HCL OneDB 2.0.1

Getting Started

Contents

Chapter 1. Getting Started

This document provides a brief tutorial to help you get familiar with HCL OneDB™ products.

Introduction to HCL OneDB™

HCL OneDB™ is a cloud native, high performance scalable database server that manages relational, object-relational, and dimensional databases.

Its small footprint, scalability, and self-managing capabilities are suited to manage your data from small embedded data-management solutions to enterprise scale solutions.

The HCL OneDB™ database server runs in container and has support for AIX, Linux and Windows.

HCL OneDB™ set of the products include:

- HCL OneDB™ Server
- HCL OneDB™ Client SDK
- HCL OneDB™ JDBC
- HCL OneDB™ APIs
- HCL OneDB™ Explore

What's new in HCL OneDB™ 2.0.1

This section includes information about the new, enhanced features in HCL OneDB™ 2.0.1.

What's new in HCL OneDB™ 2.0.1.3

- Flexnet is deprecated from HCL OneDB 2.0.1.3 onwards.
- OneDB Explore Swagger (API documentation) will be deprecated from the next release.

What's new in HCL OneDB™ 2.0.1.2

- Enhanced integration experience for HCL Commerce and HCL Unica customers using HCL OneDB.
- New Capabilities added in Helm Chart for better deployment experience of HCL OneDB in cloud native environment. For more information, see Deploying HCL OneDB using Helm charts.
- Apache Log4j vulnerability issue is addressed in this release as the Log4j library used is upgraded to v2.17.1.

What's new in HCL OneDB™ 2.0.1.1

Apache Log4j vulnerability (December 2021) issue is addressed in this release as the Log4j library used is upgraded to v2.16.

What's new in HCL OneDB™ 2.0.1.0

No new features have been added to HCL OneDB™ 2.0.1.0 release.

HCL OneDB Editions

HCL OneDB™ Server is available in two editions:

- HCL OneDB™ Enterprise Server
- HCL OneDB™ Standard Server

Both editions include the same features; however, OneDB Standard Server places limits on the number of CPU Virtual Processors (Cores / CPUVPs) allowing a maximum of twenty four (24) per instance.

Note: There is a single edition of HCL OneDB Client SDK and HCL OneDB JDBC which will work with both Enterprise and Standard editions of the Server.

Installing HCL OneDB™

About this task

You can install the HCL OneDB™ server by running the interactive installation application.

You can include options to the installation command if you run the command from the command line. For example, you can change the name or location of the installation log file.

Prerequisites: A Compatible JRE must be available in your system PATH.

Note: For information about system requirements, see System requirements for HCL OneDB database server.

Downloading the product

Download the HCL OneDB™ software for your specific platform from the HCL License and Delivery portal

AIX, Linux

1. If necessary, extract the product files. For example, run the **tar** command:

   tar xvf *filename*

   Where *filename* is the name of the product tar file.
2. Run the following command:

   onedb_install

a. **If you run the install as root this will install a root-based installation**

   create a user and group informix used to configure HCL OneDB™ server .

   HCL OneDB™ server processes will run with root privileges.

b. **If you run the install as any other, then this user will be used to**

   configure HCL OneDB™ server.

   HCL OneDB™ server processes will run with this users privileges.

Windows

1. Double-click the onedb_install.exe file.
   This will run the installer as an administrator.
2. Select the setup method, for example, Typical, and follow the instructions in the installation application.
3. Complete the installation and exit the installation application.
4. Log files are created during the installation process. You can use the log files to troubleshoot installation errors.
5. If you created a database server during installation, the database server is configured, initialized, started, and ready for client connections.
   An environment file will be created in the installation folder.
   onedb.ksh onedb.csh or onedb.cmd (name of the file depends on the server name either chosen or configured at install time).
6. To see the connectivity information, look at the ${ONEDB_ SQLHOSTS} file or %ONEDB_ SQLHOSTS% file for Windows.
7. To see the configuration information, look at the onconfig file ${ONEDB_HOME}/etc/${ONCONFIG} or the %ONEDB_HOME%\etc\%ONCONFIG% (Windows).
8. You can customize the database server properties by setting configuration parameters in the onconfig file, setting environment variables in your environment, and adding connectivity information to the sqlhosts file.
9. If you did not create a database server during installation, you must configure and initialize the database server.

Migrating to HCL OneDB™ 2.0.0.0

About this task
This topic describes the steps to migrate to HCL OneDB™ 2.0.0.0 from HCL Informix 14.10 or previous versions of HCL OneDB™.

Before you start the migration procedure:

- Make sure that you have enough available space for the new server, your data, and any other network and data tools that you use.
- Take a backup of the source (HCL Informix or previous versions of HCL OneDB).

1. Use onmode -kuy to bring down the source database server.
2. Install HCL OneDB 2.0.0.0. See Installing HCL OneDB on page 4.
3. Use the same **onconfig** and **sqlhost** files from the source database server.
4. Set environment variables for OneDB 2.0.0.0. See Setting environment variables.
    - Set ONEDB_HOME to OneDB 2.0.0.0 directory.
    - Set PATH as PATH ${ONEDB_HOME}/bin:${ONEDB_HOME}/extend/krakatoa/jre/bin:${PATH}.
    - Set ONEDB_SERVER with the same value as that of INFORMIXSERVER in the source database server.
    - Set ONEDB_SQLHOST with the same value as that of INFORMIXSQLHOST value in the source database server.
    - Set ONCONFIG with the same value as that of ONCONFIG value in the source database server.
5. Use oninit -vy to start OneDB 2.0.0.0 database server.

Naming Conversion for HCL OneDB™

With version HCL OneDB™ 2.0.0.0, naming of some environment variables, configuration parameters, file and path names are changed.

Note: You have to use the new names when installing and using the HCL OneDB™ 2.0.0.0 database server as well as HCL OneDB™ 2.0.0.0 database client products.

Table 1. UNIX/Linux environment variables and corresponding Windows registry entries:

Informix/OneDB Versions prior to 2.0.0.0	OneDB 2.0.0.0
INFORMIXASFDBG	ONEDB_ASFDBG
INFORMIXASFDBG	ONEDB_ASFDBG
INFORMIXBROWSER	ONEDB_BROWSER
INFORMIXCMCONUNITNAME	ONEDB_CMCONUNITNAME
INFORMIXCMNAME	ONEDB_CMNAME
INFORMIXCOB	ONEDB_COB
INFORMIXCOBDIR	ONEDB_COBDIR
INFORMIXCOBSTORE	ONEDB_COBSTORE
INFORMIXCOBTYPE	ONEDB_COBTYPE
INFORMIXCONCSMCFG	ONEDB_CONCSMCFG
INFORMIXCONRETRY	ONEDB_CONNECT_RETRIES
INFORMIXCONTIME	ONEDB_CONNECT_TIMEOUT
INFORMIXCPPMAP	ONEDB_CPPMAP
INFORMIXC	ONEDB_C

Table 1. UNIX/Linux environment variables and corresponding Windows registry entries: (continued)

Informix/OneDB Versions prior to 2.0.0.0	OneDB 2.0.0.0
INFORMIXCP	ONEDB_CP
INFORMIXCPP	ONEDB_CPP
INFORMIXFAILOVERTIME	ONEDB_FAILOVERTIME
INFORMIXFORTRAN	ONEDB_FORTRAN
INFORMIXFORTYPE	ONEDB_FORTYPE
INFORMIXKEYTAB	ONEDB_KEYTAB
INFORMIXMEMSIZE	ONEDB_MEMSIZE
INFORMIX_NOUNLOAD	ONEDB_NOUNLOAD
INFORMIXNSF	ONEDB_NSF
INFORMIX_NSF_SPINCOUNT	ONEDB_NSF_SPINCOUNT
INFORMIXONGUARDNAME	ONEDB_ONGUARDNAME
INFORMIXOPCACHE	ONEDB_OPCACHE
INFORMIXOPTDB	ONEDB_OPTDB
INFORMIXDIR	ONEDB_HOME
INFORMIXQATERM	ONEDB_QATERM
INFORMIXSERVER	ONEDB_SERVER
INFORMIXSERVERNUM	ONEDB_SERVERNUM
INFORMIXSHMBASE	ONEDB_SHMBASE
INFORMIX_SLOW_CLEAR	ONEDB_SLOW_CLEAR
INFORMIXSQLHOSTS	ONEDB_SQLHOSTS
INFORMIXSTACKSIZE	ONEDB_STACKSIZE
INFORMIXTERM	ONEDB_TERM
INFORMIXTEST	ONEDB_TEST
INFORMIXUNALIGNED	ONEDB_UNALIGNED

Table 2. Configuration parameters

Informix	OneDB
INFORMIXCONTIME	CONNECT_TIMEOUT

Table 2. Configuration parameters (continued)

Informix	OneDB
INFORMIXCONRETRY	CONNECT_RETRIES

Table 3. SQL "SET ENVIRONMENT" session parameter names

Informix	OneDB
INFORMIXCONTIME	CONNECT_TIMEOUT
INFORMIXCONRETRY	CONNECT_RETRIES

Note: INFORMIXCONTIME and INFORMIXCONRETRY are still supported for backward compatibility

Table 4. File and path names

Informix	OneDB
$INFORMIXDIR/etc/informixdir-is-insecure	$ONEDB_HOME/etc/installdir-is-insecure (script)
$INFORMIXDIR/etc/make-informixdir-secure	$ONEDB_HOME/etc/fix-installdir-security (script)
$INFORMIXDIR/extend/krakatoa/informix.policy[.std]	$ONEDB_HOME/extend/krakatoa/onedb.policy[.std]
$INFORMIXDIR/etc/informix.rc	$ONEDB_HOME/etc/onedb.rc
~/.informix	~/.onedb
/etc/informix/...	/etc/onedb/...
/usr/informix/...	/usr/onedb/...

Table 5. JDBC connection URL string

Informix	OneDB
jdbc:informix-sqli://...	jdbc:onedb://...

HCL OneDB™ Explore

HCL OneDB™ Explore is a modern web console for visualizing, monitoring, and managing your HCL OneDB™ server instances. It is purpose built for ease-of-use, scaling out, and optimizing DevOps needs. It provides critical performance management capabilities, monitoring how key performance metrics are changing over time and tracking how efficiently HCL OneDB™ is running your workload even when you've stepped away from your screen. Its monitoring system feeds directly into a customizable alerting system so you can be immediately alerted via email, Twilio, or PagerDuty whenever an issue occurs on one of your HCL OneDB™ database server instances. HCL OneDB™ Explore is designed to be scalable to efficiently manage and monitor as many HCL OneDB™ database server instances as you need. Moreover, it's a tool that can be shared by the DBAs, the app developers, the ops engineers, and management and accessed from any desktop, laptop, or mobile device.

HCL OneDB™ Explore is the centralized hub for graphical monitoring, alerting, and administration of your HCL OneDB™ database servers.

HCL OneDB™ Explore Server

The OneDB Explore server is a Java 8 based Jetty web server. The server is the heart of HCL OneDB™ Explore. It manages the monitoring profiles for all instances, communicates with agents, handles all alerting activities, hosts the web UI, and provides the REST services that the web UI depends on.

HCL OneDB™ Explore Agent

The HCL OneDB™ Explore agent is a lightweight Java 8 based monitoring agent. It should be installed on the host machine for each HCL OneDB™ database server that you want monitored by HCL OneDB™ Explore. The agent runs alongside the database server, gathering database statistics through JDBC connections to sysmaster. The HCL OneDB™ Explore agent only needs read access to the database server. By running the agent directly on the HCL OneDB™ server host machine, the agent is also able to gather and monitor OS statistics which can be just as critical in evaluating and tuning the HCL OneDB™ database server performance metrics.

Getting started with HCL OneDB™ Explore

About this task

This topic provides a brief tutorial to help you get started with HCL OneDB™ Explore.

Prerequisites

The following table lists the software prerequisites for HCL OneDB™ Explore:

Table 6. Prerequisites for HCL OneDB™ Explore

Software	Required Version
HCL OneDB™ Database Server	1.0
Java	1.8

Starting the HCL OneDB™ Explore Server

This section walks you through the easiest way to start the HCL OneDB™ Explore server.

For more advanced topics, like configuring logging or starting HCL OneDB™ Explore Server from the command line, see Starting the OneDB Explore Server .

1. Create a HCL OneDB™ Explore Server properties file.

```
# required initial password for the admin user
initialAdminPassword=myPassword
# optionally, uncomment these properties to have OneDB Explore encrypt its
    internal H2 database
#h2.encrypt.enable=true
```

```
#h2.encrypt.password=password
```

For more information see, HCL OneDB™ Explore Server Configuration.

2. Locate the start/stop script in the `$ONEDB_HOME/explore` directory and run the following command to start the server.
   OneDBExplore startServer=serverName where **serverName** is the name of your process for HCL OneDB™ Explore Server.
3. In a web browser, go to **http://localhost:8080** and login with your admin credentials.
4. From the main dashboard, click create server and/or create groups to add connection information to your OneDB servers.

Starting the HCL OneDB™ Explore Server

About this task

This section walks you through the easiest way to start the HCL OneDB™ Explore server.

For more advanced topics, like configuring logging or starting HCL OneDB™ Explore Server from the command line, see Starting the HCL OneDB™ Explore Server .

1. Create a HCL OneDB™ Explore Server properties file.

```
# required initial password for the admin user
initialAdminPassword=myPassword
# optionally, uncomment these properties to have OneDB Explore encrypt its
    internal H2 database
#h2.encrypt.enable=true
#h2.encrypt.password=password
```

   For more information see, HCL OneDB™ Explore Server Configuration.
2. Locate the start/stop script in the `$ONEDB_HOME/explore` directory and run the following command to start the server.
   OneDBExplore startServer=serverName where **serverName** is the name of your process for HCL OneDB™ Explore Server.
3. In a web browser, go to **http://localhost:8080** and login with your admin credentials.
4. From the main dashboard, click create server and/or create groups to add connection information to your HCL OneDB™ servers.

Starting the HCL OneDB™ Explore Agent

About this task

This section walks you through two ways to start the HCL OneDB™ Explore agent.

For more advanced topics, like configuring logging or starting HCL OneDB™ Explore Agent from the command line, see Getting Started section in the HCL OneDB™ Explore Agent documentation.

Prerequisites

1. Navigate to the HCL OneDB™ database server's **Setup > Agent** page in UI and click on select button to choose repository server. The The repository server is the database server that will store the monitoring data collected by the agent is the database server that will store the monitoring data collected by the agent.
2. Choose any repository server from popup of which you want to choose the repository database.
3. Now select any database from the **Select Database** dropdown which you want to make it as a repository database then click on save button. The repository database is where the monitored data will be stored.

Deploying and starting the HCL OneDB™ Explore agent automatically from the UI

About this task

Prerequisites

1. SSH must be installed on the database server's host machine.
2. The `onedb-explore-agent.jar` file and the `onedb-explore-server.jar` file must be located in the same directory.

Steps :

1. From the server's **Setup > Agent** page, you can also try to deploy the agent with default configuration by clicking the Deploy Agent button. If the automatic deployment is successful, you are done. Otherwise, proceed to the next step.
2. To configure agent deployment, click on the gear icon and provide the following information:
    - Username: remote user which will own and run the agent.
    - Password or Identity File/Passphrase: remote user's passphrase or remote user's identity file (e.g. private key) and its optional passphrase.
    - Remote directory: Directory to deploy the agent files to. This directory will be created if it does not exist.
3. If SSL is enabled, keystore configuration may be required as well. If keystore configuration is not provided, OneDB Explore will try to generate and deploy a keystore for you.
    - Keystore file: Location of an existing keystore on the remote machine (can be relative to the remote directory)
    - Keystore password: Password used to access the existing keystore
    - Keystore type: Existing keystore's type. Default is JKS
4. Click **Deploy Agent**
5. Once the agent is ready, use the HCL OneDB™ Explore UI in your web browser to configure the monitoring profile and alerts for this server.

Starting the OneDB Explore Agent with Start/Stop Script

About this task

This is an alternative step to the previous section to start the HCL OneDB™ Explore Agent.

1. Create a HCL OneDB™ Explore Agent properties file.

```
Sample properties file:
# host and port of the OneDB Explore server
server.host=localhost
server.port=8080
# The id of the OneDB database server as defined in OneDB Explore
onedbServer.id=1
```

Where onedbServer.id is the id of the HCL OneDB™ database server in HCL OneDB™ Explore. You can find the server's id on the server's Setup page in the OneDB Explore UI. For more information, see HCL OneDB™ Explore Agent Configuration .

2. Locate the start/stop script in the **$ONEDB_HOME/explore** directory and run the following command to start the server. **OneDBExplore startAgent=agentName** where **agentName** is the name of your process for HCL OneDB™ Explore Agent.
3. At this point the agent is ready and running. Use the HCL OneDB™ Explore UI in your web browser to configure the monitoring profile and alerts for this server.

Stopping the HCL OneDB™ Explore Server and Agent

1. To stop the HCL OneDB™ Explore Server, locate the start/stop script in the `$ONEDB_HOME/explore` directory and run the following command to stop the server. **OneDBExplore stop=serverName** where **serverName** is the name for your process for HCL OneDB™ Explore Server which you have used to start the OneDB Explore Server.
2. To stop the HCL OneDB™ Explore Agent, locate the server's **Setup > Agent page**. If the agent is online, you can see the "Shutdown agent" button in UI. To stop the agent you can click on this button and it will stop the agent running for that server.
3. An alternative way to stop the HCL OneDB™ Agent is to use the start/stop script. You can only use this step to stop the agent if you have started the agent by using the start/stop script. To stop the agent, locate the start/stop script in the `$ONEDB_HOME/explore` directory and run the following command to stop the agent **OneDBExplore stop=agentName** where **agentName** is the name for your process for HCL OneDB™ Explore Agent which you have used to start the agent.

Manging Database using HCL OneDB™ Explore

You can use OneDB Explore to manage the database.

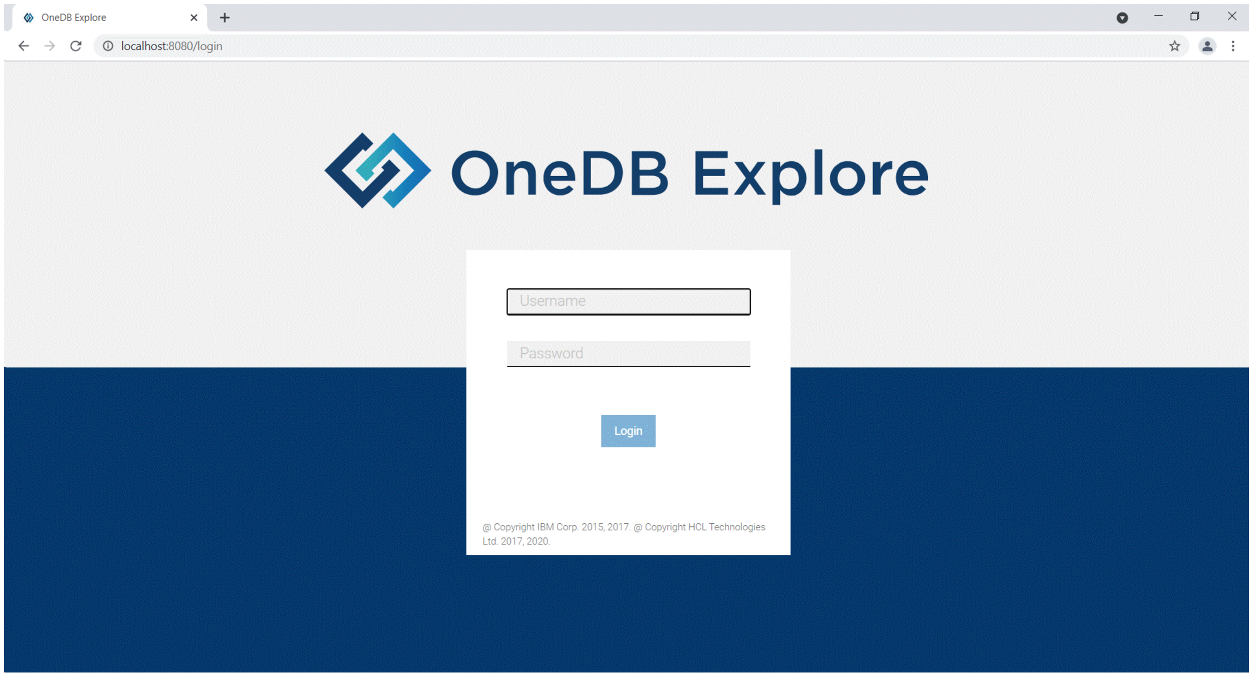
This section provides a detailed explanation on creating a new database, new table, adding a record and viewing it using OneDB Explore.

Creating an OneDB Database using OneDB Explore

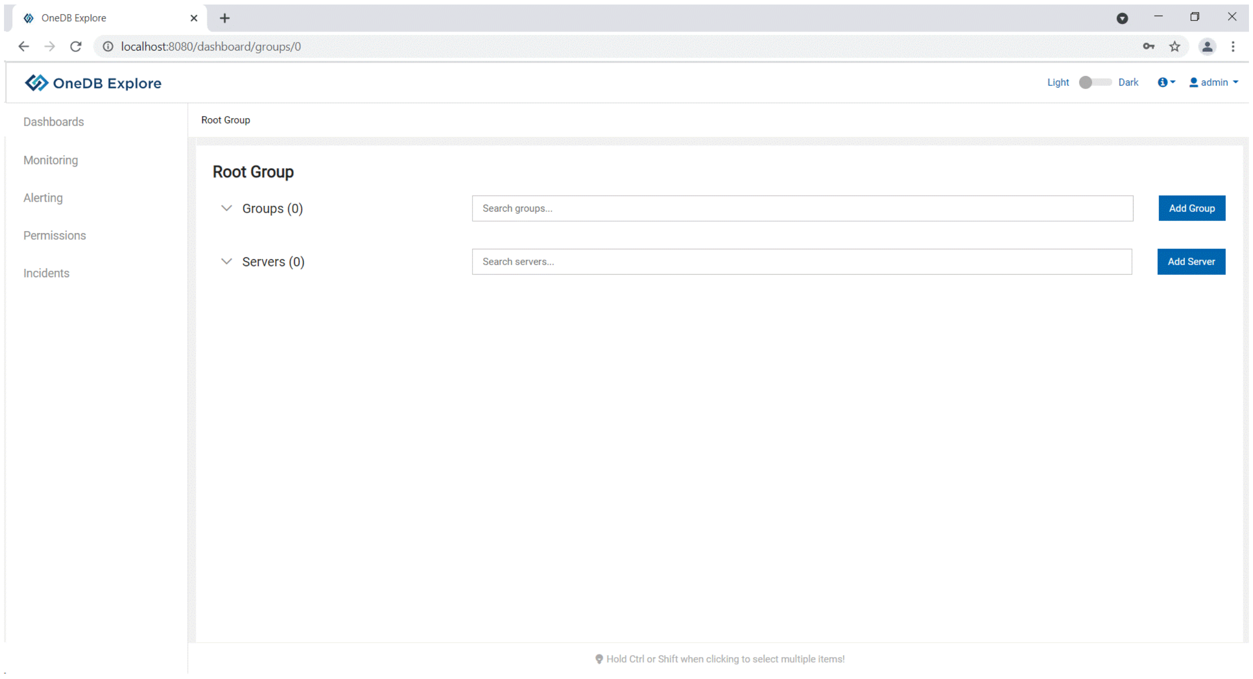
About this task

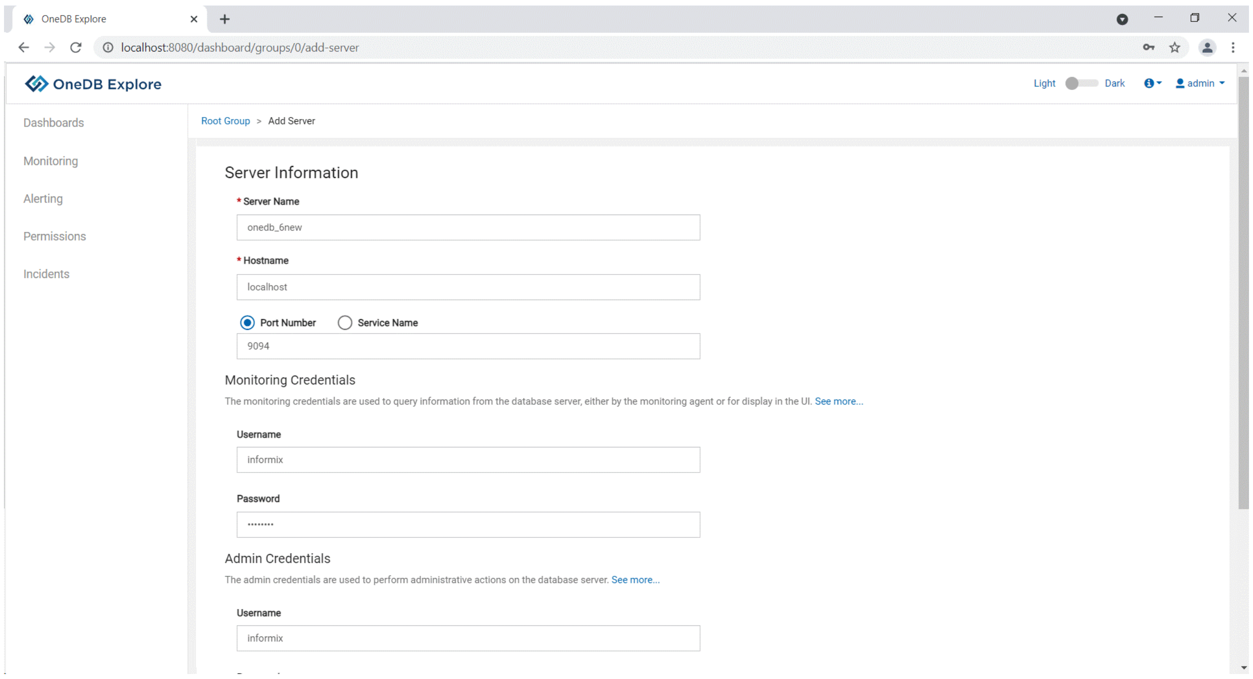
This topic provides the steps to create a new database using OneDB Explore.

1. Login to OneDB Explore using admin credentials.


OneDB Explore
localhost:8080/login
OneDB Explore
Username
Password
Login
@ Copyright IBM Corp. 2015, 2017. @ Copyright HCL Technologies Ltd. 2017, 2020.

2. Click the **Add Server** button to enter the server information and credentials to establish the connection.





Server Information

* Server Name

onedb_6new

* Hostname

localhost

Port Number   Service Name

9094

Monitoring Credentials

The monitoring credentials are used to query information from the database server, either by the monitoring agent or for display in the UI. See more...

Username

informix

Password

••••••••

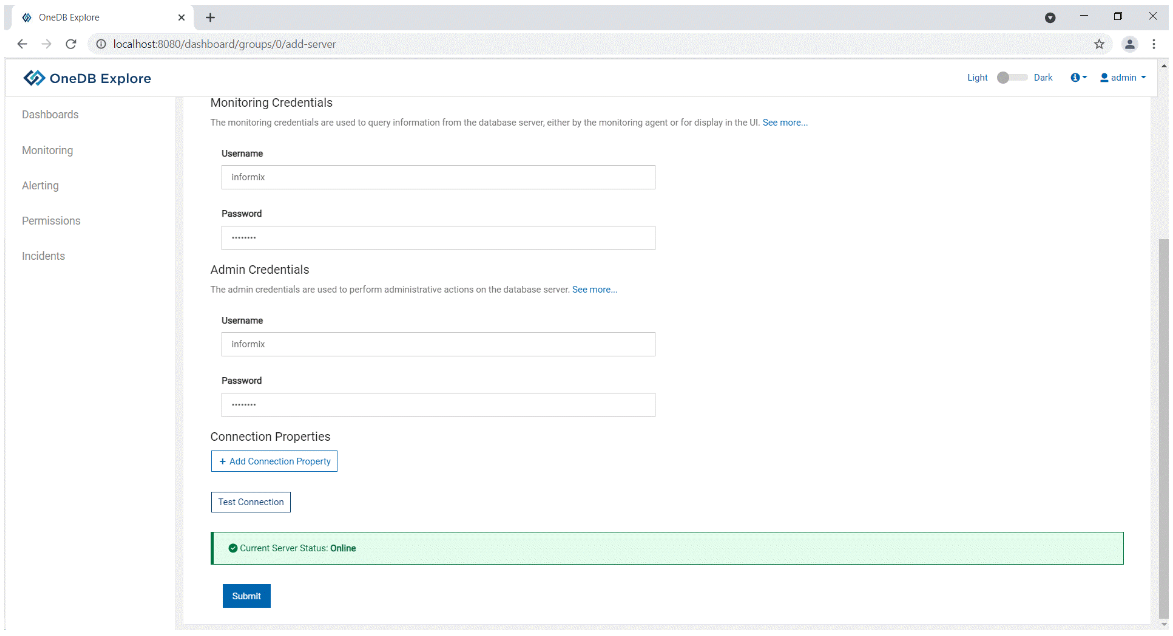
Admin Credentials

The admin credentials are used to perform administrative actions on the database server. See more...

Username

informix

a. Click the **Test Connection** button to verify if the server is online and click **Submit**.



Monitoring Credentials

The monitoring credentials are used to query information from the database server, either by the monitoring agent or for display in the UI. See more...

Username

informix

Password

••••••••

Admin Credentials

The admin credentials are used to perform administrative actions on the database server. See more...

Username

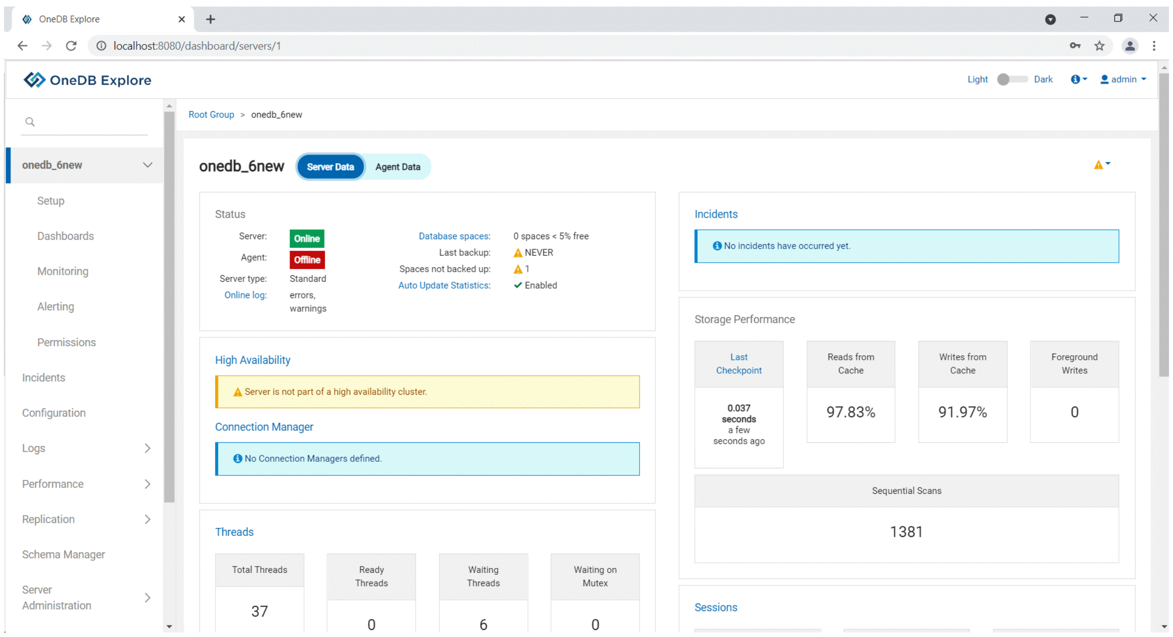
informix

Password

••••••••

Connection Properties

Current Server Status: Online



onedb_6new

Status

Server: Online
Agent: Offline
Server type: Standard
Online log: errors, warnings
Database spaces: 0 spaces < 5% free
Last backup: NEVER
Spaces not backed up: 1
Auto Update Statistics: Enabled

Incidents

No incidents have occurred yet.

High Availability

Server is not part of a high availability cluster.

Connection Manager

No Connection Managers defined.

Storage Performance

Last Checkpoint	Reads from Cache	Writes from Cache	Foreground Writes
0.037 seconds a few seconds ago	97.83%	91.97%	0

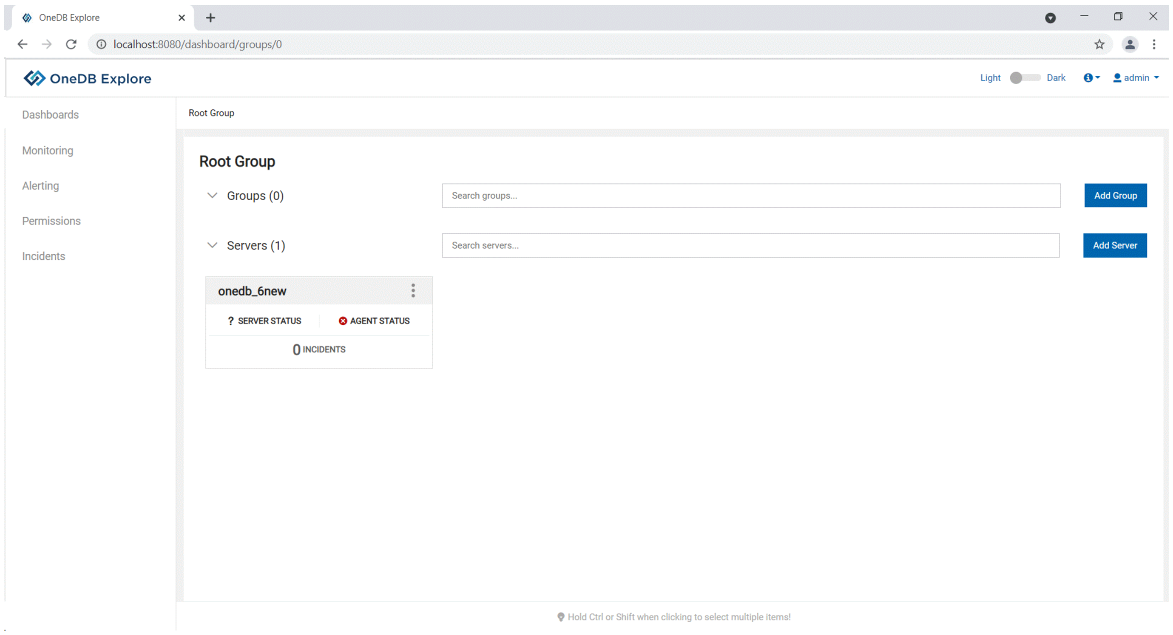
Sequential Scans
1381

Threads

Total Threads	Ready Threads	Waiting Threads	Waiting on Mutex
37	0	6	0

Sessions

b. If the server is already configured, select the server and proceed to next step.



Root Group

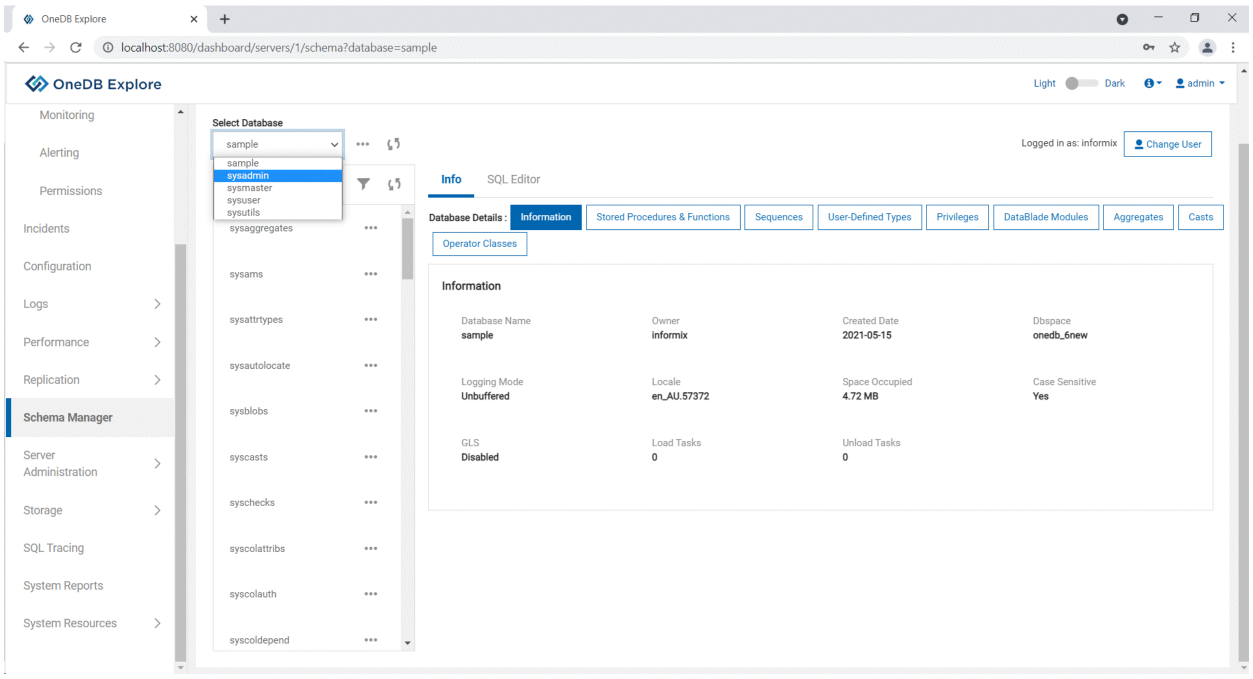
Groups (0)

Servers (1)

onedb_6new
? SERVER STATUS
AGENT STATUS
0 INCIDENTS

Hold Ctrl or Shift when clicking to select multiple items!

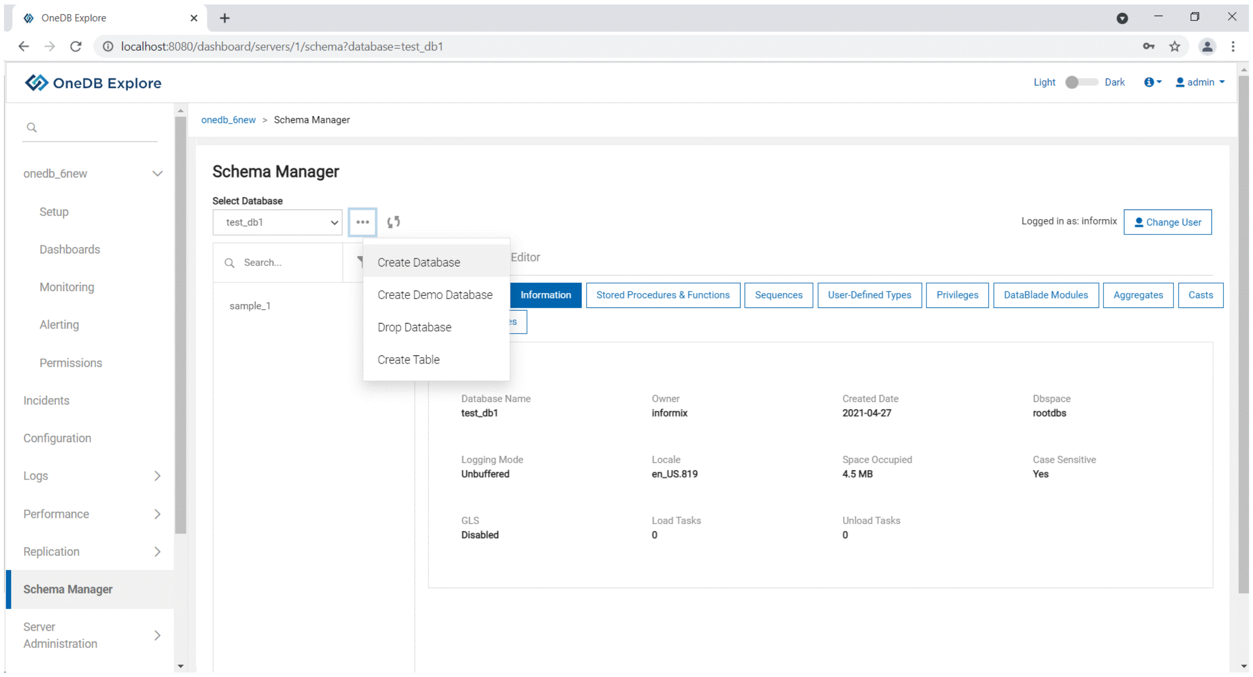
3. Select **Schema Manager** from the left pane and existing databases will be displayed in the **Select Database** drop-down list.



Information

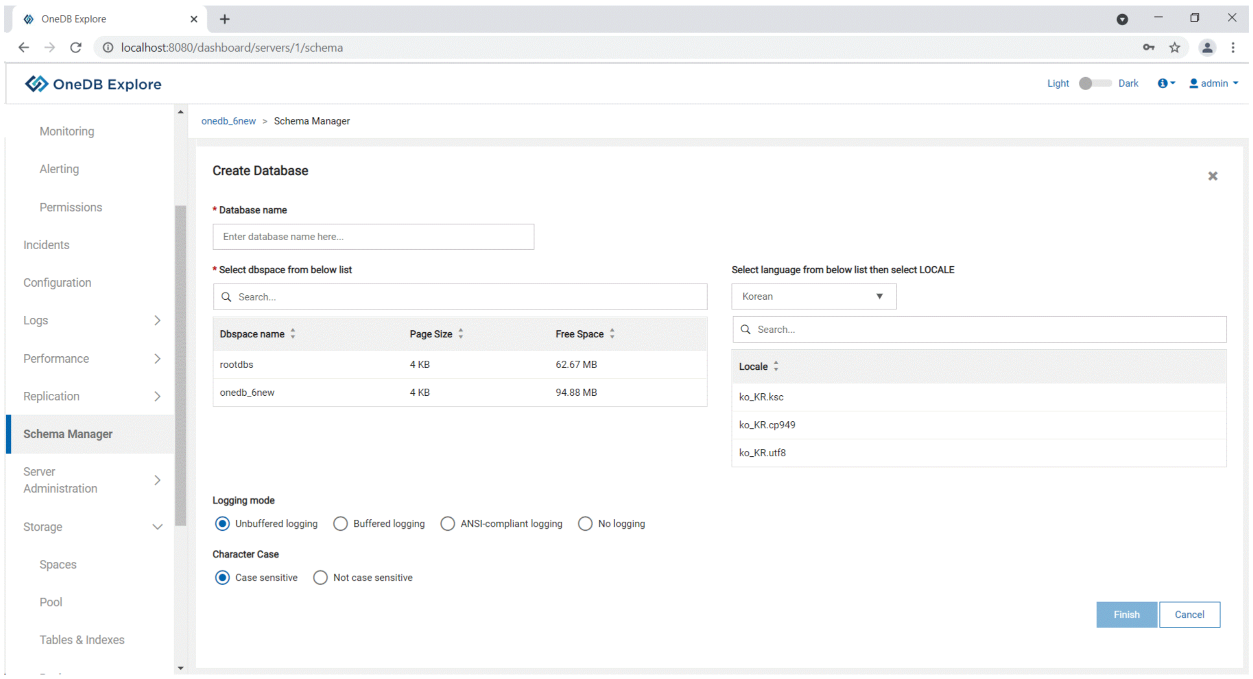
Database Name	Owner	Created Date	Dbspace
sample	informix	2021-05-15	onedb_6new
Logging Mode	**Locale**	**Space Occupied**	**Case Sensitive**
Unbuffered	en_AU.57372	4.72 MB	Yes
GLS	**Load Tasks**	**Unload Tasks**	
Disabled	0	0	

4. Click on **...** and select **Create Database** to create a new database.



Database Name	Owner	Created Date	Dbspace
test_db1	informix	2021-04-27	rootdbs
Logging Mode	**Locale**	**Space Occupied**	**Case Sensitive**
Unbuffered	en_US.819	4.5 MB	Yes
GLS	**Load Tasks**	**Unload Tasks**	
Disabled	0	0	

5. Enter the details and then click **Finish** button to create the database.



Create Database

* Database name

Enter database name here...

* Select dbspace from below list

Dbspace name	Page Size	Free Space
rootdbs	4 KB	62.67 MB
onedb_6new	4 KB	94.88 MB

Select language from below list then select LOCALE

Korean

Locale
ko_KR.ksc
ko_KR.cp949
ko_KR.utf8

Logging mode

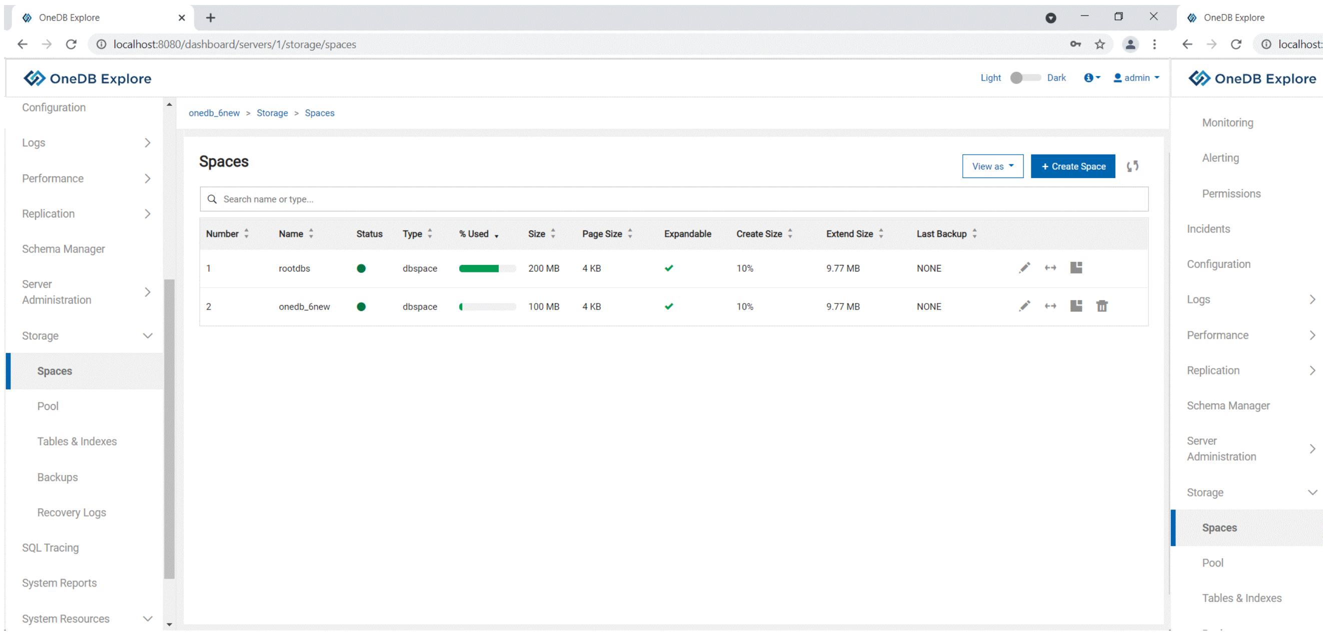
[✓ Unbuffered logging] Buffered logging ANSI-compliant logging No logging

Character Case

[✓ Case sensitive] Not case sensitive

- Database name –Name of the new database. Must be a unique name.
- Dbspace – Storage space allocated to store all the data related to this new database. It lists down all the available dbspaces in the server and the default is root dbspace.

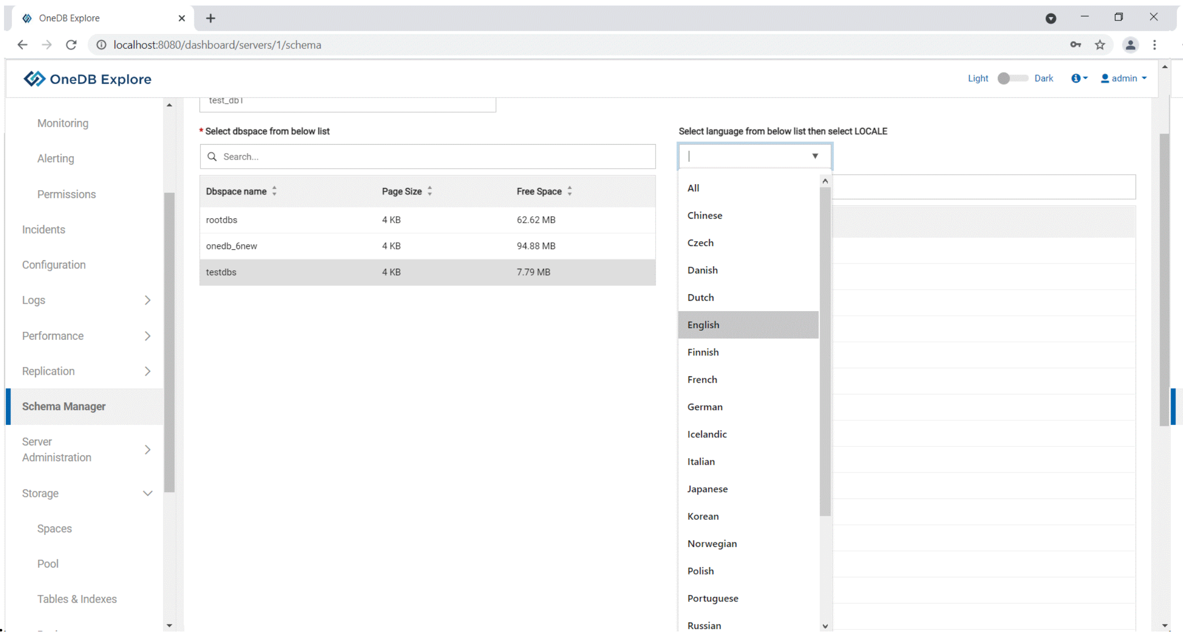
Note: User can create a dbspace of required size using the **Storage – Spaces** option from the left pane and click **Create Space**.



Spaces

Number	Name	Status	Type	% Used	Size	Page Size	Expandable	Create Size	Extend Size	Last Backup
1	rootdbs	●	dbspace		200 MB	4 KB	✓	10%	9.77 MB	NONE
2	onedb_6new	●	dbspace		100 MB	4 KB	✓	10%	9.77 MB	NONE

- Language and Locale – Select a language, the database uses to store the data and then select the locale with the required code set to correctly interpret locale-sensitive data types (NCHAR and NVARCHAR).



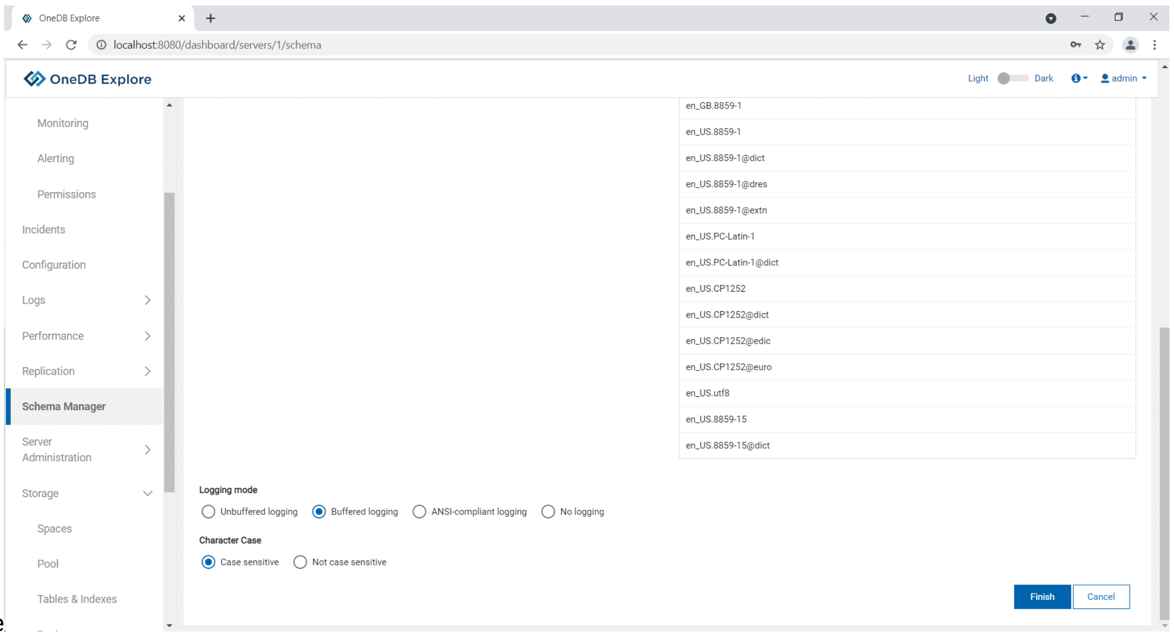
test_db1

* Select dbspace from below list

Dbspace name	Page Size	Free Space
rootdbs	4 KB	62.62 MB
onedb_6new	4 KB	94.88 MB
testdbs	4 KB	7.79 MB

Select language from below list then select LOCALE

- Logging mode – Select one of the four options:
  - Unbuffered logging – All transactions are written to disk which makes it slower but recommended.
  - Buffered logging – Transactions can be used, and all logs are stored in buffer before written to disk.
  - ANSI-compliant logging – It enables implicit transaction.
  - No logging – Fast but not safe as transactions are not allowed.
- Character Case

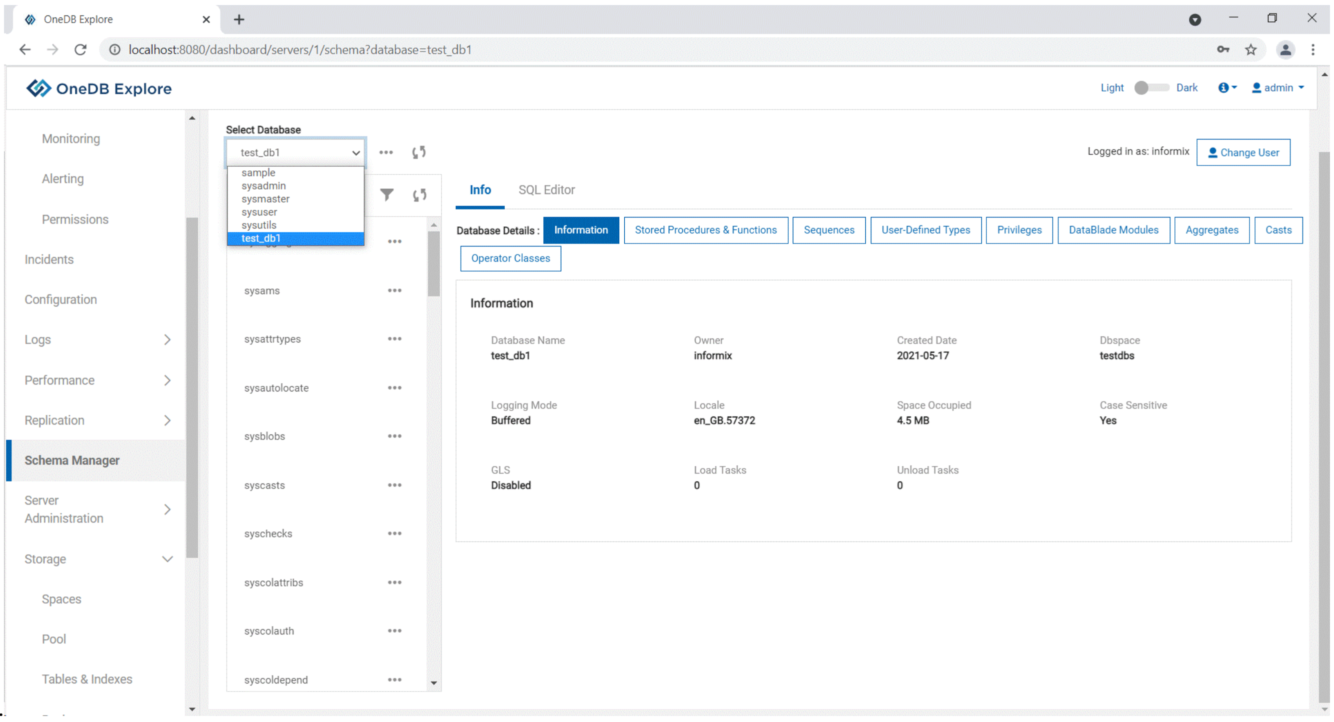


Logging mode
Unbuffered logging [✓ Buffered logging] ANSI-compliant logging No logging

Character Case
[✓ Case sensitive] Not case sensitive

Results

A new database is created, and select **Info** tab to view the



Information

Database Name	Owner	Created Date	Dbspace
test_db1	informix	2021-05-17	testdbs

Logging Mode	Locale	Space Occupied	Case Sensitive
Buffered	en_GB.57372	4.5 MB	Yes

GLS	Load Tasks	Unload Tasks
Disabled	0	0

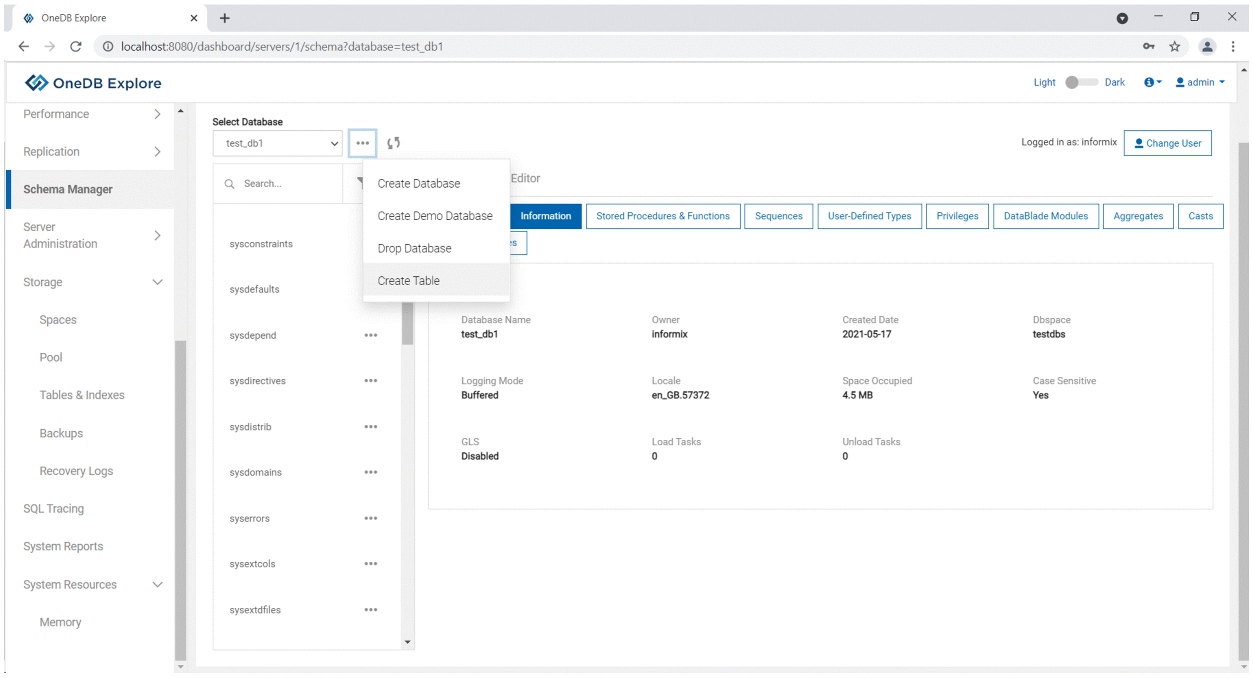
details.

Creating a table using OneDB Explore

About this task

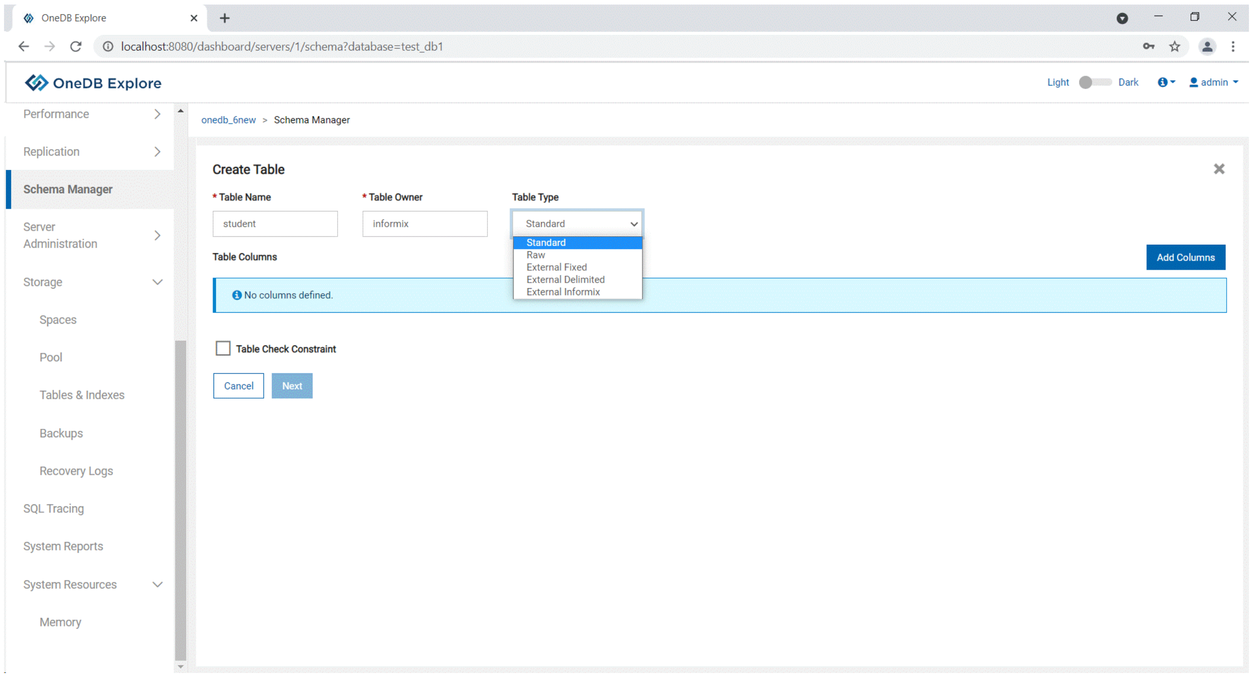
This topic provides the steps to create a table.

1. Select the required database from the **Select Database** drop-down list and click on **...** to select the **Create Table** option.



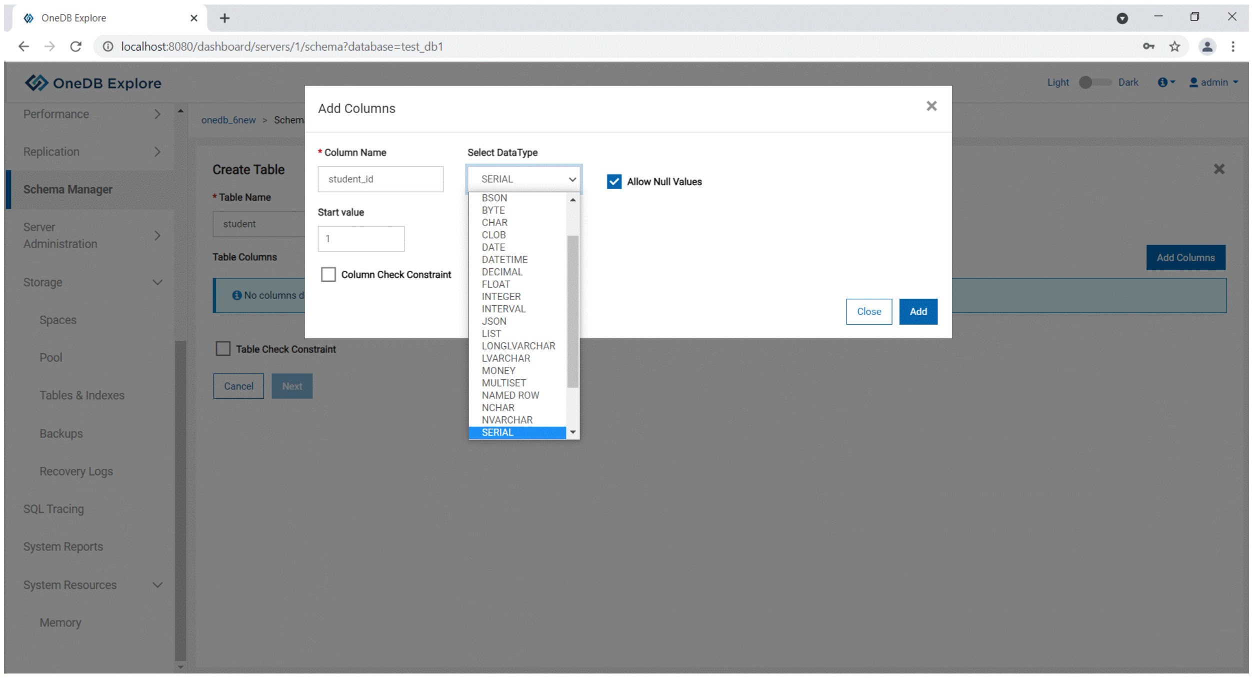
Database Name test_db1	Owner informix	Created Date 2021-05-17	Dbspace testdbs
Logging Mode Buffered	Locale en_GB.57372	Space Occupied 4.5 MB	Case Sensitive Yes
GLS Disabled	Load Tasks 0	Unload Tasks 0	

2. Enter a unique table name and select the table type from the drop-down list. Following is an example of Standard table type.



* Table Name: student
* Table Owner: informix
Table Type: Standard
Table Columns
No columns defined.

3. Click **Add Columns** to add the columns for this table. Enter the column name and select datatype from the given list and click **Add**. Enter the require details.



Add Columns

* Column Name: student_id

Select DataType: SERIAL

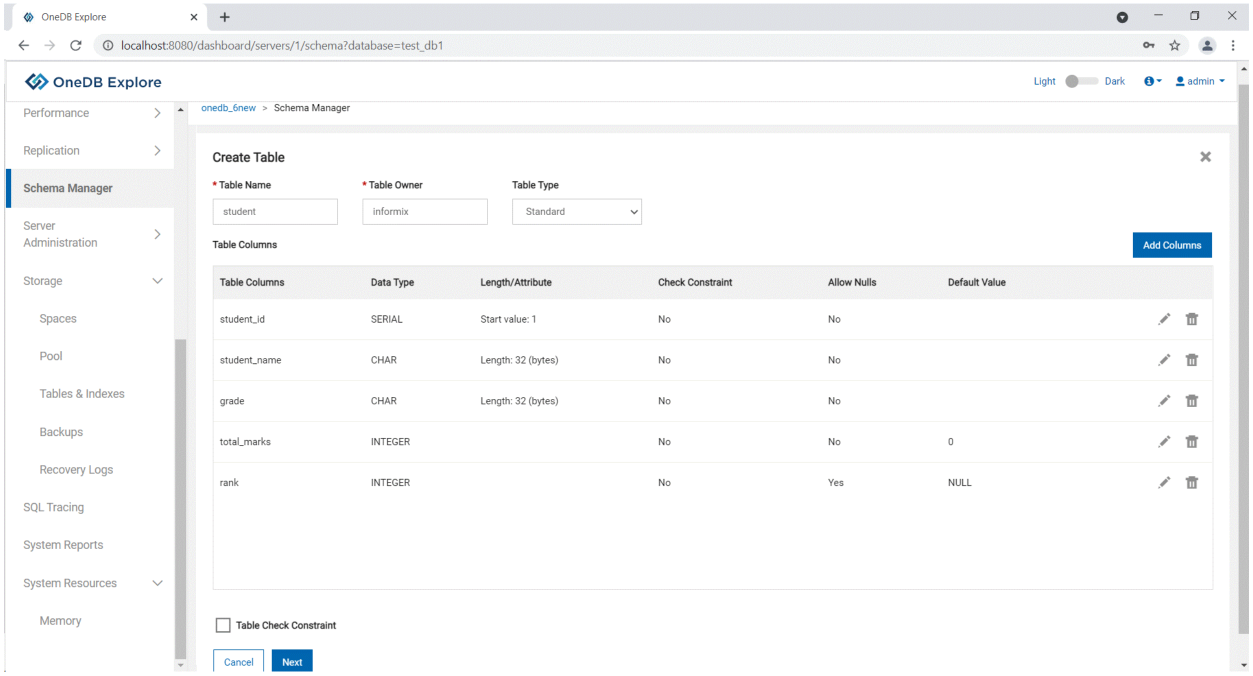
Start value: 1

[✓] Allow Null Values

[ ] Column Check Constraint

* Table Name: student

4. Click **Close** and click **Next**.



Create Table

* Table Name: student

* Table Owner: informix

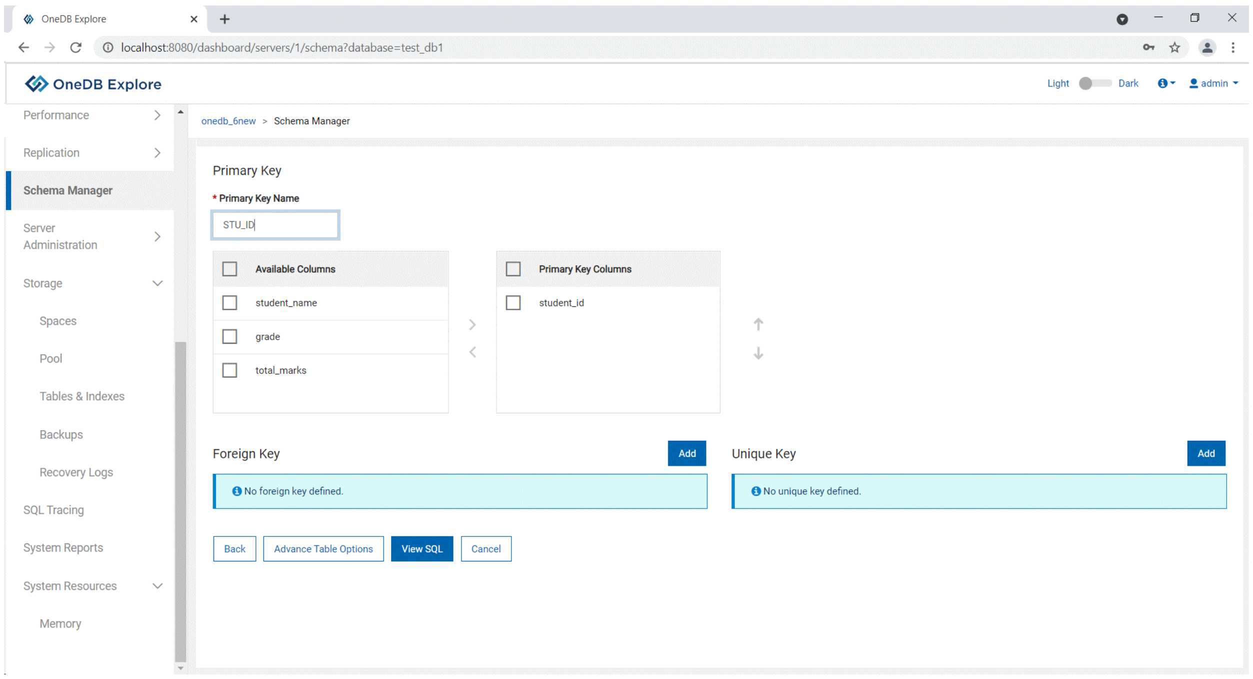
Table Type: Standard

Table Columns

Table Columns	Data Type	Length/Attribute	Check Constraint	Allow Nulls	Default Value
student_id	SERIAL	Start value: 1	No	No	
student_name	CHAR	Length: 32 (bytes)	No	No	
grade	CHAR	Length: 32 (bytes)	No	No	
total_marks	INTEGER		No	No	0
rank	INTEGER		No	Yes	NULL

[ ] Table Check Constraint

5. You can choose the Primary key from the available column list and define Foreign Key and Unique Key as per the requirement. Either click **View SQL** button to directly view the SQL query constructed with the default settings or click **Advance Table Options** button.



Primary Key

* Primary Key Name: STU_ID

Available Columns	Primary Key Columns
student_name	student_id
grade	
total_marks	

Foreign Key

No foreign key defined.

Unique Key

No unique key defined.

6. You can modify the default table options in this page and click **View Query & Create** button.

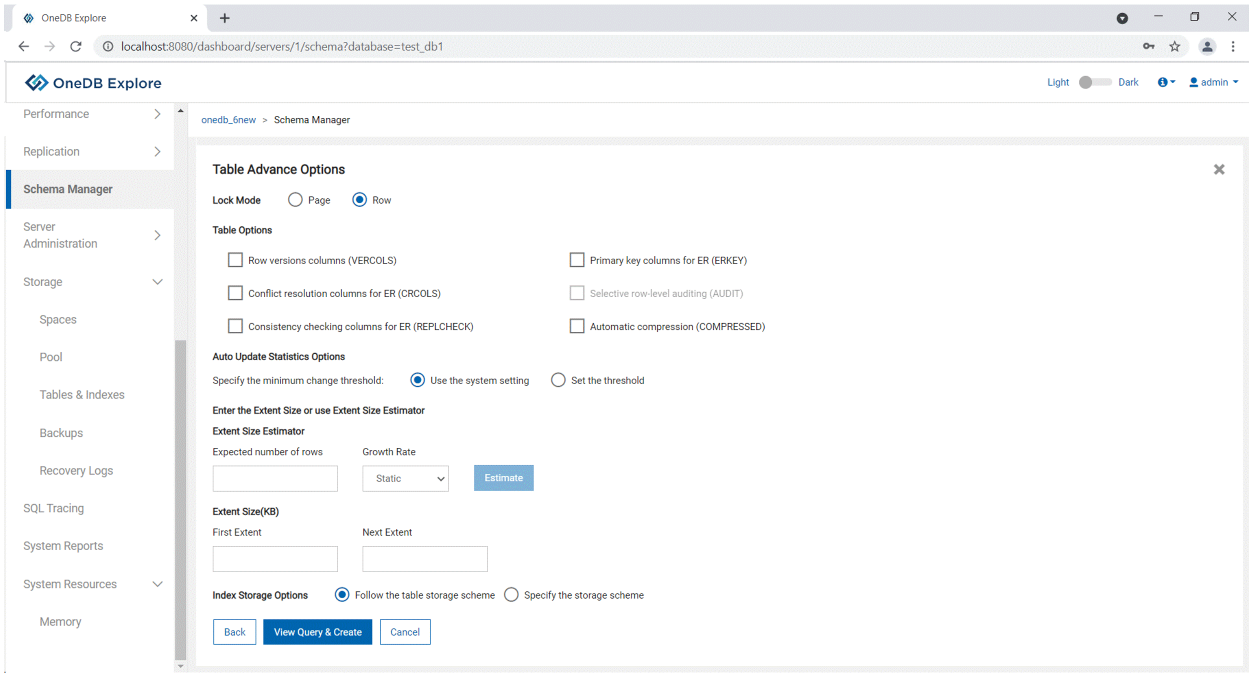


Table Advance Options

Lock Mode: Page [✓ Row]

Table Options

- [ ] Row versions columns (VERCOLS)
- [ ] Primary key columns for ER (ERKEY)
- [ ] Conflict resolution columns for ER (CRCOLS)
- [ ] Selective row-level auditing (AUDIT)
- [ ] Consistency checking columns for ER (REPLCHECK)
- [ ] Automatic compression (COMPRESSED)

Auto Update Statistics Options

Specify the minimum change threshold: [✓ Use the system setting] Set the threshold

Enter the Extent Size or use Extent Size Estimator

Extent Size Estimator

Expected number of rows:

Growth Rate: Static

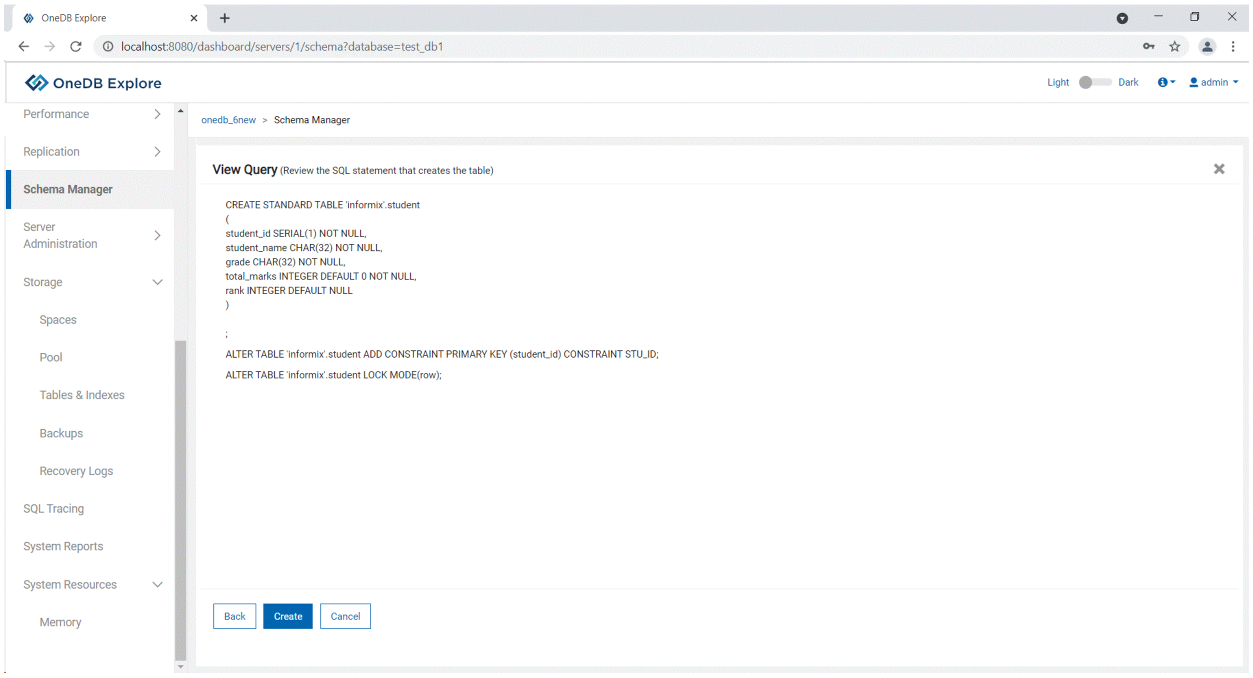
Extent Size(KB)

First Extent:

Next Extent:

Index Storage Options: [✓ Follow the table storage scheme] Specify the storage scheme

7. If the constructed query is as expected, click **Create** to create the table. Otherwise click **Back** to modify as per the requirement.

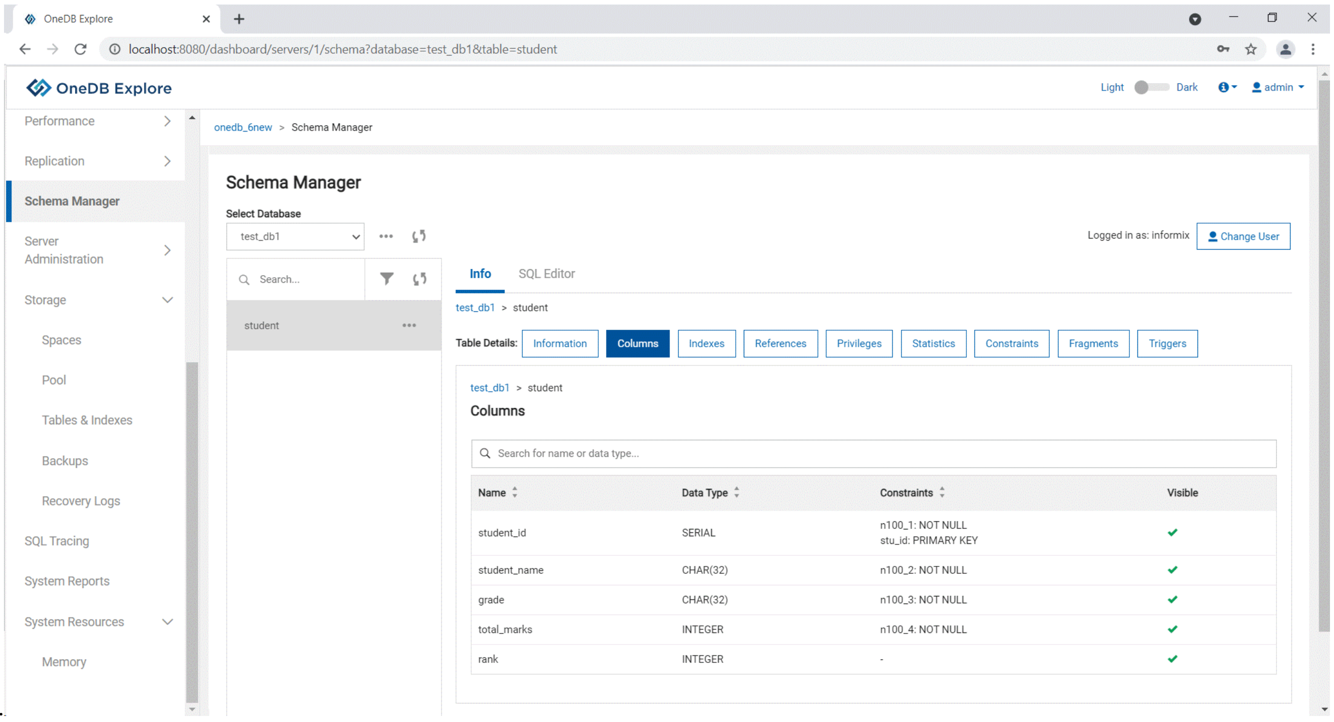


```
CREATE STANDARD TABLE 'informix'.student
(
student_id SERIAL(1) NOT NULL,
student_name CHAR(32) NOT NULL,
grade CHAR(32) NOT NULL,
total_marks INTEGER DEFAULT 0 NOT NULL,
rank INTEGER DEFAULT NULL
)

;
ALTER TABLE 'informix'.student ADD CONSTRAINT PRIMARY KEY (student_id) CONSTRAINT STU_ID;
ALTER TABLE 'informix'.student LOCK MODE(row);
```

Results

A new table is created, and select **Info** tab to view the



Name	Data Type	Constraints	Visible
student_id	SERIAL	n100_1: NOT NULL stu_id: PRIMARY KEY	✓
student_name	CHAR(32)	n100_2: NOT NULL	✓
grade	CHAR(32)	n100_3: NOT NULL	✓
total_marks	INTEGER	n100_4: NOT NULL	✓
rank	INTEGER	-	✓

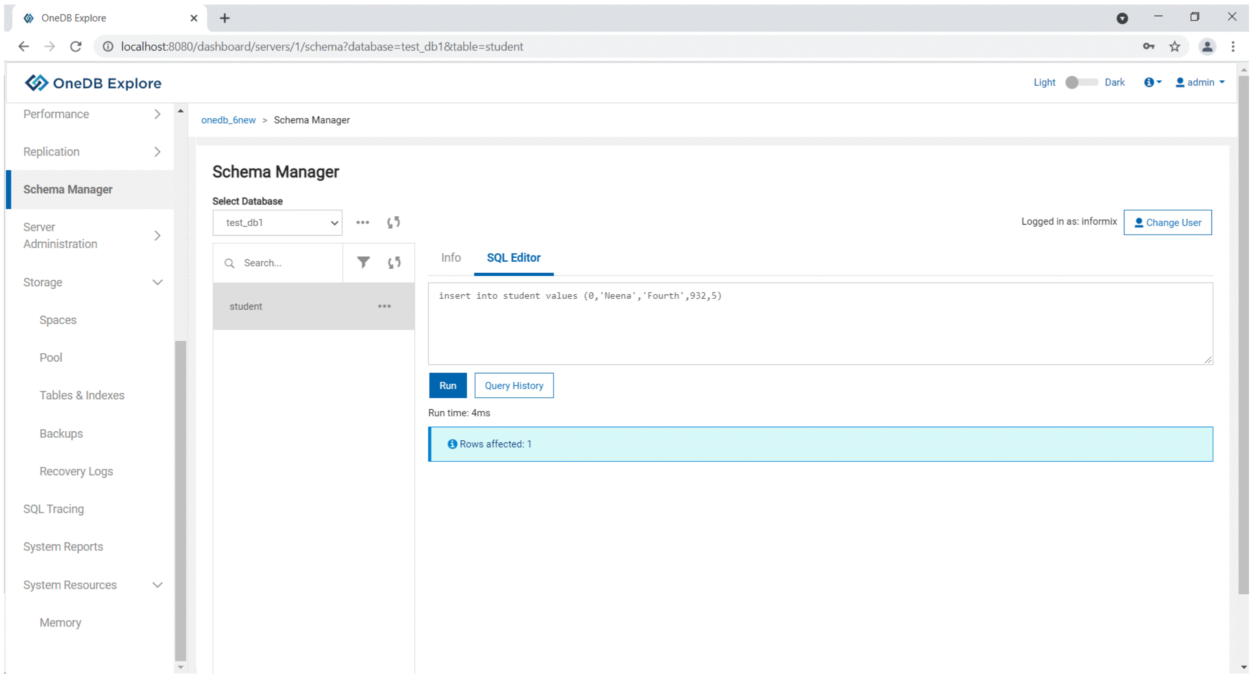
details.

Adding a record using OneDB Explore

About this task

This topic provides the steps to create a record.

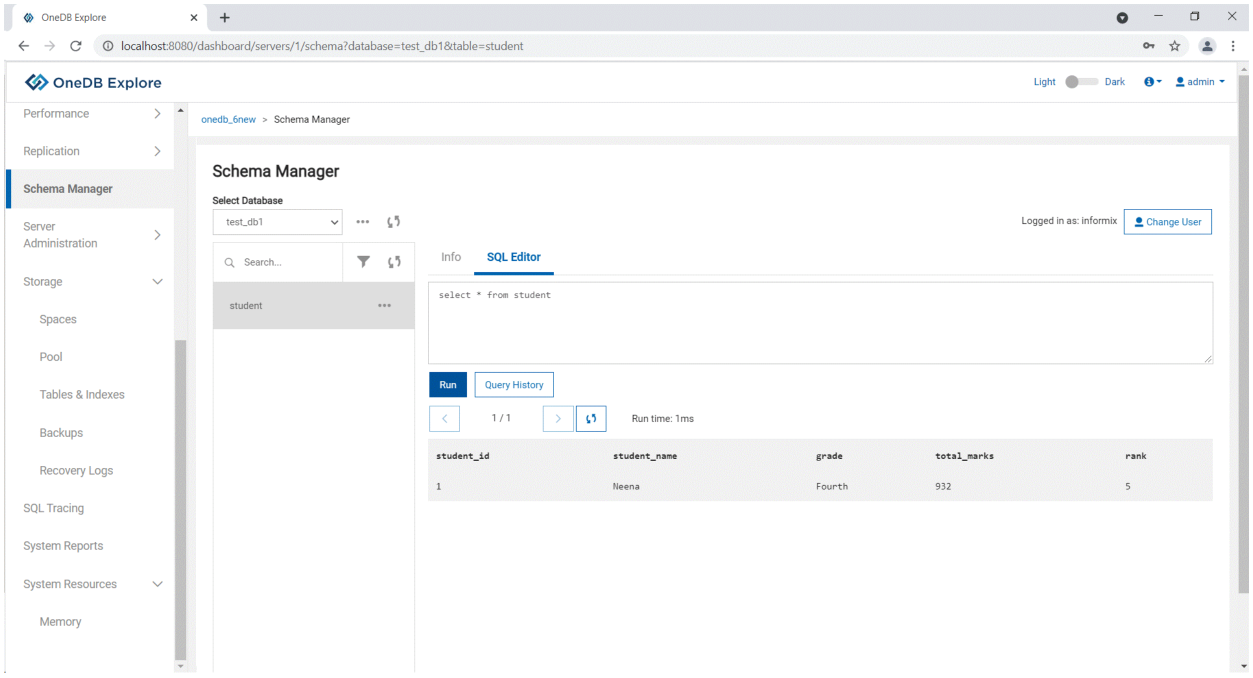
1. Select the database from the **Select Database** drop-down list and select the **table** for which you want to add a record.
2. Select **SQL Editor** tab to execute the sql queries.



```
insert into student values (0,'Neena','Fourth',932,5)
```

Run time: 4ms

Rows affected: 1



```
select * from student
```

Run time: 1ms

student_id	student_name	grade	total_marks	rank
1	Neena	Fourth	932	5

Data Replication

HCL OneDB™ has built-in support for variety of business requirements related to fault tolerance, data replication, and database scale-out using sharding technology.

Customers rely HCL OneDB™ replication technologies to meet each one of the below mentioned business requirements:

- For transactional database, businesses demand zero data loss, and less than one minute recovery time in case of failure
- Businesses demand rolling application upgrade, and rolling server upgrade support – No planned downtime for maintenance operations
- Businesses do require transactional databases to be protected from natural disasters, and human errors
- Businesses do require transactional databases to scale out quickly to handle spikes during special events
- For geographically distributed applications, businesses do require transactional database to be available even if network goes down between datacenters
- For regulatory reasons, businesses do require certain data to be stored local to the country and still support seamless access to the data for applications residing across geographic boundaries

Physical and logical replication

HCL OneDB™ replication technology is broadly classified into physical and logical replication.

With physical replication, logical log records get replicated to secondary server, and secondary server continuously apply logical log records as if server is in crash recovery mode. Physical replication requires database storage, hardware architecture, operating system and HCL OneDB™ server version to be same.

Logical replication does capture data from logical log records, but log records gets converted to a format understood by SQL engine. Logical replication does support heterogeneous hardware and software. Customers can pick and choose which tables to be replicated, and can replicate only few columns or few rows in a table.

HCL OneDB™ High Availability Data replication(HDR), Remote Standalone Secondary(RSS) and Shared Disk Secondary(SDS) servers fall under physical replication category.

Enterprise replication(ER) and flexible grid technology fall under logical replication category.

HDR, and SDS technologies support zero data loss (RPO=0) configuration.

RSS supports asynchronous replication across wide area networks and ideal deployment option to protect transactional database from natural disasters. RSS “delayed apply” configuration protects transactional database from human errors.

Shared disk technology(SDS) is ideal fit to scale-out transactional database to support spike in transactional volume during special events.

From CAP theorem perspective, HCL OneDB™ physical replication technologies support CP configuration. In case of network partitioning, secondary servers cannot accept write activity till new primary server was elected. With physical replication, primary server is the designated write server for transactional data.

HCL OneDB™ Enterprise Replication technology supports AP configuration. In case of network partitioning, applications can continue performing database write activity at all ER servers and ER rely on conflict resolution rules to resolve conflicting updates. ER technologies supports near-zero downtime maintenance operations like rolling application and schema upgrade, rolling server upgrade, and thus eliminate requirement for planned downtime for maintenance operations.

Enterprise Replication (ER)

Enterprise Replication (ER) technology, which is both flexible and powerful, allows businesses to replicate data from branch offices to corporate office, replicate data from corporate office to branch offices, replicate data closer to customers.

ER works by you first defining the servers among which you would like data to be replicated. This creates a network topology — of root, leaf, and non-root, non-leaf nodes — that will transfer the data. Each ER *`node`* may be a single server or a cluster of servers, which this article further discusses below. All the interconnected nodes together are called an ER *`domain`*. The domain does not define what data will be transferred, just the paths or routes along which the data will flow.

Next, you decide what data you would like to be replicated. This is defined by an SQL query involving a server, database, table, and columns. The query acts as a filter whose output decides the data to replicate and proves to be a very flexible tool. If you'd like an entire table to be replicated, your query would be `SELECT * FROM ...`. If you'd like just the first three columns to be replicated, your query would now be `SELECT column1, column2, column3 FROM ...`. Or, if you'd like only certain rows to be replicated, you just use a `WHERE` clause in your query. ER can replicate data with very broad or very fine filters. To help with replication ER requires that the table have a primary key defined.

After the query is written, determine what nodes should participate in replicating the data. Let's say you want nodes A, B, and C to all have the same data in their Employee table. So A, B, and C are your *`participants`*. ER can be configured so that if data changes are made on any server, the other participants become updated too. This is called an *`update anywhere`* scenario. What happens if you only want updates to flow from A, into both B and C, but back never to A? ER gives you the flexibility to do these `primary-target` scenarios as well. In situations where data changes occur on more than one participant, it may happen that a row changed on server B conflicts with a change occurring on server C. ER allows you to define rules for automatically handling such conflicts. These include comparing timestamps, running a stored procedure, and ignoring conflicts.

Now that you know the query, the participants, and the scenario, take all this information and use it to create something called a *`replicate`*. Think of a replicate as a faucet that controls the flow of data from the query. The faucet can be turned on or off, and you can have as many faucets as you want. Replicates can be grouped into sets, which make it easier for users to control multiple replicates. Also, you can use templates to help you quickly create replicates for multiple tables. All this work is done through the server utility `cdr`.

After a replicate has been created and started, how does the data actually get moved? Well, first of all, the replication happens asynchronously. This means there may be a delay between data being committed on one node and it appears on another node. And only committed data, of course, is replicated. ER works by reading the logical logs, testing if a log record needs to be replicated, and finally sending that information to the appropriate participants.

A simple Example

About this task

Let's set up a simple example of ER. We'll replicate the employees table of the "db1" database in an update-anywhere fashion between server1 and server2. (The following steps assume the DBA user account is used.)

1. Prepare the SQLHOSTS files

   The file for each server should contain an entry for both servers as well as two group definitions. Each group represents an ER `node`:

   ```
   grp1  group   -  -  i=1 server1  onsoctcp host port g=grp1
   grp2  group  -  -  i=2 server2  onsoctcp host port g=grp2
   ```
2. Prepare the ER disk space

   For each server, perform the following steps:
   1. Edit the ONCONFIG to contain a smart blob space:

      ```
      CDR_QDATA_SBSPACE sbs1
      ```
   2. Start the server
   3. Create a chunk for that smartblob space and add the space to the server. For example, on UNIX:

      ```
      touch /onedb/chunks/sbs1_chunk1 chmod 660 /onedb/chunks/sbs1_chunk1
      onspaces -c -S sbs1 -p /onedb/chunks/sbs1_chunk1 -s 100000 -o 0
      ```
3. Add the nodes to a new ER domain

   On server1, execute the following command:

   ```
   cdr define server --init grp1
   ```

   On server2, execute the following command:

   ```
   cdr define server --sync=grp1 --init grp2
   ```

   These commands create an ER `domain` of two root nodes.
4. Prepare the table to replicate

   On both server1 and server2, run the following SQL statements:

   ```
   CREATE DATABSE db1 WITH LOG;
   CREATE TABLE employees(id int primary key, ...);
   ```
5. Create the replicate

   On server1, execute:

   ```
   cdr define replicate --conflict=always r_employees \
   "db1@grp1:informix.employees"  "select * from employees" \
   "db1@grp2:informix.employees"  "select * from employees"
   ```
6. Start the replicate

   On either server, run:

```
cdr start replicate r_employees
```

Now you're replicating data! An insert, update, or delete to a row on either server1 or server2 will soon appear on the other server.

High availability cluster

In a cluster, you gain scale-out, high availability, disaster recovery, and load-balancing capabilities. This technology is built right into the database and requires no additional hardware, software, or network infrastructure. In contrast to ER, in a cluster, you cannot control the granularity of data replication; the entire server's data is always replicated. Cluster and ER technologies, as this article shows later, can be used side by side.

A cluster consists of one primary server and a mix of three different types of secondary servers:

- Shared Disk (SD) secondary server
- High-Availability Data Replication (HDR) server
- Remote Standalone (RS) secondary server

Servers must be the same HCL OneDB™ version and run on the same operating system, hardware, and disk architecture. Applicants can also insert, update, and delete data on secondary servers. Thus, all three secondary server types can be used to increase the capacity of your primary as well as balance the load.

Table 1, below, compares the server types. Table 2 lists some common `ONCONFIG` settings.

Table 7. Overview of the characteristics of cluster server types

	Primary	**SD secondary**	**HDR**	**RS secondary**
Primary purpose		Increased capacity/scaleout	High availability/hot standby	Disaster recovery/remote backup
Maximum number of nodes per cluster	One	No limit	One	No limit
Possible type changes in one step	SD secondary, HDR, standard	Primary	Primary, RS secondary, standard	HDR, standard
Supported isolation levels	All	Dirty read, Committed Read, Committed Read, Last Committed	Dirty read	Dirty read

Table 8. ONCONFIG parameters common to cluster server types

Configuration parameter	Server type	Supported values	Description
HA_ALIAS	SD and RS secondary servers	DBSERVERNAME or one of the DBSERVERALIAS name. Must be a network protocol.	Used to define the name by which a secondary server is known within a cluster. Gives you flexibility to use something other than DBSERVERNAME.
LOG_INDEX_BUILDS	Primary, HDR, and RS secondary	• 0- Disable index page logging • 1 - Enable index page logging	While creating a new index at primary server, Index Page Logging (IPL) writes index pages to logical log file. Enabling index page logging is required for RSS nodes and optional for HDR secondary server.
UPDATABLE_SECONDARY	HDR, SD and RS secondary servers	Primary	Primary, RS secondary, standard
Supported isolation levels	All	• 0- Disable write functionality at secondary server • >=1 - Enable write functionality at secondary server	Determines the number of network connections used between the primary and secondary server to support data updates.
TEMPTAB_NOLOG	Primary, HDR, SD, and RS secondary servers	• 0- Create temporary tables with logging enabled by default • 1 - Create temporary tables without logging.	Required to be set to 1 at HDR, RSS, and SD secondary servers. Controls the default logging mode for explicit temporary tables created with CREATE TEMP TABLE or SELECT INTO TEMP statements. Secondary servers must not have logged temporary tables.

Shared Disk (SD) secondary

SD secondary servers share disk space — except temporary dbspaces — with the primary server. This is typically done through a network-based clustered file system. Adding a new SD secondary to a cluster is very easy and can be done in a few seconds once the shared disk is prepared. Because SD secondary nodes leverage the primary's disks and can be brought up easily and quickly, they are well-suited for scale-out scenarios. In SD secondary servers checkpoints are synchronized. This means a checkpoint at the primary server completes only after the checkpoint at the SD server completes. SD secondary supports committed read and committed read last committed isolation levels as well as dirty read. An SD secondary can be promoted to a primary server with one single command: `onmode -d make primary <name of SD server>`. Because an SD secondary server is so close to the primary (in other words, it shares the same disk), it is often the best type of server to initially fail over to if the primary should encounter a problem.

Table 9. Important ONCONFIG parameters for the SD secondary

Configuration parameter	Server type	Supported values	Description
SDS_ENABLE	Primary, SD secondary	• 0- Disable SDS functionality • 1 -Enable SDS functionality	Use this to allow SD secondaries to be added to the cluster
SDS_PAGING	SD secondary	<absolute path for paging file1>,<absolute path for paging file 2>	Two paging files must be configured to bring up SDS node
SDS_TEMPDBS	SD secondary	<dbspace_name>,<path>,<pagesiz in KB>,<offset in KB>,<size in KB>	eTemporary dbspace information for SD secondary node. You can configure up to 16 SDS_TEMPDBS entries. Example: SDS_TEMPDBS sdstmpdbs1, / work/dbspaces/ sdstmpdbs1,2,0,16000
SDS_TIMEOUT	Primary	>= 0 seconds	This configuration parameter is used at the primary to decide how long to wait for an acknowledgement from an SD server. If no acknowledgement occurs,

Table 9. Important ONCONFIG parameters for the SD secondary (continued)

Configuration parameter	Server type	Supported values	Description
			the primary acts to shut the SD server down. The default value is 20 seconds

Adding an SD secondary to a cluster

About this task

Let's take a standalone OneDB server and turn it into a cluster. First let's add an SD secondary server. (This scenario assumes that the shared disk file system has already been set up and the DBA account is used.)

1. Prepare the SQLHOSTS file.

   Make sure the SQHOSTS file at both the primary and SDS node has entries for the other server:

   ```
   g_mach11  group     -          -          i=10
    myprim    ontlitcp  primhost  prim_port  g=g_mach11
    sds1      ontlitcp  sds1host  sds1_port  g=g_mach11
   ```

   Note:

   Note that the use of groups here is optional. However, we include it to prepare for an example below.

2. Set the primary as the owner of the shared disk.

   On the primary, run:

   ```
   onmode -d set SDS primary myprim
   ```

3. Configure the SD secondary.
   a. ▪ Make sure these match the `ONCONFIG` of the primary: `ROOTNAME`, `ROOTPATH`, `ROOTOFFSET`, `ROOTSIZE`, `PHYSDBS`, `PHYSFILE`, `LOGFILES`, and `LOGSIZE`.
   b. Set `SDS_ENABLE` to 1.
   c. Configure `SDS_PAGING` and `SDS_TEMPDBS`.

   For example:

   ```
   SDS_ENABLE          1
   SDS_PAGING          /onedb/sds/dbspaces/page_1,/onedb/sds/dbspaces/page_2
   SDS_TEMPDBS         sdstmpdbs1,/onedb/sds/dbspaces/sdstmpdbs1,2,0,16000
   UPDATABLE_SECONDARY 1
   TEMPTAB_NOLOG       1
   ```

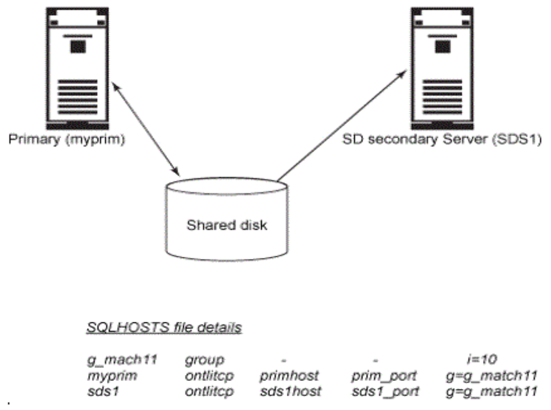
4. Start the SD secondary using:

   ```
   oninit
   ```

   Result

The SD secondary is now up and running! You can check the cluster status using the `onstat -g sds` command at both the primary and secondary servers. Here's what our cluster looks like:

Figure 1. The cluster: Primary and one SD secondary



SQLHOSTS file details

```
g_mach11   group      -          -          i=10
myprim     ontlitcp   primhost   prim_port  g=g_match11
sds1       ontlitcp   sds1host   sds1_port  g=g_match11
```

High-Availability Data Replication (HDR)

HDR consists of a pair of servers — the primary and the HDR secondary — and supports both synchronous and asynchronous replication modes. In synchronous mode, transactions on the primary server will not commit until it receives an acknowledgement from the HDR secondary server. Thus, the HDR secondary is immediately ready to take the place of the primary server — what is called a "hot" standby. In asynchronous mode, only checkpoints are synchronized between the primary and HDR secondary. One characteristic of HDR is that it uses a half-duplex communication protocol and thus is sensitive to network latency. This is not the case for SD and RS secondary servers.

Table 1, below, compares the server types. Table 2 lists some common `ONCONFIG` settings.

Table 10. Important ONCONFIG parameters for the HDR server

Configuration parameter	Server type	Supported Values	Description
DRAUTO	Primary, HDR server	1. - Manual 2. - Automatic failover ultimatelyretaining server types 3. - Automatic failover ultimatelyrever	Controls how the primary and HDR secondary behave in a failure scenario.

Table 10. Important ONCONFIG parameters for the HDR server (continued)

Configuration parameter	Server type	Supported Values	Description
		sing server types 4. - Have the Connection Manager Arbitrator control the failover	
DRIDXAUTO	Primary, HDR server	1. - Disable automatic index repair 2. - Enable automatic index repair	Automatically repair an index, if the HDR secondary server detects a corruption.
DRINTERVAL	Primary	-1 - Operate in synchronous mode > 0 - Operate in asynchronous mode	The maximum interval, in seconds, between flushing of the highavailability data-replication buffer.
HDR_TXN_SCOPE	Primary	ASYNC - Commits are not synced NEAR_SYNC - The committed transaction has been sent to the HDR secondary but not yet applied. FULL_SYNC - The transaction has been sent and applied on the HDR secondary.	Defines transactional synchronization in the HDR primary when DRINTERVAL is turned off. The default is NEAR_SYNC. To enable HDR_TXN_SCOPE behavior DRINTERVAL config parameter value must be set to 0.
DRLOSTFOUND	Primary, HDR server	<path for lost and found file>	The path for the HDR lost-andfound

Table 10. Important ONCONFIG parameters for the HDR server (continued)

Configuration parameter	Server type	Supported Values	Description
			transaction file. This file is created during a failover and contains the transactions committed on the primary but not the HDR server.
DRTIMEOUT	Primary	>= 0 seconds Default value = 30 seconds	The time, in seconds, before a network timeout occurs. Used by DRAUTO to detect failover.

Adding an HDR server to a cluster

About this task

Let's add an HDR secondary node to our cluster

1. Prepare the SQLHOSTS files

   Update the SQLHOSTS file at primary. and at both SDS and HDR secondary servers:

   ```
   g_mach11  group     -         -          i=10
    myprim    ontlitcp  primhost  prim_port  g=g_mach11
    hdr       ontlitcp  hdrhost   hdr_port   g=g_mach11
    sds1      ontlitcp  sds1host  sds1_port  g=g_mach11
   ```

2. Configure the ONCONFIG files

   In order for HDR to work, certain `ONCONFIG` parameters must be exactly the same on both the primary and secondary. In many cases, this means they cannot be changed while either server is running. These parameters include `DRAUTO`, `DRINTERVAL`, `DRTIMEOUT`, settings related to the root dbspace, settings related to logical logs, and more. To ensure these settings match, one technique is to copy the onconfig of the primary to the secondary, and then modify the few things that must be different, such as `DBSERVERNAME`. You should plan for these settings before bringing the primary up. This example just uses the defaults.

3. Register the HDR secondary server with the primary

   On the primary, run:

   ```
   onmode -d primary hdr
   ```

4. Prepare the HDR secondary server's disk

The storage used on the HDR secondary must match that of the primary (for example, number of dbspaces, number of chunks, chunk sizes, pathnames, and offsets). Since this example uses the backup to restore the HDR secondary server, it is only necessary that the chunk files exist and have the proper permissions.

5. Restore the backup on the HDR secondary server

   On the HDR server, perform a physical restore of the level 0 backup:

```
ontape -p
  Three questions will be asked.  Answer as shown below:
   Continue restore? (y/n) y
   Do you want to back up the logs? (y/n) n
   Restore a level 1 archive (y/n) n
```

6. Bring the HDR secondary server into online mode

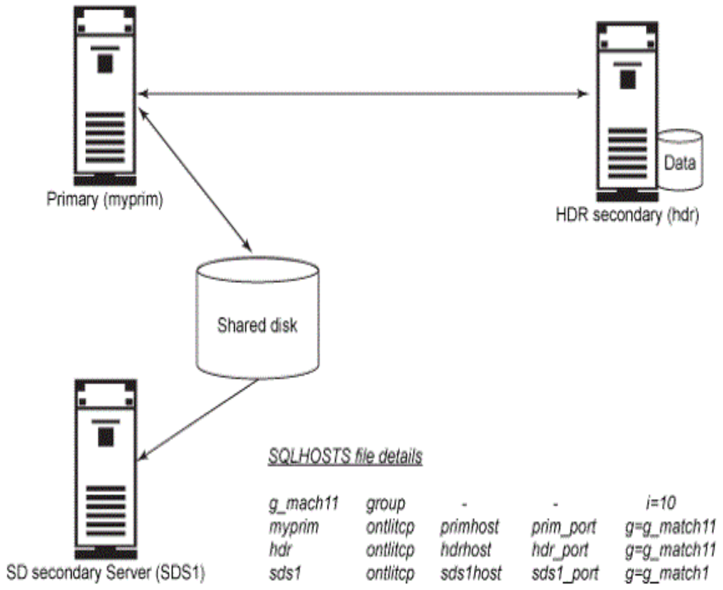
   After the restore is complete, the HDR secondary will be in recovery mode. Run the following command:

```
onmode -d secondary myprim
```

   Result

   HDR secondary is up and running! Run `onstat -g dri`at the primary and HDR secondary to check HDR status. Our cluster now looks like this:

Figure 2. The cluster: Primary and one SD secondary



```
g_mach11  group     -          -          i=10
myprim    ontlitcp  primhost   prim_port  g=g_match11
hdr       ontlitcp  hdrhost    hdr_port   g=g_match11
sds1      ontlitcp  sds1host   sds1_port  g=g_match1
```

Remote Standalone (RS) secondary

The primary purpose of the RS secondary server is to provide a disaster recovery solution. As in HDR, the primary server sends all its logical log records to the RS secondary server continuously, except now only in an asynchronous manner. Unlike HDR, the communication occurs using a full duplex protocol. Thus, the RS secondary is less sensitive to network latency and can more easily reside in a distant geography. One characteristic of an RS secondary is that unlike in SD and HDR servers, the primary server doesn't synchronize checkpoints with an RS secondary. Thus, it is not immediately ready to take the place of the primary; it must be changed into an HDR server first. Should multiple failures in a cluster occur, however, an RS node can prevent your database from being entirely unavailable.

Table 11. Important ONCONFIG parameters for the RS secondary

Configuration parameter	Server type	Supported values	Description
LOG_INDEX_BUILDS	Primary	1. - Disable index page logging 2. - Enable index page logging	While creating a new index at primary server, Index Page Logging(IPL) writes index pages to logical log file. Index page logging must be enabled at primary server to add RS server to a cluster.

Adding an RS secondary to a cluster

About this task

Let us add an RS node to our cluster.

1. Prepare the SQLHOSTS files
   All servers in the cluster must have SQLHOSTS entries for all others.

   ```
   g_mach11  group     -         -          i=10 myprim    ontlitcp
               primhost  prim_port  g=g_mach11 hdr       ontlitcp  hdrhost   hdr_port
             g=g_mach11 sds1      ontlitcp  sds1host
               sds1_port  g=g_mach11 rss1      ontlitcp
               rss1host  rss1_port  g=g_mach11
   ```

2. On the primary, enable index page logging

   ```
   onmode -wf LOG_INDEX_BUILDS=1
   ```

3. On the primary, register the new RS secondary

   ```
   onmode -d add RSS rss1
   ```

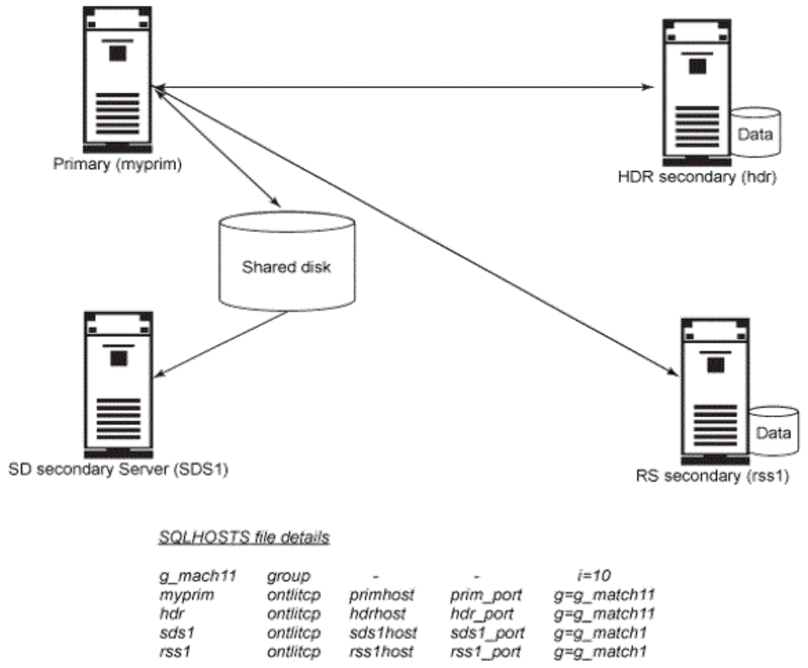
4. Bring the RS secondary into online mode

   ```
   onmode -d RSS myprim
   ```

   Result

Now RSS node is up and running! Run the `onstat -g rss` command to check RSS node status at both primary and RSS nodes. Now our cluster looks like this:

Figure 3. The cluster: Primary and one SD secondary



SQLHOSTS file details

```
g_mach11   group      -          -           i=10
myprim     onlitcp    primhost   prim_port   g=g_match11
hdr        ontlitcp   hdrhost    hdr_port    g=g_match11
sds1       ontlitcp   sds1host   sds1_port   g=g_match1
rss1       ontlitcp   rss1host   rss1_port   g=g_match1
```

Connection Manager

Connection manager is one of the distinguished technologies in HCL OneDB™. With connection manager, customers can configure applications to connect to a service level agreement (SLA) defined by the business logic rather than to a specific host and port number of a transactional database. For example, OLTP applications can be configured to connect to primary server, and reporting application can be configured to connect to secondary server. Application connectivity can be configured based on geography as well. This logical separation of application connectivity from database cluster allows transactional database to failover transparently, and support zero-downtime maintenance operations like rolling application and server upgrades.

Avoiding split brain situation during automated failovers

In physical replication scenario where there is a designated write server for transactional data, split brain situation is one of major nightmare for DBAs in case of network partitioning. Connection manager helps avoid this problem by monitoring activity at both primary and secondary servers, and failover primary server role only when majority servers agree that primary server isn't reachable. Automated failover configuration with connection manager helps with reduced recovery time objective (RTO less than 1 minute), and avoid split brain situation.

Connection manager configuration

In this example, we will configure two connection managers to monitor HDR cluster, and support client connection routing and automated failover functionality.

We will configure OLTP SLA group to redirect clients to current primary server and REPORT SLA group to redirect reporting applications to HDR secondary server.

SQLHOSTS file for the OneDB Primary + HDR cluster:

```
$ cat $ONEDB_ SQLHOSTS
g_cluster  group     -            -
# primary server connection details:
primary  onsoctcp  pri_machine    pri_port g=g_cluster
#HDR server connection details
hdr      onsoctcp  hdr_machine    hdr_port g=g_cluster
```

Configuration file for first connection manager (cm1):

```
$cat $ONEDB_HOME/etc/cm1.cfg
NAME     cm1
LOGFILE /opt/onedb/cm1.log
CLUSTER my_cluster1
{
  ONEDB_SERVER pri,hdr
  SLA oltp1 DBSERVERS=primary
  SLA report1 DBSERVERS=HDR,primary
  FOC ORDER=ENABLED TIMEOUT=5 PRIORITY=1
  CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
}
```

SQLHOSTS file for cm1 connection manager

```
$ cat $ONEDB_ SQLHOSTS
# primary server connection details:
primary  onsoctcp  pri_machine    pri_port
#HDR server connection details
hdr      onsoctcp  hdr_machine    hdr_port
#host and port number details SLAs
oltp1    onsoctcp  cm1_machine    cm1_port1
report1  onsoctcp  cm1_machine    cm1_port1
```

Command to start connection manager

`$ oncmsm -c $ONEDB_HOME/etc/cm1.cfg`

Check connection manager log /opt/onedb/cm1.log file to verify connection manager started correctly. Also verify connection manager status from primary and HDR servers using onstat command ‘onstat -g cmsm’.

Configuration file for second connection manager (cm2)

```
$cat $ONEDB_HOME/etc/cm2.cfg
NAME     cm2
LOGFILE /opt/onedb/cm2.log
CLUSTER my_cluster2
{
  ONEDB_SERVER pri,hdr
  SLA oltp2 DBSERVERS=primary
  SLA report2 DBSERVERS=HDR,primary
  FOC ORDER=ENABLED TIMEOUT=5 PRIORITY=2
 CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
}
```

SQLHOSTS file for cm2 connection manager

```
$ cat $ONEDB_ SQLHOSTS
# primary server connection details:
primary  onsoctcp  pri_machine    pri_port
#HDR server connection details
hdr      onsoctcp  hdr_machine    hdr_port
#host and port number details SLAs
oltp2    onsoctcp  cm2_machine    cm2_port1
report2  onsoctcp  cm2_machine    cm2_port1
```

Command to start cm2 connection manager

```
$ oncmsm -c $ONEDB_HOME/etc/cm2.cfg
```

Check connection manager log /opt/onedb/cm2.log file to verify connection manager started correctly. Also verify connection manager status from primary and HDR servers using onstat command ‘onstat -g cmsm’.

SQLHOSTS file for client applications

```
$ cat $ONEDB_ SQLHOSTS
# oltp SLA group
oltp             group     -              -              c=1
oltp1            onsoctcp  cm1_machine    cm1_port1      g=oltp
oltp2            onsoctcp  cm2_machine    cm2_port2      g=oltp
# report SLA group
report             group     -              -              c=1
report1            onsoctcp  cm1_machine    cm1_port1      g=report
report2            onsoctcp  cm2_machine    cm2_port2      g=report
```

Applications connecting to “oltp” group gets redirected to primary server in the high availability cluster, and applications connecting to “report” group gets redirected to secondary server, and if secondary server isn’t reachable then “report” applications gets redirected to primary server.

Ifxclone

Ifxclone perform a clone copy of the source database server instance to target server. Target server can be

1. Standard stand-alone server
2. Remote Standalone Server (RSS)
3. High-Availability Data Replication (HDR) server
4. ER and Flexible Grid participating server

Cloning OneDB server

About this task

In this topic, let's explore steps required to clone OneDB server to setup HDR secondary.

Prerequisites

1. HARDWARE:

   A second machine with:

   - Sufficient storage capacity to match the storage allocated on the original machine
   - Adequate memory and processing capacity to handle your workload in event of the primary server failing, or needing to be shutdown
2. SOFTWARE:
   - Install the same version of OneDB software on the second machine as the original machine
   - Install all User-defined types, user-defined routines, and DataBlade modules that are installed on the original/primary machine (They do not need to be registered on the secondary server)
3. Customized Scripts:
   - alarmprogram.sh
   - evidence.sh
4. Ownership and Permission of storage paths:
   - Owner and group should match the primary server (informix:informix)
   - Directories containing cooked files must have 770 permissions

Common Setup (From Source Server)

Some changes and files will be the same on both servers. We will make the changes to the files on "host1" (our "server1" machine), and then "ifxclone" will copy them to the new target host (host2). We are avoiding naming our systems "primary" and "secondary" because these are roles that can switch between the servers over time, and perhaps for long periods of time.

Trusted Server Connection

The primary server must trust the connection from the new database server. We need mutual trust when the roles of the two servers reverse. While you can do this at the OS level, we will assume you want this trust limited to the database service. This can be done by creating a file you identify in the ONCONFIG file's REMOTE_SERVER_CFG parameter in the $ONEDB_HOME/etc directory. For this example, will create a file named "trusted.hosts" in the "$ONEDB_HOME/etc" directory. This also allows HCL OneDB tools to update the trust information later.

Create an empty "trusted.hosts" file, and set the file permissions to limit write access to the instance owner(informix) and group (informix):

cp /dev/null $ONEDB_HOME/etc/trusted.hosts

chmod 640 $ONEDB_HOME/etc/trusted.hosts

Now configure the server to use the new file:

$ onmode -wf REMOTE_SERVER_CFG=trusted.hosts

Enable “ifxclone” and the “sysadmin:admin” function to setup the connectivity information on all servers in the cluster (for now it is just one).

$ onmode -wf CDR_AUTO_DISCOVER=1

You can manually edit the file the first time, but we will execute the “admin” SQL function in the “sysadmin” database using dbaccess to update the REMOTE_SERVER_CFG on all servers in the cluster (I am also adding our current server as trusted so either server can initiate a connection to the other, and this trust information will eventually be copied to our new server too):

execute function sysadmin:admin('cdr add trustedhost', 'host2 informix, host1 informix');

HDR ONCONFIG Changes

Disable temp table logging, and enable snapshot copy to be made by the “ifxclone” command:

$ onmode -wf TEMPTAB_NOLOG=1

$ onmode -wf ENABLE_SNAPSHOT_COPY=1

NOTE:

1. The TEMPTAB_NOLOG setting is only really required on the secondary. However, when the roles are reversed, leaving this enabled would affect applications that depend on rollback on TEMP Tables. So, either plan to correct immediately after making a new primary or be consistent and don’t rely on TEMP Table transactions.
2. If you are going to setup an RSS server instead of HDR secondary, we need to enable the logging of index builds. This causes all index creation operations to go through the logs (hope you have automatic log backups configured):

$ onmode -wf LOG_INDEX_BUILDS=1

ONEDB_ SQLHOSTS

The “ifxclone” command will attempt to add the new server to the ONEDB_ SQLHOSTS file, and if it already exists will not properly modify the file to define a server group the way we want. If you already have the entry for the new server we should delete it, or comment it out for now:

Database

server1 onsoctcp host1 9088

# server2 onsoctcp host2 9088

Database Logging Mode

Only databases using logging will be replicated. Databases using buffered logging can lose transactions if the log has not been flushed to disk and the server fails. This same window of vulnerability extends to the replica as well. If all databases are logging, or you don’t need the non-logging databases to be replicated, you can move on to the next step. You can verify the list of the non-logging databases, and those that are using buffered logging by running the following query against the "sysmaster" database table "sysdatabases" using "dbaccess":

select name, is_buff_log from sysmaster:sysdatabases where is_logging = 0 or is_buff_log = 1;

Notes:

- Stop the secondary servers
- Change the logging mode and take a Level 0 backup
- Restore the secondary from backup (or re-clone)

With Data Replication (DR) you should create all new Databases with some form of logging (ANSI, Unbuffered, or Buffered).

Chunk Path/Device Information

While we are still on server1, you should collect the path information for the chunks since these paths need to exist on the new server. You can collect this with the "onstat -d" command. We will discuss the output during setup of server2.

TARGET SETUP

Most, if not all, of the new server configuration can be handled by "ifxclone":

1. Copies the REMOTE_SERVER_CFG trusted host information
2. Copies the ONCONFIG: Due to the number of configuration parameters that need to be the same, it is probably easiest to let "ifxclone" copy the ONCONFIG from the original server. Otherwise you need to verify each of the ONCONFIG parameters which must be identical (listed in the Administrators Reference) have been correctly set. If needed you can have "ifxclone" override some settings using the "-c" option.
3. Setup the source and local ONEDB_ SQLHOSTS file

The "ifxclone" command cannot setup the symlinks to raw devices, or chunkfiles if they are in directories that do not exist.

Create Needed Directories and Symlinks to Raw Devices

All dbspace chunk paths used on the primary must be the same on our new server. You will need to create the matching symlinks to the corresponding physical devices. For cooked files, the "ifxclone" can create the missing chunk files automatically, even for the root dbspace, but it will not create the parent directories. The parent directories for those paths **must already exist** for "ifxclone" to successfully create the chunk files**.**

You can find the paths in the "onstat -d" output from "**server1**". For this example, "onstat -d" produced the following output, showing that the chunk paths are all under the "/chunkdir" directory:

Then, for each of those paths, make sure the parent directories exist with adequate permissions. The above example only needs the "/chunkdir" directory to exist. It also must have the ownership, and permissions, so you will need to do these operations as **root/administrator**:

$ mkdir /chunkdir

```
$ chown informix:informix /chunkdir
```

```
$ chmod 770 /chunkdir
```

Under those directories you can manually create the needed raw device symlinks (need permission of 660, same as any cooked files you choose to create manually.)

If the list of chunks is long, you can collect the paths via the following command on "server1", and use the output to create a script to handle the above steps for creating the required parent directories:

```
$ onstat -d | awk '$NF ~ /\// { print $NF; }'
```

Run "ifxclone":

The "ifxclone" will make "server1" a primary, perform "fake" backup of "server1", and restore it to "server2" as a new HDR secondary. All changes to logged spaces on "server1" will be continually applied to "server2". The "ifxclone" will add the secondary to the ONEDB_ SQLHOSTS file on the "source" server, and copy the "source" server's ONCONFIG information to the target:

```
$ ifxclone --source=server1 --sourceIP=host1 --sourcePort=9088 \

--target=server2 --targetIP=host2 --targetPort=9088 \

--trusted \

--createchunkfile \

--disposition=HDR \

--autoconf
```

Check the Progress

To monitor the Data Replication Information and message log, run the following (Ctrl-C to break out):

```
$ onstat -g dri -m -r 1
```

The server state at the top of each output burst should change from "Initialization", to "Fast Recovery", then eventually to "Read-Only (Sec)", and show it is paired with server1 and the Data Replication state is "on":

Server Failed to Initialize?

If the server failed to initialize, check the message log for errors. Correct any missing "paths", or permissions on the filesystem, or any other changes needed to the ONCONFIG file. Then rerun the "ifxclone" command, but **you must add "--useLocal"** so you don't overwrite any changes you fixed in the local ONCONFIG.

ONEDB_ SQLHOSTS is Updated

The “autoconf” option added a “group” to the server ONEDB_ SQLHOSTS file, and assigned each server in the group using the “g=” option. The group name is the name of the source server but with “g_” prepended. The ONEDB_ SQLHOSTS file from our example contains three lines now:

g_server1 group - -i=1

server1 onsoctcp host1 9088 g=g_server1

server2 onsoctcp host2 9088 g=g_server1

Verify Backup Device Configuration

Verify the ONCONFIG LTAPEDEV and TAPEDEV values (copied from server1 to server2) exist and have the correct permissions (RWX for owner and group) on the new server. This will ensure that any automatic and manual backups run as intended. If you need to change them on the secondary later, you can use “onmode -wf” command while the server is running to update the value.

Copying the alarmprogram.sh ensures that the same automatic log backup configuration is used on server2. In the event server2 becomes the primary, the log backups are already configured for operation.

REST API

The OneDB REST API

HCL OneDB provides a REST API for easy access to your data from any HTTP client. You can use REST to query your OneDB database server and get the query results back in an easy-to-use JSON format. The REST interface also supports inserts, updates, deletes, and creating new tables and databases for relational data, for TimeSeries data, and for JSON/BSON document collections.

Starting the OneDB REST API

About this task

The OneDB REST API is a Java-based mid-tier gateway server that will process HTTP requests from clients. The OneDB REST API is provided as an executable JAR file in the HCL OneDB™ APIs package.

Table 12. Prerequisites for starting the OneDB REST API:

Java 11
A OneDB server instance that is up and running

To start the OneDB REST API :

1. Create a REST configuration file.

   Sample yaml configuration file:

   ```
   # REST port number
   port: 8080

   # List of database servers that the REST API can connect to
   onedb.servers:
     -
       alias: server1
       host: host1.mycompany.com
       port: 9088
     -
       alias: server2
       host: host2.mycompany.com
       port: 9088

   # Optional, but recommended, properties to enable HTTPS
   tls.enable: true
   tls.keystore.type: pkcs12
   tls.keystore.file: rest-keystore.pkcs12
   tls.keystore.password: myPassword

   # Optional, but recommended, property to enable anti-CSRF tokens
   security.csrf.token.enable: true
   ```

   The **onedb.servers** property is required. This specifies the alias, host, and port number for the databases servers that HTTP clients will be able to access through the REST API.

   Important: In addition to **onedb.servers**, it is recommended that you configure the TLS (HTTPS) and anti-CSRF configuration properties in order to secure your REST API when running it in a production environment. See the Securing the REST API topic for more information on how to properly secure your REST API server.

   The REST API uses HTTP Basic Authentication. The REST server will connect to the OneDB database server through JDBC using the provided user and password credentials provided by the HTTP client.

2. Start the REST listener using Java 11 or higher

   ```
   java -jar onedb-rest.jar -config rest-config.yaml
   ```

   where `rest-config.yaml` is the name of your REST configuration file.

Using the OneDB Rest API

Once the REST API server is started, you can use any HTTP client to access your data through REST. Authenticate using HTTP Basic Authentication and a user and password known on the HCL OneDB™ database server. Here are some examples to get you started.

The following examples assume the REST API is running on the localhost at port 8080 and that you have a server alias called **server1** in your REST configuration file.

Example 1: Get the list of databases on the server

`GET http://localhost:8080/api/servers/server1/databases`

Response:

```
[
    "db1",
    "db2",
    "stores_demo"
]
```

Example 2: Get the list of tables in the "stores_demo" database

`GET http://localhost:8080/api/servers/server1/databases/stores_demo/tables`

Response:

```
[

    "call_type",
    "catalog",
    "classes",
    "cust_calls",
    "customer",
    ...
]
```

Example 3: Query for all rows in the "customer" table

`POST http://localhost:8080/api/servers/server1/databases/stores_demo/tables/customer/query`

Response:

```
{
    "results": [
        {
            "customer_num": 101,
            "fname": "Ludwig",
            "lname": "Pauli",
            "company": "All Sports Supplies",
            "address1": "213 Erstwild Court",
            "address2": null,
            "city": "Sunnyvale",
            "state": "CA",
            "zipcode": "94086",
            "phone": "408-789-8075"
        },
        ...
    ],
    "hasMore": false,
    "responseTime": 31
}
```

Example 4: Query for all rows in the "customer" table, sorting by "customer_num" and only including certain fields in the result

`POST http://localhost:8080/api/servers/server1/databases/stores_demo/tables/customer/query`

Request body:

```
{
    "fields": [ "customer_num", "lname", "state" ],
    "orderBy": [ { "key": "customer_num", "direction": "asc"} ]
}
```

Response:

```
{
    "results": [
        {
            "customer_num": 101,
            "lname": "Pauli",
            "state": "CA"
        },
        {
            "customer_num": 102,
            "lname": "Sadler",
            "state": "CA"
        },
        {
            "customer_num": 103,
            "lname": "Currie",
            "state": "CA"
        },
        ...
    ],
    "hasMore": false,
    "responseTime": 514
}
```

Example 5: Query the “customer” table with query condition

`POST http://localhost:8080/api/servers/server1/databases/stores_demo/tables/customer/query`

Request body:

```
{
    "fields": [ "customer_num", "fname", "lname", "city", "state" ],
    "filter": {
        "op": "and",
        "filters": [
            { "key": "state", "op": "=", "value": "CA"},
            { "key": "customer_num", "op": ">=", "value": 110 }
        ]
    }
}
```

Response:

```
{
    "results": [
        {
            "customer_num": 110,
            "fname": "Roy",
            "lname": "Jaeger",
            "city": "Redwood City",
            "state": "CA"
        },
        {
            "customer_num": 111,
            "fname": "Frances",
            "lname": "Keyes",
            "city": "Sunnyvale",
            "state": "CA"
        },
        {
            "customer_num": 112,
            "fname": "Margaret",
            "lname": "Lawson",
            "city": "Los Altos",
            "state": "CA"
        },
        {
            "customer_num": 113,
            "fname": "Lana",
            "lname": "Beatty",
            "city": "Menlo Park",
            "state": "CA"
        },
        ...
    ],
    "hasMore": false,
    "responseTime": 35
}
```

Example 6: Insert one row into a table or JSON collection

`POST http://localhost:8080/api/servers/server1/databases/mydb/collections/people/insert`

Request body:

```
{ "firstName": "John", "lastName": "Doe", "age": 31 }
```

Response:

```
{
    "n": 1,
    "responseTime": 65
}
```

Example 7: Insert multiple rows into a table or JSON collection

`POST http://localhost:8080/api/servers/server1/databases/mydb/collections/people/load`

Request body:

```
{
    "data": [
        {"firstName": "Jane", "lastName": "Doe", "age": 28},
        {"firstName": "Joseph", "lastName": "Doe", "age": 32}
    ]
}
```

Response:

```
{
    "n": 2,
    "responseTime": 72
}
```

Example 8: Update one or more rows or documents in a table or JSON collection

`POST http://localhost:8080/api/servers/server1/databases/mydb/collections/people/update`

Request body:

```
{
    "filter": { "key": "firstName", "op": "=", "value": "John" },
    "updates": {
        "age": 32
    }
}
```

Response:

```
{
    "n": 1,
    "responseTime": 44
}
```

Example 9: Delete one or more rows or documents in a table or JSON collection based on a query condition

`POST http://localhost:8080/api/servers/server1/databases/mydb/collections/people/delete`

Request body:

```
{
    "filter": { "key": "age", "op": "eq", "value": 32 }
}
```

Response:

```
{
    "n": 1,
    "responseTime": 44
}
```

Example 10: Run a SQL query directly through REST

`POST http://localhost:8080/api/servers/server1/databases/stores_demo/sql`

Request body:

```
{
    "sql": "select count(*) as count from customer"
}
```

Response:

```
{
    "results": [
        {
            "count": 28
        }
    ],
    "hasMore": false,
    "responseTime": 363
}
```

Example 11: Run a SQL query with host variables through REST

`POST http://localhost:8080/api/servers/server1/databases/stores_demo/sql`

Request body:

```
{
    "sql": "select count(*) as count from customer where state = ? and customer_num > ?",
    "hostVariables": [ "CA", 110 ]
}
```

Response:

```
[
    {
        "count": 8
    }
]
```

You can find the full OneDB REST API syntax and many more examples here or you can access the REST API's OpenAPI documentation directly from your REST API server at `http://localhost:8080/openapi`.

Using MongoDB Applications

The OneDB Wire Listener for MongoDB

HCL OneDB™ provides a wire listener that implements the MongoDB protocol. The HCL OneDB™ wire listener is a java-based mid-tier gateway server that enables communication between MongoDB clients and the HCL OneDB™ database server. This allows you to connect MongoDB applications and MongoDB client drivers directly to your HCL OneDB™ database server and communicate with HCL OneDB™ as if it was a MongoDB server. You can use MongoDB clients to interact with HCL OneDB™'s JSON/BSON collections or with relational data.

Starting the HCL OneDB™ Wire Listener for MongoDB

About this task

Starting the OneDB Wire Listener for MongoDB

The HCL OneDB™ Mongo interface is part of the HCL OneDB™ wire listener. The HCL OneDB™ wire listener comes with the HCL OneDB™ APIs package.

Table 13. Prerequisites for starting the OneDB Wire Listener:

Java 1.8
A OneDB server that is up and running

To start the OneDB Wire Listener for MongoDB:

1. Create a listener properties file

   Sample properties file:

   ```
   url=jdbc:onedb://localhost:9088/sysmaster;USER=informix;PASSWORD=password
   listener.hostName=*
   listener.port=27017
   listener.type=mongo
   mongo.api.version=3.6
   security.sql.passthrough=true
   ```

   The wire listener configuration file on page property is the JDBC url to connect to your OneDB database server. The listener.hostName property determines the network adapter or interface that the wire listener binds the server socket to (a value of * directs the wire listener to bind to all interfaces or addresses). The listener.port property specifies the port number to listen on for incoming connections from clients.

   The mongo.api.version sets your MongoDB API compatibility level. The default MongoDB compatibility level is version 3.6, but any of the following versions are supported: 3.0, 3.2, 3.4, 3.6, 4.0, and 4.2.

   Thesecurity.sql.passthrough is optional, but by setting it to true, you enable support for issuing SQL statements directly through the wire listener.

   There are also authentication options that can be specified in the properties file, which you will want to configure once you get beyond the testing stage. See User authentication with the wire listener for more details on authentication options.

2. Start the listener using java.

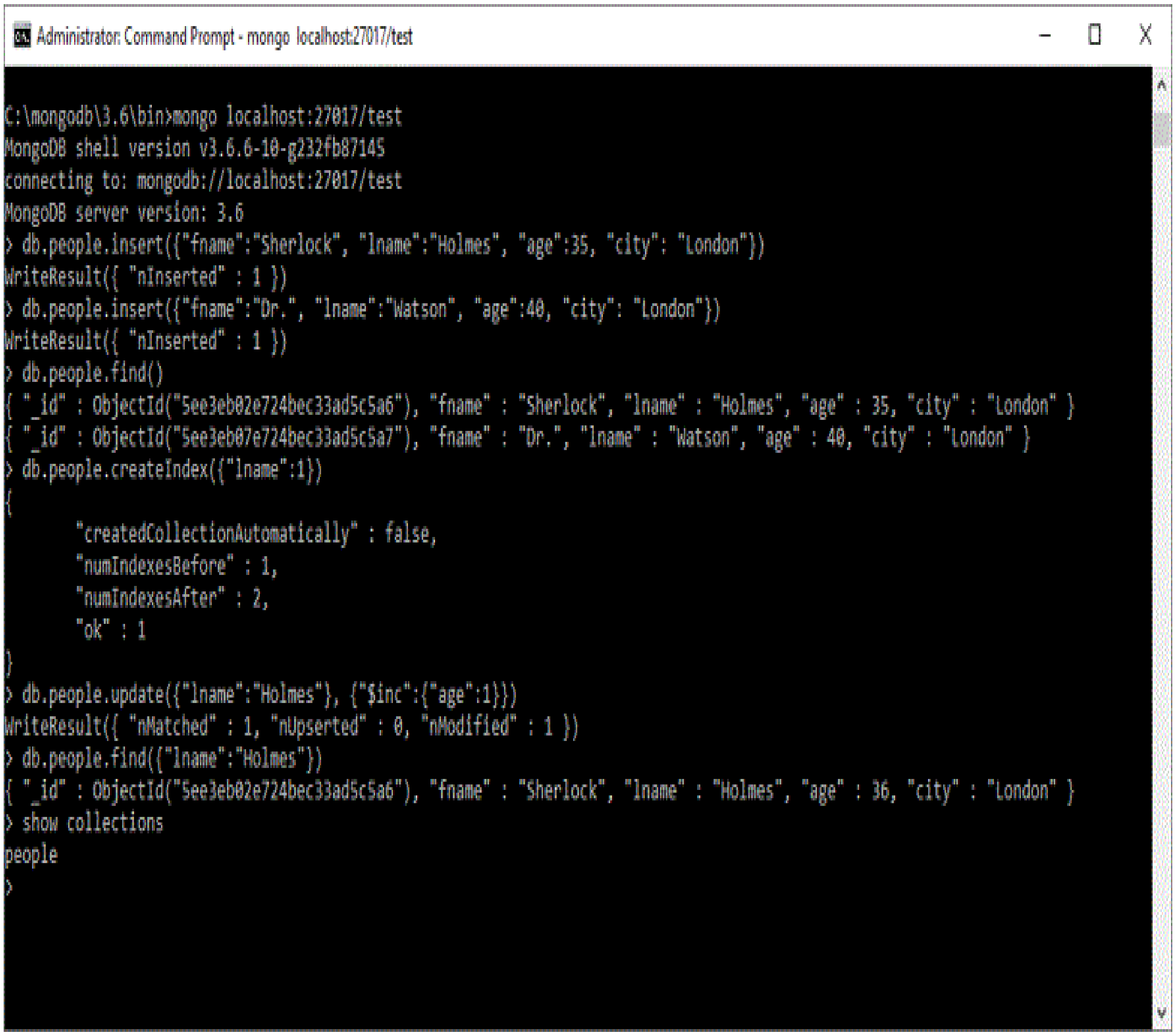
   ```
   java -jar onedb-wire-listener.jar -config mongo.properties -logfile wire_listener_mongo.log -start
   ```

Using the OneDB Wire Listener for MongoDB

About this task

Once the OneDB Wire Listener for MongoDB is started, you can use any MongoDB client or application to access your OneDB data through MongoDB queries and commands.

For example, you can connect the MongoDB shell to the wire listener and run interactive MongoDB commands against the OneDB database as shown in the screenshot below.



```
C:\mongodb\3.6\bin>mongo localhost:27017/test
MongoDB shell version v3.6.6-10-g232fb87145
connecting to: mongodb://localhost:27017/test
MongoDB server version: 3.6
> db.people.insert({"fname":"Sherlock", "lname":"Holmes", "age":35, "city": "London"})
WriteResult({ "nInserted" : 1 })
> db.people.insert({"fname":"Dr.", "lname":"Watson", "age":40, "city": "London"})
WriteResult({ "nInserted" : 1 })
> db.people.find()
{ "_id" : ObjectId("5ee3eb02e724bec33ad5c5a6"), "fname" : "Sherlock", "lname" : "Holmes", "age" : 35, "city" : "London" }
{ "_id" : ObjectId("5ee3eb07e724bec33ad5c5a7"), "fname" : "Dr.", "lname" : "Watson", "age" : 40, "city" : "London" }
> db.people.createIndex({"lname":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.people.update({"lname":"Holmes"}, {"$inc":{"age":1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.people.find({"lname":"Holmes"})
{ "_id" : ObjectId("5ee3eb02e724bec33ad5c5a6"), "fname" : "Sherlock", "lname" : "Holmes", "age" : 36, "city" : "London" }
> show collections
people
>
```

You can find the full documentation on OneDB’s Mongo support here: MongoDB API and commands.

OneDB Client Software Development Kit (CSDK)

OneDB Client Software Development Kit (CSDK), is set of connectivity APIs for OneDB database server. All applications written using, C, C++, C#, Visual Studio, MS Access, MS Excel, .NET, MSDTC, etc., requires one or more of these APIs to communicate with OneDB Server. This Getting Started guide will demonstrate how to make use of OneDB CSDK in such application scenarios. This session will cover installation, configuration and sample programs using various CSDK APIs (i.e. .NET Core Provider, ODBC and ESQL/C). ODBC and ESQL/C is shipped on all platforms, .NET Core is available on Windows and Linux platforms.

Sections in this document:

- Installing OneDB Client SDK (CSDK).
- Checking your CSDK version.
- Testing your OneDB Server connection.
- Creating an ODBC Data Source Name (DSN).
- Writing a C program using ESQL/C.
- Writing a C# program using .NET Core Provider.

This document assumes that you are running on Windows. You may need to adjust (e.g. change PATH formats) for other platforms.

Following are expected to be pre-installed to run the sample programs.

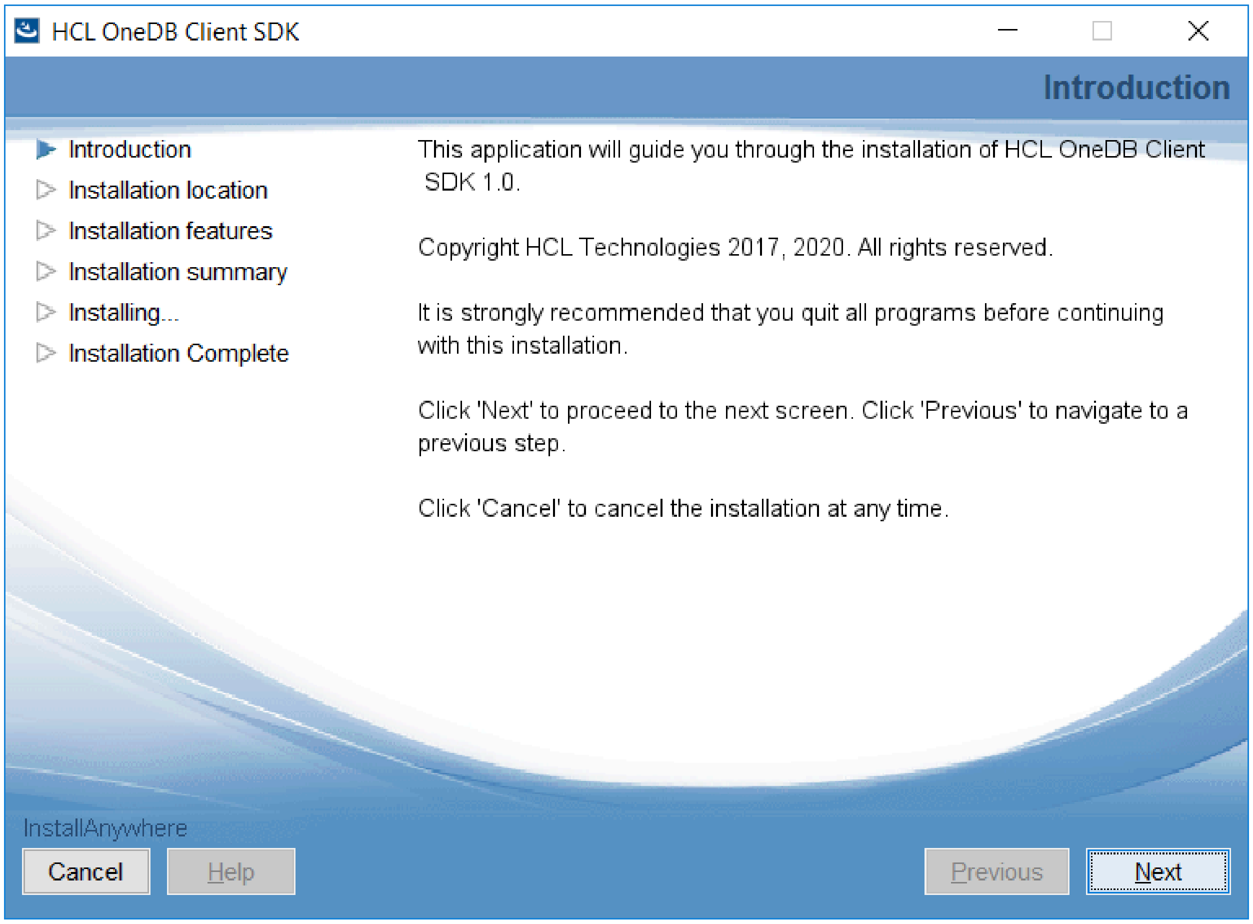
- OneDB Server must be installed and online.
- Microsoft Visual Studio 2017 is used in this document.
- .NET Core SDK/Runtime 3.1.302 is used in this document.

OneDB Client SDK (CSDK) installation and configuration

About this task

Following steps are based on the assumption that you already have OneDB CSDK installer with you.

1. Once you extract the installer, you will see file “installclientsdk.exe” file.
2. Double click in the installclientsdk.exe file to start the installation. Depending on the privilege level of your machine for the given user, you may have to run installclientsdk.exe as “Run as administrator”.
3. Click “Next”.



This application will guide you through the installation of HCL OneDB Client SDK 1.0.

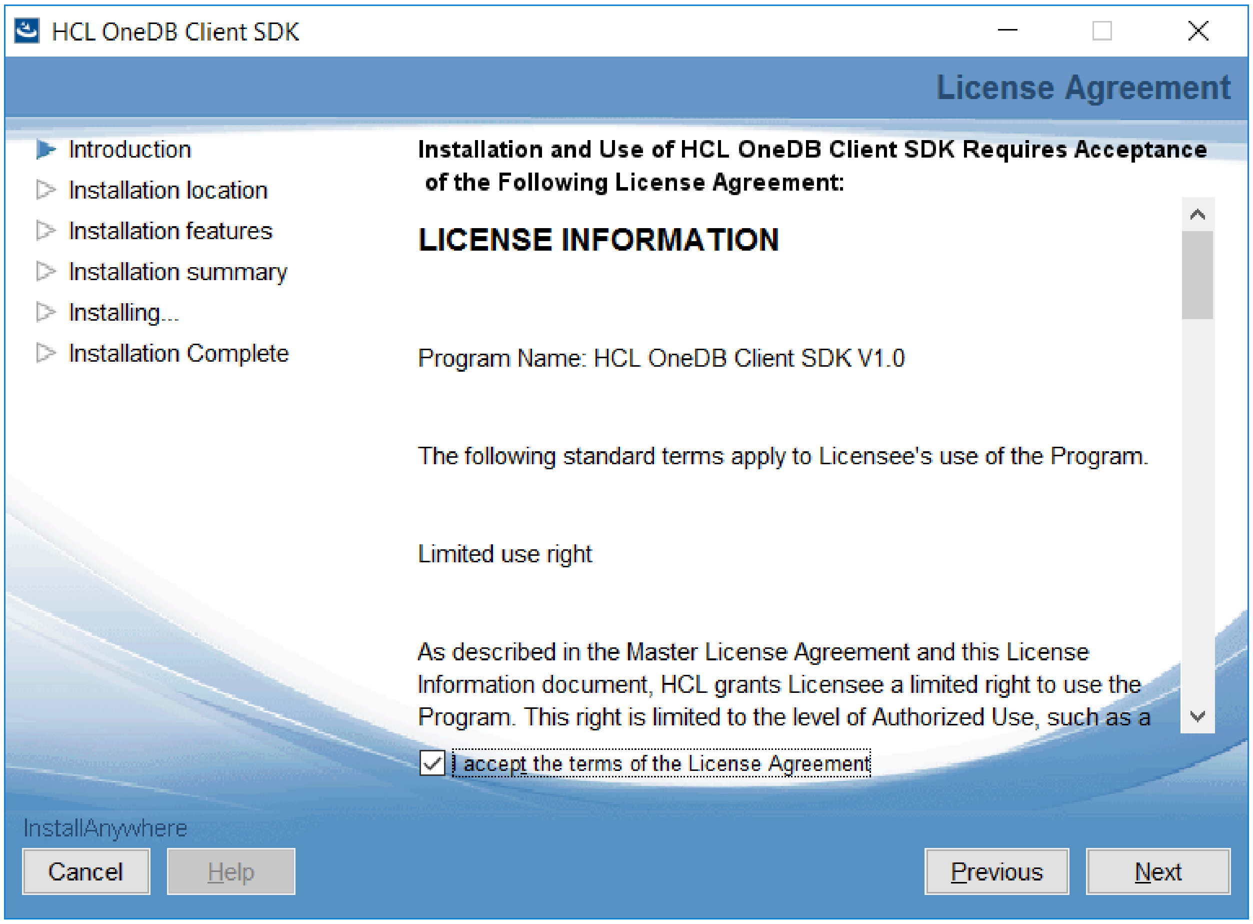
Copyright HCL Technologies 2017, 2020. All rights reserved.

It is strongly recommended that you quit all programs before continuing with this installation.

Click 'Next' to proceed to the next screen. Click 'Previous' to navigate to a previous step.

Click 'Cancel' to cancel the installation at any time.

4. Review the license Information. Choose "I accept…" radio button and press "Next" to proceed.



Installation and Use of HCL OneDB Client SDK Requires Acceptance of the Following License Agreement:

LICENSE INFORMATION

Program Name: HCL OneDB Client SDK V1.0

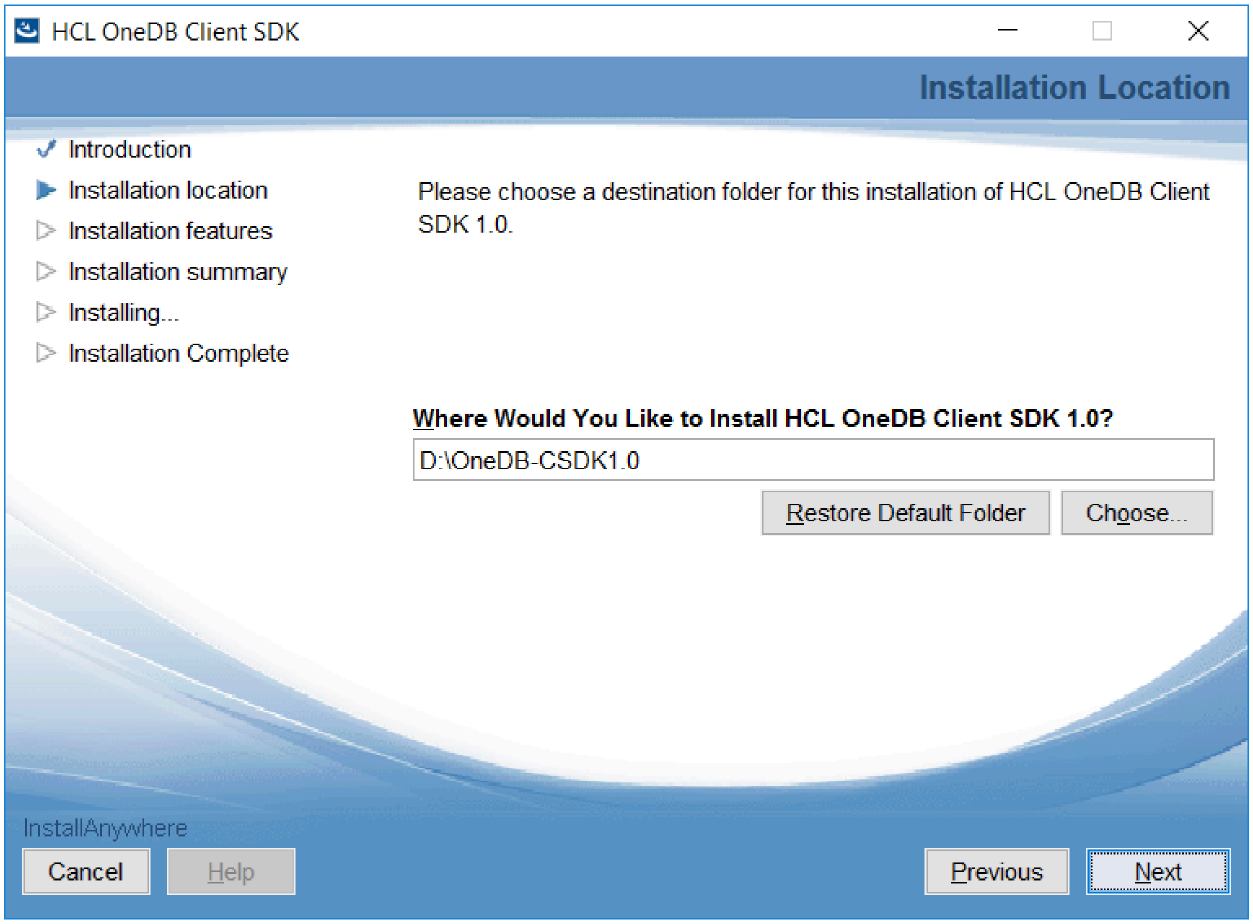
The following standard terms apply to Licensee's use of the Program.

Limited use right

As described in the Master License Agreement and this License Information document, HCL grants Licensee a limited right to use the Program. This right is limited to the level of Authorized Use, such as a

[✓] I accept the terms of the License Agreement

5. In the next window (below), provide the folder location to be installed and press “Next”


HCL OneDB Client SDK

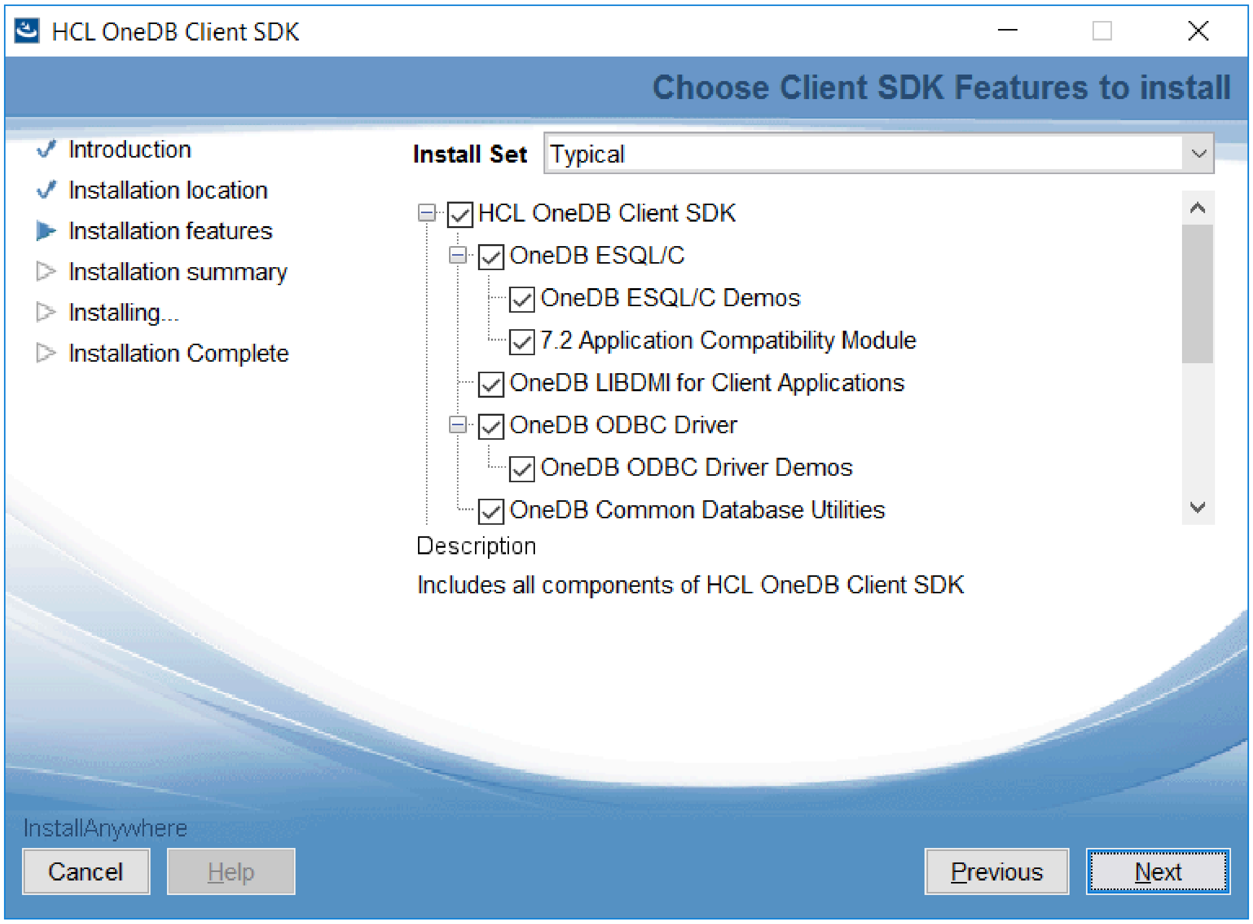
Installation Location

Please choose a destination folder for this installation of HCL OneDB Client SDK 1.0.

Where Would You Like to Install HCL OneDB Client SDK 1.0?

D:\OneDB-CSDK1.0

6. In the next window (below), press “Next”, and install all APIs of CSDK.



Choose Client SDK Features to install

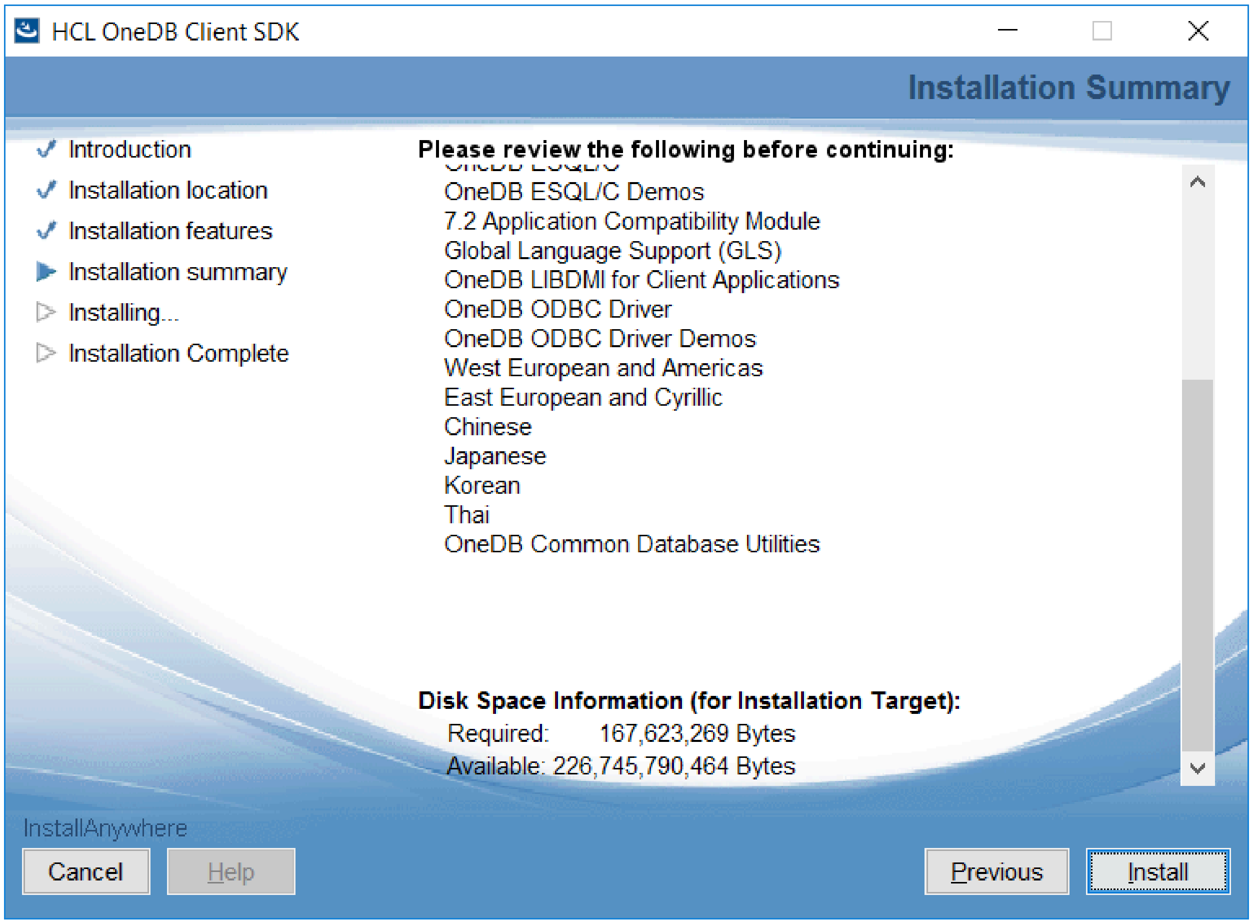
Install Set: Typical

- ☑ HCL OneDB Client SDK
  - ☑ OneDB ESQL/C
    - ☑ OneDB ESQL/C Demos
    - ☑ 7.2 Application Compatibility Module
  - ☑ OneDB LIBDMI for Client Applications
  - ☑ OneDB ODBC Driver
    - ☑ OneDB ODBC Driver Demos
  - ☑ OneDB Common Database Utilities

Description

Includes all components of HCL OneDB Client SDK

7. Click Install


HCL OneDB Client SDK

Installation Summary

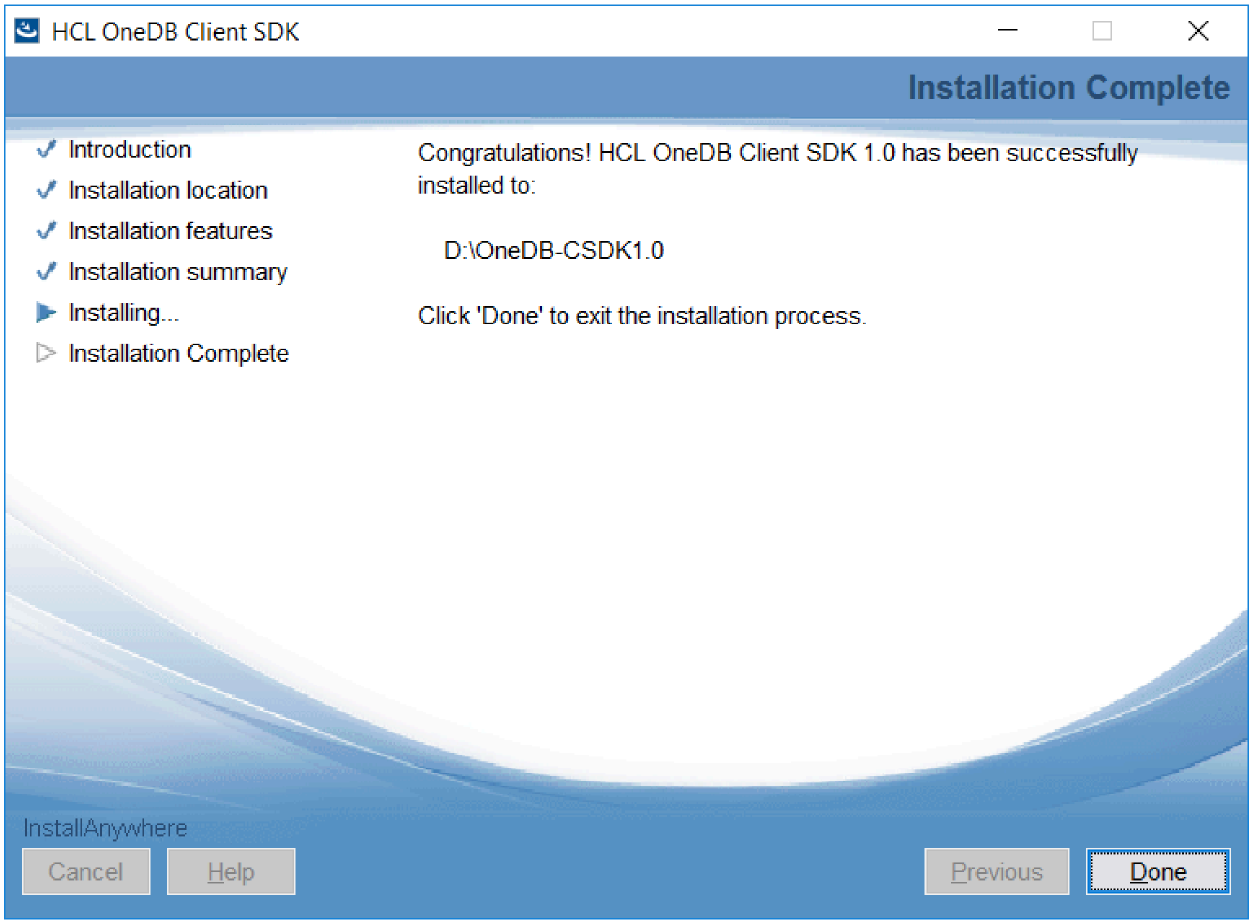
Please review the following before continuing:

OneDB ESQL/C Demos
7.2 Application Compatibility Module
Global Language Support (GLS)
OneDB LIBDMI for Client Applications
OneDB ODBC Driver
OneDB ODBC Driver Demos
West European and Americas
East European and Cyrillic
Chinese
Japanese
Korean
Thai
OneDB Common Database Utilities

Disk Space Information (for Installation Target):

Required: 167,623,269 Bytes
Available: 226,745,790,464 Bytes

8. Click on “Done” as it prompts for successful installation.

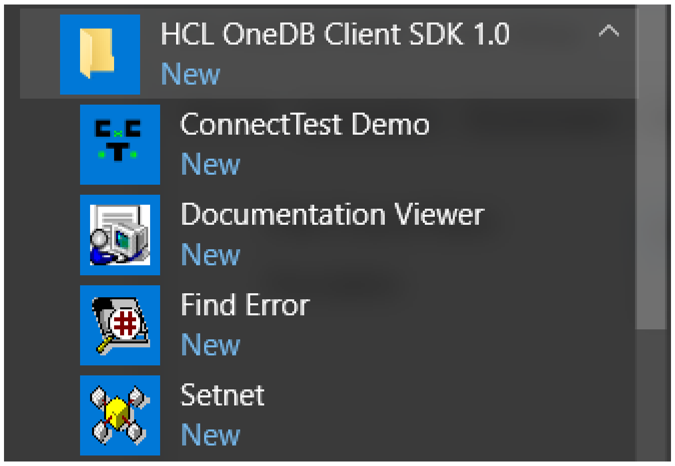


Congratulations! HCL OneDB Client SDK 1.0 has been successfully installed to:

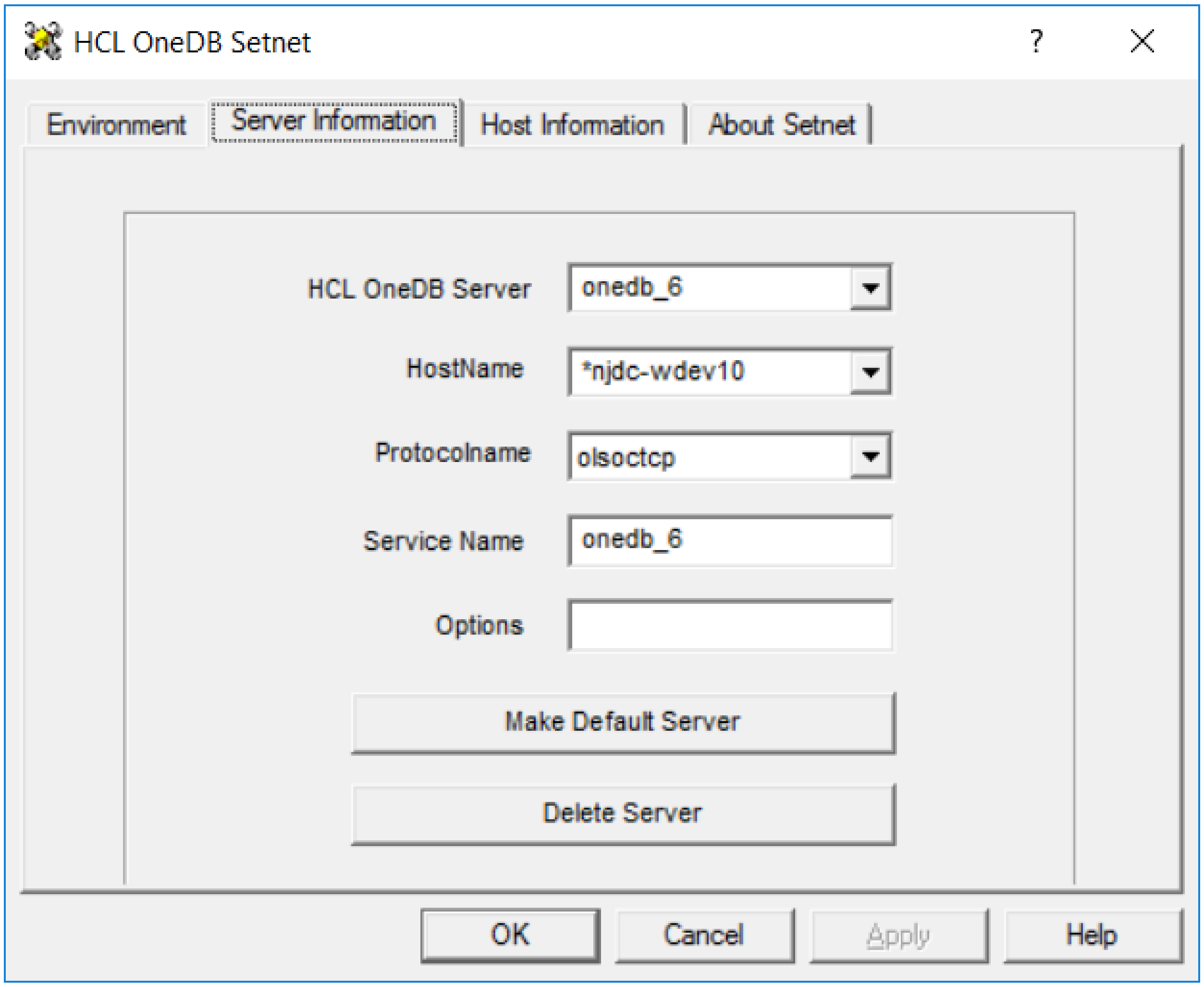
D:\OneDB-CSDK1.0

Click 'Done' to exit the installation process.

9. Post successful installation, you can go to "Start Program/Windows Symbol" at the bottom left corner of your computer and you can browse to see below list under "HCL OneDB Client SDK 1.0" menu.



10. In the above step click on "Setnet", Setnet is utility to set OneDB Client environment variables and OneDB Server configuration details. Once you invoke "Setnet", Go to "Server Information" tab and provide required information for your OneDB Server instance as mentioned below. Click on "Make Default Server" followed by "Apply" and "OK". Check the version of OneDB Client SDK installed using "esql -V" & "esql -version"
11. Using the "System Environment" of Windows, you should set environment variables:

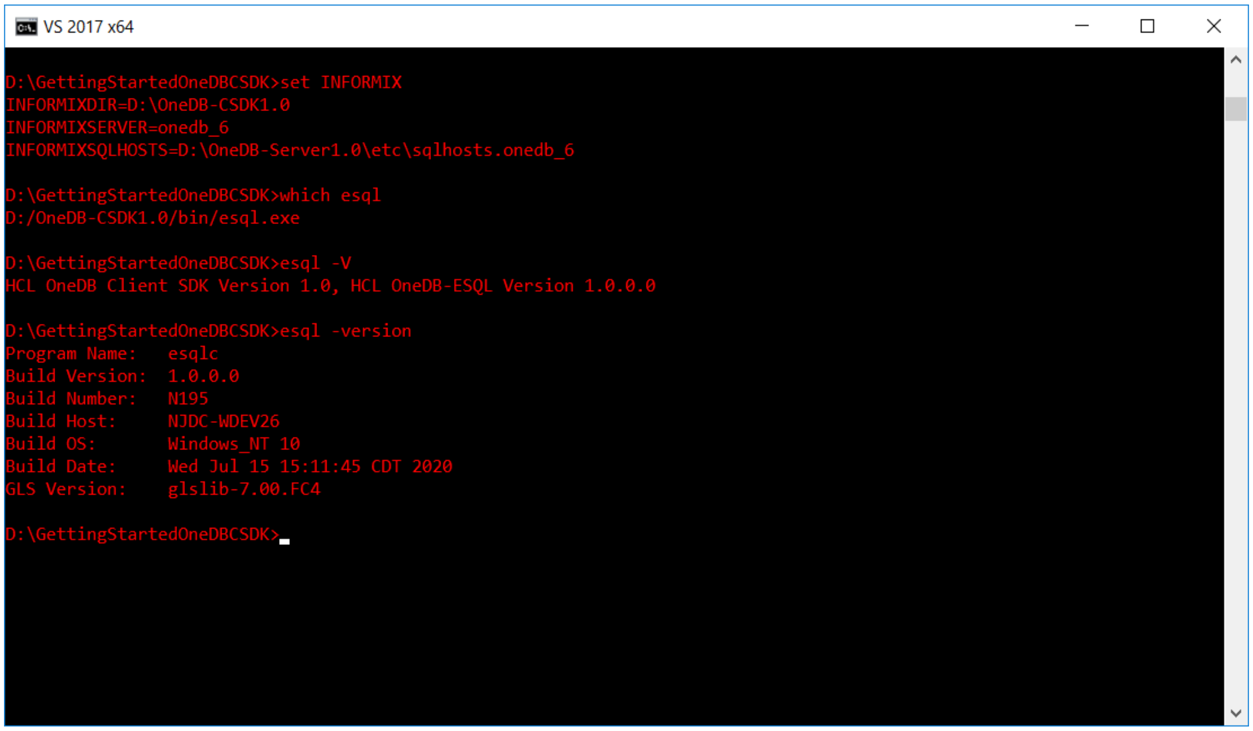


HCL OneDB Server: onedb_6
HostName: *njdc-wdev10
Protocolname: olsoctcp
Service Name: onedb_6
Options:

- ONEDB_HOME (PATH where OneDB CSDK is installed)
- ONEDB_SERVER (OneDB Server instance name)
- ONEDB_ SQLHOSTS (OneDB Server connection end point information)
- PATH (add %ONEDB_HOME%\bin)

12. Once above environment variables are set, Open the command prompt of Visual Studio 2017 (or later) and cross verify the environment set by running “set INFORMIX” command. Followed by “which esql” (this output will ensure

PATH is correctly set). **Followed by “esql -V” and “esql -version” commands**. These commands output provides the version information of OneDB Client SDK. Refer below command prompt for the commands executed and its results.



```
D:\GettingStartedOneDBCSDK>set INFORMIX
INFORMIXDIR=D:\OneDB-CSDK1.0
INFORMIXSERVER=onedb_6
INFORMIXSQLHOSTS=D:\OneDB-Server1.0\etc\sqlhosts.onedb_6

D:\GettingStartedOneDBCSDK>which esql
D:/OneDB-CSDK1.0/bin/esql.exe

D:\GettingStartedOneDBCSDK>esql -V
HCL OneDB Client SDK Version 1.0, HCL OneDB-ESQL Version 1.0.0.0

D:\GettingStartedOneDBCSDK>esql -version
Program Name:   esqlc
Build Version:  1.0.0.0
Build Number:   N195
Build Host:     NJDC-WDEV26
Build OS:       Windows_NT 10
Build Date:     Wed Jul 15 15:11:45 CDT 2020
GLS Version:    glslib-7.00.FC4

D:\GettingStartedOneDBCSDK>
```

ConnectTest Demo – Client-Server connectivity testing

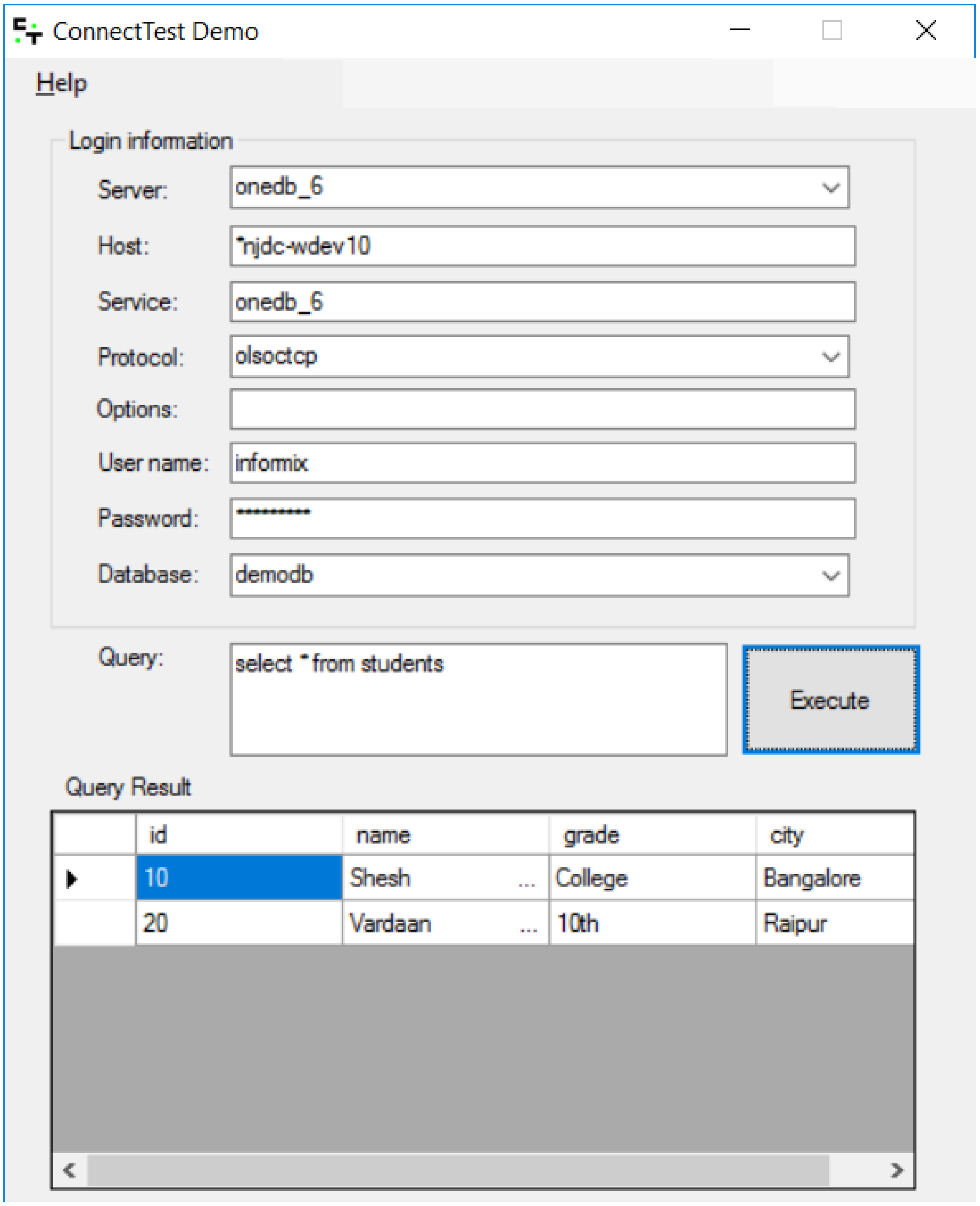
About this task

This topic provides steps for Client-Server connectivity testing.

1. Go to “Start Program/Windows Symbol” at the bottom left corner of your computer, from “HCL OneDB Client SDK 1.0” drop-down select “ConnectTest Demo”. Using Setnet we have already configured OneDB Server details.
2. Your sever name should be listed under “Server” drop down (in the below image its onedb_6), select the same and fill the “User name” and Password” fields.
3. Select the “Database” from the drop-down list, at this time, internally ConnectTest Demo program will connect to OneDB Server and get all the database names to be displayed.
4. Choose one of your databases or if you have not created one already, choose the “sysmaster” from the list.

    ```
    SELECT * FROM systables
    ```

    If you choose your own database, accordingly you need to use the table name in the SELECT query. Once you enter all the details, click on “Execute”, you should see the result of your SELECT query in the Query Result window.



Server: onedb_6

Host: *njdc-wdev10

Service: onedb_6

Protocol: olsoctcp

Options:

User name: informix

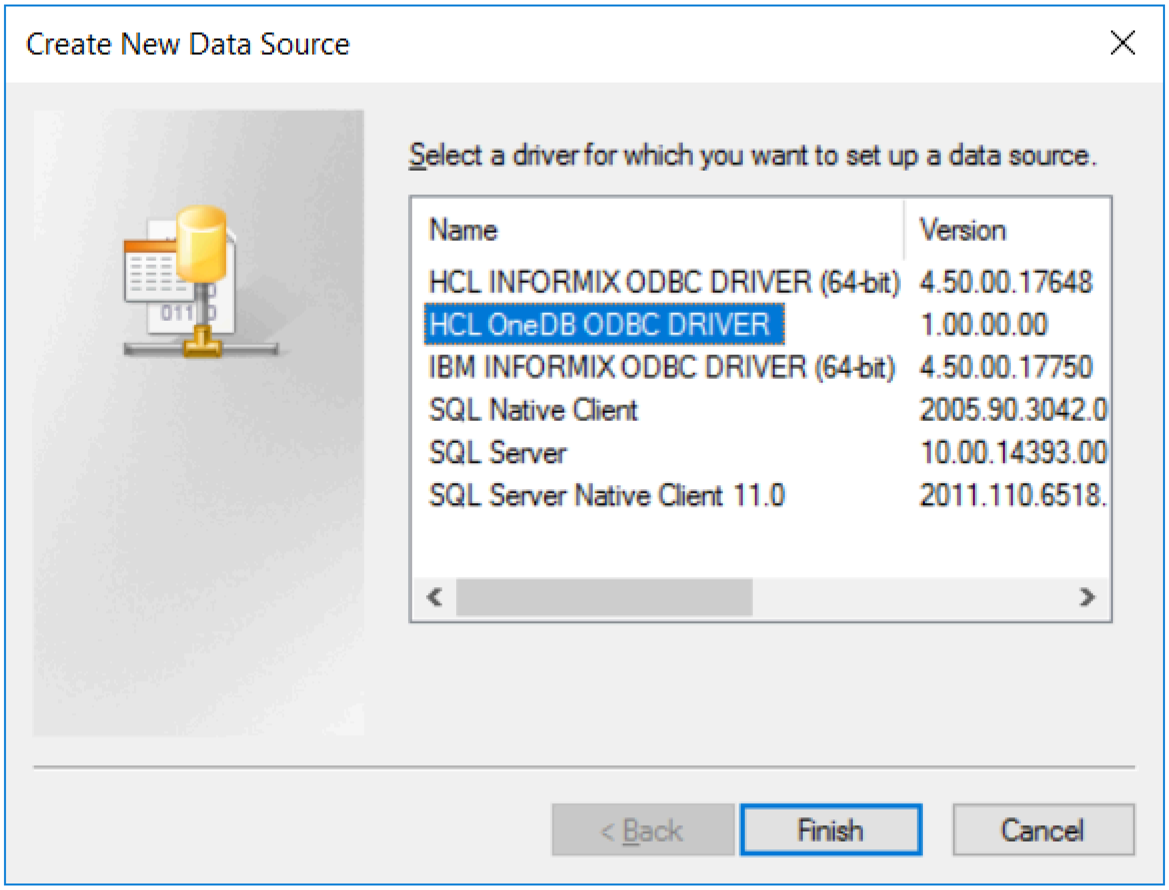
Password: **********

Database: demodb

Query: select * from students

	id	name	grade	city
▶	10	Shesh ...	College	Bangalore
	20	Vardaan ...	10th	Raipur

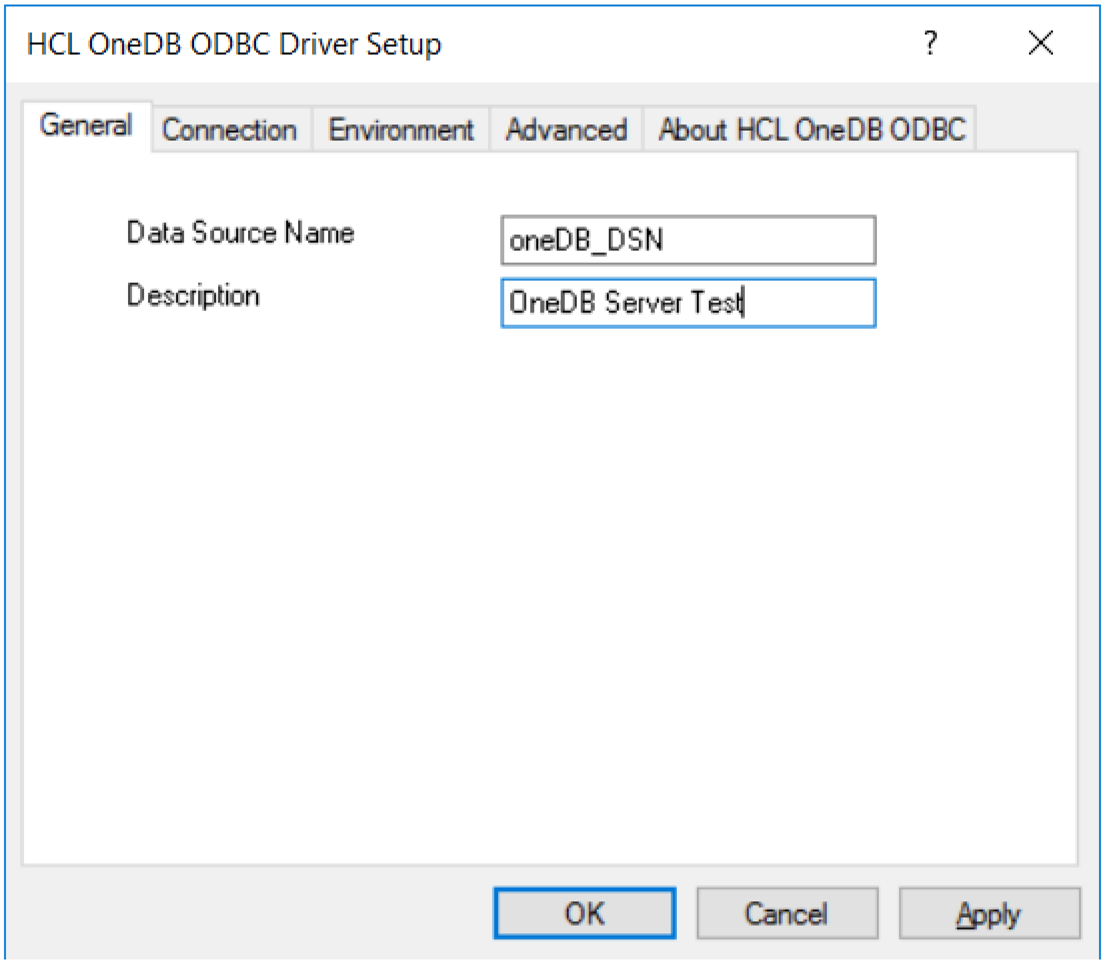
5. Create ODBC Data Source Name (DSN) and test the connection.
6. Run the “ODBC Data Source (64-bit)” utility which is typically located at `%windir%\system32\odbcad32.exe` location. Click on “New”, select “HCL OneDB ODBC DRIVER” and click on “Finish”.


Create New Data Source

Select a driver for which you want to set up a data source.

Name	Version
HCL INFORMIX ODBC DRIVER (64-bit)	4.50.00.17648
HCL OneDB ODBC DRIVER	1.00.00.00
IBM INFORMIX ODBC DRIVER (64-bit)	4.50.00.17750
SQL Native Client	2005.90.3042.0
SQL Server	10.00.14393.00
SQL Server Native Client 11.0	2011.110.6518.

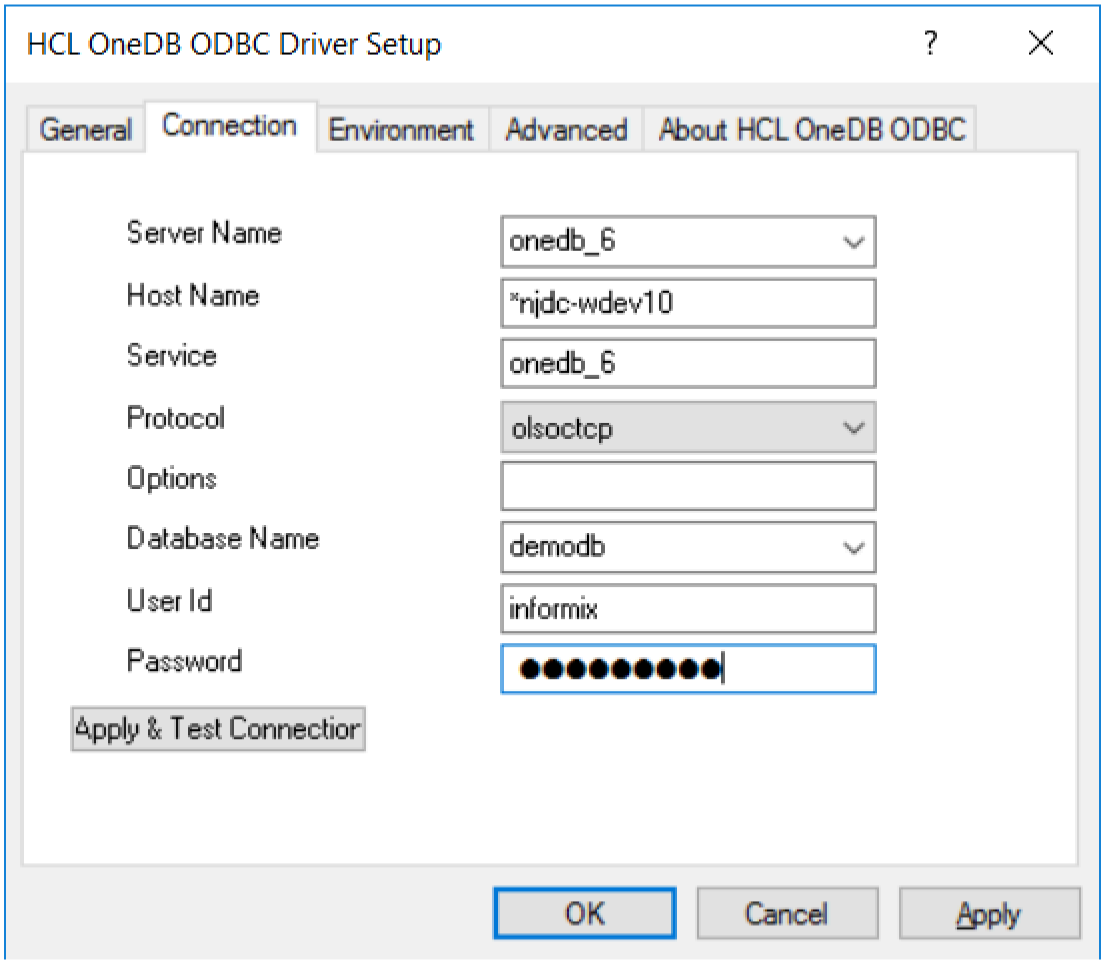
7. Provide “Data Source Name” and “Description” information and click on next tab i.e. “Connection”.



Data Source Name: oneDB_DSN

Description: OneDB Server Test

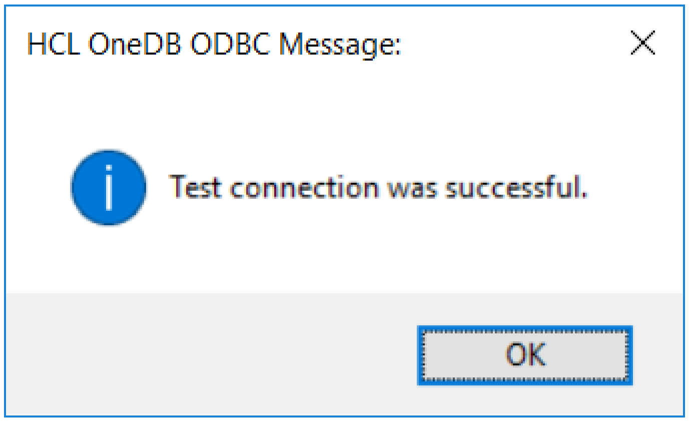
8. Select “Server Name” from the drop-down list (in the below image onedb_6), provide “User Id” and “Password” details and click on drop-down list of “Database Name”, all your database should be listed here, which the one which is of your importance. Click on “Apply and Test Connection”. It should display the success message as mentioned below.



HCL OneDB ODBC Driver Setup

Server Name	onedb_6
Host Name	*njdc-wdev10
Service	onedb_6
Protocol	olsoctcp
Options	
Database Name	demodb
User Id	informix
Password	●●●●●●●●●●

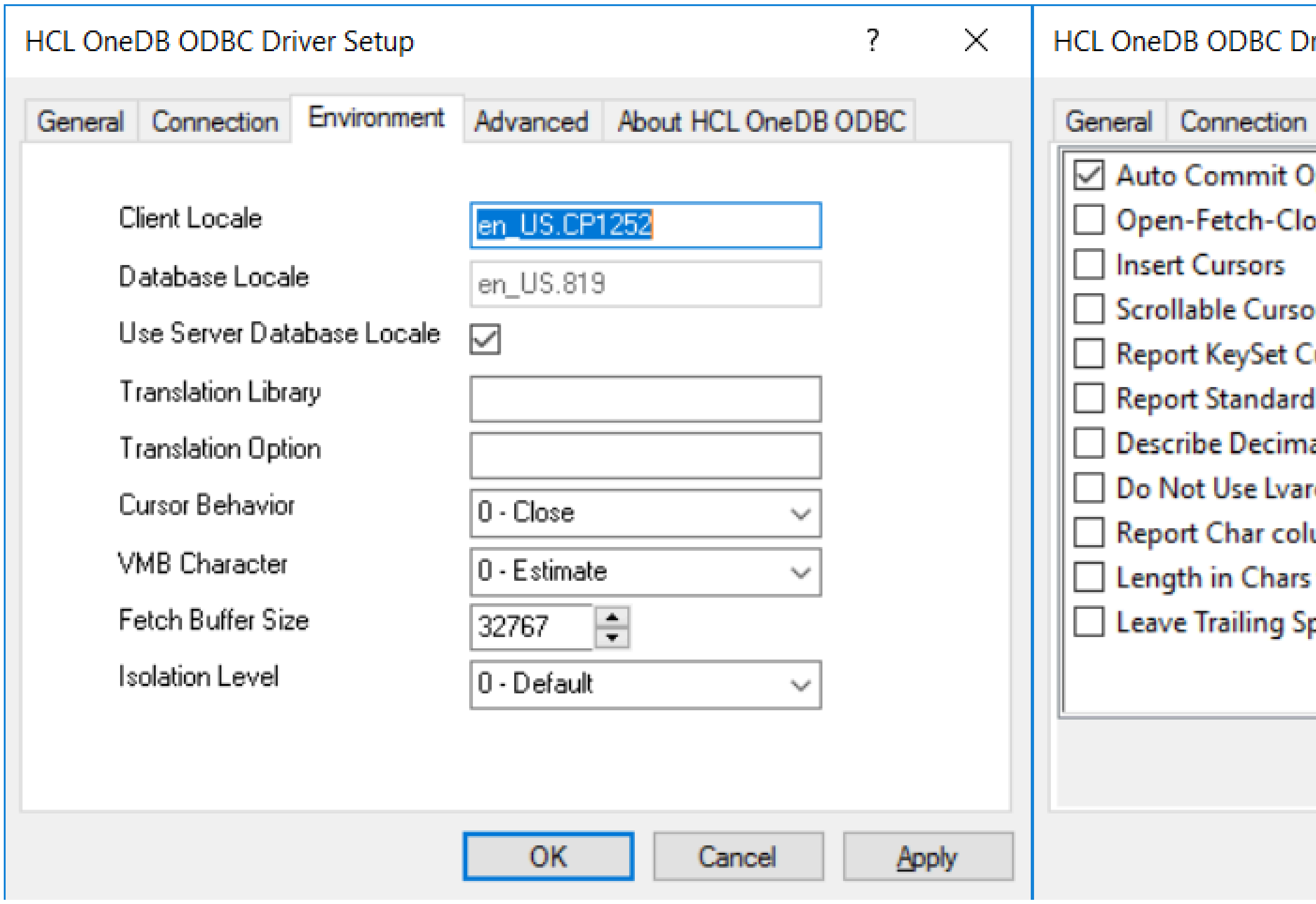
9. Click OK.



HCL OneDB ODBC Message:

Test connection was successful.

10. You can click on “OK” and close the window or feel free to visit other tabs as mentioned below for your reference. These are default values.



HCL OneDB ODBC Driver Setup

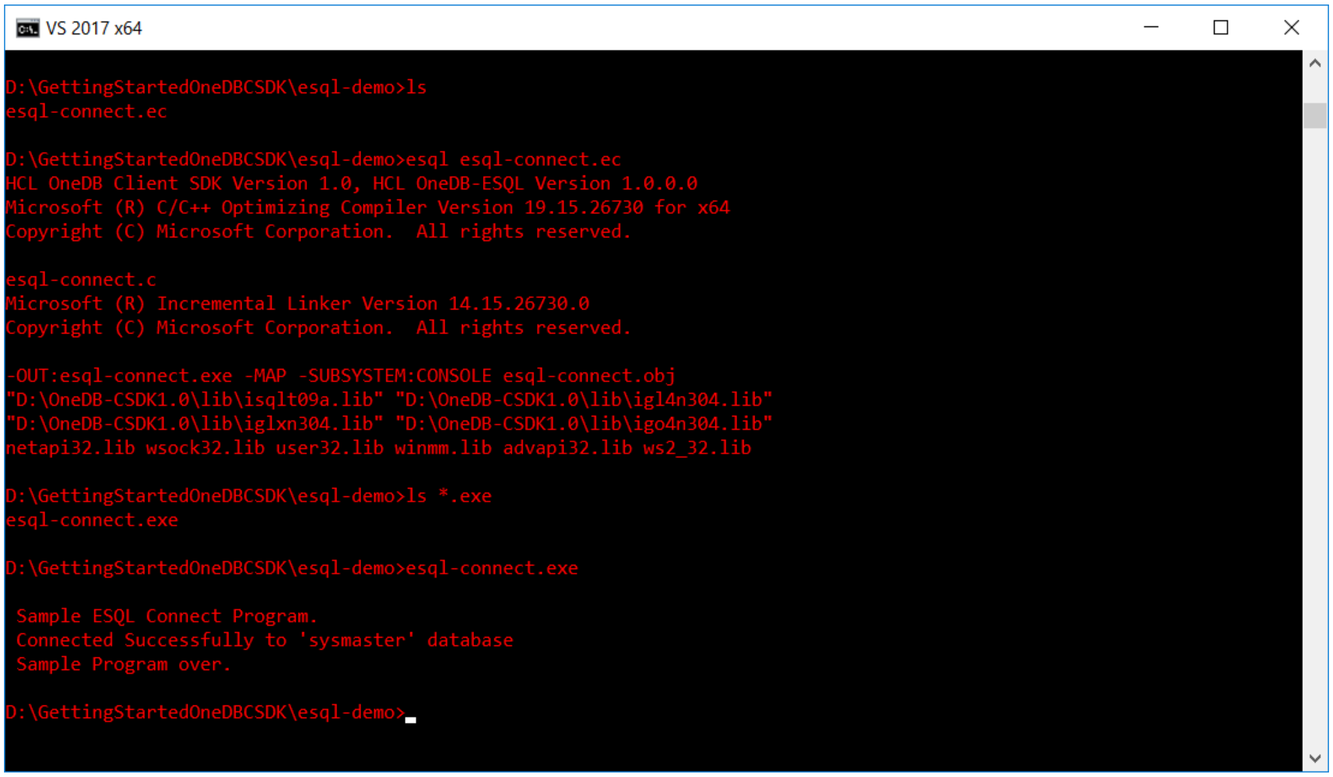
Client Locale	en_US.CP1252
Database Locale	en_US.819
Use Server Database Locale	✓
Translation Library	
Translation Option	
Cursor Behavior	0 - Close
VMB Character	0 - Estimate
Fetch Buffer Size	32767
Isolation Level	0 - Default

- ✓ Auto Commit O
- ☐ Open-Fetch-Clo
- ☐ Insert Cursors
- ☐ Scrollable Curso
- ☐ Report KeySet C
- ☐ Report Standard
- ☐ Describe Decima
- ☐ Do Not Use Lvar
- ☐ Report Char colu
- ☐ Length in Chars
- ☐ Leave Trailing Sp

Sample ESQL/C program to connect to OneDB Server

Open “VS2017 Command Prompt” (or later version) window. You can use any text editor to use the below sample ESQL/C code, name it “esql_connect.ec “ and save the file. Run “**esql esql-connect.ec**” command, it will generate **esql-connect.exe** executable file. Run this executable file. You should get the SUCCESS message. Refer below image for more information.

```
#include <stdio.h>
#include <windows.h>
main()
{
printf("\n Sample ESQL Connect Program.\n");
EXEC SQL connect to 'sysmaster';
if (SQLCODE == 0)
printf(" Connected Successfully to 'sysmaster' database\n");
else
{
printf(" Connection error !!\n");
exit(-1);
}
EXEC SQL disconnect current;
printf(" Sample Program over.\n");
}
```



VS 2017 x64

```
D:\GettingStartedOneDBCSDK\esql-demo>ls
esql-connect.ec

D:\GettingStartedOneDBCSDK\esql-demo>esql esql-connect.ec
HCL OneDB Client SDK Version 1.0, HCL OneDB-ESQL Version 1.0.0.0
Microsoft (R) C/C++ Optimizing Compiler Version 19.15.26730 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

esql-connect.c
Microsoft (R) Incremental Linker Version 14.15.26730.0
Copyright (C) Microsoft Corporation.  All rights reserved.

-OUT:esql-connect.exe -MAP -SUBSYSTEM:CONSOLE esql-connect.obj
"D:\OneDB-CSDK1.0\lib\isqlt09a.lib" "D:\OneDB-CSDK1.0\lib\igl4n304.lib"
"D:\OneDB-CSDK1.0\lib\iglxn304.lib" "D:\OneDB-CSDK1.0\lib\igo4n304.lib"
netapi32.lib wsock32.lib user32.lib winmm.lib advapi32.lib ws2_32.lib

D:\GettingStartedOneDBCSDK\esql-demo>ls *.exe
esql-connect.exe

D:\GettingStartedOneDBCSDK\esql-demo>esql-connect.exe

 Sample ESQL Connect Program.
 Connected Successfully to 'sysmaster' database
 Sample Program over.

D:\GettingStartedOneDBCSDK\esql-demo>_
```

C# sample program for.NET Core Provider to demonstrate retrieval/RecordSet of data from OneDB Server

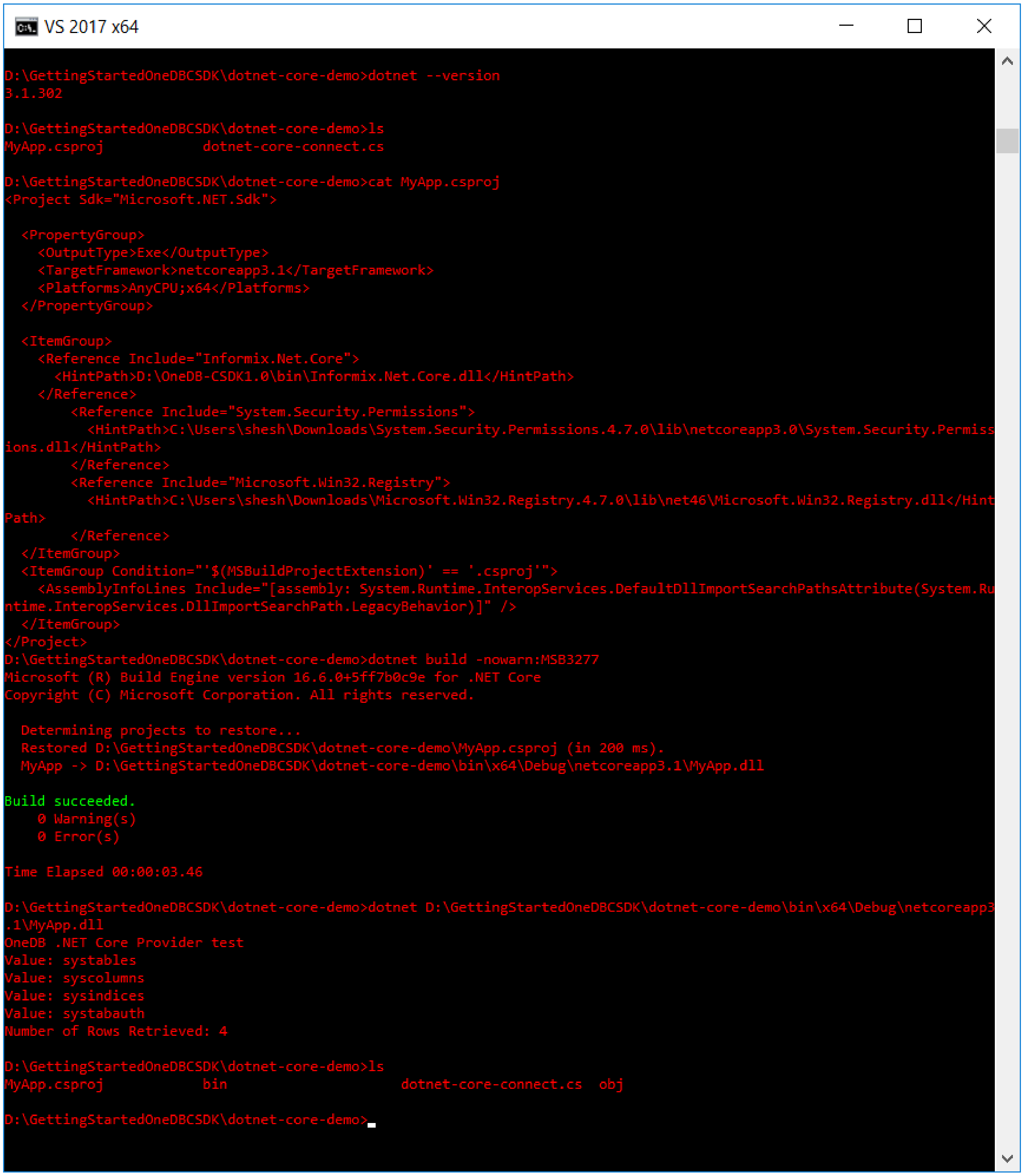
Open “VS2017 Command Prompt” (or later version) window. Below is the sample C# program to use OneDB .NET Core Provider.

It connects to “sysmaster” database and selects few records from “systables” table. You need to change the connection string in IfxConnection() call which suits your OneDB Server environment. This program uses .NET Core SDK v3.1.302 version.

You can use any text editor to use the below sample C# code, name it “dotnet-code-connect.cs “ and save the file. In the below image, you can refer the project file used, you need to have your own similar project file to suit your environment (i.e. location for files) and all dependent assemblies i.e. System.Security.Permissions, Microsoft.Win32.Registry etc installed (may be from www.nuget.org).

- In the command prompt run “dotnet build”.
- It will create the binaries for the sample program.
- You can run the binary “dotnet <name of the DLL created in the ‘dotnet build’ phase>”
- The program will display the data and number of records it fetched from the database on its successful execution.
- You can refer below image for actual outputs in the environment where its executed.

```
using System;
using System.Data;
using Informix.Net.Core;
class TestClass
{
static void Main(string[] args)
{
IfxConnection conn = null;
int rows = 0;
String value = "";
Console.WriteLine("OneDB .NET Core Provider test");
conn = new IfxConnection("Server=onedb_6;User ID=informix;password=xxxx;Database=sysmaster;");
conn.Open();
IfxCommand Selcmd = conn.CreateCommand();
Selcmd.CommandText = "select tabname from systables where tabid<5;";
Selcmd.CommandType = CommandType.Text;
Selcmd.Connection = conn;
IfxDataReader DReader = Selcmd.ExecuteReader();
while (DReader.Read())
{
rows++;
value = DReader[0].ToString();
Console.WriteLine("Value: " + value);
}
Console.WriteLine("Number of Rows Retrieved: " + rows);
DReader.Close();
conn.Close();
conn.Dispose();
}
}
```



```
D:\GettingStartedOneDBCSDK\dotnet-core-demo>dotnet --version
3.1.302

D:\GettingStartedOneDBCSDK\dotnet-core-demo>ls
MyApp.csproj            dotnet-core-connect.cs

D:\GettingStartedOneDBCSDK\dotnet-core-demo>cat MyApp.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <Platforms>AnyCPU;x64</Platforms>
  </PropertyGroup>

  <ItemGroup>
    <Reference Include="Informix.Net.Core">
      <HintPath>D:\OneDB-CSDK1.0\bin\Informix.Net.Core.dll</HintPath>
    </Reference>
        <Reference Include="System.Security.Permissions">
          <HintPath>C:\Users\shesh\Downloads\System.Security.Permissions.4.7.0\lib\netcoreapp3.0\System.Security.Permiss
ions.dll</HintPath>
        </Reference>
        <Reference Include="Microsoft.Win32.Registry">
          <HintPath>C:\Users\shesh\Downloads\Microsoft.Win32.Registry.4.7.0\lib\net46\Microsoft.Win32.Registry.dll</Hint
Path>
        </Reference>
  </ItemGroup>
  <ItemGroup Condition="'$(MSBuildProjectExtension)' == '.csproj'">
    <AssemblyInfoLines Include="[assembly: System.Runtime.InteropServices.DefaultDllImportSearchPathsAttribute(System.Ru
ntime.InteropServices.DllImportSearchPath.LegacyBehavior)]" />
  </ItemGroup>
</Project>
D:\GettingStartedOneDBCSDK\dotnet-core-demo>dotnet build -nowarn:MSB3277
Microsoft (R) Build Engine version 16.6.0+5ff7b0c9e for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  Restored D:\GettingStartedOneDBCSDK\dotnet-core-demo\MyApp.csproj (in 200 ms).
  MyApp -> D:\GettingStartedOneDBCSDK\dotnet-core-demo\bin\x64\Debug\netcoreapp3.1\MyApp.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.46

D:\GettingStartedOneDBCSDK\dotnet-core-demo>dotnet D:\GettingStartedOneDBCSDK\dotnet-core-demo\bin\x64\Debug\netcoreapp3
.1\MyApp.dll
OneDB .NET Core Provider test
Value: systables
Value: syscolumns
Value: sysindices
Value: systabauth
Number of Rows Retrieved: 4

D:\GettingStartedOneDBCSDK\dotnet-core-demo>ls
MyApp.csproj            bin                     dotnet-core-connect.cs  obj

D:\GettingStartedOneDBCSDK\dotnet-core-demo>
```

You can refer more demo/sample programs from **%ONEDB_HOME%/demo** folder of your OneDB Client SDK installation.

Server administration (from the command-line)

HCL OneDB includes utilities and applications that you can use to perform administrative tasks and capture information about configuration and performance.

onstat : Use this command to take snapshots of different conditions on your database server,

- Monitor users (onstat -u),
- transactions (onstat -x),
- Locks (onstat -k),
- Logical logs (onstat -l),
- Dbspaces & chunks (onstat -d),
- View configuration (onstat -c) and many more.

onmode: Use this command to take different actions over the database server like:

- Shutdown the server (onmode -ky),
- Create a checkpoint (onmode -c),
- Advance to next logical log (onmode -l),
- Dynamically change some configuration parameters (onmode -wm|wf) ,
- Put server in administration mode (onmode -j),
- Block/unblock the server (onmode -c block|unblock),
- Start up or remove virtual processors of a specific class (onmode -p +# class),
- Dynamic listen thread control (onmode -P [start|stop|restart] <servername>),
- Kill a specific session (onmode -z <sesid>) and many more.

onspaces: Use this command to manage storage spaces. HCL OneDB stores data in disks chunks that belongs to DBspaces, BLOBspaces, SBLOBspaces, temporary DBspaces. Use this command to:

- Add a chunk to DBspace / BLOBspace / SBLOBspace (onspaces -a <spacename>.....),
- Create DBspace / BLOBspace / SBLOBspace / PLOGspace (onspaces -c -d <spacename>....),
- Drop an empty chunk / DBspace / BLOBspace / SBLOBspace (onspaces -d <spacename>...) and many more.

onparams: Use this command to manage logical logs , physical log and buffer pools. You can:

- Add a logical log (onparams -a -d <dbspace>...),
- Add a buffer pool (onparams -b -g size ...),
- Move the physical log to a different location / size (onparams -p -s <size> -d <dbspace>...) and many more.

oncheck: Use this command to check specific disk structures for inconsistencies, repair inconsistent index structures, and display information about disk structures. You can check:

- Reserved pages (oncheck -pr / cr),
- System catalogs (oncheck -pc / cc),

- Chunk free list (oncheck -pe / ce),
- Data pages (oncheck -cd / cD),
- Index pages (oncheck -ci / cI) and many more.

onaudit: Use this command to configure auditing on the database server.

dbschema: Use this command to create a file with SQL commands to create table(s) / view(s) / database.

dbexport / dbimport: Use these commands unload / load a database from text files with included dbschema.

dbaccess: Use this command to get a a user interface for entering, running, and debugging Structured Query Language (SQL) statements and Stored Procedure Language (SPL) routines.

finderr: Use this command to look up a specific error code and display the corresponding error text.

dbimport utility

The dbimport command imports previously exported data into another database.

Syntax

```
              .----------.
              V          |
>>-dbimport----+------+-+--| Input-File Location |------------->
               +- -c--+
               +- -D--+
               +- -nv-+
               +- -q--+
               '- -X--'

                      (2)
>--| Create Options |------+-----------+--database------------><
                           +- -V-------+
                           '- -version-'
```

Element	Purpose
-c	Completes importing data even when certain nonfatal errors occur.
-D	Specifies a default extent size of 16 KB for the first and subsequent extents during the import operation, if the extent sizes are not specified in the CREATE TABLE statement.
-nv	While the dbimport -nv command is running, tables with foreign-key constraints that ALTER TABLE ADD CONSTRAINT creates in enabled or filtering mode are not checked for violations, as if NOVALIDATE has been specified.

Element	Purpose
-q	Suppresses the display of error messages, warnings, and generated SQL data-definition statements.
-V	Displays the software version number and the serial number.
-version	Extends the -V option to display additional information about the build operating system, build number, and build date.
-X	Recognizes HEX binary data in character fields.
database	Declares the name of the database to create.

dbimport input-file location options

The input-file location specifies the location of the database.exp directory, which contains the files that the dbimport utility imports.

```
|-+----------------------------------------------------------------+--|
  +- -i--directory-------------------------------------------------+
  '- -t--device--+------------------------------+--+-------------+-'
                 '- -b--blocksize-- -s--tapesize-' '- -f--pathname-'
```

Element	Purpose
-b blocksize	Specifies, in kilobytes, the block size of the tape device. If you are importing from tape, you must use the same block size that you used to export the database. If you do not use the -b option, the default block size is 1.
-f pathname	Specifies where dbimport can find the schema file to use as input to create the database when the data files are read from the tape.
-i directory	Specifies the complete path name on disk of the database.exp directory, which holds the input data files and schema file that dbimport uses to create and load the new database. The directory name must be same as the database name.
-s tapesize	Specifies, in kilobytes, the amount of data that you can store on the tape.
-t device	Specifies the path name of the tape device that holds the input files.

dbimport create options

The dbimport utility supports options for creating a database, specifying a dbspace for that database, defining logging options, and optionally, specifying ANSI/ISO-compliance or NLS case-insensitivity (or both) as properties of the database.

```
|--+--------------+--+-----------------------+--+------+--------|
   '- -d--dbspace-'  '- -l--+-+----------+-+-'  '- -ci-'
                            | '-buffered-' |
                            '- -ansi-------'
```

Element	Purpose
-ansi	Creates an ANSI/ISO-compliant database in which the ANSI/ISO rules for transaction logging are enabled. Otherwise, the database uses explicit transactions by default.
-ci	Specifies the NLS case-insensitive property. Otherwise, the database is case-sensitive by default.
-d dbspace	Specifies the dbspace where the database is created..
-l	Establishes unbuffered transaction logging for the imported database. If the -l flag is omitted, the database is unlogged.
-l buffered	Establishes the buffered transaction logging for the imported database. If -l is included but buffered is omitted, the database uses unbuffered logging.

dbexport utility

The dbexport command unloads a database into text files that you can later import into another database. The command also creates a schema file.

Remember: You must have DBA privileges or log in as user informix to export a database.

Syntax

```
>>-dbexport-----------------------------------------------------> 


     .---------------------------------------------------.
     V                                                   |
>--+---+-----------------------------------------------+-+--database-+-><
   |   +- -c    ---------------------------------------+             |
   |   +- -d-------------------------------------------+             |
   |   +- -no-data-tables--tablenames------------------+             |
   |   +- -no-data-tables-accessmethods--accessmethods-+             |
   |   +- -nw------------------------------------------+             |
   |   +- -q-------------------------------------------+             |
   |   |                                               |             |
   |   +-| Destination Options |-----------------------+             |
   |   +- -ss-----------------------------------------+             |
   |   +- -si-----------------------------------------+             |
   |   '- -X------------------------------------------'             |
   '-+-----------+--------------------------------------------------'
     +- -V-------+
     '- -version-'
```

Element	Purpose
-c	Makes dbexport complete exporting unless a fatal error occurs.
-d	Makes dbexport export simple-large-object descriptors only, not simple-large-object data.
-q	Suppresses the display of error messages, warnings, and generated SQL data-definition statements.
-ss	Generates database server-specific information for all tables in the specified database. This option specifies initial and next-extent sizes, fragmentation information if the table is fragmented, the locking mode, the dbspace for a table, the blobspace for any simple large objects, and the dbspace for any smart large objects.
-si	Excludes the generation of index storage clauses for non-fragmented tables. **Note:** The -si option is available only with the -ss option.
-X	Recognizes HEX binary data in character fields.
-no-data-tables	Prevents data from being exported for the specified tables. Only the definitions of the specified tables are exported. Accepts a comma-separated list of names of tables for which data will not be exported.
-no-data-tables-accessmethods	Prevents data from being unloaded using the specified access methods.

Element	Purpose
	Accepts a comma-separated list of names of access methods. Tables using those access methods are not unloaded.
-nw	Generates the SQL for creating a database without the specification of an owner.
-V	Displays the software version number and the serial number.
-version	Extends the -V option to display additional information about the build operating system, build number, and build date.
database	Specifies the name of the database that you want to export.

Destination Options

```
|--+-------------------------------------------------------------+--|
   +- -o--directory----------------------------------------------+
   '- -t--device-- -b--blocksize-- -s--tapesize--+---------------+-'
                                                 '- -f--pathname-'
```

Element	Purpose
-b blocksize	Specifies, in kilobytes, the block size of the tape device.
-f pathname	Specifies the name of the path where you want the schema file to be stored, if you are storing the data files on tape.
-o directory	Specifies the directory on the disk in which dbexport creates the database.exp directory. This directory holds the data files and the schema file that dbexport creates for the database.
-s tapesize	Specifies, in kilobytes, the amount of data that you can store on the tape.
-t device	Specifies the path name of the tape device where you want the text files and, possibly, the schema file stored.

When you write to disk, dbexport creates a subdirectory, database.exp, in the directory that the -o option specifies. The dbexport utility creates a file with the .unl extension for each table in the database. The schema file is written to the file database.sql. The .unl and .sql files are in the `database.exp` directory.

Troubleshooting and Getting Help

Occasionally, a series of event causes the database server to return unexpected error codes or behaviour.

HCL OneDB provides tools to gather useful information for the support team to troubleshoot and provide a solution. Some useful utilities are:

ifxcollect: This utility gathers pertinent onstats information for support engineer to diagnose specific problems such as: performance, connection failures, Enterprise Replication issues, Assert Failures, etc. It also provides option for uploading the gathered information to a support FTP site.

Output files that ifxcollect commands generate are in the $OneDB Installationdir/isa/data directory.

The type of data that is collected per category and subcategory is in predefined XML files in the $OneDB Installationdir/isa/ directory. These XML files can be modified to add or remove specific commands.

onmode -I : Use this onmode option to instruct the database server to perform the diagnostics collection procedures and generate an Assert Failure for support engineers analysis. To use onmode -I when you encounter an error number, supply the iserrno and an optional session ID.

Output files that onmode -I generates are in the location indicated by the DUMPDIR onconfig parameter, depending on the DUMPSHMEM onconfig setting you might also get a shared memory dump. Please consult with a support engineer on this command use and mentioned settings.

onmode -Y or SET EXPLAIN ON : You can use this onmode utility to dynamically set the explain output or SET EXPLAIN output to study the query plans of an application. The output of the SET EXPLAIN statement shows decisions that the query optimizer makes. It shows whether parallel scans are used, index scan(s), sequential scan(s), the maximum number of threads required to answer the query, and the type of join used for the query.

Other troubleshooting options are available, you might get instructed by a support engineer to activate any of these options in order to get additional information on the unexpected behaviour. On any case please provide us with a detailed information on the unexpected behaviour you are experiencing and get help from our support team.

For opening a support ticket visit the HCL support portal (https://www.hcltech.com/software/support)

Index