

Connect SQL to Databricks

We need to create SQL keys in Key vault for that we need to get these details from SQL database

SQL Database-> Settings -> Connection strings -> JDBC

`jdbc:sqlserver://deepthi.database.windows.net:1433;database=dbnew;`

`user=admin12@deepthi;password={your password here};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;`

here URL will be `jdbc:sqlserver://deepthi.database.windows.net:1433;database=dbnew;`

username is admin12

password is *****

as per my account details

Now got to key vault -> Objects-> Secrets-> creates keys

URL key:

Home > deepthikv | Secrets >

Create a secret ...

Upload options: Manual

Name *: URL

Secret value *: *****

Content type (optional):

Set activation date: ☐

Set expiration date: ☐

Enabled: Yes No

Tags: 0 tags

Create Cancel

Username:

Create a secret



Upload options	Manual
Name *	Username ✓
Secret value *	***** ✓
Content type (optional)	
Set activation date	<input type="checkbox"/>
Set expiration date	<input type="checkbox"/>
Enabled	Yes No
Tags	0 tags

[Create](#)[Cancel](#)

Password:

Create a secret



Upload options	Manual
Name *	Passwordq ✓
Secret value *	***** ✓
Content type (optional)	
Set activation date	<input type="checkbox"/>
Set expiration date	<input type="checkbox"/>
Enabled	Yes No
Tags	0 tags

[Create](#)[Cancel](#)

3 keys are created

[+ Generate/Import](#) [Refresh](#) [Restore Backup](#) [Manage deleted secrets](#) [View sample code](#)

i The secret 'Password' has been successfully created.

Name	Type	Status	Expiration date
Password		✓ Enabled	
Username		✓ Enabled	
URL		✓ Enabled	

Now go to data bricks and create scope is it is not created

Using: <https://adb-3370721355401501.1.azuredatabricks.net/?o=3370721355401501#secrets/createScope>

Create Secret Scope

Cancel

Create

A store for secrets that is identified by a name and backed by a specific store type. [Learn more](#)

Scope Name ?

askvconnection

Manage Principal ?

Creator

Azure Key Vault ?

DNS Name

https://deepthikv.vault.azure.net/

Resource ID

9ae2b/resourceGroups/DKResources/providers/Microsoft.KeyVault/vaults/deepthikv

Now open a notebook and check if the scope is created correctly using

`Dbutils.secrets.listScope()` command

```
▶ 12:54 PM (2s) 2
dbutils.secrets.listScopes()
[SecretScope(name='askvconnection')]
```

```
▶ 12:55 PM (1s) 3 Python
dbutils.secrets.list('askvconnection')
[SecretMetadata(key='Password'),
 SecretMetadata(key='URL'),
 SecretMetadata(key='Username')]
```

Now we need to make a connection with SQL using the key which we created

Using code

```
username = dbutils.secrets.get(scope = "askvconnection", key = "Username")
```

```
password = dbutils.secrets.get(scope = "askvconnection", key = "Password")
```

```
jdbcUrl = dbutils.secrets.get(scope = "askvconnection", key = "URL")
```

```
df = (spark.read
      .format("jdbc")
      .option("url", jdbcUrl)
      .option("dbtable", "SalesLT.Product")
      .option("user", username)
      .option("password", password)
      .load()
    )
```

Instead of creating variables separately we directly give the key values in read code

Like below

```
df = (spark.read
      .format("jdbc")
      .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
      .option("dbtable", "SalesLT.Product")
      .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
      .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
      .load()
    )
```



The screenshot shows a Jupyter Notebook interface. At the top, it displays the time '12:59 PM (1s)', a tab count of '5', and the language 'Python'. Below this is a code cell containing the second code snippet. The code is as follows:

```
df = (spark.read
      .format("jdbc")
      .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
      .option("dbtable", "SalesLT.Product")
      .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
      .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
      .load()
    )
```

Below the code cell, the output is displayed as a single line: `df: pyspark.sql.dataframe.DataFrame = [ProductID: integer, Name: string ... 15 more fields]`.

In the above code we are using table name directly instead we can use a query to get the data as well

```
df_Blackprod= (spark.read
               .format("jdbc")
```

```

.option("url", jdbcUrl)

.option("query", "SELECT * FROM SalesLT.Product where color='Black'")

.option("user", username)

.option("password", password)

.load()

)

display(df_Blackprod)

```

The screenshot shows a Databricks notebook interface. At the top, a status bar indicates 'Just now (1s)' and '7' (likely a cell number). Below the code editor, a message says '(1) Spark Jobs'. A variable declaration shows 'df_Blackprod: pyspark.sql.dataframe.DataFrame = [ProductID: integer, Name: string ... 15 more fields]'. The main part of the image is a table view of the DataFrame. The table has 8 columns: ProductID, Name, ProductNumber, Color, StandardCost, and ListPrice. It contains 7 rows of data, all representing black products.

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000
2	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900
3	722	LL Road Frame - Black, 58	FR-R38B-58	Black	204.6251	337.2200
4	723	LL Road Frame - Black, 60	FR-R38B-60	Black	204.6251	337.2200
5	724	LL Road Frame - Black, 62	FR-R38B-62	Black	204.6251	337.2200
6	736	LL Road Frame - Black, 44	FR-R38B-44	Black	204.6251	337.2200
7	737	LL Road Frame - Black, 48	FR-R38B-48	Black	204.6251	337.2200

We can also write back to SQL using code:

```

(df_Blackprod.write

.format("jdbc")

.option("url", jdbcUrl)

.option("dbtable", "dbo.BlackProduct")

.option("user", username)

.option("password", password)

.save()

)

```

Here we are creating a table dbo.BlackProduct with only black products data

```

(df_Blackprod.write
    .format("jdbc")
    .option("url", jdbcUrl)
    .option("dbtable", "dbo.BlackProduct")
    .option("user", username)
    .option("password", password)
    .save()
)

```

▼ (1) Spark Jobs
 ▶ Job 2 [View](#) (Stages: 1/1)

Table is created in SQL database we can check by using SSMS

SQLQuery2.sql - deepthi.database.windows.net (admin12 (89)) - Microsoft SQL Server Management Studio

File Edit View Project Tools Window Help

Quick Launch (Ctrl+Q)

Execute

dbnew

Object Explorer

Connect

deepthi.database.windows.net (SQL Se)

Databases

System Databases

dbnew

Database Diagrams

Tables

System Tables

External Tables

Graph Tables

BlackProduct

dbo.BuildVersion

dbo.ErrorLog

Sales.T.Address

Sales.T.Customer

Sales.T.CustomerAddress

Sales.T.Product

Sales.T.ProductCategory

Sales.T.ProductDescription

Sales.T.ProductModel

Sales.T.ProductModelProduct

Sales.T.SalesOrderDetail

Sales.T.SalesOrderHeader

Dropped Deleted Tables

Views

External Resources

Synonyms

Programmability

Query Store

Extended Events

SQLQuery2.sql - de...new (admin12 (89))

SQLQuery1.sql - de...new (admin12 (66))

SELECT TOP (1000) [ProductID], [Name], [ProductNumber], [Color], [StandardCost], [ListPrice], [Size], [Weight], [ProductCategoryID], [ProductModelID], [SellStartDate], [SellEndDate], [DiscontinuedDate], [ThumbnailPhoto], [ThumbnailPhotoFileName], [rowguid]

100 %

Results Messages

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID	SellStartDate	SellEndDate
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000	58	1016.04	18	6	2002-06-01 00:00:00.000	NULL
2	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900	NULL	NULL	35	33	2005-07-01 00:00:00.000	NULL
3	722	LL Road Frame - Black, 58	FR-R38B-58	Black	204.6251	337.2200	58	1115.83	18	9	2005-07-01 00:00:00.000	NULL
4	723	LL Road Frame - Black, 60	FR-R38B-60	Black	204.6251	337.2200	60	1124.90	18	9	2005-07-01 00:00:00.000	NULL
5	724	LL Road Frame - Black, 62	FR-R38B-62	Black	204.6251	337.2200	62	1133.98	18	9	2005-07-01 00:00:00.000	NULL
6	736	LL Road Frame - Black, 44	FR-R38B-44	Black	204.6251	337.2200	44	1052.33	18	9	2005-07-01 00:00:00.000	NULL
7	737	LL Road Frame - Black, 48	FR-R38B-48	Black	204.6251	337.2200	48	1070.47	18	9	2005-07-01 00:00:00.000	NULL
8	738	LL Road Frame - Black, 52	FR-R38B-52	Black	204.6251	337.2200	52	1088.62	18	9	2005-07-01 00:00:00.000	NULL
9	814	ML Mountain Frame - Black, 38	FR-M63B-38	Black	185.8193	348.7600	38	1238.30	16	15	2006-07-01 00:00:00.000	2007-06-30
10	830	ML Mountain Frame - Black, 40	FR-M63B-40	Black	185.8193	348.7600	40	1256.44	16	14	2006-07-01 00:00:00.000	2007-06-30
11	743	HL Mountain Frame - Black, 42	FR-M94B-42	Black	739.0410	1349.6000	42	1233.76	16	5	2005-07-01 00:00:00.000	NULL
12	744	HL Mountain Frame - Black, 44	FR-M94B-44	Black	699.0928	1349.6000	44	1251.91	16	5	2005-07-01 00:00:00.000	2006-06-30
13	745	HL Mountain Frame - Black, 48	FR-M94B-48	Black	699.0928	1349.6000	48	1270.05	16	5	2005-07-01 00:00:00.000	2006-06-30

Query executed successfully.

deepthi.database.windows.net admin12 (89) dbnew 00:00:00 89 rows

As read and write codes can be used multiple times for different tables, we can use functions to reuse the same code multiple times, we can achieve this by writing the reusable code in different notebook and use it in any notebook.

We define function using def key word.

```
def read_sql(tablename):
```

```
    df = (spark.read
```

```
        .format("jdbc")
```

```
        .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
```

```
        .option("dbtable", tablename)
```

```
        .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
```

```
        .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
```

```
.load()
)
return(df)
```

This is how a function is defined, run this code and can call this function.

Read using table name



```
def read_sql(tablename):
    df = (spark.read
          .format("jdbc")
          .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
          .option("dbtable", tablename)
          .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
          .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
          .load()
    )
    return(df)
```

Read data using query

def read_query_sql(query):

```
df = (spark.read
      .format("jdbc")
      .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
      .option("query", query)
      .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
      .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
      .load()
)
return(df)
```

Read using Query



```
def read_query_sql(query):
    df = (spark.read
          .format("jdbc")
          .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))
          .option("query", query)
          .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))
          .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))
          .load()
    )
    return(df)
```

Write to SQL

Write to SQL

```
def write_sql(df, mode, tablename):  
    (df.write  
     .mode(mode)  
     .format("jdbc")  
     .option("url", dbutils.secrets.get(scope = "askvconnection", key = "URL"))  
     .option("dbtable", tablename)  
     .option("user", dbutils.secrets.get(scope = "askvconnection", key = "Username"))  
     .option("password", dbutils.secrets.get(scope = "askvconnection", key = "Password"))  
     .save()  
    )
```

Now to use these functions code in another notebook we need to use line command
%run/notebook path

%run/[/Workspace/Users/deepthi.reddy128@gmail.com/MetaCode](#)

Use this code in the notebook where we want to use the reusable code

The screenshot shows a Databricks notebook interface. At the top, the title is "SQL connection to Data Bricks". Below the title, there's a toolbar with "Run all", "demo cluster", "Schedule", and "Share" buttons. The notebook content includes a code cell with the following text:

```
%run /Workspace/Users/deepthi.reddy128@gmail.com/MetaCode
```

Below the code cell, there's a list of topics:

- Using functions here
- Read using table name
- Read using Query
- Write to SQL

Using read_sql() function in another notebook to read data

SQL connection Python ☆

File Edit View Run Help [Last edit was now](#)

▶ Run all demo cluster Schedule Share

Just now (2s) 10 Python

```
display(read_sql("SalesLT.Product"))
```

▶ (1) Spark Jobs

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	1059.3100	1431.5000
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	13.0863	34.9900
4	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900
5	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000
6	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000
7	711	Sport-100 Helmet, Blue	HL-U509-B	Blue	13.0863	34.9900
8	712	AWC Logo Cap	CA-1098	Multi	6.9223	8.9900
9	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	38.4923	49.9900
10	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900
11	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900
12	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900
13	717	HL Road Frame - Red, 62	FR-R92R-62	Red	868.6342	1431.5000
14	718	HL Road Frame - Red, 44	FR-R92R-44	Red	868.6342	1431.5000

Using read_query_sql() function

SQL connection Python ☆

File Edit View Run Help [Last edit was now](#)

▶ Run all demo cluster Schedule Share

Just now (1s) 11 Python

```
display(read_query_sql("SELECT * FROM SalesLT.Product where color='Black'"))
```

▶ (1) Spark Jobs

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000
2	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900
3	722	LL Road Frame - Black, 58	FR-R38B-58	Black	204.6251	337.2200
4	723	LL Road Frame - Black, 60	FR-R38B-60	Black	204.6251	337.2200
5	724	LL Road Frame - Black, 62	FR-R38B-62	Black	204.6251	337.2200
6	736	LL Road Frame - Black, 44	FR-R38B-44	Black	204.6251	337.2200
7	737	LL Road Frame - Black, 48	FR-R38B-48	Black	204.6251	337.2200
8	738	LL Road Frame - Black, 52	FR-R38B-52	Black	204.6251	337.2200
9	743	HL Mountain Frame - Black, 42	FR-M94B-42	Black	739.0410	1349.6000
10	744	HL Mountain Frame - Black, 44	FR-M94B-44	Black	699.0928	1349.6000
11	745	HL Mountain Frame - Black, 48	FR-M94B-48	Black	699.0928	1349.6000
12	746	HL Mountain Frame - Black, 46	FR-M94B-46	Black	739.0410	1349.6000
13	747	HL Mountain Frame - Black, 38	FR-M94B-38	Black	739.0410	1349.6000
14	765	Road-650 Black, 58	BK-R50B-58	Black	486.7066	782.9900

Using write_sql() function

Just now (1s) 12 Python

```
display(write_sql(df, "overwrite", "dbo.BlackProduct"))
```

▶ (1) Spark Jobs

To check if the data is written to SQL need to check using SSMS

The screenshot shows the Microsoft SQL Server Enterprise Manager (SSMS) interface. On the left, the Object Explorer displays the database structure for 'deepthi.database.windows.net (SQL Se)'. The central pane shows a query window with the following SQL statement:

```
SELECT [ProductModelID],
       [SellStartDate],
       [SellEndDate],
       [DiscontinuedDate],
       [ThumbnailPhoto],
       [ThumbnailPhotoFileName],
       [rowguid],
       [ModifiedDate]
FROM [dbo].[BlackProduct]
```

Below the query window, the 'Results' tab displays a grid of 18 rows of data. The columns are: ProductID, Name, ProductNumber, Color, StandardCost, ListPrice, Size, Weight, ProductCategoryID, ProductModelID, SellStartDate, and SellEndDate. The data shows various bicycle models, all with a 'Color' of 'Black'.

ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID	SellStartDate	SellEndDate
680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000	58	1016.04	18	6	2002-06-01 00:00:00.000	NULL
708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900	NULL	NULL	35	33	2005-07-01 00:00:00.000	NULL
722	LL Road Frame - Black, 58	FR-R38B-58	Black	204.6251	337.2200	58	1115.83	18	9	2005-07-01 00:00:00.000	NULL
723	LL Road Frame - Black, 60	FR-R38B-60	Black	204.6251	337.2200	60	1124.90	18	9	2005-07-01 00:00:00.000	NULL
724	LL Road Frame - Black, 62	FR-R38B-62	Black	204.6251	337.2200	62	1133.98	18	9	2005-07-01 00:00:00.000	NULL
736	LL Road Frame - Black, 44	FR-R38B-44	Black	204.6251	337.2200	44	1052.33	18	9	2005-07-01 00:00:00.000	NULL
737	LL Road Frame - Black, 48	FR-R38B-48	Black	204.6251	337.2200	48	1070.47	18	9	2005-07-01 00:00:00.000	NULL
738	LL Road Frame - Black, 52	FR-R38B-52	Black	204.6251	337.2200	52	1088.62	18	9	2005-07-01 00:00:00.000	NULL
814	ML Mountain Frame - Black, 38	FR-M63B-38	Black	185.8193	348.7600	38	1238.30	16	15	2006-07-01 00:00:00.000	2007-06-30
830	ML Mountain Frame - Black, 40	FR-M63B-40	Black	185.8193	348.7600	40	1256.44	16	14	2006-07-01 00:00:00.000	2007-06-30
743	HL Mountain Frame - Black, 42	FR-M94B-42	Black	739.0410	1349.6000	42	1233.76	16	5	2005-07-01 00:00:00.000	NULL
744	HL Mountain Frame - Black, 44	FR-M94B-44	Black	699.0928	1349.6000	44	1251.91	16	5	2005-07-01 00:00:00.000	2006-06-30
745	HL Mountain Frame - Black, 46	FR-M94B-46	Black	699.0928	1349.6000	46	1270.05	16	5	2005-07-01 00:00:00.000	2006-06-30
746	HL Mountain Frame - Black, 48	FR-M94B-48	Black	739.0410	1349.6000	48	1288.20	16	5	2005-07-01 00:00:00.000	NULL
747	HL Mountain Frame - Black, 50	FR-M94B-50	Black	739.0410	1349.6000	50	1215.62	16	5	2005-07-01 00:00:00.000	NULL
765	Road-650 Black, 58	BK-R50B-58	Black	486.7066	782.9900	58	8976.55	6	30	2005-07-01 00:00:00.000	2007-06-30
766	Road-650 Black, 60	BK-R50B-60	Black	486.7066	782.9900	60	9026.44	6	30	2005-07-01 00:00:00.000	2007-06-30
767	Road-650 Black, 62	BK-R50B-62	Black	486.7066	782.9900	62	9071.80	6	30	2005-07-01 00:00:00.000	2007-06-30

The status bar at the bottom indicates: 'Query executed successfully. deepthi.database.windows.net admin12 (89) dbnew 00:00:00 89 rows'.

Only black color data is stored in dbo.BlackProduct table