

2.10 - SQL Testing with Appvance

SQL Testing with Appvance — Appvance IQ documentation.

- [SQL Testing with AIQ](#)
- [How to Create a Valid SQL Script](#)
- [Setting up the Environment](#)
- [Creating a New Database Scenario](#)
- [Reviewing SQL Test Execution Results](#)
- [Sample SQL Scripts for AIQ Testing](#)
- [Reduce MySQL Results Database Storage Usage](#)
- [SQL Testing Appendix for AIQ](#)

SQL Testing with AIQ

SQL Testing with AIQ

Use SQL testing in AIQ when a scenario must validate database state, prepare controlled test data, or confirm that an application workflow created the expected records. Keep these tests focused on non-production data unless the activity has been explicitly approved.

Recommended guidance

- Use a least-privilege database account that can access only the schema and operations required by the test.
- Keep connection details in secured AIQ configuration or environment-specific variables instead of hard-coding credentials in the scenario.
- Run read-only validation queries whenever possible. Use write, update, or delete statements only for controlled setup and cleanup steps.
- Parameterize test data with DPLs or variables so the same scenario can run safely across environments.
- Validate expected row counts, field values, and error handling before using the scenario in a scheduled run.

Validation workflow

1. Confirm the target database, schema, and account permissions.
2. Create or select the SQL step in the AIQ test flow.
3. Define the query, bind any variables, and set the expected result.
4. Run a small smoke test and review the output before increasing test volume.
5. Clean up generated test data when the scenario requires it.

Note: Avoid using broad production queries or unrestricted database accounts for performance or functional test validation.

Last updated: October 8, 2026

How to Create a Valid SQL Script

To create a valid SQL script in Appvance AIQ, you must configure the database connection properties correctly. These properties are based on Hibernate and are used to map the database configuration and establish a connection through the appropriate JDBC driver.

The following configuration values are required:

- User Name
- Password
- URL (Connection String)
- Driver Class
- Database Name
- Dialect
- Connection Delimiters

All of these values must be correctly configured in the SQL script when using ASE to establish a database connection.

Database Connection Properties

1. User Name

The **User Name** is the database account used to authenticate and access the required database.

Make sure that:

- The user exists in the database.
- The user has the required permissions to perform the requested operations.
- The database allows connections from the machine where the script is executed.
- For remote databases, remote access is enabled and permitted by the database and network configuration.

2. Password

The **Password** is the password associated with the specified database user.

The credentials must match a valid database account. If authentication fails, verify the username, password, and database access permissions.

3. URL

The **URL** specifies the JDBC connection string used to connect to the database.

The format of the URL depends on the database engine and JDBC driver being used.

For example, a typical MySQL JDBC connection URL can be structured as follows:

```
jdbc:mysql://<host>:<port>/<database>
```

Additional connection parameters can be included when required by the database or JDBC driver.

4. Driver Class

The **Driver Class** identifies the JDBC driver that Appvance uses to establish the database connection.

Each database engine has its own JDBC driver class.

For example, the current MySQL Connector/J driver class is:

```
com.mysql.cj.jdbc.Driver
```

“ **Note:** Older MySQL documentation may reference `com.mysql.jdbc.Driver`. This class name was deprecated in MySQL Connector/J 8.0 and should not be used for current configurations.

Make sure the appropriate JDBC driver is available to the AIQ environment where the SQL script is executed.

5. DB Name

The **DB Name** is the name of the database that the script connects to.

It must correspond to an existing database that is accessible by the configured user.

For example:

```
my_database
```

The database name is also commonly included as part of the JDBC connection URL.

6. Dialect

The **Dialect** specifies the Hibernate dialect used to communicate with the selected database engine.

For Appvance AIQ, the dialect must use the Appvance-specific Hibernate dialect package:

```
org.appvance.hibernate.dialect
```

The Appvance dialect package is based on Hibernate dialect implementations and was modified to include the `appvance` package name to avoid runtime conflicts.

The appropriate dialect must be selected according to the database engine being used. Refer to the supported dialect table in this documentation for the available values.

7. Connection Delimiters

The SQL script must include the commands required to explicitly open and close the database connection.

The connection delimiters are:

```
Connect  
Disconnect
```

Connect starts the database connection using the configured connection properties.

Disconnect closes the database connection after the SQL operations have completed.

These commands should be used to clearly define the beginning and end of the database interaction.

Example Configuration

A valid SQL script must contain the required connection information and use the appropriate connection delimiters. The exact syntax and values depend on the database engine and JDBC driver being used.

Before executing the script, verify that:

1. The database user and password are valid.
 2. The database is reachable from the AIQ execution environment.
 3. The JDBC URL is correctly formatted.
 4. The required JDBC driver is available.
 5. The database name is correct.
 6. The appropriate Appvance Hibernate dialect is configured.
 7. The `Connect` and `Disconnect` delimiters are present.
 8. The database user has sufficient permissions for the SQL operations performed by the script.
-

Last updated: August, 17 2026

If you have any feedback on this topic, please contact us at documentation@appvance.ai.

Setting up the Environment

Now let's see how to create an SQL script using the required configuration settings. In this example, we use a MySQL database named `appvancetest` with a `worker` table containing the following columns:

- `idworker`
- `name`
- `lastname`

The database can be hosted on a local or remote machine. MySQL uses port `3306` by default, although the port may vary depending on your database configuration.

1. Configure the database connection

The first step is to add the Hibernate settings required to connect to the database and execute the script.

The configuration settings must be defined at the beginning of the SQL script, before any SQL actions are executed.

As shown in the example, provide the appropriate value for each setting required to establish the database connection.

These settings are specific to your database environment. For example, the database host, database name, username, password, and port must correspond to the MySQL instance you want to access.

```
-- username=rootRemote
-- password=admin
-- url=jdbc:mysql://appvance00:3306/appvancetest
-- driver_class=com.mysql.jdbc.Driver
-- database=appvancetest
-- dialect=org.appvance.hibernate.dialect.MySQLDialect
-- connect

INSERT INTO `appvancetest`.`worker`
    (`idworker`, `name`, `lastname`)
VALUES
    (1, 'John', 'Smith');

-- disconnect
```

2. Add the SQL actions

After defining the connection settings, add the **connect delimiter** before the SQL actions.

You can then add the SQL statements that you want to execute. Once all SQL actions have been defined, add the **disconnect delimiter** to close the database connection.

For example, a script that inserts a new record into the `worker` table follows this structure:

```
-- connect

INSERT INTO `appvancetest`.`worker`
    (`idworker`, `name`, `lastname`)
VALUES
    (1, 'John', 'Smith');

-- disconnect
```

```
-- Hibernate configuration settings

connect

INSERT INTO worker (idworker, name, lastname)
VALUES (1, 'John', 'Doe');

disconnect
```

The exact configuration values and SQL statement will depend on your database environment and the operation you want to perform.

These are the basic steps for creating a valid SQL script for ASE and using it in a scenario to execute database-related test actions.

```
-- username=rootRemote
-- password=admin
-- url=jdbc:mysql://appvance00:3306/appvancetest
-- driver_class=com.mysql.jdbc.Driver
-- database=appvancetest
-- dialect=org.appvance.hibernate.dialect.MySQLDialect
```

Important

- The configuration settings must be placed before the SQL actions.
- Type `--` before each configuration setting so that ASE recognizes the setting.
- The SQL script must use the `.sql` file extension.
- Make sure the database host, port, credentials, and database name are valid and accessible from the environment where the script is executed.

Last updated: August 13, 2026

If you have any feedback on this topic, please contact us at documentation@appvance.ai.

Creating a New Database Scenario

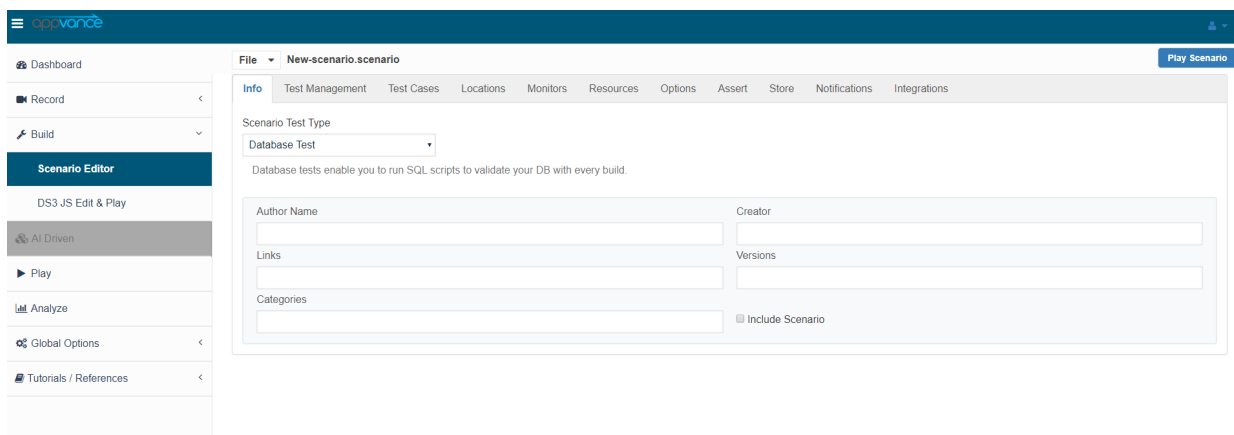
Once you have created a valid SQL script, you can create an AppVance Scenario to execute it using the Scenario Builder.

This article assumes basic knowledge of AppVance Scenario Builder and focuses only on the configuration required to create a Database Scenario and execute a SQL script.

Configure the Scenario

1. Select the Database Scenario Type

In the **Info** section of the Scenario Builder, select **Database** as the **Scenario Test Type**.



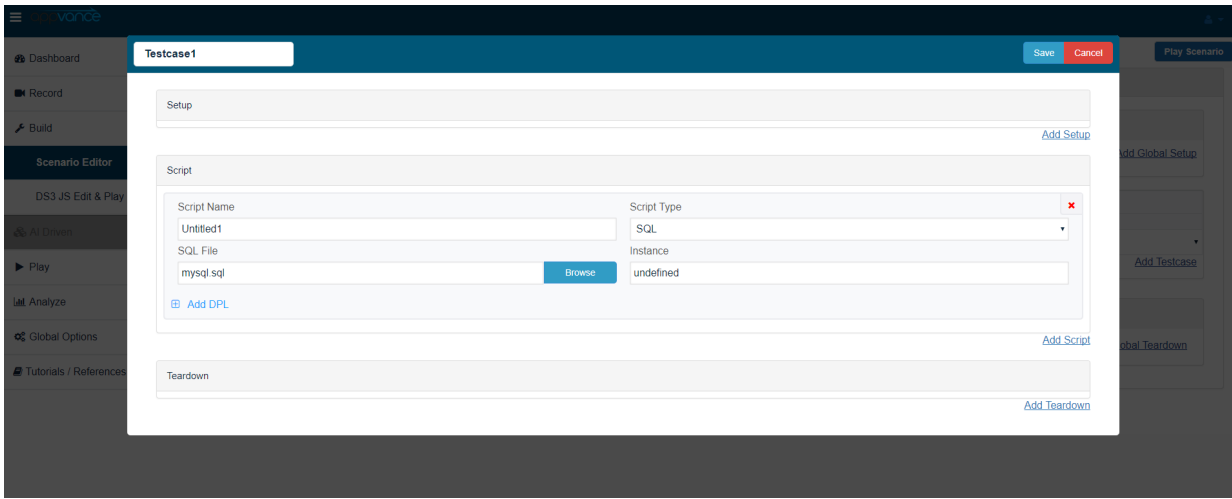
The screenshot shows the AppVance Scenario Builder interface. On the left is a sidebar with navigation options: Dashboard, Record, Build, Scenario Editor (highlighted), DS3 JS Edit & Play, AI Driven, Play, Analyze, Global Options, and Tutorials / References. The main area is titled 'File - New-scenario.scenario' and has a 'Play Scenario' button in the top right. Below the title bar is a tabbed interface with 'Info' selected. The 'Info' tab contains a 'Scenario Test Type' dropdown menu set to 'Database Test'. Below this, a text box states: 'Database tests enable you to run SQL scripts to validate your DB with every build.' Further down are input fields for 'Author Name', 'Creator', 'Links', 'Versions', and 'Categories'. There is also a checkbox labeled 'Include Scenario'.

2. Configure the SQL Test Case

In the **Test Case** section:

1. Select **SQL** from the **Test Type** drop-down menu.
2. In the **SQL File** field, click **Browse**.
3. Select and upload the `.sql` script you created previously.

Selecting **SQL** as the Test Type tells AppVance to use the **ASE** database execution mechanism.



After configuring the Scenario and uploading the SQL file, save the Scenario.

Click **Play** to execute the Scenario and run the SQL script.

Before executing the Scenario, make sure that the database connection settings defined in the SQL script are valid and that the target database is accessible from the environment where the Scenario is running.

Last updated: September 2, 2026

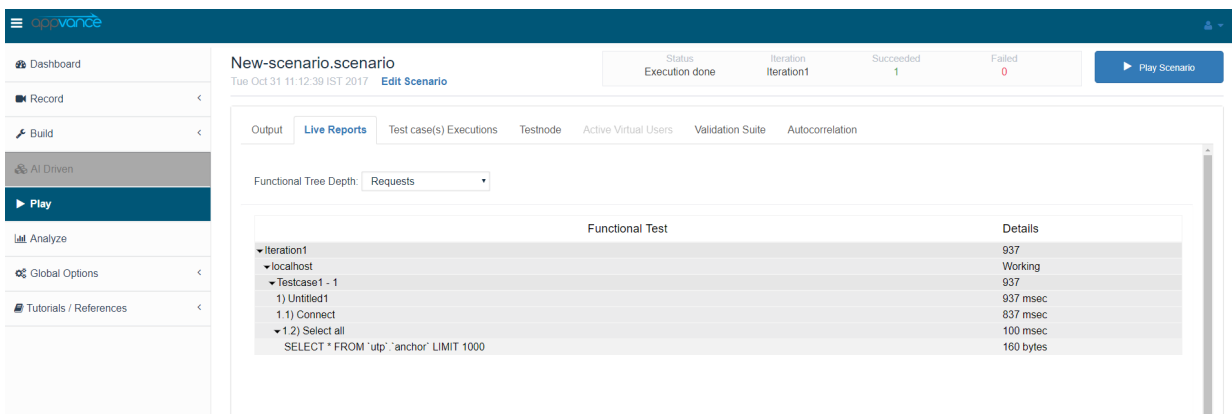
If you have any feedback on this topic, please contact us at documentation@appvance.ai

Reviewing SQL Test Execution Results

This section explains how to review the results of a SQL test execution and how to customize the step names displayed in the execution results.

In this example, the SQL test inserts a new record into the database and then retrieves the records using a `SELECT` statement. The execution results can then be reviewed to validate the data returned by the database.

The following example shows the results of a SQL test executed against a MySQL database:



The screenshot displays the Appvance web interface for reviewing test results. The left sidebar contains navigation links: Dashboard, Record, Build, AI Driven, Play (highlighted), Analyze, Global Options, and Tutorials / References. The main content area is titled 'New-scenario.scenario' and shows a summary of the test execution: Status 'Execution done', Iteration 'Iteration1', Succeeded '1', and Failed '0'. A 'Play Scenario' button is visible. Below the summary, there are tabs for 'Output', 'Live Reports' (selected), 'Test case(s) Executions', 'Testnode', 'Active Virtual Users', 'Validation Suite', and 'Autocorrelation'. The 'Live Reports' tab shows a 'Functional Tree Depth' dropdown set to 'Requests'. The main table displays the 'Functional Test' results, including a tree view of the test structure and a 'Details' column showing execution metrics.

Functional Test	Details
Iteration1	937
localhost	Working
Testcase1 - 1	937
1) Untitled1	937 msec
1.1) Connect	837 msec
1.2) Select all	100 msec
SELECT * FROM 'ulp' 'anchor' LIMIT 1000	160 bytes

Customizing SQL Step Names

By default, the step name displayed in the execution results corresponds to the SQL action or request defined in the script.

You can assign a custom name to a SQL step by adding a `-- step` instruction immediately before the corresponding SQL statement.

Use the following syntax:

```
-- step=Custom Step Name
```

For example:

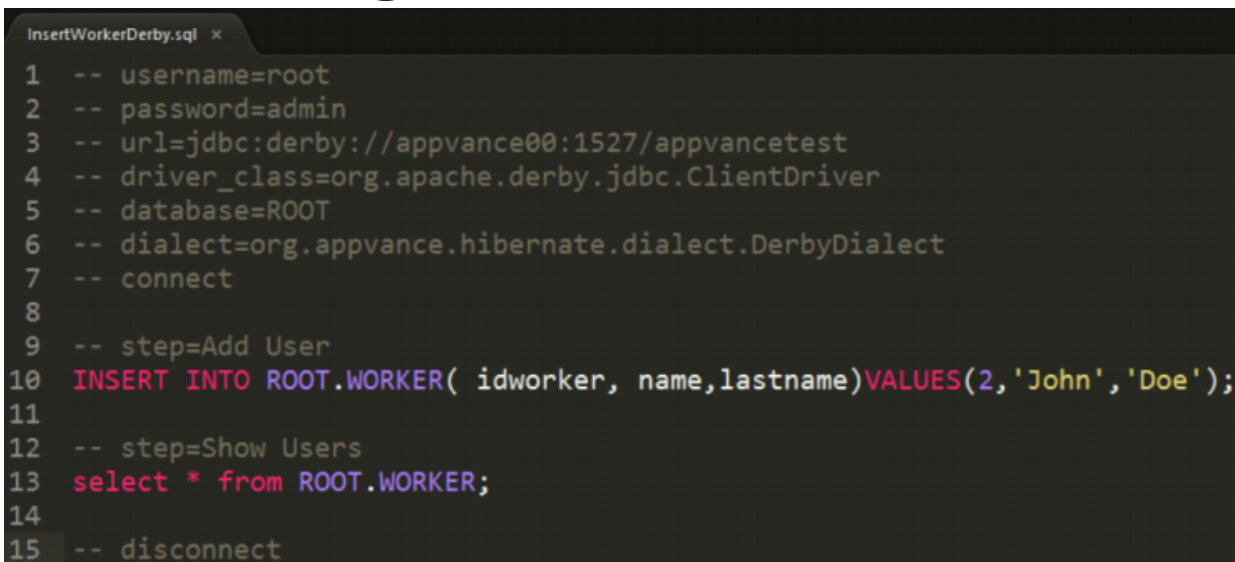
```
-- step=Select All
SELECT * FROM users;
```

The SQL action is displayed in the execution results using **Select All** as the step name.

This can make execution results easier to understand, especially when a SQL script contains multiple SQL statements.

The `.sql` script settings used for this execution are shown below:

Reviewing



```
InsertWorkerDerby.sql x
1  -- username=root
2  -- password=admin
3  -- url=jdbc:derby://appvance00:1527/appvancetest
4  -- driver_class=org.apache.derby.jdbc.ClientDriver
5  -- database=ROOT
6  -- dialect=org.appvance.hibernate.dialect.DerbyDialect
7  -- connect
8
9  -- step=Add User
10 INSERT INTO ROOT.WORKER( idworker, name,lastname)VALUES(2,'John','Doe');
11
12 -- step=Show Users
13 select * from ROOT.WORKER;
14
15 -- disconnect
```

SELECT Results

You can review the data returned by a `SELECT` statement from the **Request & Response** section of the execution results.

To review the response:

1. Open the SQL test execution results.
2. In the panel on the right side of the results screen, select **Test-case Executions**.
3. Select the test case execution you want to review.
4. Open the **Request & Response** tab.
5. Select the step associated with the `SELECT` statement.

The **Request & Response** tab displays the requests and responses associated with the individual execution steps. The custom step names assigned with the `-- step` instruction can be used to identify the corresponding SQL statements.

Selecting the request associated with the `SELECT` statement downloads the corresponding request data as a plain-text file.

The downloaded file does not have a file extension, so it can be opened with a text editor such as Notepad or Sublime Text.

The file contains the records returned by the `SELECT` statement during the test execution. You can use this information to verify that the database returned the expected data and validate the results of the SQL operation.

Last updated: September 3, 2026

If you have any feedback on this topic, please contact us at documentation@appvance.ai.

Sample SQL Scripts for AIQ Testing

Recommended guidance

Sample SQL scripts should demonstrate safe, repeatable validation patterns for AIQ testing. Examples must avoid production data, destructive statements, and customer-specific credentials.

Example hygiene

- Use placeholder connection values, schema names, table names, and credentials.
- Prefer read-only queries unless the article explicitly covers controlled setup or cleanup.
- Explain expected results, assumptions, and failure conditions.
- Include transaction or rollback guidance when data changes are required.

Validation workflow

1. Run scripts in a non-production database first.
2. Confirm permissions match the intended AIQ execution account.
3. Review output in AIQ and database logs.
4. Remove or update obsolete examples that no longer match current SQL testing workflows.

Last updated: October 7, 2026

Reduce MySQL Results Database Storage Usage

Reduce MySQL Results Database Storage Usage

Use this procedure when the AIQ results database is consuming excessive storage. Treat database cleanup as a controlled maintenance task because deleting execution data can permanently remove historical evidence and reports.

Before making changes

- Confirm the affected MySQL instance, schema, and available disk space.
- Create a verified backup or snapshot before deleting or archiving results.
- Confirm the retention window with the customer or project owner.
- Prefer supported AIQ cleanup, archival, or retention settings before using direct database changes.

Recommended cleanup workflow

1. Identify the largest result tables and the date range that can be archived or removed.
2. Export any reports or evidence that must be retained.
3. Run cleanup in a maintenance window with AIQ services stopped or quiesced if required by the procedure.
4. Optimize or analyze tables only after the cleanup approach has been validated.
5. Restart services and verify that recent executions, reports, and login flows still work.

Warning: Do not run broad delete statements without an approved backup and an exact retention target.

Last updated: October 8, 2026

SQL Testing Appendix for AIQ

Recommended guidance

Use the SQL Testing appendix for supplemental reference material that supports active SQL testing workflows. Keep appendix content concise, current, and linked to the main SQL testing guidance.

What belongs here

- Reference examples that are too detailed for the main article.
- Terminology, troubleshooting notes, or environment-specific reminders.
- Links to approved SQL call, data, and validation articles.
- Warnings about database permissions, sensitive data, and cleanup.

Maintenance checklist

1. Remove duplicate examples that already exist in active SQL pages.
2. Replace Appvance wording with AIQ where appropriate.
3. Use placeholders instead of customer-specific connection strings or credentials.
4. Validate examples against current Service Suite or SQL testing behavior.

Last updated: October 7, 2026