Course report 2026

Advanced Higher Computing Science, SCQF level 7

We've written this report for teachers and lecturers. We recommend you read this with the published course assessment documents and marking instructions.

The report:

- provides a summary of how each course component performed
- highlights areas of the question paper and coursework where candidates performed well and areas where they did not perform well
- has advice you might find useful when preparing candidates for assessment

We collect and prepare the statistics in course reports before we complete the appeals process.

Copyright © Qualifications Scotland 2026

Statistical and grade boundary information

Course entries

Number of resulted entries in 2025: 767

Number of resulted entries in 2026: 775

Distribution of course awards (including minimum mark to achieve each grade)

Course award	Number of candidates	Percentage	Cumulative percentage	Minimum mark required
A	207	26.7	26.7	94
B	198	25.5	52.3	80
C	175	22.6	74.8	67
D	116	15.0	89.8	53
No award	79	10.2	100	Not applicable

We've not applied rounding to these statistics.

In this report:

- 'Most' means greater than or equal to 70%.
- 'Many' means 50% to 69%.
- 'Some' means 25% to 49%.
- 'Few' or 'a few' means less than 25%.

You can find statistical reports on our website.

General commentary on grade boundaries

Our main aim when setting grade boundaries is to be fair to candidates across all subjects and levels, and to maintain comparable standards across the years, even as arrangements evolve and change.

For most National Courses, we aim to set examinations and other external assessments and create marking instructions that allow:

- a competent candidate to score a minimum of 50% of the available marks (the notional grade C boundary)
- a well-prepared, very competent candidate to score at least 70% of the available marks (the notional grade A boundary)

It's very challenging to get the standard on target every year, in every subject, at every level. Therefore, we hold a grade boundary meeting for each course to bring together all the information available (statistical and qualitative) and make final decisions on grade boundaries based on this information. Senior Qualifications Scotland staff normally chair these meetings.

Principal assessors use their subject expertise to evaluate the performance of the assessment and propose suitable grade boundaries based on the full range of evidence. We can adjust the grade boundaries as a result of the discussion at these meetings. This allows the pass rate to be unaffected in circumstances where there is evidence that the question paper or other assessment has been more, or less, difficult than usual.

- We can adjust the grade boundaries downwards if there's evidence that the question paper or other assessment has been more difficult than usual.
- We can adjust the grade boundaries upwards if there's evidence that the question paper or other assessment has been less difficult than usual.
- If levels of difficulty are comparable to previous years, we maintain similar grade boundaries.

Every year, we evaluate the performance of our assessments in a fair way, while ensuring standards are maintained so that our qualifications remain credible. To do

this, we measure evidence of candidates' knowledge and skills against the national standard.

For full details of the approach, please refer to the Awarding and Grading for National Courses Policy.

How the course assessment performed

Question paper

The question paper performed as expected. Marker feedback indicated that it was fair and accessible for candidates. Most candidates understood what was required and completed the mandatory section and their chosen optional section in the allocated time.

The C-level questions at the start of each section helped candidates focus on the specialist content before progressing to the more demanding problem-solving questions. 60.6% of candidates attempted questions in section 2, ‘Database design and development’, while 39.4% of candidates attempted questions in section 3, ‘Web design and development’. Overall attainment was broadly similar across both options, which was also reflected in similar performance on the questions common to both sections.

Project

The introduction of a mandatory template to assist candidates in September 2025 resulted in a small improvement in results. Template prompts helped clarify the required evidence at each stage and reminded candidates to track requirements introduced during analysis through each stage of project development.

Despite the template’s clear statement that candidates should have no more than six end-user requirements, with four mandatory, three integrative and a maximum of eight additional functional requirements, many candidates did not follow those restrictions:

- some candidates presented additional requirements that required several separate processes or additional queries
- some candidates added to the pre-printed mandatory requirements and, in a few cases, candidates deleted the mandatory requirements and replaced them with additional requirements of their own

- many candidates did not match their end-user requirements to any of their functional requirements, thereby introducing several undeclared additional functional requirements into their solutions

As a result, many candidates submitted an excessive volume of evidence that went well beyond the scope and standard of the Advanced Higher Computing Science project.

How candidates performed

Project

Analysis

Many candidates performed well in aspects of the analysis section of the project. Candidates accurately described constraints on the development, and many focused their end-user requirements more clearly on the tasks that users would carry out using the system, rather than on aspects of the user interface. This resulted in more specific requirements that matched those identified in the UML use case diagram.

However, most candidates continued to undertake extremely complex projects that generated an excessive number of requirements and greatly exceeded the scope and standard required for the project.

In addition, many candidates' project plans were generic, with the tasks for each stage simply repeating the headings used in the marking instructions. In some cases, the time scales indicated in those plans were unrealistic and did not include the ongoing testing required during implementation.

The additional functional requirements required candidates to include the intention to validate keyboard input. Despite a clear reminder of this in each template, some candidates did not include this requirement, while many candidates referred instead to user authentication for checking credentials rather than to validating the planned input values. As a result of these omissions at the analysis stage, candidates could not gain marks for input validation at subsequent stages of the project's development.

Design

For procedural software design and development projects, the mandatory requirement to create a modular solution with parameter passing resulted in more candidates gaining marks for presenting a top-level design that showed the intended parameter passing. Similarly, for web design and development projects, clear prompts in the template led more candidates to present a navigation structure

showing the planned passing of values between webpages and gain marks for the design evidence presented.

Most candidates who attempted software design and development projects continued to present reverse-engineered code that they had edited extensively to present as the design for their Advanced Higher concepts, integration and functional requirements.

A procedural programming solution requires a top-level design that indicates the intended data flow between modules. In addition, the design of the Advanced Higher data structure should indicate the planned data types, structure and dimensions that the candidate intends to implement. In many cases, candidates either omitted this evidence or provided code from their final solution. Code presented as design does not gain marks.

Similarly, an object-oriented programming solution requires a class-level design in the form of a UML class diagram that indicates the properties and methods, including a constructor, for any class that the candidate intends to implement from scratch.

This year, more candidates presented evidence that they intended to create their own classes rather than rely on classes generated automatically by frameworks. However, although many candidates had indicated the use of programmer-defined classes, they omitted method-level designs or a main program or class algorithm showing how the planned methods would be used by the main program or class to interact with the instance variables. To complete the design of an object-oriented solution, candidates must design each non-trivial method that they intend to implement from scratch. They must also design the algorithm for the main program or class to indicate object creation, initialisation, sequencing of method calls, looping, branching and any user interactions.

Many candidates who undertook a project focused on database design and development did not arrange the required attributes correctly within their solution. As a result, relationships between tables were sometimes overly complex and, in other cases, incorrect. These fundamental errors at design caused problems during implementation and, in many cases, made the proposed design impossible to implement.

For most candidates whose projects focused on web design and development, the planned use of session variables in the solution remained unclear. Many candidates indicated in their pseudocode that they intended to set session variables, but did not explain their purpose or planned use. In their designs, candidates must clearly indicate how they will use the session data on at least one subsequent page.

When creating their user interface designs, although the template prompted candidates to include error messages resulting from input validation in their output screens, many candidates did not do this, even when their pseudocode for the input validation design included those error messages.

Implementation

Most candidates achieved high marks for implementing the Advanced Higher concepts and the user interface, with many candidates also achieving high marks for integration. In addition, moving the table used to record ongoing testing evidence to the start of the implementation section of the template meant that many candidates considered the testing required as they coded individual functions and added them to their solution. As a result, markers allocated more marks for this aspect of the project than in previous years.

All candidates are required to provide evidence of the initial contents of implemented tables, together with the data used to populate the tables before any queries have been executed. This evidence allows markers to check that candidates executed queries successfully; without it, candidates cannot access the full range of the marks for this work at the implementation stage. In addition, some candidates submitted incomplete evidence of the structure of the implemented tables and therefore did not demonstrate that their implemented table structure matched the intended structure shown in the earlier data dictionary.

Although most candidates who implemented an Advanced Higher search or sort algorithm submitted evidence to show how the results of the algorithm performed, many did not provide the ‘before’ evidence of the data searched or the unsorted data. Markers must have this evidence to check that the implemented algorithm worked correctly; without it, candidates cannot access the full range of the marks for this work at the implementation stage.

Testing

Most candidates achieved high marks for completing all tests listed in their test plan and providing the planned screenshot evidence. In addition, restructuring this section of the template prompted candidates to link the persona and planned test cases to the end-user requirements identified during analysis. This resulted in more realistic persona descriptions, which many candidates then adopted when carrying out the identified test cases.

Candidates must produce a comprehensive test plan before carrying out final testing of their solution. Although many candidates included all mandatory, integrative and additional requirements identified at the analysis stage in their test plans, many plans lacked detail and were not comprehensive.

Evaluation

Most candidates who referred to each of the requirements identified at the analysis stage gained 1 mark for accurately describing the fitness for purpose of their solution. Many candidates also provided an evaluative discussion of the results of final testing.

In their evaluation of the maintainability of their solution, many candidates focused at length on readability but did not explain how this would support future maintenance or refer to maintenance activities appropriate to Advanced Higher level. A few candidates referred to appropriate maintenance activities and suggested reasons why future adaptive, corrective and perfective maintenance may be needed, but did not explain how features of their own solution would support future developers.

Question paper

Areas where candidates performed well

Software design and development

Question 1(b): Most candidates correctly described the comparisons made during the first pass of the bubble sort algorithm.

Question 2(a): Most candidates explained why technical feasibility must be considered before an app is developed, with many also explaining the need for economic feasibility.

Question 2(b): Many candidates accurately described how the test cases would be used in final testing.

Question 3(a)(i): Many candidates described the effect of the code provided.

Question 3(a)(ii): Most candidates correctly wrote the output produced by the code.

Question 3(b)(i): Many candidates accurately wrote the code for the `countInteractions()` method.

Question 3(c): Many candidates correctly stated the content missing from the cells of the table by applying the insertion sort algorithm the data.

Question 4(a)(i): Most candidates accurately described one benefit of user surveys for the developer.

Question 4(a)(ii): Most candidates accurately described one benefit of user surveys for potential users of the game.

Database design and development

Question 5(a): Most candidates correctly identified two errors in the UML case diagram provided.

Question 5(b)(i): Most candidates accurately explained why ‘Performer’ would be a strong entity.

Question 5(b)(ii): Most candidates accurately explained why ‘Concert’ would be a weak entity.

Question 5(b)(iii): Many candidates accurately described one benefit of a surrogate key.

Question 5(c): Many candidates correctly used `BETWEEN` in the search criteria and `HAVING COUNT(*) >5` to complete the query design.

Question 6(f): Many candidates correctly identified that corrective maintenance would be needed and provided good justification for their choice.

Web design and development

Question 7(a): Most candidates correctly identified two errors in the UML case diagram provided.

Question 7(b): Many candidates wrote an accurate media query to alter how the webpage would be displayed on a mobile device.

Question 7(c): Many candidates provided the missing details for validation of the email, with some candidates also providing the details needed to complete the underlying process for the 'Forgotten password' link.

Question 7(d): Many candidates accurately described how session variables store the email address used to display the personalised message shown in the screenshot.

Question 8(a): Many candidates provided the correct version of the `mysqli_connect()` function by correctly sequencing the connection details.

Question 8(e): Many candidates correctly identified that perfective maintenance would be needed and provided good justification for their choice.

Areas where candidates did not perform well

Software design and development

Question 1(a): Although some candidates described one difference between an array and a linked list, many candidates stated a fact about only one data structure and did not compare it with the other.

Question 3(b)(ii): Although some candidates explained that the `posts` array stores objects of type `Post` and `VideoPost` because the array was a superclass of type `Post`, many of these candidates did not explain that the posts array was unable to access the `addView()` method because it was only accessible to objects of type `VideoPost`.

Question 3(d): Some candidates indicated the correct list of fields and data types needed to complete the UML class diagram for the `Creator` class, but many candidates did not indicate that the references to posts should represent an array of `Post` objects. Only a few candidates indicated that the `Creator` class would require a constructor method.

Question 4(b): Although some candidates correctly declared a 2D array of type `Square` with dimensions 7 x 9, many candidates used an incorrect data type.

Question 4(c)(i): Some candidates correctly assigned the pointers to the correct element of the array declared in question 4(b), but many candidates gave inconsistent responses because they changed the `gameBoard` data structure from the one declared in question 4(b).

Question 4(c)(ii): A few candidates created an algorithm to correctly assign position numbers to the array structure declared in question 4(b). Some candidates managed to correctly assign position numbers to either odd or even rows of the gameboard, but not both.

Question 4(d)(i): Although many candidates gained 1 or more marks, inconsistent use of the data structure declared in question 4(b) resulted in many candidates not achieving full marks for this question. Most marks were awarded for correctly indicating the value of `newPosition` or for correctly searching the `gameBoard` array and matching it with the colour of the current player.

Question 4(d)(ii): Many candidates gained 1 mark for correctly checking for a win, but did not calculate the required bounce back correctly.

Question 4(d)(iii): Many candidates gained 1 mark for indicating the correct comparison needed to check for a ladder or a snake, but only a few candidates correctly updated `newPosition` by assigning the stored `pointer` value needed to gain the second mark.

Question 4(e): Although many candidates gained 1 mark for a partial response that correctly identified the need for integrative testing or explained its purpose, only some candidates provided a full explanation that merited full marks.

Database design and development

Question 6(a): A few candidates explained the impact of the data types and sizes indicated in the data dictionary by referring to relevant implications for implementing the `Member` table and testing the values stored in its fields.

Question 6(b): Some candidates gained 1 mark for correctly using `classID NOT IN` in the `WHERE` clause of their SQL query, with a few candidates gaining a second mark for correctly using a subquery with `classID` in its `SELECT` clause.

Question 6(c): Although many candidates provided an accurate description of how either the `ANY` or `EXISTS` operator processed the subquery results, many candidates did not describe the results returned by both subqueries or explain how both operators processed those results to generate the required output.

Question 6(e)(i): Some candidates correctly explained why the use of `GET` would be appropriate when submitting search criteria to a website.

Question 6(e)(ii): Although some candidates described one additional type of data validation for the category name entry, many candidates

incorrectly suggested adding a presence check despite the code already indicating that category was required.

Web design and development

Question 8(b)(i): Although many candidates gained 1 mark for correctly using the `mysqli_query()` function, few candidates indicated that the `die()` function would terminate query execution and inform the user that the query failed.

Question 8(b)(ii): Although some candidates gained 1 mark for correctly displaying the appointment time and dog name, most candidates did not correctly process appointment times where no appointment had been made.

Question 8(c): Many candidates gained 1 mark for either indicating the correct use of the HTML `SELECT` element with its `name` and `value` attributes or for indicating the correct use of the `$ratings` array to populate the options, but only a few candidates managed to achieve full marks.

Question 8(d)(i): Many candidates gained 1 mark for correctly indicating all three output messages generated by their algorithms. In addition, some candidates gained a further mark for correctly indicating that the algorithms should process only rows of the `$allpets` array for pets that had not yet been banned. However, most candidates did not indicate that update queries were required to change the values stored in the database.

Question 8(d)(ii): Many candidates gained 1 mark for a correct `WHERE` clause in their SQL query, with a few candidates correctly using `COUNT(*)` in the `SELECT` clause.

Preparing candidates for assessment

Question paper

You should continue to ensure that candidates have opportunities to apply searching and sorting algorithms to different Advanced Higher data structures, including 2D arrays, arrays of objects and arrays of records. Candidates should have opportunities to practise designing and implementing sort algorithms that arrange data in both ascending and descending order.

Ensure that candidates have opportunities to generate UML class diagrams for unfamiliar tasks. At Advanced Higher level, candidates are not expected to indicate the type of relationships between linked classes, but they must correctly list the properties, data types and methods, including a constructor, for any given class. In addition, candidates must indicate the visibility of each field and method as either public (+) or private (-).

You should encourage candidates to pay close attention to the wording of each question. When asked to describe a difference between two items, candidates must provide a correct fact about the first item and a contrasting fact about the second.

Overall, candidates performed well in problem solving on question 4, which required them to design algorithms for unfamiliar tasks and processes. Continue to encourage candidates to attempt these more challenging questions, as statistical evidence shows that most candidates gained partial marks for correct sections of their designs, even when the designs were incomplete.

Encourage candidates to read all parts of larger questions to ensure that they do not overlook any questions that require no problem solving. Questions such as 4(e), at the end of question 4, focus on knowledge and understanding of analysis, testing or evaluation within the Advanced Higher course and are often less demanding than questions requiring candidates to design or code an unfamiliar process.

You should provide candidates with opportunities to design algorithms to process data stored in 2D array structures, including 2D arrays of records. This helps reinforce the need to use nested loops to process 2D array data: one loop for each

row of the array and another for each column within that row, followed by the required processing.

When preparing candidates to attempt questions in section 2, ‘Database design and development’, ensure that they have opportunities to explain how the SQL engine processes complex queries with subqueries, especially the queries that use `NOT, IN, ANY`, and `EXISTS` operators. Understanding the step-by-step evaluation of the outer query, the subquery and the way an operator processes the subquery results will help candidates understand why, and how, the query returns the data it does.

Each operator has a distinct mental model:

- with the `IN` operator, the SQL engine checks whether the value from the outer row appears in the data set returned by the subquery
- with the `ANY` operator, the SQL engine compares the outer value with each value returned by the subquery and returns TRUE if any comparison meets the condition
- with the `EXISTS` operator, the SQL engine checks whether the subquery returns at least one row and does not consider the values within those rows
- NOT tells the SQL engine to keep rows where the condition would normally fail, so it reverses the result of a condition and can be used with other operators, for example `NOT IN`, `NOT ANY` and `NOT EXISTS`, when comparing the outer query with the results returned by the subquery

In preparing candidates to attempt questions in section 3, ‘Web design and development’, ensure candidates have opportunities to design unfamiliar server-side processes that can be implemented using the PHP functions `die()`, `mysqli_connect()`, `mysqli_close()`, `mysqli_query()`, `mysqli_fetch_array()` and `mysqli_num_row()`.

Project

You must use the correct version of the coursework assessment. Version 4.0 was issued in September 2026, as well as four updated templates. These revised templates are mandatory for all project submissions from session 2026–27 and provide details of the evidence candidates must submit to demonstrate the skills

assessed at each stage of the project. In addition, each template clearly indicates the scope that applies to the development, and candidates must understand the implications of that scope. All stated mandatory functional requirements must feature in the solution that is developed, and candidates should not introduce any additional functionality. The requirements stated in the scope ensure that the project assesses the skills, knowledge and understanding of the Advanced Higher Computing Science course and generates the evidence required for a successful Advanced Higher project submission.

When selecting projects, you should advise candidates to be realistic about what they can achieve working independently within the constraints and scope of the Advanced Higher course. We have provided examples of possible projects for each type, which you can use to illustrate the scope of the development. You must help candidates understand that the main purpose of the Advanced Higher project is to assess the skills, knowledge and understanding associated with each stage of the software development process in the Advanced Higher course.

Since all Advanced Higher projects require validation of all inputs to the final solution, you should advise candidates that they are expected to apply the input validation skills gained at National 5 level by applying input validation routines to two separate input values used to form a parameterised query. Candidates should indicate the intended input validation on the wireframes that show the planned user interface design, together with any anticipated error messages. You should ensure that candidates understand the difference between user authentication and input validation.

When preparing their project evidence for submission, candidates must use one of the four mandatory templates to organise and present their evidence. To help markers track requirements at each stage of development, each template includes page numbering in the document footer, which must not be removed before submission.

Markers must be able to read the code that is presented as part of the implementation evidence. When candidates provide screenshots of code, they should make them large enough to read and, where possible, avoid using a black

background with coloured text. Where candidates use screenshots of the user interface as evidence of testing, markers should be able to read the input values and output messages without magnifying the page.

Analysis

Encourage candidates to use a UML use case diagram early in the analysis stage to help identify end-user goals. The end-user goals identified should provide candidates with a high-level understanding of how end-users will interact with the system and the tasks that end-users should be able to perform. Candidates should include these tasks as end-user requirements in the requirements specification and match each one to one or more mandatory functional requirements.

Design

Encourage candidates to use structured English statements to generate the design of any processes, functions or methods that form part of their solution. At the design stage, candidates are required to describe the logic of each process and function without using programming language keywords, syntax or structures. Importantly, candidates should produce each process design before coding the process; designs created by reverse-engineering the finished code will gain no marks.

Candidates who complete a ‘Database design and development’ project must be able to correctly arrange the attributes and entities needed in their solution. Ensure that these candidates have the data modelling skills to identify the entities, attributes and relationships needed to support the functionality they intend to implement.

The data modelling process requires candidates to:

- identify the entities needed — the real-world ‘things’ that the system needs to store data about
- list the attributes required for each entity, ensuring that attributes do not appear in more than one entity
- create a first-draft entity-relationship diagram to identify the cardinality of relationships and any many-to-many relationships

- introduce an intersection or link entity to resolve each many-to-many relationship; this entity should contain two foreign keys referencing the original entities, creating a compound key
- produce a detailed entity-relationship diagram showing the name and cardinality of each relationship, weak and strong entities, and whether each relationship is mandatory or optional

Implementation

Encourage candidates to provide evidence demonstrating that the standard algorithm in their solution works correctly. For example, a project that uses the bubble sort algorithm must include evidence of the data held in the Advanced Higher data structure before the sort, evidence of the same data after the sort, the sort activation code (for example, evidence of the function call with parameter passing within the main control flow or, in an object-oriented solution, evidence of the sort method trigger within the main program or class flow), together with the full code for the sort algorithm.

Remind candidates that they must submit evidence to show that the structure of the implemented tables matches the structure indicated in the data dictionary produced at the design stage. This evidence can include SQL `CREATE` statements or screenshots of the implemented tables showing the required structural details.

In addition, candidates must also present evidence to show that all queries are fit for purpose. Candidates must present and clearly label the initial values stored in tables before any of the implemented queries are executed (for example, a list of quiz questions or a list of stock items) to indicate that these data values are the initial values. This applies even if the tables initially have no records (for example, a member table that initially has no records or a high score table with no entries). This evidence could be the SQL `INSERT` code used to populate the tables, or screenshots showing the initial values or empty tables.

To demonstrate that their queries are working, candidates should also include screenshot evidence to show that each implemented query is fit for purpose:

- Evidence for `INSERT` queries should show additional records in the table.

- Evidence for `UPDATE` queries should show that existing records in the table have been edited.
- Evidence for `DELETE` queries should show that records have been removed from the table.
- Evidence for `SELECT` queries should show that the returned results have been retrieved from the underlying table.

Throughout the implementation of their solution, candidates should carry out ongoing testing. As they code their project solution, all candidates automatically run any new sections of code to check that they are working correctly. Candidates should add details of the ongoing testing carried out as they added functional requirements to the solution.

Encourage candidates to use breakpoints or watchpoints to pause the code during execution, and generate evidence showing the values stored in internal data structures. Also, depending on the sequence of implementation, candidates may need to add temporary print lines, stubs or drive code to test certain sections of the solution.

Candidates should provide a brief description of any issues they encountered as they implemented the requirements, along with the full details of any references that they used to resolve the problem. If candidates resolved issues themselves or did not encounter difficulties implementing a particular component, they should briefly note this.

Testing

Candidates should carry out end-user requirement testing themselves by adopting the realistic persona of a typical end-user and completing each listed test case. Remind candidates that a suitable persona would be a typical end-user of the system, and the description of the persona should indicate key information about the user's characteristics, goals, intended use of the system, technical knowledge and familiarity with the system. Candidates should identify several test cases and match each of the end-user requirements identified at the analysis stage against at least one test case.

Encourage candidates to consider how to test their solution comprehensively. You should remind candidates to specify the actual input values they will use to test input validation. These values should include normal, extreme and exceptional values to ensure comprehensive testing of input validation routines. Vague statements such as ‘values like 42 will be entered to test that the input validation of pupil age works correctly’ do not provide sufficient detail for comprehensive testing of an input validation routine.

Evidence of testing of the functional requirements in the solution should include screenshot evidence for each test in the test plan.

Where appropriate, this evidence should include:

- screenshots matching the actual input values stated in the test plan
- screenshots of validation error messages
- evidence of breakpoints
- evidence generated by adding stubs or driver code to test individual components
- underlying database tables
- output showing the results of processing

Where appropriate, candidates may use evidence from ongoing testing carried out during implementation as evidence of final testing.

Evaluation

In the evaluation section of the project, the revised template will require candidates to discuss how their solution could be maintained in the future. They should refer to specific features of their solution that would make maintenance easier and give examples of maintenance tasks that may be needed.

You should encourage candidates to consider perfective maintenance tasks that could improve the solution in the future. Candidate evidence should describe these possible improvements and explain how they could be implemented.