**Basic/core set of test queries with expected responses**

// Create and use a test database

*CREATE DATABASE markbook;*

[OK]

*USE markbook;*

[OK]

// Create a table to store some data

*CREATE TABLE marks (name, mark, pass);*

[OK]

// Populate the table with a few entries

*INSERT INTO marks VALUES ('Steve', 65, true);*

[OK]

*INSERT INTO marks VALUES ('Dave', 55, true);*

[OK]

*INSERT INTO marks VALUES ('Bob', 35, false);*

[OK]

*INSERT INTO marks VALUES ('Clive', 20, false);*

[OK]

// Perform various test queries on the table…

*SELECT \* FROM marks;*

[OK]

id name mark pass

1 Steve 65 true

2 Dave 55 true

3 Bob 35 false

4 Clive 20 false

*SELECT \* FROM marks WHERE name != 'Dave';*

[OK]

id name mark pass

1 Steve 65 true

3 Bob 35 false

4 Clive 20 false

*SELECT \* FROM marks WHERE pass == true;*

[OK]

Id name mark pass

1 Steve 65 true

2 Dave 55 true

*UPDATE marks SET mark = 38 WHERE name == 'Clive';*

[OK]

*SELECT \* FROM marks WHERE name == 'Clive';*

[OK]

id name mark pass

4 Clive 38 false

*DELETE FROM marks WHERE name == 'Dave';*

[OK]

*SELECT \* FROM marks;*

[OK]

id name mark pass

1 Steve 65 true

3 Bob 35 false

4 Clive 38 false

*DELETE FROM marks WHERE mark < 40;*

[OK]

*SELECT \* FROM marks;*

[OK]

id name mark pass

1 Steve 65 true

**More substantial sample database**

// The following queries are not directly assessed, but are run to set up a sample database for subsequent testing

*USE imdb;*

*DROP TABLE actors;*

*DROP TABLE movies;*

*DROP TABLE roles;*

*DROP DATABASE imdb;*

*CREATE DATABASE imdb;*

*USE imdb;*

*CREATE TABLE actors (name, nationality, awards);*

*INSERT INTO actors VALUES ('Hugh Grant', 'British', 3);*

*INSERT INTO actors VALUES ('Toni Collette', 'Australian', 12);*

*INSERT INTO actors VALUES ('James Caan', 'American', 8);*

*INSERT INTO actors VALUES ('Emma Thompson', 'British', 10);*

*CREATE TABLE movies (name, genre);*

*INSERT INTO movies VALUES ('Mickey Blue Eyes', 'Comedy');*

*INSERT INTO movies VALUES ('About a Boy', 'Comedy');*

*INSERT INTO movies VALUES ('Sense and Sensibility', 'Period Drama');*

*SELECT id FROM movies WHERE name == 'Mickey Blue Eyes';*

*SELECT id FROM movies WHERE name == 'About a Boy';*

*SELECT id FROM movies WHERE name == 'Sense and Sensibility';*

*SELECT id FROM actors WHERE name == 'Hugh Grant';*

*SELECT id FROM actors WHERE name == 'Toni Collette';*

*SELECT id FROM actors WHERE name == 'James Caan';*

*SELECT id FROM actors WHERE name == 'Emma Thompson';*

*CREATE TABLE roles (name, movieID, actorID);*

*// Note: ids used in the following four lines are the ones returned by the previous SELECT queries*

*INSERT INTO roles VALUES ('Edward', 3, 1);*

*INSERT INTO roles VALUES ('Frank', 1, 3);*

*INSERT INTO roles VALUES ('Fiona', 2, 2);*

*INSERT INTO roles VALUES ('Elinor', 3, 4);*

**Advanced set of test queries with expected responses**

// ALTER, JOIN and *compound* WHERE statements are considered advanced features and so are tested in this section

//    >   <     >=    <=    LIKE  are also considered advanced functionality and so are tested in this section

// For simplicity >   <     >=    <=    only work with numerical data (when used, attempt to interpret data as numerical)

// We recommend you use Float.parseFloat(String s) to convert Strings to numbers for these comparisons

*SELECT \* FROM actors WHERE awards < 5;*

[OK]

id name nationality awards

1 Hugh Grant British 3

*ALTER TABLE actors ADD age;*

[OK]

*SELECT \* FROM actors;*

[OK]

id name nationality awards age

1 Hugh Grant British 3

2 Toni Collette Australian 12

3 James Caan American 8

4 Emma Thompson British 10

*UPDATE actors SET age = 45 WHERE name == 'Hugh Grant';*

[OK]

*SELECT \* FROM actors WHERE name == 'Hugh Grant';*

[OK]

id name nationality awards age

1 Hugh Grant British 3 45

*SELECT nationality FROM actors WHERE name == 'Hugh Grant';*

[OK]

nationality

British

*ALTER TABLE actors DROP age;*

[OK]

*SELECT \* FROM actors WHERE name == 'Hugh Grant';*

[OK]

id name nationality awards

1 Hugh Grant British 3

*SELECT \* FROM actors WHERE (awards > 5) AND (nationality == 'British');*

[OK]

id name nationality awards

4 Emma Thompson British 10

// Use of brackets ensures desired execution ordering is achieved

*SELECT \* FROM actors WHERE (awards > 5) AND ((nationality == 'British') OR (nationality == 'Australian'));*

[OK]

id name nationality awards

2 Toni Collette Australian 12

4 Emma Thompson British 10

// The LIKE operation only works with substrings (when used, attempt to interpret data as textual)

*SELECT \* FROM actors WHERE name LIKE 'an';*

[OK]

id name nationality awards

1 Hugh Grant British 3

3 James Caan American 8

*SELECT \* FROM actors WHERE awards >= 10;*

[OK]

id name nationality awards

2 Toni Collette Australian 12

4 Emma Thompson British 10

// Remove some data entries ready to test a JOIN query

*DELETE FROM actors WHERE name == 'Hugh Grant';*

[OK]

*DELETE FROM actors WHERE name == 'James Caan';*

[OK]

*DELETE FROM actors WHERE name == 'Emma Thompson';*

[OK]

// The JOIN query should perform an inner join on the two specified tables

// You need not store the combined table on the filesystem, but just return the complete table to the user

// This is equivalent to performing SELECT \* on the combined table

// The entry IDs should be generated for this table (they are NOT the IDs from the original two tables)

// Note: generated attribute names are just examples, but you might like to use those suggested

*JOIN actors AND roles ON id AND actorID;*

[OK]

id actors.name actors.nationality actors.awards roles.name roles.movieID roles.actorID

1 Toni Collette Australian 12 Fiona 2 2

// Another JOIN with multiple matches in the 2nd table

*JOIN movies AND roles ON id AND movieID;*

[OK]

id movies.name movies.genre roles.name roles.movieID roles.actorID

1 Mickey Blue Eyes Comedy Frank 1 3

2 About a Boy Comedy Fiona 2 2

3 Sense and Sensibility Period Drama Edward 3 1

4 Sense and Sensibility Period Drama Elinor 3 4

*DROP TABLE actors;*

[OK]

*SELECT \* FROM actors;*

[ERROR]: Table does not exist

*DROP DATABASE imdb;*

[OK]

*USE imdb;*

[ERROR]: Unknown database

**Robustness testing queries (note: imdb database is recreated afresh before running these)**

// These check for a range of query errors – it is essential that the server does not crash when encountering these

// All error messages are purely for the benefit of the user – please choose a suitably helpful message

// Testing will only check to make sure that the [ERROR] keyword has been returned

// Missing semi-colon

*SELECT \* FROM actors*

[ERROR]: Semi colon missing at end of line

*SELECT \* FROM crew;*

[ERROR]: Table does not exist

*SELECT spouse FROM actors;*

[ERROR]: Attribute does not exist

// Rogue bracket at the end of the line

*SELECT \* FROM actors);*

[ERROR]: Invalid query

// Missing close quote

*SELECT \* FROM actors WHERE name == 'Hugh Grant;*

[ERROR]: Invalid query

// In situations where it is not possible to convert strings to numbers an error should be generated

*SELECT \* FROM actors WHERE name > 10;*

[ERROR]: Attribute cannot be converted to number

// Missing comma between attribute names !

*SELECT name age FROM actors;*

[ERROR]: Invalid query

// Missing WHERE keyword

*SELECT \* FROM actors awards > 10;*

[ERROR]: Invalid query

// Use of LIKE on numerical data

*SELECT \* FROM actors WHERE name LIKE 10;*

[ERROR]: String expected

// Leading whitespace should not affect success of query

*SELECT \* FROM actors WHERE awards > 10;*

[OK]

id name nationality awards

2 Toni Collette Australian 12

*USE ebay;*

[ERROR]: Unknown database