);


CREATE TABLE Hotel (
        hotelRef varchar(4) NOT NULL PRIMARY KEY,
        hotelName varchar(20) NOT NULL,
        resortID int NOT NULL,
    starRating int NOT NULL CHECK(starRating >=1 AND starRating <=
    5),
        seasonStartDate date,
    mealPlan varchar(17) NOT NULL CHECK(mealPlan IN('Room Only',
    'Bed and Breakfast', 'Half Board', 'Full Board')),
        checkInTime time NOT NULL,
    pricePersonNight float(6,2) NOT NULL CHECK(pricePersonNight >=50
    AND pricePersonNight <= 250),
        FOREIGN KEY (resortID) REFERENCES Resort(resortID)
);
```

```
CREATE TABLE Customer (
        customerNo int AUTO_INCREMENT PRIMARY KEY,
        firstname varchar(20) NOT NULL,
        surname varchar(20) NOT NULL,
        address varchar(40) NOT NULL,
        town varchar(20) NOT NULL,
        postcode varchar(8) NOT NULL
);


CREATE TABLE Booking (
        hotelRef varchar(4) NOT NULL,
        customerNo int NOT NULL,
        startDate date NOT NULL,
        numberNights int NOT NULL CHECK(numberNights >=1),
        numberInParty int NOT NULL CHECK(numberInParty >=1),
        PRIMARY KEY (customerNo, hotelRef, startDate),
        FOREIGN KEY (customerNo) REFERENCES Customer(customerNo),
        FOREIGN KEY (hotelRef) REFERENCES Hotel(hotelRef)
);
```

**The following example queries match the examples in appendix 5: query design.**

**Queries making use of the HAVING clause**

**Query 7**: display the resort name and number of hotels in any resort that has at least two hotels.

```
SELECT resortName, COUNT(*) AS [Number of Hotels]

FROM Resort, Hotel

WHERE Resort.resortID = Hotel.resortID

GROUP BY resortName

HAVING COUNT(*) >= 2;
```

**Query 8**: display the full name and the total cost of all bookings for each customer. The query should only list details of customers whose total cost exceeds £2000 and should list the details of the biggest spending customer first.

```
SELECT firstName, surname, SUM(pricePersonNight * numberNights *
numberInParty) AS [Total cost of all Bookings]

FROM Customer, Booking, Hotel

WHERE Customer.customerNo = Booking.customerNo

AND Booking.hotelRef = Hotel.HotelRef

GROUP BY firstName, surname

HAVING SUM(pricePersonNight * numberNights * numberInParty) >=
2000

ORDER BY SUM(pricePersonNight * numberNights * numberInParty)
DESC;
```

**Query 9**: display the average price per person, per night for each holiday resort. Display only those resorts with an average price per person, per night that exceeds £100.

```
SELECT resortName, ROUND(AVG(pricePersonNight),2) AS [Average
Price]

FROM Resort, Hotel

WHERE Resort.resortID = Hotel.resortID

GROUP BY resortName

HAVING AVG(pricePersonNight) > 100;
```

## Queries using logical operators

**Query 10**: display the name and type of non-coastal resort, together with the name and meal plan for each hotel that meets these criteria.

```
SELECT resortName, resortType, hotelName, mealPlan

FROM Resort, Hotel

WHERE Resort.resortID = Hotel.resortID

AND NOT resortType = "coastal";
```

**Query 11**: display the full name and total number of bookings made by each customer who has made between two and four bookings.

```
SELECT firstName, surname, COUNT(*) AS [Total Bookings]

FROM Customer, Booking

WHERE Customer.customerNo = Booking.customerNo

GROUP BY surname, firstName

HAVING COUNT(*) BETWEEN 2 AND 4;
```

**Query 12**: display the surname, postcode, and town of customers who live in towns that begin with the letters 'E' through to 'M'. The query should list customers in alphabetical order of town.

```
SELECT surname, postcode, town

FROM Customer

WHERE town BETWEEN "E" AND "M"

ORDER BY town;
```

**Query 13**: display the hotel name and meal plan for hotels that offer room only, half board or full board.

```
SELECT hotelName, mealPlan
FROM Hotel
WHERE mealPlan IN ("Room Only", "Half Board", "Full Board");
```

**Query 14**: display the name and type of resorts that are neither city nor country resorts.

```
SELECT resortName, resortType
FROM Resort
WHERE NOT resortType IN ("city", "country");
```

## Queries with a subquery in the WHERE clause

**Query 15:** display the hotel name, star rating, and price per person for the most expensive hotel.

```
SELECT hotelName, starRating, pricePersonNight
FROM Hotel
WHERE pricePersonNight =
      (SELECT MAX(pricePersonNight)FROM Hotel);
```

**Query 16**: display the resort name, hotel name, and star rating of all hotels that have a below-average star rating.

```
SELECT resortName, hotelName, starRating
FROM Resort, Hotel
WHERE Resort.resortID = Hotel.resortID
AND starRating <
      (SELECT AVG(starRating)FROM Hotel);
```

**Query 17**: display the full name and postcode of the customer who booked the same hotel as the customer with ID 111.

```
SELECT firstName, surname, postcode
FROM Customer, Booking
WHERE Customer.customerNo = Booking.customerNo
AND NOT Customer.customerNo = 111
AND hotelRef =
      (SELECT hotelRef FROM Booking
       WHERE customerNo = 111);
```

**Query 18**: display the name and star rating of all hotels booked by the customer with ID 315.

```
SELECT hotelName, starRating

FROM Hotel

WHERE hotelName IN

    (SELECT hotelName FROM Hotel, Booking

     WHERE Hotel.hotelRef = Booking.hotelRef

    AND customerNo = 315);
```

**Query 19**: display the names and types of resort not booked by the customer with ID 315.

```
SELECT resortName, resortType

FROM Resort

WHERE resortName NOT IN

    (SELECT resortName FROM Resort, Hotel, Booking

     WHERE Resort.resortID = Hotel.resortID

     AND Hotel.hotelRef = Booking.hotelRef

     AND customerNo = 315);
```

**Query 20**: display the customer number, hotel reference, and booking cost for any booking that costs more than any bookings made by customers with surnames Lowden, Shawfair or Sheriffhall.

```
SELECT customerNo, Hotel.hotelRef,
pricePersonNight*numberNights*numberInParty AS [Booking Cost]

FROM Booking, Hotel

WHERE Booking.hotelRef = Hotel.hotelRef

AND pricePersonNight*numberNights*numberInParty > ANY

(SELECT pricePersonNight*numberNights*numberInParty

    FROM Booking, Hotel, Customer

    WHERE Booking.hotelRef = Hotel.hotelRef

    AND Booking.customerNo = Customer.customerNo

    AND surname IN ("Danderhall", "Lowden", "Shawfair"));
```

**Query 21**: display the details (hotel name, star rating, meal plan, and resort name) of all 3-star hotel bookings. The query should list the hotels in alphabetical order of meal plan.

```
SELECT hotelname, mealPlan, starRating, resortName
FROM Hotel, Resort
WHERE Hotel.resortID = Resort.resortID
AND starRating = 3
AND EXISTS
     (SELECT * FROM Booking
      WHERE Booking.hotelRef = Hotel.hotelRef)
ORDER BY mealPlan ASC;
```

**Query 22**: display the full name and address of customers who have never made a booking.

```
SELECT firstName, surname, address
FROM Customer
WHERE NOT EXISTS
     (SELECT * FROM Booking
      WHERE Customer.customerNo = Booking.customerNo);
```

**Query 23**: display the name, star rating, and total of nights booked for hotels that have:

♦ a total number of customer nights booked that is more than the total number of nights booked by the customer with ID 290 (number of nights booked multiplied by number in party)

  and

♦ a star rating which is less than that of the hotel with the highest star rating

The query should list the hotels from lowest star rating to the highest.

```
SELECT hotelName, starRating, SUM(numberNights*numberInParty)
AS[Nights x Number in Party]
FROM Hotel, Booking
WHERE Hotel.hotelRef= Booking.HotelRef
AND numberNights*numberInParty >(
    SELECT SUM(numberNights*numberInParty) FROM Booking
    WHERE customerNo =290)
AND starRating < (SELECT MAX(starRating) FROM Hotel)
GROUP BY hotelName, starRating
ORDER BY starRating;
```

# Appendix 11: HTML forms (WDD)

## Continuation from Higher

The Higher Computing Science course defined the use of form elements and input types to include validated input for text, numeric, and restricted-choice entry (select and radio). There are no additional input methods or validation in the Advanced Higher course.

The focus of web content in the Advanced Higher course is the use of PHP to integrate with an SQL database. This includes server-side processing of HTML form code introduced at Higher.

## Form action and method attributes

To process the contents of an HTML form, an action and method must be initiated when the form's 'submit' button is clicked. These are coded as attributes of the <form> element.

```
<form action="registerStudent.php" method="POST">
```

The action shown above states the file "registerStudent.php" will be opened when the form is submitted. As this is a PHP file, the web server where it is stored will automatically execute the PHP code contained in the file.

You can submit a form using one of two HTTP methods:

♦ GET

♦ POST

The submission process for both methods begins the same way, with the browser constructing a form data set.

### GET

If you submit a form with `method="GET"`, the browser constructs a URL by taking the value of the action attribute, appending a ? to it, then appending the form data set. It then processes this URL as if following a link. The browser divides the URL into parts and recognises a host, then sends a GET request to that host, with the rest of the URL as an argument.

**Advantages of using GET:**

♦ If security is not an issue, the URL can be bookmarked, allowing it to be re-used without having to complete and submit the original form.

♦ If there is a network connection issue when a form is submitted, the browser will automatically resend the form, as it assumes it does not contain sensitive data.

♦ GET submissions can usually be cached. If the same submission is used regularly (for example form data used to generate the same database query), this could have a significant effect on efficiency.

**Disadvantages of using GET:**

♦ The form data in the constructed URL is visible and so less secure.

♦ URLs can only contain ASCII codes, which will cause issues if the form data contains non-ASCII characters.

♦ The URL constructed will be stored in the user's web browsing history, making it inappropriate for sensitive data.

♦ URLs have a limited number of characters, which limits the form data submitted.

**Recommended use**

GET is usually used when non-sensitive data (like the parameters of a database query) is sent to a server.

## POST

When you submit a form using the POST method, the form data set is encoded within a message that is sent to the server.

**Advantages of using POST:**

♦ The submitted form data is not visible and so more secure than GET.

♦ Non-ASCII characters can be submitted within the form data set.

♦ There is no URL character limit, so form data can be much larger.

**Disadvantages of using POST:**

♦ The submitted form cannot be bookmarked for later use.

♦ If there is a network issue while the form is being submitted, the browser will ask the user to resubmit the form.

**Recommended use**

POST is usually used when sensitive data (like personal information) is sent to a server. A database update would usually be initiated with the POST method.

POST can also be used for non-sensitive data: if the submitted data is likely to contain non-ASCII characters or the length is over the limit of a URL.

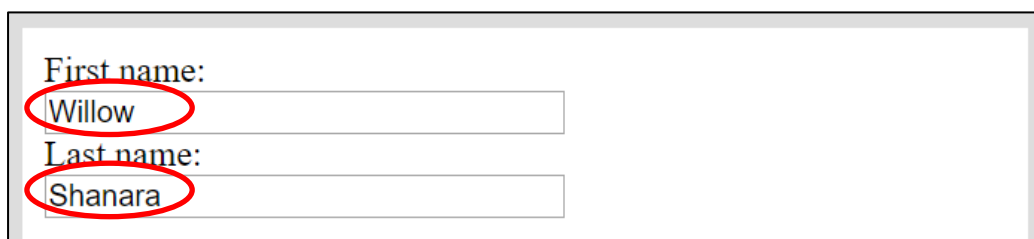# Name and value attributes for form element and input types

The form data set is comprised of key/value pairs, where key is a declared attribute of the form input called `name` and `value` is the data entered by the user.

In the case of text and numeric input, the `name` attribute is defined within the input type. These attributes are used when processing the form and must match the attributes used when the submitted form data is assigned to server-side variables.

```
First name:<br>
<input type="text" name="firstname" size="30" maxlength="15" required>
<br>

Last name:<br>
<input type="text" name="lastname" size="30" maxlength="15" required>
<br><br>
```

The `value` can be a number, a character or a string that the user types into the form's field or one that has been defined in the HTML form.

```
First name:
Willow
Last name:
Shanara
```

In the case of a drop-down menu or a radio button input, both the `name` and `value` are defined in the HTML code.

```
Select play:
<select name="play">
  <option value="hamlet">Hamlet</option>
  <option value="godo">Waiting for Godo</option>
  <option value="brothers">Blood Brothers</option>
  <option value="curious">Curious Incident</option>
</select>
<br><br>
```

```
Choose your age:<br>
<input type="radio" name="age" value="12 to 14" checked> 12-14
<input type="radio" name="age" value="15 or 16"> 15,16
<input type="radio" name="age" value="over 16"> over 16
```

# Appendix 12: PHP form processing (WDD)

The Advanced Higher course requires candidates to execute server-side code to:

♦ process HTML form data, using PHP
♦ store submitted form data within a database table, using SQL and PHP
♦ query a database, using SQL and PHP
♦ display the results of a query within HTML table elements, using PHP

## Database and web servers

To execute PHP files, you need a database server (connected to a database to store or retrieve data) and a web server. Although we usually think of a server as hardware, a web and/or database server setup is a collection of software technologies that may be:

♦ installed on and run from a local PC or hardware server
♦ installed on and run from a USB flash drive
♦ installed on and run from an external PC or hardware server across the World Wide Web

There are many ways to install the required software. These range from builds of individual components (which requires knowledge, expertise and time), to prebuilt, simple installations that require a single install such as XAMPP, WampServer or EasyPHP.

## Executing PHP files

You must have the following to execute .php files:

♦ web server software installed and running
♦ .php files saved to a specific folder within the installed server folders

The following examples demonstrate this for an XAMPP setup.

**XAMPP control panel showing both Apache (web server) and MySQL (database server) applications running**
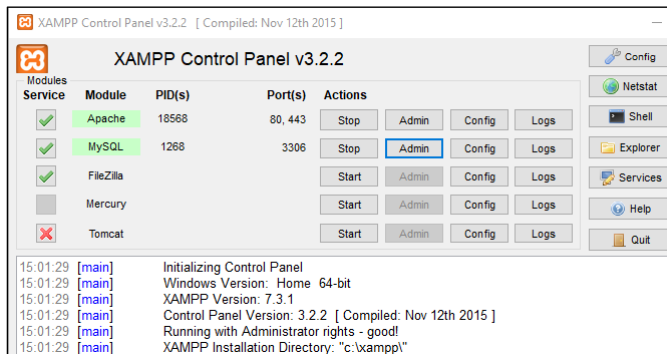


**XAMPP folder htdocs where .php files are located**
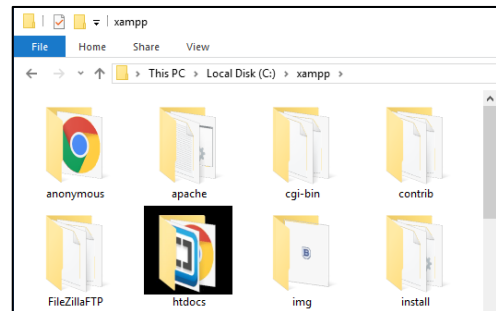


## What is a PHP file?

PHP files are text files that can contain HTML, CSS, JavaScript, and PHP code.

When a .php file is executed on a server, the PHP code it contains can:

♦ collect form data
♦ add, delete and modify data in your database
♦ generate dynamic page content

When a .php file is executed, the results are returned to the browser as a plain HTML file.

The following examples use code taken from the Advanced Higher example website. You can download the example website from SQA's secure site.

## HTML forms

The Drama page on the Advanced Higher example website contains the following form.

```
<form action="registerStudent.php" method="POST">

  First name:<br>
  <input type="text" name="firstname" size="30" maxlength="15" required>
  <br>

  Last name:<br>
  <input type="text" name="lastname" size="30" maxlength="15" required>
  <br><br>

  Select play:
  <select name="play">
    <option value="hamlet">Hamlet</option>
    <option value="godo">Waiting for Godo</option>
    <option value="brothers">Blood Brothers</option>
    <option value="curious">Curious Incident</option>
  </select>
  <br><br>

  Number of Tickets Required (between 1 and 3):
  <input type="number" name="tickets" value="1" min="1" max="3">
  <br><br>

  Choose your age:<br>
  <input type="radio" name="age" value="12 to 14" checked> 12-14
  <input type="radio" name="age" value="15 or 16"> 15,16
  <input type="radio" name="age" value="over 16"> over 16

  <br><br>

  If required, please delete and state any special requirements:<br>
  <textarea name="message" rows="6" cols="30">none</textarea>
  <br><br>

  <input type="submit" value="Register">
</form>
```

To process a form, the server requires the following:

♦ `action=""`

This contains the name of the file to be executed when the form is submitted. This can be the current file or a different file.

♦ `method=""`

The method used to submit the form can be GET or POST:

— GET — the submitted data is visible to the user and therefore not secure

— POST — the submitted data is hidden from the user (forms are almost always submitted using the POST method)

♦ `name=""` and `value=""`

When the data in the form is submitted to the server, it is converted into an array of key/value pairs (where key is the `name` of the form controls and `value` is the data entered by the user).

Note: with <select> and radio input, the values are defined in the form code.

# PHP form processing

When the form on the Drama web page is submitted, the following file is executed:

```
registerStudent.php
```

A .php file may contain HTML, CSS and JavaScript, so you must identify any PHP code by placing it inside a PHP script.

```php
<?php
    // PHP code goes here
?>
```

## Assign form data to server-side variables

The values in the array passed from the submitted form are assigned to separate variables in the lines below. Each of these lines uses the `$_POST[" "]` method to extract values from matching variables first declared in the form.

```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $forename = $_POST["firstname"];
    $surname = $_POST["lastname"];
    $play = $_POST["play"];
    $tickets = $_POST["tickets"];
    $age = $_POST["age"];
    $requirements = $_POST["message"];
}
```

Note: these assignments are placed within a conditional statement, which checks that the form has been submitted:

If a form is submitted using the $_GET method, POST would be changed to GET as shown below.

```php
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    $forename = $_GET["firstname"];
    $surname = $_GET["lastname"];
    $play = $_GET["play"];
    $tickets = $_GET["tickets"];
    $age = $_GET["age"];
    $requirements = $_GET["message"];
}
```

## Open and close connection to database

To connect to a database, you need to define the following parameters. You can enter these directly into the connection function or store them in variables, as shown below.

```
3        $servername = "localhost";
4        $username = "root";
5        $password = "";
6        $dbname = "studentList";
```

Line 3 defines the host name of the server. Line 4 defines the username used to connect to the server. "root" is a default value that is usually set with administration rights for the server.

For security purposes, server access is usually password protected (for example `$password="hsd56XC89"`). For teaching purposes, the password string can be left empty as shown in line 5. The name of the database the script will connect to is stored in line 6.

The function `mysqli_connect()` is used to connect to the server:

```
$conn = mysqli_connect($servername, $username, $password);
```

or to connect to a database stored on the server:

```
$conn = mysqli_connect($servername, $username, $password, $dbname);
```

This function returns a Boolean True if the connection is successful. You can use the Boolean value to:

♦ ensure the script only proceeds when a proper connection is made
♦ return an error message if the function returns false
♦ kill the script using `die()` if a connection is not made, as shown below

```
24       // Create connection
25       $conn = mysqli_connect($servername, $username, $password);
26
27       // Check connection
28 ▼     if (!$conn) {
29           die("Connection failed");
30       }
```

`$conn` stores a single instance of a connection.

Connections should be closed using the function `mysqli_close()` at the end of a script.

```
73       mysqli_close($conn);
74   ?>
```

## Executing an SQL query to insert submitted form data into a database table

After the submitted form data has been assigned to PHP variables and a connection to the database has been established, you can use SQL to add the form data to a database table.

The function `mysqli_query()` is used to execute an SQL statement, as shown below.

```
61      $sql = "INSERT INTO studentData (forename, surname, play, tickets, age, requirements)
62      VALUES ('$forename', '$surname', '$play', '$tickets', '$age', '$requirements')";
63
64 ▼    if (mysqli_query($conn, $sql)) {
65          echo '<script type="text/javascript">alert("Sucessfully Registered");</script>';
66 ▼    } else {
67          echo "Error inserting data";
68      }
```

The function requires two parameters:

♦ the connection ($conn) used to identify the connection to the database being used
♦ the SQL statement including the PHP variables, that now store the form data

Line 67 shows the use of echo to output a message. The echo statement is often used in PHP coding to output HTML code, which is then interpreted by the browser and displayed.

```
echo "<p>Hello world</p>";
```

Additional notes:

1   You can write the SQL statement directly into the function, but it is common practice to assign the SQL statement to a variable, which is then used in the function. This makes the code more readable.

2   For an SQL INSERT statement, the function returns a Boolean value (True = success, False = failed). This can then be used to return messages.

3   The example above uses a JavaScript alert to inform the user that their drama trip details have been successfully added to the database. This is not a requirement of the Advanced Higher course, but may be a useful tool to visually demonstrate the success of the `mysqli_query()` function, without using echo.

# Executing an SQL query and displaying formatted results using PHP

The staff page on the Advanced Higher example website includes two further examples of web and database integration.

## Check ticket purchases

This example uses a simple form to input the name of a play. The web page outputs a list of students, with the number of tickets each student has purchased.



When the 'Generate List' button is clicked, the page is reloaded, with the query output displayed in a table.

The results of an SQL SELECT statement returns output in the form of an array. The following code was used to display the returned data.

```php
62 ▼    if ($_SERVER["REQUEST_METHOD"] == "POST") {
63      $play = $_POST["play"];
64
65      $conn = mysqli_connect($servername, $username, $password, $dbname);
66      if (!$conn) {die("Connection failed");}
67
68      $sql = "SELECT forename, surname, tickets FROM studentData WHERE play = '$play'";
69      $result = mysqli_query($conn, $sql);
70
71      echo "<br><br>Students With Tickets ".mysqli_num_rows($result)."<br><br>";
72
73 ▼    if (mysqli_num_rows($result) > 0) {
74
75          echo "<table>";
76          echo "<tr><th style='width:100px;text-align:left'>Forename</th> <th
            style='width:100px;text-align:left'>Surname</th> <th style='width:50px;text-
            align:left'>Tickets</th></tr>";
77          // output data of each row
78 ▼        while($row = mysqli_fetch_array($result)) {
79              echo "<tr><td>".$row["forename"]."</td> <td>".$row["surname"]."</td>
                <td>".$row["tickets"]."</td></tr>";
80          }
81          echo "</table>";
82
83 ▼    } else {
84      echo "0 results";
85      }
```

When the 'Generate List' button is clicked, line 63 assigns the selected play to the PHP variable `$play`.

An SQL statement is used with `mysqli_query()` to query the database for students who have tickets for the selected play.

```
SELECT forename, surname, tickets FROM studentData WHERE play =
'$play'
```

The PHP function `mysqli_num_rows()` is used in line 71, to display the number of rows returned by the query — which is the number of students found.

Lines 75 to 81 use PHP to display an HTML table. This output is built in three stages:

♦ the static top part of the table

♦ the dynamic middle part of the table, where the number of rows displayed will depend on the query result

♦ the static bottom part of the table

A `while` loop on line 78 uses the function `mysqli_fetch_array()` to extract each row returned by the SQL select statement in turn. Each extracted row is stored as an associative array. The contents of the array are concatenated with the HTML table elements; this is required to create a single row of a three-column table.

```
<tr>    <td></td>   <td></td>   <td></td>    </tr>
```

Note: the first row of the table is displayed as a header row using <th> in place of <td>. Also, `mysqli_num_rows()` is used to ensure the table is only displayed when >0 rows are returned by the query (line 73). If zero rows are returned, `"0 results"` is displayed instead of the table.

## Check if places are available

In addition to the name of the play, this form also includes numerical input. The web page counts the total number of tickets purchased for the selected play and calculates the number of places remaining.

This is achieved using the code below.

```
72        $play = $_POST["play"];
73        $maxTickets = $_POST["maxTickets"];
```

```
81        // Create SQL statement returns the number of rows matching a play name.
82        $sql = "SELECT SUM(tickets) FROM studentData WHERE play = '$play'";
83
84        // The query is passed to the server and the result stored as a MySQL result object
85        $result = mysqli_query($conn, $sql);
86        $countArray = mysqli_fetch_array($result);
87        $placesLeft = $maxTickets - $countArray[0];
```

```
90        echo "<br><br><table style='margin-left:30px'>";
91        echo "<tr><td>Tickets Reserved</td></tr>";
92        echo "<tr><td>".$countArray[0]."</td></tr>";
93        echo "<tr><td>Places Still Available</td></tr>";
94        echo "<tr><td>".$placesLeft."</td></tr>";
95        echo "</table>";
```

Note: the first element of the array returned by `mysqli_fetch_array()` stores the numeric, aggregate result of the query.

`$countArray[0]`

## Building web pages generated by PHP code

You can generate the HTML returned to a browser by a .php file in the following ways:

♦   If the PHP script is contained within the same page as a submitted form, then the entire page will be reloaded when the GET or POST script is executed. Any output produced by the script will be included according to the position of the script within the HTML. This is the simplest solution if you wish to stay on the same web page when a form is submitted.

♦   If you want to generate a completely different page, then the form should load a different .php file. In this case, the PHP file will have to contain all the HTML elements required to build the new page.

### The PHP include function

This is not a requirement of the Advanced Higher course, but is an efficient way to build pages without repeating lots of code.

The Advanced Higher example website separates out the <header>, <nav> and <footer> elements of each page, storing them in separate HTML files.

| attendanceList.php | 20/03/2019 10:45 | PHP File |
| banner.html | 18/03/2019 10:21 | Chrome HTML Do… |
| bottomOfPage.html | 15/02/2019 11:30 | Chrome HTML Do… |
| countAttendance.php | 20/03/2019 11:31 | PHP File |
| drama.php | 18/03/2019 15:06 | PHP File |
| home.php | 18/03/2019 10:36 | PHP File |
| music.php | 18/03/2019 10:36 | PHP File |
| navigation.html | 20/02/2019 10:31 | Chrome HTML Do… |
| registerStudent.php | 20/03/2019 09:57 | PHP File |
| removeTrip.php | 18/03/2019 11:21 | PHP File |
| sport.php | 18/03/2019 10:36 | PHP File |
| staff.php | 18/03/2019 10:41 | PHP File |
| study.php | 18/03/2019 10:36 | PHP File |
| studyQuiz.php | 18/03/2019 10:25 | PHP File |

You can include these elements in each page using the PHP function `include`.

```
14   <body>
15
16   <?php
17   include 'banner.html';
18   include 'navigation.html';
19   ?>
```

In addition to substantially reducing the amount of code in each page, this also makes maintenance of these three elements easier, as their contents are stored in a single location and not repeated across every page of the website.

# Appendix 13: PHP sessions (WDD)

## Definition and use

When a browser loads a new web page, it forgets all the information from the previous page. A PHP session is a way of storing information within a website, so that it can be retained and used across multiple pages.

Sessions work in a similar way to a program. The website code opens (starts) the session. Information is generated, stored and sometimes changed. The website code then closes (destroys) the session to end it.

Examples of session use are:

♦ retaining selected items in a shopping cart, as the user navigates from page to page
♦ displaying a user's id on multiple pages, following a successful login
♦ retaining values, such as a user's quiz Score, when each new question page loads

## Starting a session

The following PHP function is used to start a session. This should be placed at the top of a page, before any HTML code. If data is being passed between multiple pages, each page that requires access to the session should contain the PHP code below.

```
<?php
// Start a new session
    session_start();
?>

<!DOCTYPE html>
<html>
<head>
```

When a new session starts, a user key is stored on the user's computer. The `session_start()` function looks to see if a user key exists. If it does, the current session is continued. If no user key exists, a new session is started.

## Session variables

Session variables are assigned values, as shown below.

```
$_SESSION['staffLogin']="False";
```

A PHP file that contains `session_start()` has access to any session variables previously created.

## Ending a session

The PHP function `session_destroy()` is used to end a session. On the Advanced Higher example website, clicking the 'Log Out' button calls the file `logout.php`. The code below destroys the session and then reloads the staff page.

```php
1  <?php
2  // Start the session
3      session_start();
4  ?>
5
6  <?php
7      if ($_SERVER["REQUEST_METHOD"] == "POST") {
8          session_destroy();
9          include 'staff.php';
10         die();
11         }
12 ?>
```

Note: this page must also include `session_start()`, as the current session must be continued before it can be destroyed.

## Worked example

The following examples use code taken from the Advanced Higher example website. You can download the example website from [SQA's secure site](#).

To view the staff page, a password is required. The original page content remains hidden until the correct password is entered. On the example website, the staff password is 'password' and has been implemented using session variables.

The page initially hides the content, and instead displays a simple form. The form calls the PHP file `login.php` when the 'staff password' button is clicked.

```
<div>
    <h2>Staff Page</h2>
    <p>You need to enter the staff password to view this area.</p>
    <form action="login.php" method="POST">
        <input class="signInGap" type="text" name="staffpass"
        value="">
        <input class="signButton" type="submit" value="Staff
        Password">
    </form>
</div>
```

When executed, the `login.php` file:

1  connects to the current session
2  compares the user's password (from the form) with the string `"password"`
3  sets the session variable `staffLogin` to `True`, if the user's input and the string match
4  uses the PHP function `include 'staff.php'` to display the staff page

```
1   <?php
2   // Start the session
3       session_start();
4   ?>
5
6   <?php
7       if ($_SERVER["REQUEST_METHOD"] == "POST") {
8           $staffpass = $_POST["staffpass"];
9           if (empty($staffpass)) {
10              include 'home.php';
11              die();
12          } else {
13              if ($staffpass == "password") {
14                  $_SESSION['staffLogin']="True";
15                  include 'staff.php';
16                  die();
17              }
18              else {
19                  include 'staff.php';
20                  die();
21              }
22          }
23      }
24  ?>
```

The above code contains alternative outcomes if the user's password field is empty or the password is incorrect.

Note: the `empty()` function used in this example is not a requirement of the Advanced Higher course.

This example could be extended to retrieve users' names and passwords stored within a database. PHP and SQL could be used to retrieve and then compare a stored password to the user's login attempt.

When the `staff.php` file is reloaded, the session variable `staffLogin` now stores `True`, indicating the user has successfully logged in. The staff page uses the value stored in the session variable to determine if the login form or the page content is displayed.

```
if ($_SESSION['staffLogin']=="True") {

        displays the page content

} elseif ($_SESSION['staffLogin']=="False") {

        displays the login form
}
```

The full code can be viewed using the Advanced Higher example website. You can download the example website from [SQA's secure site](#).

# Appendix 14: media queries (WDD)

## The function of media queries

The `@media` rule is used to define alternative CSS rules that are only implemented when certain defined expressions are true.
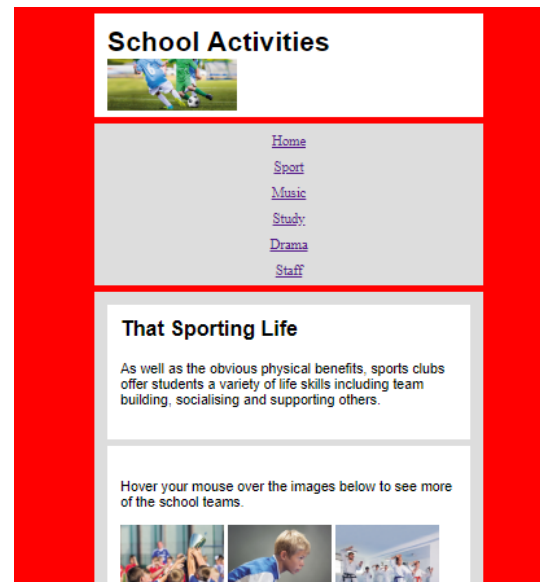
For example, alternative CSS rules could be declared if the width of the viewport (usually a browser window or screen) is less than a maximum of 600 pixels (px).

```
@media screen and (max-width:600px) {   }
```

**Screen width greater than or equal to 600px — original CSS applied**



**Screen width less than 600px — media query CSS applied**

## Media query syntax and code structure

A media query is formatted as:

```
@media not|only mediatype and (expressions) {
  CSS Code;
}
```

Three media types are in the Advanced Higher course: `all`, `screen` and `print`.

Only one media feature has been defined in the Advanced Higher course: max-width. This limits using media query expressions to checking the width of the viewport.

If candidates wish to explore media queries further, a complete list of media types and features are available using the following resource:

https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

Within the CSS, default values are written first, with media queries defined underneath.

When coding media queries, only the changes are styled. All the original styles are still applied to the page elements when the media query is triggered, so they do not need to be repeated within the `@media` rule.

The following example uses the Advanced Higher example website to demonstrate how a media query could be used within a candidate's project. You can download the example website from SQA's secure site.

Note: only the media query declaration itself is included in the Advanced Higher course.

```
body{margin:auto;background-color:LightBlue}

body{width:800px}
header {height:80px}
footer {height:60px}
nav {height:35px}

nav ul {list-style-type:none}
nav ul li {float:left;width:80px;text-align:center}
nav ul li a {display:block;padding:8px}
nav ul li a:hover {background-color:#000;color:White}
...
```



```
@media screen and (max-width:600px) {

    /*Alternative Body Styles     */
    body {background-color:red;width:300px}

    /* Alternative Navigation Styles */
    nav {height:125px}
    nav ul li {width:100%;height:20px;font-
    size:8pt}

    ...

}
```
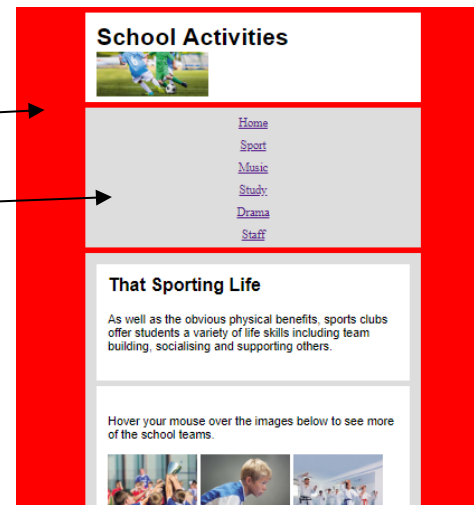


When the screen width is reduced below the maximum 600px, the above media query is triggered. Any declarations within the `@media` rule then become active, overriding the original declarations in the code.

When the screen width is greater than 600px, the trigger is no longer active and the original declarations once again become active.

# Media queries and design

Implementing interactive layouts should be based on multiple wireframe designs.

All Text
Font - Helvetica
Text Colour - Black

Body - width 800px

Page Background Colour - Light Blue

**School Activities - Sports Page (>= 600px )**   _ ◱ ✕

Main Heading
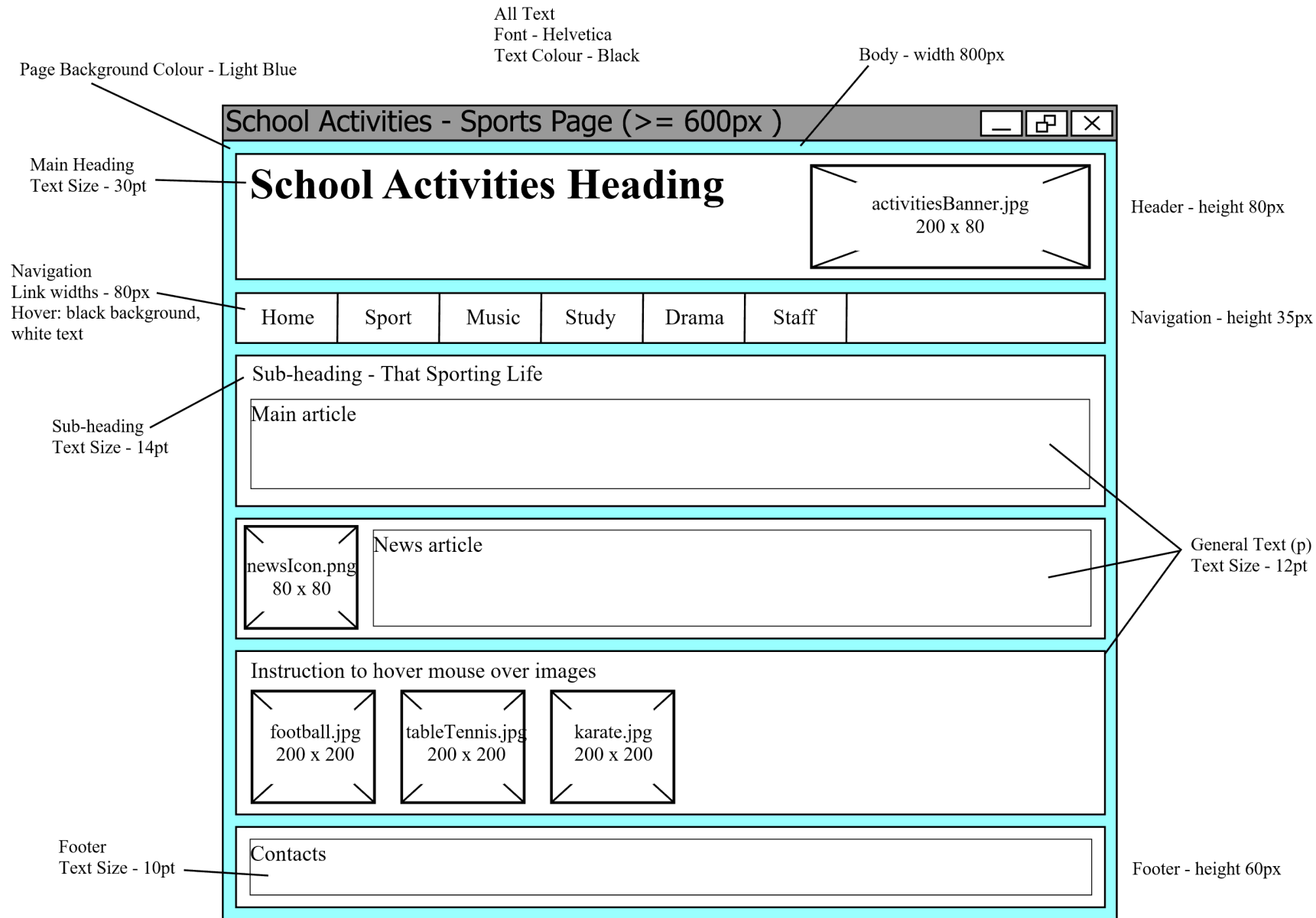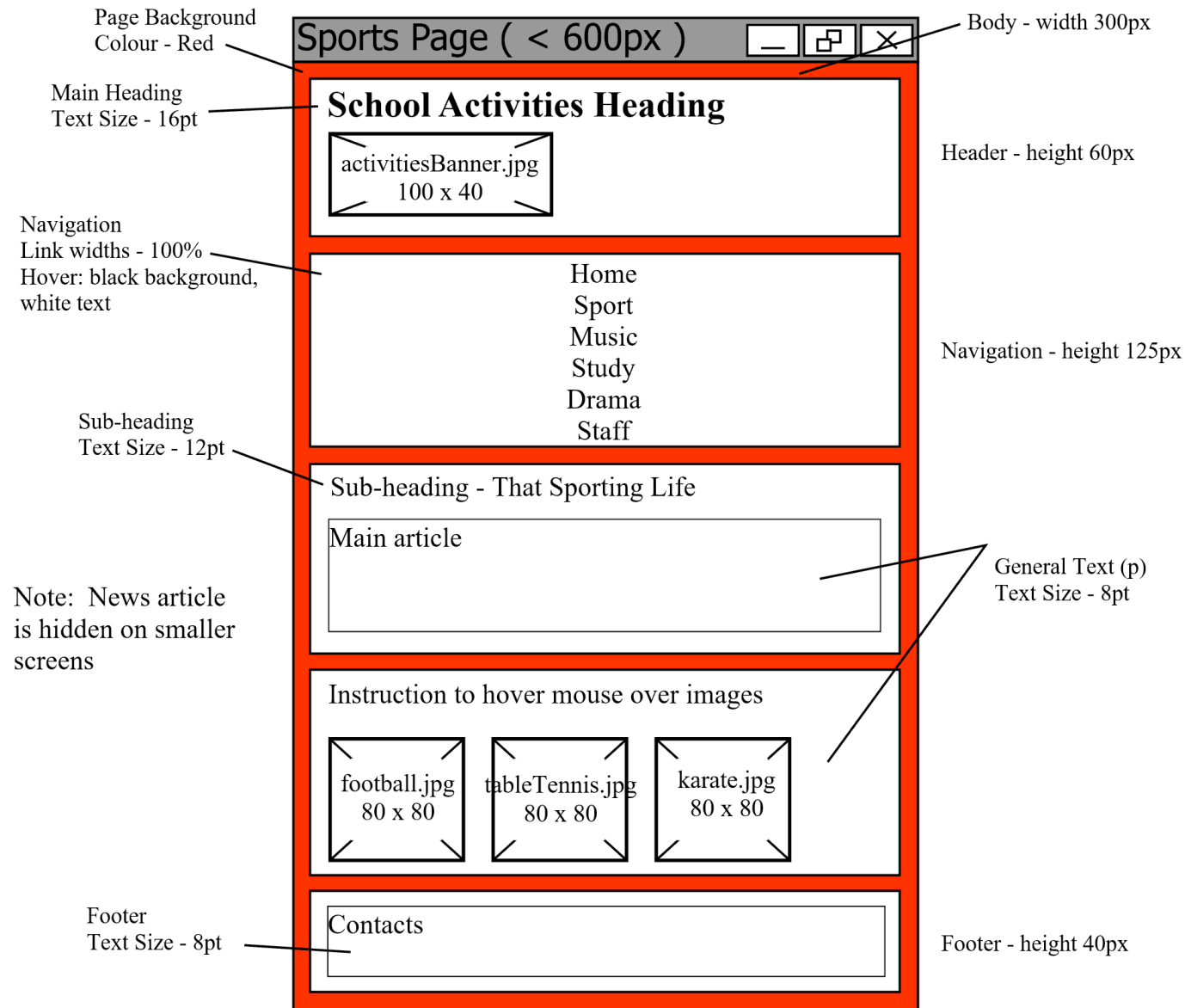Text Size - 30pt

## School Activities Heading

activitiesBanner.jpg
200 x 80

Header - height 80px

Navigation
Link widths - 80px
Hover: black background,
white text

| Home | Sport | Music | Study | Drama | Staff | |

Navigation - height 35px

Sub-heading - That Sporting Life

Main article

Sub-heading
Text Size - 14pt

newsIcon.png
80 x 80

News article

General Text (p)
Text Size - 12pt

Instruction to hover mouse over images

football.jpg
200 x 200

tableTennis.jpg
200 x 200

karate.jpg
200 x 200

Footer
Text Size - 10pt

Contacts

Footer - height 60px

Page Background
Colour - Red

Main Heading
Text Size - 16pt

Navigation
Link widths - 100%
Hover: black background,
white text

Sub-heading
Text Size - 12pt

Note: News article
is hidden on smaller
screens

Footer
Text Size - 8pt

Body - width 300px

Header - height 60px

Navigation - height 125px

General Text (p)
Text Size - 8pt

Footer - height 40px

**Sports Page ( < 600px )** ▭ ◲ ✕

## School Activities Heading

activitiesBanner.jpg
100 x 40

Home
Sport
Music
Study
Drama
Staff

Sub-heading - That Sporting Life

Main article

Instruction to hover mouse over images

football.jpg
80 x 80

tableTennis.jpg
80 x 80

karate.jpg
80 x 80

Contacts

# Media query examples

The following examples use code taken from the Advanced Higher example website. You can download the example website from SQA's secure site.

**Example 1 — general page structure**

The wireframes show that when the media query is triggered, some general changes are applied to the page styles.

```css
@media screen and (max-width:600px) {

/* Alternative Body Styles */
body {background-color:red;width:300px}

/* Alternative Navigation Styles */
nav {height:125px}
nav ul li {width:100%;height:20px;font-size:8pt}

/*Header with small title and small image banner below title */
header h1 {display:block}
.imageBanner {width:100px;height:40px;float:left}
header {height:60px}

/*Footer Contact List  */
footer ul li {width:70px;font-size:8pt}
footer {height:40px}

/*Test Sizes for smaller page*/
h1 {font-size:16pt}
h2 {font-size:12pt}
h3, p, ul {font-size:8pt}

/*News Articles are hidden on smalll screen*/
#newsArticle{display:none}
```

This changes the heights and widths of the general page structure.

A decision has been made to hide the news articles on smaller screen sizes, this is done by styling the id of the <section> that contains both the news icon and text as `display:none`.

**Example 2 — navigation bar**

The horizontal navigation is not appropriate for smaller screen sizes. When the media query is triggered, the CSS of the `<nav>` element is styled to create a vertical layout.

```css
/* Alternative Navigation Styles */
nav {height:125px}
nav ul li {width:100%;height:20px;font-size:8pt}
```

Note: only some of the `<nav>` styles need to be changed to achieve the different layout.

A height is added to each list item (`<li>`) to control the vertical spacing. This is not required in the default rules, as the height of the `<nav>` element limits the height of each link.

**Example 3a — alternative layouts**

The Drama web page was previously styled to position the `<section>` elements containing the `<form>` and the Drama Opportunities information side by side. Reducing the width of the screen automatically forces the form section to appear below the paragraph.

Some styling is still required to control the width of the Drama Opportunities section (which was previously wider than the new body width) and the margins of the form section.



```
/*Alternative Drama Layout*/
#dramaLeft{width:260px}
#dramaRight{margin-left:0px;margin-top:15px}
```

**Example 3b — alternative layouts**

The study page previously had three graphics with onclick JavaScript events that were used to reveal text. In the narrower layout, these were changed to simple text links.
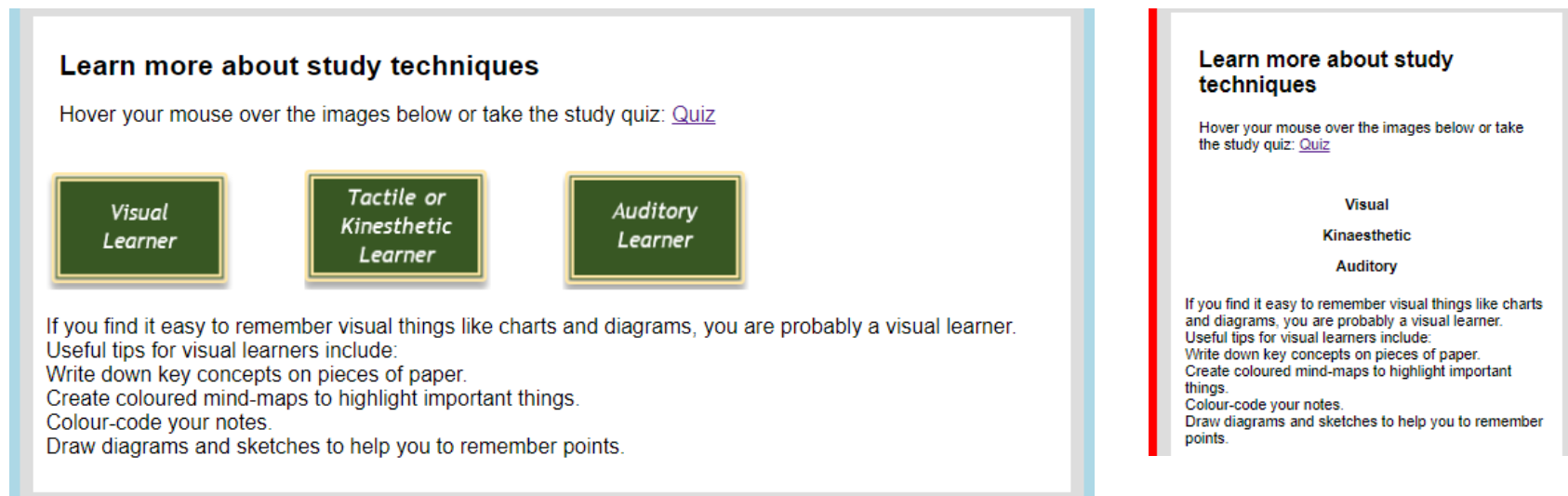


This was achieved by creating two matching `<section>` elements within the HTML file. One section includes three `<img>` elements (with JavaScript code) and the other includes three `<h3>` elements (with JavaScript code). The original styles and the media query styles alternately hide or show one of these two `<section>` elements.

**Original styles**

```
#studyWithImages{display:block}
#studyWithText{display:none}
```

**Media query styles**

```
#studyWithImages{display:none}
#studyWithText{display:block}
```

# Appendix 15: integrative testing (SDD, DDD and WDD)

Integrative testing is the second level of testing used in any development and takes place after component testing has been carried out. Component testing takes place during the development of the solution, when individual functions or modules are created.

Integrative testing is needed for projects that require integration of separate components. Testing is carried out to verify interaction between components and to detect interface defects. These tests determine whether independently developed units of software work correctly when connected to each other.

Due to the nature of integrative testing, some of the test cases needed require temporary code. This code can generate results not explicitly mentioned in the requirements specification. Once these tests are proved successful, any temporary code is removed.

In the examples that follow, test cases that require temporary code are marked with an asterisk*.

## Example 1: a project that combines SDD and DDD

A program is being developed to act as a personal diary app. The program will:

♦ allow the user to add new diary entries with a date, title, and description
♦ search diary entries by date
♦ list diary entries with the most recent entry first
♦ delete any diary entries that have expired
♦ allow a maximum of three images to be associated with each diary entry
♦ store details of all diary entries in a secure database server

This development meets the requirements of the Advanced Higher project, because:

♦ it is based on the SDD content of the course:
  — details of diary entries are stored and processed in an array of records
  — a sort algorithm is used to display diary entries with the most recent entry first
♦ it integrates with the DDD content of the course:
  — details of diary entries are stored in a database table
  — a connection with the database server is used to execute SQL queries
  — SQL queries are used to add and delete diary entries
  — an SQL query is used to search for diary entries using date entered by the user
  — an SQL query is used to select all diary entries, to enable processing to take place

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates SDD and DDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
|---|---|---|---|
| 1 | Check communication from the program to the secure database server* | Use program code to connect to the secure database server | Message is displayed confirming a successful connection with the database server |
| 2 | Check that all diary entries are selected from the database and stored in the array of records* | Use program code to execute the SQL query, store the query results in the array of records, and then display the contents | Details of all diary entries in the database are stored in the array of records and are displayed successfully |
| 3 | Check the details of a new diary entry have been added to the database | Enter details of a new diary entry, use program code to execute the SQL query to store the details in a new record of the database table | A new record has been added to the database table to store the new diary entry (check contents of the database table) |

## Example 2: a project that combines SDD and WDD

An object-oriented program is being developed to act as a recipe manager. The program will:

♦ use a recipe class to define the data types and methods associated with a recipe
♦ store recipe details in an array of objects for processing
♦ allow the user to add new recipes
♦ save recipe details to a sequential file
♦ allow the user to search for recipes by ingredient or category (starter, main course or dessert)
♦ display recipe details in alphabetical order of recipe title

This development meets the requirements of the Advanced Higher project, because:

♦ it is based on the SDD content of the course:
— a recipe class is defined
— an array of objects is used to store and process recipe details
— the linear search algorithm is used to search the recipe details
— a sort algorithm is used to arrange the search results, in alphabetical order of recipe title
♦ it integrates with the WDD content of the course. A web page is used to:
— present the user with output
— allow the user to enter details of a new recipe and indicate search criteria
— display the search results

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates SDD and WDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
|---|---|---|---|
| 1 | Check communication between the program code and the web page* | Use a HTML textbox to enter the recipe title, then use Java code to display the title entered | Message is displayed on the web page showing the recipe title entered |
| 2 | Check that all recipe details are stored in the array of objects* | Use Java code to import recipe details, store in the array of objects, and then display the array contents | Details of all recipes in the array of objects are displayed correctly on the web page |
| 3 | Check the recipes are displayed in alphabetical order of title | Use Java code to sort contents of the array of objects in alphabetical order of recipe title, and display the results | Recipe details are displayed on the web page in alphabetical order of recipe title |

## Example 3: a project that combines DDD and SDD

A movie review database is being developed. The database will:

♦ store details of movies, actors, reviews, and reviewers in five linked tables of a relational database
♦ allow users to search for details of individual movies
♦ allow users to add details of new movies to the database
♦ allow users to add a review and rating for any movie
♦ use forms to create an interface for all SQL functionality
♦ use subqueries to display details of the highest rated movie(s) and details of any movie that has at least five reviews
♦ use a query to display details of any movie that was not made in the UK

This development meets the requirements of the Advanced Higher project, because:

♦ it is based on the DDD content of the course:
  — details of movies, actors, reviews, and reviewers are stored in five linked tables of a relational database
  — subqueries are used to extract details from the database
  — queries make use of logical operators to search for required details
  — queries and subqueries make use of at least three database tables
♦ it integrates with the SDD content of the course:
  — forms created using toolbox controls provided within the integrated development environment provide an interface for all SQL functionality
  — the program forms a connection with the secure database server to execute SQL queries
  — the program is used to format and display query results

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates DDD and SDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
|---|---|---|---|
| 1 | Check communication between the program form used to enter details of a new review and the secure database server* | Use program code to connect to the secure database server | Message is displayed confirming a successful connection with the database server |
| 2 | Check that the query selected by the user has executed correctly* | Use program code to generate the query required, execute the query and display a query confirmation message | Message is displayed to confirm a successful execution of the query |
| 3 | Check the details of the highest rated movie(s) have been displayed correctly | Use program code to execute the SQL query required and then display the results | Details of the highest rated movie(s) are formatted and displayed |

# Example 4: a project that combines DDD and WDD

A music albums database is being developed. The database will:

♦ store details of albums, artists, and tracks in five linked tables of a relational database
♦ allow the user to search for details of individual albums, artists, and tracks
♦ allow details of new albums to be added to the database and stored in the relevant tables
♦ use web pages to create an interface for all SQL functionality
♦ use subqueries to display details of the most popular albums, artists, and tracks
♦ use a subquery to display details of the tracks in any album, that has at least ten tracks

This development meets the requirements of the Advanced Higher project, because:

♦ it is based on the DDD content of the course:
  — details of albums, artists, and tracks are stored in five linked tables of a relational database
  — subqueries are used to extract required album, artist and track details
  — queries and subqueries use at least three database tables
♦ it integrates with the WDD content of the course:
  — web pages provide an interface to display results of SQL queries
  — online forms are used to enter query search criteria
  — online forms are used to gather details of new albums
  — PHP code is used to form a connection with the secure database server and to execute SQL queries
  — PHP is used to format and display the results returned by SQL queries

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates DDD and WDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
|---|---|---|---|
| 1 | Check communication between the online form used to enter details and the secure database server* | Enter the title of a new album, then use PHP code to connect to the database server | Message is displayed confirming a successful connection with the database server |
| 2 | Check that the query selected by user has been formed correctly and has executed successfully* | Search for details of all albums by the band Genesis, use PHP code to generate the SQL query required. Use an 'echo' statement to display the syntax of the query formed, then execute the query and display the query confirmation message | The 'echo' statement is used to display the correct SQL query and a message is displayed on the web page confirming successful execution of the query |
| 3 | Check the details of the most popular artist are displayed correctly | Use PHP code to generate the SQL query required, to execute the query and display the results | Details of the most popular artist are formatted and displayed on the web page |

## Example 5: a project that combines WDD and SDD

A website is being developed to allow the user to play a game of 'Connect Counters'.

This is a 2-player game played on a 5 x 5 grid. Users take it in turns to either position a coloured counter in the grid to form a continuous sequence of counters (horizontally, vertically or diagonally), or block their opponent's sequence. Once the grid is full, the player who has the longest sequence of counters gets a point (in the event of a draw, both players receive points).

The software will:

♦ allow each player's name to be entered at the start of the game, together with the number of rounds being played (the maximum number of rounds is three)
♦ allow players to take it in turn to indicate the position of their coloured counter in the grid
♦ control the game play and award points
♦ display the name of the winner(s) and points awarded at the end of each round
♦ display the name of the overall winner, once all rounds of the game have been played

This development meets the requirements of the Advanced Higher project, because:

- it is based on the WDD content of the course:
  — an online form is used to gather and submit details at the start of the game
  — players use an online form of submit buttons to indicate the grid position they want to use on the 'Game Play' page
  — PHP is used to assign variables and process the form data
  — session variables are used to store details entered by the players for the duration of the game
  — external CSS is used to format the layout of all web pages in the website
  — a media query is used to create multiple layouts
- it integrates with the SDD content of the course:
  — a 2-D array is used to represent the position of the players' counters

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates WDD and SDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
| --- | --- | --- | --- |
| 1 | Check that the number of rounds entered by the user is passed to the game code successfully* | Enter the number of rounds = 2, assign to a PHP session variable and use this to control a fixed loop displaying the round number being played | Messages 'Round 1 being played' 'Round 2 being played' are displayed successfully on the 'Game Play' page of the website |
| 2 | Check that the grid position selected by player 1 is updated correctly in the 2-D array* | Once the game starts, the first grid position selected by player 1 is (2,4), a value of 1 should be assigned to position (1,3) of the 2-D array and the full contents of the array displayed | Contents of 2-D array are displayed correctly on the 'Game Play' page of the website<br>0 0 0 0 0<br>0 0 0 1 0<br>0 0 0 0 0<br>0 0 0 0 0<br>0 0 0 0 0 |
| 3 | Check that players' names entered at the start are displayed correctly at the end of each round | Enter players' names and number of rounds = 2 at the start of the game, play the game for two rounds, and player details displayed at the end of each round | At the end of rounds 1 and 2, a message is displayed showing the correct player names and both scores on the 'Game Play' page of the website |

## Example 6: a project that combines WDD and DDD

A photo gallery website is being developed. The website will:

♦ allow all users to view thumbnails of all publicly available images
♦ allow new users to create an account for the website
♦ allow registered users to login and view thumbnails of images that they have stored and have marked as 'private'
♦ allow all users to click a thumbnail image and display a full-sized image
♦ allow registered users to add details of new images to the database, indicating whether access to the images is 'public' or 'private'
♦ use web pages to create an interface for all SQL functionality

This development meets the requirements of the Advanced Higher project, because:

♦ it is based on the WDD content of the course:
  — users login to the website using an online form
  — PHP is used to assign variables and process the form data
  — session variables are used to store a user's login details while they are logged in to the website
  — external CSS is used to format the layout of all web pages in the website
  — media query is used to create multiple layouts
♦ it integrates with the DDD content of the course:
  — details of uploaded images are stored in a database table
  — a separate database table is used to store users' login details
  — a connection with the database server is used to execute SQL queries
  — an SQL query is used to check a user's login credentials
  — SQL queries are used to select the images to be displayed

On-going testing is used throughout the development to test each component as it is created. Integrative testing is needed, as the development integrates WDD and DDD content. The following three examples describe integrative tests for this development.

| Test case ID | Test case objective | Test case description | Expected result |
|---|---|---|---|
| 1 | Check communication between the online form used to login to the website and the secure database server* | Login to the website with any username and password, then use PHP code to connect to the database server | Message is displayed confirming successful connection with the database server |
| 2 | Check that the user is successfully logged into the website and their login details have been passed to a new page* | Login to the website with a stored username and password, execute the SQL query to check the login details, assign details to the session variables. Display a personalised confirmation message on a separate page | Personalised message is displayed on the 'User Gallery' page of the website, confirming successful login to the website |
| 3 | Check that only thumbnails of images marked 'public' are displayed on the 'Public Gallery' page of the website | Click the link to load the 'Public Gallery' page, execute the SQL query to identify the images marked 'public', then only display thumbnails of these images | Only thumbnails of the images marked 'public' are displayed on the 'Public Gallery' page of the website. (compare with the details stored in the Photos database table) |

# Appendix 16: fitness for purpose (SDD, DDD and WDD)

## Functional requirements and fitness for purpose

During the analysis stage of the development cycle, candidates identify the functional requirements when creating a requirements specification. The functional requirements are the inputs, processes, and outputs that **must** be included in the design and implementation of any solution to a problem.

A solution is fit for purpose if (following design, implementation and testing) it meets **all** the functional requirements. In the evaluation stage of the Advanced Higher project, candidates discuss if their solution is fit for purpose.

The following examples use functional requirements identified in appendix 1. Both examples assume that a program, website, and database are designed, implemented, and tested.

## Example of an evaluation of a solution that is fit for purpose (SDD)

### Functional requirements (from appendix 1)

The functional requirements are defined in terms of the inputs, processes, and outputs listed below.

All inputs are imported from a sequential file and all outputs are displayed on the screen. The program is activated by double clicking on the file icon and then selecting 'Run' from the menu. Each process should be a separate procedure or function that is 'called' from the main program.

Inputs:

♦ itemID
♦ price
♦ number in stock

Processes:

♦ read in data from external file to a 2-D array
♦ sort the data in order of itemID, from low to high
♦ search a 2-D array, based on end-user input, for the required itemID

Output:

♦ If a match is found, the data (itemID, price, number in stock) corresponds to the end-user input.
♦ If no match is found, a suitable message informs the end-user.

**Fitness for purpose**

Following comprehensive testing, the program is fit for purpose.

The solution:

♦ reads data from an external stock file, splits the data and allocates it to a 2-D array
♦ sorts the data in numerical order using the itemID
♦ allows the user to display a stock item by selecting an itemID
♦ searches the data in the 2-D array for the itemID selected and returns the result
♦ displays formatted output, showing the itemID, price, and number in stock for the selected items
♦ displays a message 'sorry your item has not been found' if the stock item is not found in the 2-D array

# Example of an evaluation of a solution that is NOT fit for purpose (DDD)

### Functional requirements (from appendix 1)

The functional requirements are defined in terms of the inputs, processes, and outputs detailed below.

Inputs (customer):

♦ register: user email, password, password re-entered, firstName, lastName, address, postcode, email
♦ search details: category
♦ search details: itemName
♦ sort details: Field (price or rating) and order required (ascending or descending)

Input (administrator):

♦ edit item details: itemID, price
♦ edit customer details: customerID, address, postcode, email
♦ add item details: itemID, itemName, description, category, price
♦ delete item details: itemID
♦ delete customer details: customerID
♦ monthly orders: month

Processes:

♦ auto generate the customerID when a new customer registers
♦ queries to insert records into the Customer and Item tables
♦ queries to sort the item details in order of price and rating
♦ queries to delete specific customer and item records from the database
♦ queries to edit records in the Customer and Item tables
♦ queries to search Item table
♦ queries to display details of all orders placed in a particular month

Output:

♦ confirmation of successful insertions
♦ confirmation of successful deletions
♦ confirmation of successful edits
♦ answer tables showing details of sorted items (sorts)
♦ answer tables showing details of required items (searches)

**Fitness for purpose**

Following comprehensive testing, the database-driven website and its user interface are not fit for purpose.

Although the solution successfully implements all insertions, deletions, and edits of the back-end database table data, it does not:

♦ provide confirmation of these actions
♦ allow the customer to sort the results of a stock item search

The solution does successfully:

♦ store the required information for each new customer (including an auto-generated customerID) when they register
♦ allow a customer's address, postcode, and email to be edited and deleted by an administrator
♦ store the required information for each new stock item
♦ allow stock item details to be edited and deleted by an administrator
♦ display stock items (descriptions, categories and prices) following a customer search for stock items by either name or category
♦ display all the orders for a month selected by an administrator

# Copyright acknowledgements

Appendix 10: all images copyright SQA

Appendix 13 and 14:   all images Shutterstock:
School activities banner — 553188940

Appendix 14:          Footballers lifting trophy — 370092275
Table tennis — 87611998
Karate — 740133061
Drama opportunities — 329907647

# Administrative information

**Published:**    September 2024 (version 3.2)

## History of changes

| Version | Description of change | Date |
|---|---|---|
| 2.0 | Course support notes added as appendix — it includes the 'Resources to support the Advanced Higher Computing Science course' appendices.<br><br>Diagram in appendix 1 altered to move actors outside system process box.<br><br>Amended 'entries' to 'entities' in the 'Skills, knowledge and understanding for course assessment' section. | August 2019 |
| 3.0 | Amendments to the 'Course overview' section, 'Course content' section, 'Course Assessment' section and 'Course support notes'. This provides information on the structure of the question paper, the scope for integration in each optional section, and reflects changes to the marks and duration.<br><br>Appendix 3: amendments to exemplification of mandatory/optional relationships<br><br>Appendices 12 and 13: deletions of additional HTML/PHP functions.<br><br>Appendices 17, 18 and 19: deleted. | May 2023 |
| 3.1 | Appendix 3: text changed to remove reference to 'dotted line', to match diagrams. | September 2023 |
| 3.2 | Additional guidance on time and volume for project component<br><br>Appendix 3: amended to provide greater clarity on strong/weak entities.<br><br>Appendix 9: amended to ensure consistency across pseudocode design and SQA Reference Language examples of standard algorithms. | September 2024 |

Note: please check SQA's website to ensure you are using the most up-to-date version of this document.