Oracle® NoSQL Database

Plugins Guide

Release 26.1
G12046-09
April 2026

Oracle NoSQL Database Plugins Guide, Release 26.1

G12046-09

Contents

List of Tables

1

About IntelliJ Plugin

Browse tables and execute queries on your Oracle NoSQL Database data store from IntelliJ.

The Oracle NoSQL Database IntelliJ plugin connects to a running instance of your data store and allows you to:

- View the tables in a well-defined tree structure with Table Explorer.
- View information on columns, indexes, primary key(s), and shard key(s) for a table.
- View column data in a well-formatted JSON Structure.
- Create tables using form-based schema entry or supply DDL statements.
- Drop tables.
- Add new columns using form-based entry or supply DDL statements.
- Drop Columns.
- Create Indexes.
- Drop Indexes.
- Execute SELECT SQL queries on a table and view query results in tabular format.
- Execute DML statements to update, insert, and delete data from a table.

Topics:

Setting Up IntelliJ Plugin

Learn how to set up the IntelliJ plugin for your data store.

Prerequisite:
Install IntelliJ IDEA. You can download IntelliJ IDEA from JetBrains.

Procedure:

You can install the plugin as follows:

1. Open the *Oracle NoSQL Database Connector* page, JetBrains plug-in repository.
2. Select **Get**.
3. From the *Versions* menu, select **Download** for the latest version of the Oracle NoSQL Database Connector.

4. In the pop-up window, follow the on-screen instructions under **How to Install 'Oracle NoSQL Database Connector'**.

Restart the IDE. You will see the **Schema Explorer** icon on the right panel.

After you have successfully installed your IntelliJ plugin, you can create a NoSQL project and connect it to your data store.

Creating a NoSQL Project in IntelliJ

Learn how to create a NoSQL project in IntelliJ.

Perform the following steps:

1. Open IntelliJ IDEA. Click **File** > **New** > **Project**.
2. Enter a value for **Project Name** and **Project Location**. Select **Create**.
3. Select a build system and JDK path.
4. IntelliJ creates your NoSQL project directory, which includes a sample Java file. If you have selected Maven as the build system, your project directory also includes a *pom.xml* file.
5. Make sure Notifications are enabled for your Oracle NoSQL project. To enable Notifications, press `Alt+\` to open Main Menu. Select **View**, expand **Tool Windows** > **Notifications**. A Notification icon  appears on the right tool window bar.

After you successfully created a NoSQL project in IntelliJ, you can connect your project to your data store.

Connecting to Oracle NoSQL Database from IntelliJ

Learn how to connect your NoSQL project to Oracle NoSQL Database the data store using the IntelliJ plugin.

Prerequisites:
To create a successful connection to your Oracle NoSQL Database data store, ensure that:

- The data store is deployed and running.
- The Oracle NoSQL Database Proxy is started. See Configuring the Proxy. Starting the release 19.5, Proxy is bundled along with the Oracle NoSQL Database download package.

Perform the following steps:

1. Open your NoSQL project in IntelliJ.
2. Click the **task** icon  in the **Schema Explorer** window to open the **Settings** dialog for the plugin.
3. Expand **Tools** > **Oracle NoSQL** in the Settings Explorer, and click **Connections**. You can view all the existing connections for the on-premises profile type under the Connections dropdown.
4. Select **Onprem** from the drop-down menu for the Profile type.
5. Click **Add Connection**. Enter values for the following connection parameters and select **ADD**. Then select **OK**.

Table 1-1 Connection Parameters

Parameter	Description
Connection Name	A unique name, that is given to a specific connection specification is mandatory from the plugin version 1.5.1. Updating the Connection Name field is recommended after upgrading the plugin from version 1.4.0 or lower. **Note** You can add multiple connections and the stored connection specifications are persistent.
Proxy URL	`http://<proxy_host>:<proxy_http_port>` or `https://<proxy_host>:<proxy_http_port>` where: • `http` or `https` indicates the store security. For a secure data store, the proxy URL begins with `https`. • `proxy_host` is the host name of the machine to host the proxy service. This should match the host you configured earlier. See Configuring the Proxy.
Namespace	Provide the name of your namespace. This is optional. If no value is provided then the default namespace `sysdefault` is used. **Note** You can add one or more namespaces to your store, create tables within them, and grant permission for users to access namespaces and tables.
Security	Select **SSL** for secure data store. In case, you are creating connection to a non-secure KVStore, select **None**. The default value is **SSL**. **Note** In case of secure data store, the proxy URL must begin with `https`.
Username	User name to connect to the secure store. This value is required only if you select **SSL** for the Security parameter.
Password	Password to connect to the secure store. This value is required only if you select SSL for the Security parameter.

Table 1-1 (Cont.) Connection Parameters

Parameter	Description
TrustStore	Browse to the location where the certificate trust file is placed. See Using the Proxy in a secure data store.
Passphrase	Give a value for the passphrase. A passphrase refers to a secret used to protect an encryption key. This is optional.

> **Note**
>
> - Starting with version 1.5.4, the IntelliJ plugin automatically downloads the latest Oracle NoSQL Java SDK and sets the SDK path when creating a connection.
> - If you are updating the plugin from version 1.4.0 or lower, all the stored connections migrate to the new version. In this case, the Connection Name will be the same as Proxy URL. Follow the below step to change the Connection Name.

6. The IntelliJ Plugin saves the connection details in the connection name specified. To modify the connection details, choose the connection name in the drop-down for **Connections**. Click **Modify Connection**. You can change any of the connection parameters (mentioned above) and click **OK** to save the settings. To remove a connection name from the plugin, choose the connection name and click **Delete Connection**. Once you confirm the action to delete, the connection name is removed from the plugin.
7. Click the Web icon in the **Schema Explorer**. The list of existing connections is displayed in the drop-down box. The connection will be displayed in the NoSQL tool window in the following format:

   `Connection Name:Proxy URL:Namespace` (if specified).

   Choose the connection and click **OK**. The Intellij plugin connects your project to the Oracle NoSQL Database data store and displays its schema in the **Schema Explorer** window.
8. Click the **Database icon in the Schema Explorer**. A calculator tool is opened in a new window. This advanced tool provides values for cache size, storage size and the number of shards based on the provided inputs. These numbers are useful when doing capacity planning. See the Capacity Planning section in the Administrator's Guide for additional information.

- In the Je Cache Size Calculator panel, enter the values in bytes for Primary Key Size, Row Size, and Index (this is optional), along with the number of rows. Click **Add Table** to add values for another table.
- Click **CALCULATE** to view the output values. To start a new calculation, click **New Calculation**.

After you successfully connect your project to your Oracle NoSQL Databasedata store, you can manage the tables and data in your schema.

Creating applications using Oracle NoSQL Java SDK from IntelliJ

Learn to create applications from IntelliJ plugin using the Oracle NoSQL Java SDK package.

After connecting to the Oracle NoSQL Database, you can create applications using Oracle NoSQL Java SDK.

In this topic, you will run a sample application available within the Oracle NoSQL Java SDK package to create a table in Oracle NoSQL Database.

Perform the following steps:

1. Download the latest Oracle NoSQL Database Java SDK from GitHub.
2. Extract (unzip) the downloaded file in a local repository.
3. Launch IntelliJ IDEA and open your NoSQL project.
4. If you are using the Maven build system, ensure that you have added the required dependencies in pom.xml. For details, see Oracle NoSQL Java SDK GitHub.
5. From the extracted Oracle NoSQL Database Java SDK directory, navigate to `oracle-nosql-java-sdk > examples` folder. Copy `BasicTableExample` and `Common` java files to your NoSQL project directory.

   The `BasicExampleTable` program creates a table called `audienceData`, adds two rows into this table, queries the inserted rows, deletes the inserted rows, and finally drops the `audienceData` table.
6. Before running the application program, you must provide the necessary arguments to connect:
   a. From the menu, select **Run** > **Edit Configurations**.
   b. Select **Add New Configuration** and then select **Application**.
   c. In the window that appears on the right, enter your main class in the **Name** field.
   d. Under **Build and run**, specify the Java JDK path and enter the main class.
   e. Enter the required program arguments. Select **Apply** and then select **Okay**.

Table 1-2 Program Arguments

Program Arguments	More Information

On-premises non-secure connection:
 	
`http://<proxy_host>:<proxy_http_port> -useKVProxy`
 	
On-premises secure connection:
 	
`http://<proxy_host>:<proxy_http_secureport> -useKVProxy [-user <user>] [-password password]`
	For example, if your Proxy URL is `http://<proxy_host>:<proxy_http_port>`, the program argument must be:
	Non-secure: `http://<proxy_host>:8080 -useKVProxy`
	Secure: `http://<proxy_host>:443 -useKVProxy -user <user> -password <password>`
	.

7. To run this program, select **Run** or press **Shift + 10**.

> **Note**
>
> If you want to run the program from a terminal, use the following sample commands:
>
> Compile the program using the maven build system using:
>
> ```
> mvn compile
> ```
>
> Run the program by supplying the program arguments:
>
> ```
> mvn exec:java -Dexec.mainClass=BasicTableExample -
> Dexec.args="http://<proxy_host>:<proxy_http_port>"
> ```

8. View the output in the **Run tool** window at the bottom. Verify the logs to confirm that the code execution is successful. You can see the display messages that indicate table creation, rows insertion, and so on.

   Sample output:

```
Creating table audienceData
Created table audienceData
Put row: {"cookie_id":123,"audience_data":{"audience_segment":
{"sports_lover":"2018-11-30","book_reader":"2018-12-01"},"ipaddr":"10.0.00.
xxx"}} result=non-null Version
Got row: {"cookie_id":123,"audience_data":{"audience_segment":
{"sports_lover":"2018-11-30","book_reader":"2018-12-01"},"ipaddr":"10.0.00.
xxx"}}
Inserted row via query, result:
    {"NumRowsInserted":1}
Got row inserted by query: {"cookie_id":106,"audience_data":
{"audience_segment":
{"foodie":"2020-06-01","sports_lover":"2020-05-10"},"ipaddr":"10.0.00.xxx"}
}
Put row from json: {"cookie_id":456,"audience_data":
{"ipaddr":"10.0.00.yyy","audience_segment":
{"sports_lover":"2019-01-05","foodie":"2018-12-31"}}}
Got row inserted as JSON: {"cookie_id":456,"audience_data":
{"audience_segment":
{"sports_lover":"2019-01-05","foodie":"2018-12-31"},"ipaddr":"10.0.00.yyy"}
}
Query results for SELECT * from audienceData WHERE cookie_id = 456:
    {"cookie_id":456,"audience_data":{"audience_segment":
{"foodie":"2018-12-31","sports_lover":"2019-01-05"},"ipaddr":"10.0.00.yyy"}
}
Deleted key {"cookie_id":456} result=true
Dropping table audienceData

Process finished with exit code 0
```

> **Note**
>
> As the `BasicExampleTable` deletes the inserted rows and drops the `audienceData` table, you can't view this table in the **Schema Explorer**. If you want to see the table in the **Schema Explorer**, comment the code that deletes the inserted rows and drops the table, and rerun the program.

Managing Tables Using the IntelliJ Plugin

Learn how to view table data in your data store from IntelliJ **Schema Explorer**.

Perform the following steps:

1. Locate the **Schema Explore**r, and click the **Refresh** icon to reload the schema.
2. Locate your table under your tenant identifier, and expand it to view its columns, primary key, and shard key details. If you created a sample application as described in the Creating applications using Oracle NoSQL Java SDK from IntelliJ topic, you will be able to see the `audienceData` table.
3. Double-click the table name to view its data. Alternatively, you can right-click the table and select **Browse Table**.
4. A record viewer window appears in the main editor. Click **Execute** to run the query and display table data.
5. To view individual cell data separately, double-click the cell.

Perform DDL operations using IntelliJ

You can use IntelliJ to perform DDL operations.

Some of the DDL operations that can be performed from inside the IntelliJ plugin are

- CREATE TABLE
- DROP TABLE
- CREATE INDEX
- DROP INDEX
- ADD COLUMN
- DROP COLUMN
- EXECUTE DDL

CREATE TABLE

- Locate the Schema Explorer, and click the **Refresh** icon to reload the schema.
- Right click the connection name and choose **Create Table**.
- In the prompt, enter the details for your new table. You can create the Oracle NoSQL Database table in two modes:
  - **Simple DDL Input** : You can use this mode to create the table declaratively, that is, without writing a DDL statement.
  - **Advanced DDL Input** : You can use this mode to create the table using a DDL statement.

- You have the option to view the DDL statement before creating. Click **Show DDL** to view the DDL statement formed based on the values entered in the fields in the Simple DDL input mode. This DDL statement gets executed when you click Create.
- Click **Create** to create the table.
- To create a child table, right click on the desired table and choose **Create Child Table**. You can create a child table in two modes:
  - **Simple DDL Input** : You can use this mode to create a child table by simply entering a table name along with other required details.
  - **Advanced DDL Input** : You can use this mode to create a child table using a DDL statement.

  For more details on child tables, see Table Hierarchies in *Developer's Guide*.
- Click **Create** to create a child table.
- You have an option to the view the DDL statement after creating a table. Right click on the existing table. Choose **View Table DDL**. To copy the DDL statement, click **Copy to Clipboard**. Click **OK** to close the dialog box.

DROP TABLE

- Locate the Schema Explorer, and click the Refresh icon to reload the schema.
- Right click on the table that you want to drop. Choose **Drop Table**.
- A confirmation window appears, click **Ok** to confirm the drop action.

CREATE INDEX

- Locate the Schema Explorer, and click the Refresh icon to reload the schema.
- Right click on the table where index need to be created. Choose **Create Index**.
- In the Create Index panel, you have an option to create index in two modes:
  - **Form Based Index Creation(Simple DDL Input)** : Enter the details for creating an index without writing any DDL statement. Specify the name of the index and the columns to be part of the index. If the column is of JSON data type, you see an additional field called "`JSON Path to Index Field`" appear. Enter the path to the location of the JSON field, and choose the data type for it.
  - **Create Index as DDL Statement (For Advanced DDL input)** : Enter a valid DDL statement to create an index. It can also include complex data type i.e. array, map, and record.
- Click **Add Index**.

DROP INDEX

- Locate the Schema Explorer, and click the Refresh icon to reload the schema.
- Click on the target table to see the listed columns, Primary Keys, Indexes and Shard Keys.
- Locate the target-index which has to be dropped and right-click on it. Click **Drop Index**.
- A confirmation window appears, click **Ok** to confirm the drop action.

ADD COLUMN

- Locate the Schema Explorer, and click the Refresh icon to reload the schema.
- Right click on the table where column needs to be added. Choose **Add Column**.

- You can add new COLUMNs in two modes:
  - Simple DDL Input : You can use this mode to add new columns without writing a DDL statement. In case of binary or fixed binary select the data type as `Binary`. For fixed binary, enter the size of the file in the `Size` field and keep the field null in case of binary data type.
  - Advanced DDL Input : You can use this mode to add new columns into the table by supplying a valid DDL statement. This mode can also create columns of complex data type. For example, array, map, or record and also in nested format.
- In both the modes, specify the name of the column and define the column with its properties - datatype, default value and whether it is nullable.
- Click **Add Column**.

DROP COLUMN

- Locate the Schema Explorer, and click the Refresh icon to reload the schema.
- Click on the target table to see the listed columns, Primary Keys, Indexes and Shard Keys.
- Locate the target-column which has to be dropped and right-click on it. Click **Drop Column.**
- A confirmation window appears, click **Ok** to confirm the drop action.

EXECUTE DDL

You can execute any table-specific DDL statements and table-independent operations like managing namespaces, regions, and roles using the Execute DDL option.

1. Click on the Execute icon (a triangle) in the Schema Explorer to open the **Execute DDL** window. Alternatively, you can also right-click the connection name and choose **Execute DDL**.
2. In the new window, enter the SQL statement which is a System Request (like creating or dropping region/role/namespace, etc). Click **Execute**. The result of the DDL is displayed in the lower portion of the window.

> **Note**
>
> You can also perform any other DDL operation like CREATE TABLE from this window.

Operations supported using Execute DDL:

- **Create a Namespace:** You can add one or more namespaces to your store, create tables within them, and grant permission for users to access namespaces and tables. Example:

  ```
  CREATE NAMESPACE ns1
  ```

- **Create a remote region:** You can create remote regions from each local region in a Multi-Region NoSQL Database. Example:

  ```
  CREATE REGION fra
  ```

- **Add a local region:** You can set a name to the local region in a Multi-Region NoSQL Database. Example:

  ```
  SET LOCAL REGION lnd
  ```

- **Drop a region:** You can remove a region. Example:

  ```
  DROP REGION fra
  ```

- **Create an on-premises Multi-Region table:** You can create an on-premises Multi-Region table on a data store and specify the list of regions that the table should span. You can also expand a Multi-Region table by adding more regions or contract a Multi-Region table by removing the table from any existing region.
  Example: Create a Multi-Region Table

  ```
  CREATE TABLE users(id INTEGER, name STRING, team STRING,
  PRIMARY KEY(id))IN REGIONS fra,lnd
  ```

  Example: Expand a Multi-Region Table

  ```
  CREATE REGION dub;
  ALTER TABLE users ADD REGIONS dub;
  ```

  Example: Contract a Multi-Region Table

  ```
  ALTER TABLE users DROP REGIONS dub;
  ```

> **Note**
>
> Before using the IntelliJ plugin to create/expand or contract a Multi-Region table, you should complete all the tasks for configuring a Multi-Region data store. This includes deploying the data store, configuring the XRegion service, starting the XRegion agents, and running the proxy. See Configuring Multi-Region Data Stores for more details.

Perform DML operations using IntelliJ

You can add data, modify existing data and query data from tables using IntelliJ plugin.

Insert data

- Locate the **Schema Explorer**, and click the **Refresh** icon to reload the schema.
- Right click on the table where a row needs to be inserted. Choose **Insert Row**.
- In the Insert Row panel, enter the details for inserting a new row. You can INSERT a new ROW in two modes:
  - Simple Input : You can use this mode to insert the new row without writing a DML statement. Here a form based row fields entry is loaded, where you can enter the value of every field in the row.
    * For binary data type, the string typed in should be a valid Base64 encoding of a binary value or select the file to upload in the desired column.

* For fixed binary data type, the string typed in should be a valid Base64 encoding of a binary value or upload the file of size defined during the creation of the particular column.

> **Note**
>
> The file format you upload for binary data type should only have `.bin` extension.

  - Advanced JSON Input : You can use this mode to insert a new row into the table by supplying a JSON Object containing the column name and its corresponding value as key-value pairs. The input can also be of complex data type i.e. array, map, record.

- Click **Insert Row**.

Modify Data - UPDATE ROW/DELETE ROW

- Locate the **Schema Explorer**, and click the **Refresh** icon to reload the schema.
- Right click on the table where a row needs to be inserted. Choose **Browse Table.**
- In the textbox on the left, enter the SQL statement to fetch data from your table. Click **Execute** to run the statement.
- To view individual cell data separately, click the table cell.
- To perform DML operations like Update and Delete Row, right-click on the particular row. Pick your option from the context-menu that appears.
  - Delete Row : A confirmation window appears, click **Ok** to delete the row.
  - Update Row : A separate HTML window opens displaying the column names and its corresponding values. You can enter new values for row data in two modes: **form-based row fields entry (Simple DDL input)** and **Supply row contents as a JSON object (Advanced DDL input)**. In the advanced DDL input mode, JSON data is presented as a tree structure to simplify viewing and updating.

    Select **Execute** to refresh and view the updated data.

> **Note**
>
> In any row, PRIMARY KEY and GENERATED ALWAYS AS IDENTITY columns cannot be updated.

Query tables

- Locate the **Schema Explorer**, and click the **Refresh** icon to reload the schema.
- Right click on the table and choose **Browse Table**.
- In the textbox on the left, enter the SELECT statement to fetch data from your table. When you start typing the query, you are prompted with the list of possible words to auto-complete the SQL statement. All SQL keywords and column names for the given table are provided in the prompt to auto complete the SQL statement.
- The SQL syntax is highlighted in every query which provides a better SQL writing experience.
- You can format the query to improve its readability. Select your query, right click and select **Prettify**. The query is formatted and lines are wrapped to enhance readability.

- Click **Execute** to run the query. The corresponding data is retrieved from the table. When you double click the data retrieved, the column data is opened in a new window. Any JSON data is displayed in a tree structure in the new window. Click '**+**' to expand or '**-**' to collapse the tree structure. Click **Ctrl ^F** in the new window to enable the search option, which allows you to search for any value in the JSON tree. Click the up or down arrow to move to the previous or next search occurrence, respectively.
- Right click on any row and click **Download JSON**. In the dialog box, navigate to the location where you want to save the file and click **Save**. Once the file is downloaded, a notification appears at the bottom-right of the screen. Click the link to open the downloaded file. The file opens in the browser.
  - In case of Binary data type, simply click **Download Binary Object** in the output.
- Click **Download Query Result**, to download all the data in the query result. In the dialog box, navigate to the location where you want to save the file and click **Save**. In case of multiple rows, a progress bar appears on the bottom-right of the screen to showing the number of rows downloaded in real time. Once the file is downloaded, a notification appears at the bottom-right of the screen. Click the link to open the downloaded file. The file opens in the browser.
- Click **Show Query Plan** to view the execution plan of the query.
- Click the **Previous Commands** drop-down, to view the top 20 recently executed SQL statements that had provided an output.

> **Note**
>
> The drop-down will only show SQL statements related to the table you are working on.

Schema Explorer

- In the **Schema Explorer** window, you can verify the full data type of a particular column. Locate the particular column and the data type is followed by the column name.

Authorization Requirements for IntelliJ Plugin

Learn about the authorization requirements needed in Oracle NoSQL Database to use IntelliJ Plugin.

The IntelliJ plugin allows you to access Oracle NoSQL Database secure store with the credentials specified in the connection parameters. The success of any DDL or DML operation from the IntelliJ plugin depends on the role-based authorization granted to the authenticated user. Oracle NoSQL Database data store enforces the authorization. If a user attempts an action beyond their privileges, the IntelliJ plugin returns the corresponding authorization error received from Oracle NoSQL Database.

Administrators can further restrict access by granting scoped privileges, such as limiting access to specific tables or namespaces. For more information on role-based authorization, see Configuring Authorization.

For example, consider inserting a row into a table. If you only have the READ_ANY privilege, insert fails with an error similar to the following:

```
Error executing Insert Row : PUT: Insufficient privilege: Insufficient access
rights granted on table, id: 14 name: MigrateId (5.4.18)
```

2

About Oracle NoSQL Database Visual Studio Code Extension

The Oracle NoSQL Database provides an extension for Microsoft Visual Studio Code which lets you connect to a running instance of Oracle NoSQL Database.

You can use Oracle NoSQL Database Visual Studio (VS) Code extension to:

- View the tables in a well-defined tree structure with Table Explorer.
- View information on columns, indexes, primary key(s), and shard key(s) for a table.
- View column data in a well-formatted JSON Structure.
- Create tables and child tables using form-based schema entry or supply DDL statements.
- Create JSON Collection tables using Advanced DDL Input mode and insert, update rows using Advanced JSON Input mode.
- Drop tables, columns, rows, or indexes.
- Add new columns and indexes using form-based entry or supply DDL statements.
- Add new rows using a form-based entry or use Advanced JSON Input mode.
- Run SELECT SQL queries on a table and view query results in tabular format.
- Run DDL statements to manage regions, namespaces, and roles.
- Run DML statements to update, insert, and delete data from a table.
- Calculate recommended cache and storage settings by using the JeCacheSizingTool.
- Download SELECT query results as a JSON file or as individual JSON files per row.
- Use Prettify SQL to improve readability.

Installing Oracle NoSQL Database Visual Studio Code Extension

You can install the Oracle NoSQL Database VS Code extension from the Visual Studio Marketplace for online installation.

Before you can install the Oracle NoSQL Database Visual Studio (VS) Code extension, you must install Visual Studio Code. You can download Visual Studio Code from here.

For installation:

1. In Visual Studio Code, click the **Extensions** icon in the left navigation.

Alternatively, you can open the **Extensions** view by pressing:

- (Windows and Linux) Control + Shift + X
- (macOS) Command + Shift + X.

2. Search Oracle NoSQL Database Connector in the extension marketplace.
3. Click Install on the Oracle NoSQL Database Connector extension

Connecting to Oracle NoSQL Database from Visual Studio Code

Oracle NoSQL Database Visual Studio (VS) Code extension allows you to connect to Oracle NoSQL Database by filling in the connection information in the specific fields.

1. In Visual Studio Code, click the **Oracle NoSQL DB** view in the **Activity Bar**.

2. Open the Oracle NoSQL DB **Show Connection Settings** page from the Command Palette or the **Oracle NoSQL DB** view in the **Activity Bar**.
   - Open from Command Palette
     a. Open the **Command Palette** by pressing:
        – (Windows and Linux) Control + Shift + P
        – (macOS) Command + Shift + P
     b. From the Command Palette, select **OracleNoSQL: Show Connections Settings**.

> **Tip**
>
> Enter `oraclenosql` in the Command Palette to display all of the Oracle NoSQL DB commands you can use.

- Open from Oracle NoSQL DB View
  a. Expand the **Table Explorer** pane in the left navigation if it's collapsed.
  b. Click **Add Connection** to open the Oracle NoSQL DB **Show Connection Settings** page.

3. In the **Connection Settings** page, click **Onprem** to connect to Oracle NoSQL Database.
4. Click **Add New Connection**. Enter the connection information, and click **Add**.

Table 2-1 Oracle NoSQL Database Connection Parameters

Field	Description
Connection Name:	A unique name, for the connection specification is mandatory from the plugin version 4.0.0. Updating the Connection Name field is recommended after upgrading the plugin from version 3.0.0 or lower. **Note** You can add multiple connections and the stored connection specifications are persistent.
Endpoint:	Service URL of the Oracle NoSQL Database Proxy. • `http://<proxy_host>:<proxy_http_port>` • `https://<proxy_host>:<proxy_https_port>` where, • `http` or `https` indicates the store security. For a secure KVStore, the proxy URL begins with `https`. • *proxy_host* is the host name of the machine to host the Oracle NoSQL Database Proxy service. For more information, see Configuring the Proxy.

Table 2-1 (Cont.) Oracle NoSQL Database Connection Parameters

Field	Description
Namespace:	Add the name of an existing namespace in your store. This is an optional field. The default namespace is `sysdefault`, if not provided. A namespace is a container that contains one or more tables. **Note** You cannot create a table in a namespace that does not exist. To create a namespace, see EXECUTE DDL.
Security:	Select SSL for secure KVStores. In case you are creating the connection to a non-secure KVStore, select None. The default value is `None`.
Username:	User name to connect to the secure store. This value is required only if you select SSL for the Security parameter.
Password:	Password to connect to the secure store. This value is required only if you select SSL for the Security parameter.
Certificate File:	Browse to the location of the SSL certificate file (for example, `certificate.pem`). See Using the Proxy in a secure data store.

Note

The connection processing has changed starting with version 4.0.0. If you are using an older version of the plugin, then only the most recently used connection will be migrated during the upgrade process to version 4.0.0. The plugin will automatically assign the name `onprem_old0` to the migrated connection. You can change the connection name using the Modify Connection option in the Connections Setting page.

5. The connection details are saved. To modify existing connections, select the connection from the **Connections** drop-down. Click **Modify Connections**. Make the desired changes in the Connection Parameters and click **Save**.
6. Click the Web Icon in the **TABLE EXPLORER** to view the list of all the connections (on-premises and Cloud) present in the plugin. The connections are displayed in the dropdown in the format: `Connection Name:Endpoint:Namespace` (if specified).

   Choose the connection from the list and select **Connect**. The Visual Studio Code plugin connects to an Oracle NoSQL Database and displays all its tables in the **Table Explorer** window.

> **Note**
>
> When you want to connect to a on-premise secure cluster using Visual Studio Code Extension, you need to do one of the following if you have self-signed certificates:
>
> - Add the self-signed certificate to the trusted root store in your Windows configuration (using Microsoft Management Console)).
> - Open Visual Studio Code. Go to File > Preferences > Settings > Application > Proxy. In the **Proxy Support** menu select **off**.

Creating applications using Oracle NoSQL Java SDK from Visual Studio Code

Learn to create applications from Visual Studio Code extension using the Oracle NoSQL Java SDK package.

After connecting to Oracle NoSQL Database, you can create and run applications using Oracle NoSQL Java SDK.

Perform the following steps:

Set up the Maven Project

1. Open Visual Studio Code.
2. Install the Java Extension Pack:
   a. Open the Extensions view by clicking the **Extensions** icon in the left sidebar.
   b. Search for *Java Extension Pack* in the Extensions Marketplace.
   c. Click the **Install** button to install the Java Extension Pack
3. On the **View** tab, select **Command Palette** and press Ctrl+Shift+P.
4. Search for *Java: New Project*.
5. Choose the project template as *Maven*.
6. Enter the project details:
   - Project name.
   - Project location (default is the current workspace folder).
   - Package name (For example, `com.example.myproject`).
7. Set the Java path in your environment variable.
8. Visual Studio Code creates the project directory structure and files.
9. Open the *pom.xml* file from the project directory and add the Oracle NoSQL Java SDK dependency as follows. Use the latest version. See Oracle NoSQL Java SDK

```
<dependency>
    <groupId>com.oracle.nosql</groupId>
    <artifactId>nosqldb-driver</artifactId>
    <version>5.4.22</version>
</dependency>
```

Create and run a sample application program

1. Create a sample application code as follows. Update your connection's end point and port in the *<proxy_host>* and *<proxy_https_port>* placeholders.

```
package org.oracle.nosqljavasdk;

import oracle.nosql.driver.NoSQLHandle;
import oracle.nosql.driver.NoSQLHandleConfig;
import oracle.nosql.driver.NoSQLHandleFactory;
import oracle.nosql.driver.kv.StoreAccessTokenProvider;
import oracle.nosql.driver.ops.GetRequest;
import oracle.nosql.driver.ops.GetResult;
import oracle.nosql.driver.ops.PutRequest;
import oracle.nosql.driver.ops.PutResult;
import oracle.nosql.driver.ops.TableRequest;
import oracle.nosql.driver.ops.TableResult;
import oracle.nosql.driver.values.MapValue;

public class JavaVSCode {
    /* Name of your table */
    final static String tableName = "TestSDK";
    static NoSQLHandle handle;
    public static void main(String[] args) throws Exception {
      String kvstore_endpoint ="http:<proxy_host>:<proxy_http_port>";
      handle = generateNoSQLHandle(kvstore_endpoint);
        try {
            createTable(handle);
            writeRows(handle);
            readRows(handle);

                           /* Uncomment this if you want to drop the table
*/
            //dropTable(handle);
        } catch (Exception e) {
            System.err.print(e);
        } finally {
            handle.close();
        }
    }

   /* Create a NoSQL handle to access the onPremise Oracle NoSQL database
*/
    private static NoSQLHandle generateNoSQLHandle(String
kvstore_endpoint) throws Exception {
      NoSQLHandleConfig config = new NoSQLHandleConfig(kvstore_endpoint);
      config.setAuthorizationProvider(new StoreAccessTokenProvider());
      /* If using a secure store, uncomment the line below and pass
username,
      *  password of the store to StoreAccessTokenProvider
      */
      /* config.setAuthorizationProvider(new
StoreAccessTokenProvider(username, password)); */
      NoSQLHandle handle = NoSQLHandleFactory.createNoSQLHandle(config);
      return handle;
   }

    /**
```

```
     * Create a simple table with an integer key
     * and a single string data field
     * and set your desired table capacity
     */
    private static void createTable(NoSQLHandle handle) throws Exception {
        String createTableDDL = "CREATE TABLE IF NOT EXISTS " +
            tableName + "(employeeid INTEGER, name STRING, " +
            "PRIMARY KEY(employeeid))";
        TableRequest treq = new TableRequest()
            .setStatement(createTableDDL);

        System.out.println("Creating table " + tableName);
        TableResult tres = handle.tableRequest(treq);

        /* The request is async,
         * so wait for the table to become active.
        */
        System.out.println("Waiting for "
            + tableName + " to become active");
        tres.waitForCompletion(handle, 60000, /* wait 60 sec */
            1000); /* delay ms for poll */
        System.out.println("Table " + tableName + " is active");
    }

    /**
     * Construct a row and add it to the table
     */
    private static void writeRows(NoSQLHandle handle) throws Exception {
        MapValue value =
            new MapValue().put("employeeid", 1).put("name", "Tracy");
        PutRequest putRequest =
            new PutRequest().setValue(value).setTableName(tableName);
        PutResult putResult = handle.put(putRequest);
        if (putResult.getVersion() != null) {
            System.out.println("Wrote " + value);
        } else {
            System.out.println("Put failed");
        }
    }

    /**
     * Set a key and read the row from the table
     */
    private static void readRows(NoSQLHandle handle) throws Exception {
        MapValue key = new MapValue().put("employeeid", 1);
        GetRequest getRequest =
            new GetRequest().setKey(key).setTableName(tableName);
        GetResult getRes = handle.get(getRequest);
        System.out.println("Read " + getRes.getValue());
    }

    /**
     * Drop the table and wait for the table to move to dropped state
     */
    private static void dropTable(NoSQLHandle handle) throws Exception {
        System.out.println("Dropping table " + tableName);
```

```
            TableRequest treq = new TableRequest()
                .setStatement("DROP TABLE IF EXISTS " + tableName);
            TableResult tres = handle.tableRequest(treq);
            System.out.println("Waiting for " + tableName + " to be dropped");
            tres.waitForCompletion(handle, 60000, /* wait 60 sec */
                1000); /* delay ms for poll */
            System.out.println("Table " + tableName + " has been dropped");
        }
    }
```

2. Run the Java program. You can press F5 or select the desired option from the **Run** menu.

> **Note**
>
> If you have network proxies, include them in the *settings.xml* file located in the maven directory.

3. You can view the output in the Visual Studio Code terminal.

The application code above creates the table `TestSDK`, adds a row to the table, and reads the row from the table.

Sample output:

```
----------------< com.oracle.nosql.example:NoSQLOperation >-----------------
Building NoSQLOperation 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.1:resources (default-resources) @ NoSQLOperation ---
Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
skip non existing resourceDirectory
C:\Users\ramya\javaVSCode\src\main\resources

--- compiler:3.8.0:compile (default-compile) @ HelloWorld ---
Changes detected - recompiling the module!
File encoding has not been set, using platform encoding UTF-8, i.e. build is
platform dependent!
Compiling 1 source file to C:\Users\ramya\javaVSCode\target\classes

--- exec:3.1.0:exec (default-cli) @ NoSQLOperation ---
Creating table TestSDK
Waiting for TestSDK to become active
Table TestSDK is active
Wrote {"name":"Tracy","employeeid":1}
Read {"employeeid":1,"name":"Tracy"}
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  8.013 s
Finished at: 2025-11-24T17:01:24+05:30
------------------------------------------------------------------------
```

You can also view the table from your Visual Studio Code **Table Explorer**. For details, see Managing Tables Using Visual Studio Code Extension.

Managing Tables Using Visual Studio Code Extension

After you connect to your deployment using Oracle NoSQL Database Visual Studio (VS) Code extension, use the **TABLE EXPLORER** located on the left navigation to:

- Explore your tables, columns, indexes, primary keys, and shard keys.
- Create new tables.
- Create child tables.
- Drop existing tables.
- Create Indexes.
- Drop Indexes.
- Add columns.
- Drop columns.
- Insert data into table.
- Run SELECT SQL queries.
- Refresh tables.
- View table DDL.

Explore tables, columns, indexes and keys

When you expand an active connection, Oracle NoSQL Database VS Code shows the tables in that deployment.

- Click the table name to view its columns, indexes, primary key(s), and shard key(s). The column name displays along with its data type.
- You can refresh the schema or table at any time to re-query your deployment and populate Oracle NoSQL Database with the most up-to-date data.
- In the **TABLE EXPLORER**, locate the connection and click the Refresh icon to reload the schema. Alternatively, you can right-click the connection and select **Refresh Schema**.
- In the **TABLE EXPLORER**, locate the table name and click the Refresh icon to reload the table. Alternatively, you can right-click the table name and select **Refresh Table**.
- In the **TABLE EXPLORER** you can verify the data type of a particular column. Locate the particular column and the data type is followed by the column name.
- Locate the Table Explorer and click the **JeCacheSizingTool** icon. This is an advanced tool that aides in sizing an On-Premises database. It offers suggested values for cache size, storage size and number of shards based on the inputs.
  - In the JeCacheSizingTool panel, enter the values in bytes for Primary Key Size, Row Size, and Index (this is optional), along with the number of rows. Click **Add Table** to add values for another table.
  - Click **CALCULATE** to view the output values. To start a new calculation, click **New Calculation**.

Perform DDL operations using Visual Studio Code

You can use Visual Studio Code to perform DDL operations.

Some of the DDL operations that can be performed from inside the Visual Studio Code plugin are:

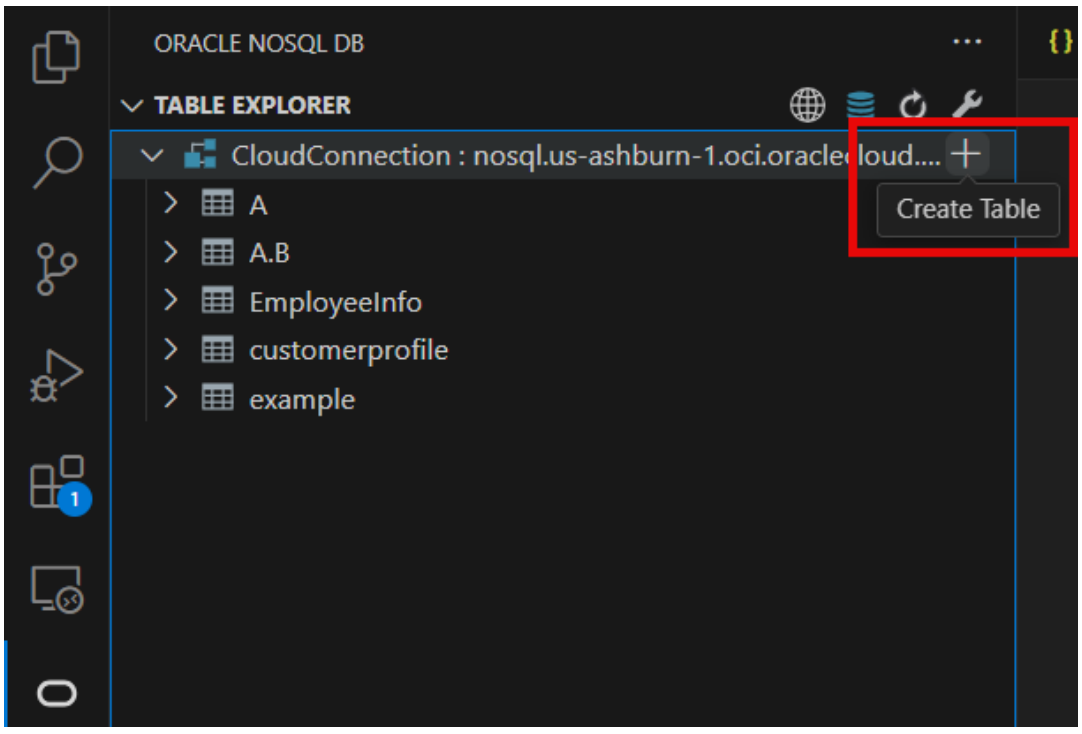
- CREATE TABLE
- CREATE CHILD TABLE
- DROP TABLE
- CREATE INDEX
- DROP INDEX
- ADD COLUMN
- DROP COLUMN
- EXECUTE DDL

CREATE TABLE

You can create the Oracle NoSQL Database table in two modes:

- **Simple DDL Input**: You can use this mode to create the Oracle NoSQL Database table declaratively, that is, without writing a DDL statement.
- **Advanced DDL Input**: You can use this mode to create the Oracle NoSQL Database table and JSON collection table using a DDL statement.

1. Hover over the Oracle NoSQL Database connection to add the new table.
2. Click the **Plus** icon that appears or right-click on the database connection name and click **Create Table**



3. In the **Create Table** page, select **Simple DDL Input**.

Table 2-2 Create an Oracle NoSQL Database Table

Field	Description
Table Name:	Specify a unique table name.
Column Name	Specify a column name for the primary key in your table.
Column Type	Select the data type for your primary key column.
Set as Shard Key	Select this option to set this primary key column as shard key. Shard key is to distribute data across the Oracle NoSQL Database cluster for increased efficiency, and to position records that share the shard key locally for easy reference and access. Records that share the shard key are stored in the same physical location and can be accessed atomically and efficiently.
Remove	Click this button to delete an existing column.
+ Add Primary Key Column	Click this button to add more columns while creating a composite (multi-column) primary key.
Column Name	Specify the column name.
Column Type	Select the data type for your column.
Default Value	(optional) Specify a default value for the column. **Note** Default values can not be specified for binary and JSON data type columns.
Not Null	Select this option to specify that a column must always have a value.
Remove	Click this button to delete an existing column.
+ Add Column	Click this button to add more columns.
Unit	Select the unit (**Days** or **Hours**) to use for TTL value for the rows in the table.

Table 2-2 (Cont.) Create an Oracle NoSQL Database Table

Field	Description
Value	Specify expiration duration for the rows in the table. After the number of days or hours, the rows expire automatically, and are no longer available. The default value is zero, indicating no expiration time. **Note** Updating Table Time to Live (TTL) does not change the TTL value of any existing data in the table. The new TTL value applies *only* to those rows that are added to the table after this value is modified and to the rows for which no overriding row-specific value has been supplied.

4. Click **Create**.
5. You have an option to the view the DDL statement before and after creating a table.
   - Before creating the table - Click **Show DDL** in the Create Table screen to view the DDL statement formed based on the values entered in the fields in the **Simple DDL input** mode. This DDL statement gets executed when you click **Create**.
   - After creating the table - Right click on the existing table. Choose **View Table DDL**.

CREATE CHILD TABLE

To create a child table, right click on the desired table and choose **Create Child Table**. You can create a child table for an already existing Oracle NoSQL Database table in one of the two modes:

- **Simple DDL Input**: You can use this mode to create a child table by simply entering a table name along with other required details.
- **Advanced DDL Input**: You can use this mode to create a child table using a DDL statement.

Click **Create** to create a child table.
For more details on child tables, see Table Hierarchies in *Developer's Guide*.

DROP TABLE

1. Right-click the target table.
2. Click **Drop Table**.
3. Click **Yes** to drop the table.

CREATE INDEX

- Locate the Table Explorer, and click the Refresh Schema icon to reload the schema.
- Right click on the table where index need to be created. Choose **Create Index**.
- In the Create Index panel, you have an option to create index in two modes:

  - **Simple Input**: Specify the name of the index and the columns to be part of the index. If the column type is JSON, you see an additional field called "JSON Path to Index Field". Enter the path to the location of the JSON field, and choose the data type for it.
  - **Using Advance DDL**: Enter a valid DDL statement to create an index on any column(s). It can also include complex data type i.e. array, map, and record.
- Click **Add Index**.
- You have an option to the view the Index DDL statement. Right click on the existing index. Choose **View Index DDL**. Click **OK** to close the dialog box.

DROP INDEX

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Click on the table where the index needs to be removed. The list of indexes are displayed below the column names.
- Right click on the index to be dropped. Click **Drop Index**.
- A confirmation window appears, click **Ok** to confirm the drop action.

ADD COLUMN

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Right click on the table where column needs to be added. Click **Add columns**.
- In the Add Column(s) Panel, you have an option to add column in two modes:
  - **Simple DDL Input**: Specify the name of the column and define the column with its properties - datatype, default value and whether it is nullable. In case of binary or fixed binary select the data type as `Binary`. For fixed binary enter the size of the file in the `Size` field and keep the field null in case of binary data type.
  - **Advanced DDL Input**: You can use this mode to add new columns into the table by supplying a valid DDL statement, as well as, create columns with complex data type (e.g. array, map, or record and also in nested format).
- Click **Add New Columns**.

DROP COLUMN

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Expand the table where column needs to be removed.
- Right click the column to be removed and choose **Drop Column**.
- A confirmation window appears, click **Ok** to confirm the drop action.

EXECUTE DDL

This feature allows you to execute DDL operations, for example, managing regions, namespaces, and roles.

To execute the DDL commands, follow these steps:

1. Click the triangle icon in the Table Explorer to open the Execute DDL panel or right-click on the Connection Name and click **Execute DDL**.
2. In the DDL text box enter the SQL DDL statement you would like to perform. Click **Execute**. You can view the output in the below textbox named *Result*.

The Execute DDL supports the following operations:

1. **Create namespace:** You can create a new namespace by using `CREATE NAMESPACE` statement and create tables within them.

   ```
   CREATE NAMESPACE ns1
   ```

2. **Create remote region:** You can create remote regions for a Multi-Region table.

   ```
   CREATE REGION fra
   ```

3. **Add local region:** You can set a name to the local region for a Multi-Region table.

   ```
   SET LOCAL REGION lnd
   ```

4. **Drop region:** You can remove a region using the `DROP REGION` statement.

   ```
   DROP REGION fra
   ```

5. **Create an on-premises Multi-Region table:** You can create an on-premises Multi-Region table on a data store and specify the list of regions that the table should span. You can also expand a Multi-Region table by adding more regions or contract a Multi-Region table by removing the table from any existing region.
   - Create a Multi-Region table

     ```
     CREATE TABLE users(id INTEGER, name STRING, team STRING,PRIMARY
     KEY(id))IN REGIONS fra,lnd
     ```

   - Expand a Multi-Region table

     ```
     CREATE REGION dub
     ALTER TABLE users ADD REGIONS dub
     ```

   - Contract a Multi-Region table

     ```
     ALTER TABLE users DROP REGIONS dub
     ```

> **Note**
>
> See Configuring Multi-Region data stores for details on deploying and configuring a data store to support Multi-Region tables.

Perform DML operations using Visual Studio Code

You can add data, modify existing data and query data from tables usingVisual Studio Code plugin.

Insert Data

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Right click on the table where a row needs to be inserted. Choose **Insert Row**.
- In the Insert Row panel, enter the details for inserting a new row. You can INSERT a new ROW in two modes:

- Simple Input : You can use this mode to insert the new row without writing a DML statement. Here a form based row fields entry is loaded, where you can enter the value of every field in the row.
  * For binary data type, the string typed in should be a valid Base64 encoding of a binary value or select the file to upload in the desired column.
  * For fixed binary data type, the string typed in should be a valid Base64 encoding of a binary value or upload the file according to the size specified for the column.

> **Note**
>
> The file format you upload for binary data type should have the .bin extension.

- Advanced JSON Input : You can use this mode to insert a new row into the table by supplying a JSON Object containing the column name and its corresponding value as key-value pairs. The input can also be a complex data type i.e. array, map, record.

> **Note**
>
> You can add a row in JSON collection table only through Advanced JSON Input mode. For more information on how to create JSON collection table, see Using JSON Collection Tables

- Click **Insert Row**.

Modify Data - UPDATE ROW/DELETE ROW:

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Click on the table where data needs to be modified.
- In the textbox on the right under SQL>, enter the SQL statement to fetch data from your table. Click **>** to run the query.
- To view individual cell data separately, click the table cell.
- To perform DML operations like Update and Delete Row, right-click on the particular row. Pick your option from the context-menu that appears.
  - Delete Row : A confirmation window appears, click **Ok** to delete the row.
  - Update Row : You can update in the separate HTML panel that opens below the listed rows, displaying the column names along with their current value in a form-based entry. The other option to edit is to provide the input as a key-value. You can choose either of the two methods and supply new values.

> **Note**
>
> * For updating the row data in a JSON collection table, you can modify the data only using **Advanced JSON Input** mode.
> * In any row, PRIMARY KEY and GENERATED ALWAYS AS IDENTITY columns cannot be updated.

Executing SQL Queries for a Table

- Locate the Table Explorer, and click the Refresh Schema to reload the schema.
- Right click on the table and choose **Browse Table**.
- In the textbox on the right under SQL>, enter the SELECT statement to fetch data from your table.
- While writing a query, the system highlights the syntax and displays a list of possible completions at the insertion point. You can select the desired option to autocomplete the code based on the context.
- After writing your query, right click anywhere in the textbox and select **Prettify SQL** to format the code, improving its spacing, line wrapping, and increasing its overall readability.
- Click **>** to run the query. The corresponding data is retrieved from the table.
- Click on any row entry with a JSON data type column to open a dialog box displaying the JSON fields in an interactive tree structure. Click '**+**' to expand or '**-**' to collapse the structure.
  - In addition to the dialog box, a search field appears at the top-right corner. This allows you to search for any value in the JSON tree in real time, even within collapsed tree structures. Use the up or down arrow to navigate to the previous or next search result, respectively.
- Right click on any row and click **Download JSON**. The single row gets downloaded into a JSON file. In the dialog box, navigate to the location where you want to save the file and click **Save**.
- Click **Download Query Result** to save the complete result of the SELECT statement as a JSON file. In the dialog box, navigate to the location where you want to save the file and click **Save**.
- Click **Fetch All Records** to retrieve all data from the table.
- Click **Show Query Plan** to view the execution plan of the query.
- Click the **Previous Commands** dropdown, to view the recently executed SQL statements that had provided an output.

> **Note**
>
> The dropdown will only show SQL statements related to the table you are dealing with.

Removing a Connection

Oracle NoSQL Database Connector provides two methods to remove a connection from Visual Studio (VS) Code.

You can:

- Remove a connection with the Command Palette, or
- Remove a connection from the Oracle NoSQL DB view in the Activity Bar.

> **Note**
>
> To remove a connection permanently from the plugin, select the connection from the **Connections** drop-down. Click **Delete Connection**. You get a confirmation to delete the plugin, click **Yes** to remove the connection from the plugin. Removing a connection from Visual Studio Code deletes the persisted connection details from the current workspace.

Remove Connection from Oracle NoSQL DB View

1. Expand the **TABLE EXPLORER** pane in the left navigation if it's collapsed.
2. Right-click the connection you want to remove, then click **Remove Connection**.

Remove Connection with Command Palette

1. Open the **Command Palette** by pressing:
   - (Windows and Linux) Control + Shift + P
   - (macOS) Command + Shift + P
2. From the Command Palette, select **OracleNoSQL: Remove Connection**.

> **Tip**
>
> Enter `oraclenosql` in the Command Palette to display all of the Oracle NoSQL DB commands you can use.

Authorization Requirements for Visual Studio Code

Learn about the authorization requirements needed in Oracle NoSQL Database to use Visual Studio Code.

The Visual Studio Code extension allows you to access Oracle NoSQL Database secure store with the credentials specified in the connection parameters. The success of any DDL or DML operation from the IntelliJ plugin depends on the role-based authorization granted to the authenticated user. Oracle NoSQL Database data store enforces the authorization. If a user attempts an action beyond their privileges, the Visual Studio Code returns the corresponding authorization error received from Oracle NoSQL Database.

Administrators can further restrict access by granting scoped privileges, such as limiting access to specific tables or namespaces. For more information on role-based authorization, see Configuring Authorization.

For example, consider inserting a row into a table. If you only have the READ_ANY privilege, insert fails with an error similar to the following:

```
Error Inserting the row : NoSQLError: [INSUFFICIENT_PERMISSION] PUT:
Insufficient privilege: Insufficient access rights granted on table, id: 14
name: MigrateId
```

Glossary

Index