SQL Queries

The SQL element allows you to execute, test, and review custom SQL queries in your applications. The element provides flexibility in data manipulation, but we recommend using Aggregates when applicable. Aggregates are highly optimized and easier to maintain.

SQL queries can access data that is sent through the input parameters only, and other logic can access only what the SQL query returns through the Output Structure.

**Input parameters**
Providing input parameters allows you to use dynamic data in the SQL query. Input parameters are optional. To reference an input parameter in your SQL statement use a `@` prefix, e.g. `@CustomInputParameter`.

**Output parameter**
SQL in OutSystems queries always have two output parameters, even when the query executed does not return a result:

- **List**: The list with the result returned by the query. The list is empty if there are no results.
- **Count**: The number of records returned by the query without considering the SQL Max Records property.

Output Structure is mandatory. You need to define the structure (data types of the columns) that your query returns. For example, if you perform a query over the Employee Entity that selects the attributes `Id, Name, Email, PhoneNumber`, specify the Output Structure with the same attributes in the same order and with matching data types. This enforces that List output parameter of the SQL query returns Employee List data type. Output Structure is needed even if your SQL statement does not return any results.

To reference an entity in your SQL query write it between curly brackets (e.g. `{User}`) and to reference an entity attribute write it between square brackets (e.g. `[PhoneNumber]`).

https://success.outsystems.com/Documentation/11/Developing_an_Application/Use_Data/Query_Data/SQL_Queries

Updated: Thu, 26 Dec 2019 01:39:13 GMT
Write Your Own SQL Query

Do the following:

1. Add a SQL element to an action flow.
2. If necessary, define the query parameters.
3. Write the SQL query.
4. Define the output structure used for the output of the SQL node.
5. Use the output list of the SQL node to access the result of the SQL query.

Notes and guidelines

Consider the following when using the SQL element:

Avoid Data Definition Language
The SQL tool should only be used to execute the following: SELECT, INSERT, UPDATE, and DELETE statements. Using Data Definition Language (DDL) statements like CREATE TABLE, ALTER TABLE, DROP TABLE, etc. might make the OutSystems metamodel inconsistent, leading to misbehavior.

No physical layer query
To ensure the OutSystems metamodel is always consistent, you cannot perform queries directly to the physical tables of the metamodel. The platform shows an error message if your query uses tables prefixed with OSLOG, OSSYS, or OSUSR.

Exception triggers a rollback
OutSystems starts a transaction when the web request arrives to the server, and commits the transaction before sending the reply to the user. If an exception is left uncaught, the transaction is rolled back to ensure your data is always consistent. This means that your queries run inside a transaction and that changes are rolled back if something goes wrong.

Check runtime user permissions
Your queries are tested and run in the database using the runtime user specified in the Configuration Tool. You need to make sure this user has permissions to run the SQL statements specified in your query, otherwise an error will occur when you execute the query at runtime or when you test it.

Database support check
If the Database Module Property is set to All, OutSystems checks the queries to ensure they are compliant with all supported databases. If not, a warning message is displayed.

Avoid enabling the Expand Inline property of query parameters
Proper use of parameters being expanded inline is hard, since you need to make sure that any user input has been properly escaped in the right way before using it in an SQL statement. If possible, avoid enabling this property altogether.
OutSystems provides ways of implementing common use cases without enabling this property.
Convert an Aggregate to SQL

As your application grows, you may need to change an existing Aggregate to fetch data in a different way. If your use case requires more advanced data fetching, such as subqueries or IN clauses, you will need to use a SQL element instead of an Aggregate. For these situations, you can convert an existing Aggregate to a SQL element, which enables you to keep evolving the SQL generated from your original Aggregate.

Although a SQL element enables you to refine your query, Aggregates have the following advantages:

• The runtime SQL statement of an Aggregate is optimized to fetch the necessary number of rows and columns.

• The effort to maintain an Aggregate is lower.

Therefore, you should always start fetching data using an Aggregate and convert to a SQL element only when necessary.

Limitations

The option to convert an Aggregate to a SQL element will only be available if your Aggregate doesn't include:

• Structures in Sources
• Calculated Attributes
• Group By attributes
• Dynamic Sorts

In Reactive Web and Mobile apps, this feature is not available for Aggregates in Client Actions or Screens.
How to Convert an Aggregate to a SQL Element

To convert an existing Aggregate to a SQL element, do the following:

1. In your action flow, double-click the Aggregate you want to convert.

2. In the Aggregate window, double-click the Executed SQL property to open the Executed SQL window.

3. Click the CONVERT AGGREGATE TO SQL button. This button is only enabled if your Aggregate doesn't include any of the limitations listed above.
4. Click the button **PROCEED**.

Your action flow now includes a SQL element based on the original Aggregate.

The original Aggregate is kept in the flow editor for your manual deletion after validating the new SQL element.

**See Also**

 SQL