UnityJDBC User Documentation

UnityJDBC User Documentation

Table of Contents

List of Tables

Chapter 1. General Information

Overview of UnityJDBC

UnityJDBC is a Type 4 JDBC driver capable of querying multiple databases in a single SQL query. The driver can be used similar to other JDBC drivers including with query, reporting, and business intelligence tools, application and web servers, or stand-alone Java programs. Internally, UnityJDBC contains a database engine and optimizer allowing it to efficiently join data from source databases to produce a single ResultSet. UnityJDBC supports updating data using results produced from cross-database queries and performs automatic dialect translation to convert queries into the proper dialect. A brief list of the major supported features is below:

- supports cross-database joins of any number or type of JDBC-accessible sources (Microsoft SQL Server, Oracle, DB2, Postgres, MySQL, Sybase, MongoDB, Cassandra, etc.)
- allows SQL-based comparison of data across databases to detect data inconsistencies, errors, or for synchronization of data between databases
- performs SQL dialect translation and automatically executes functions and features internally in the driver if the data source does not support them
- contains an advanced optimizer and query processor that performs efficient query processing by having each source process as much of the query as possible (e.g. "push-down filters")
- cross-database queries can be used to insert records into tables (INSERT INTO ... SELECT)
- supports cross-database PreparedStatements
- has a driver by-pass feature to allow direct access to individual sources
- supports connection pools and connection properties
- supports user-defined functions
- works with any data source that has a JDBC driver and will run on any Java supported platform
- works with all SQL query software including SQuirreL SQL, Aqua Data Studio, Toad, and RazorSQL
- works will business intelligence and reporting software including JasperReports, Pentaho, and Splunk

Chapter 2. Installation

Overview

The free evaluation version of UnityJDBC is a fully functioning system. The only constraint with the evaluation version is that it will only produce the first 100 results in a ResultSet. The full version has no restrictions. The evaluation version can be downloaded at http://www.unityjdbc.com/download.php. You are free to distribute the evaluation version of the software.

System Requirements

UnityJDBC requires a JRE of 1.6 or higher. UnityJDBC will run on a J2SE or J2EE platform.

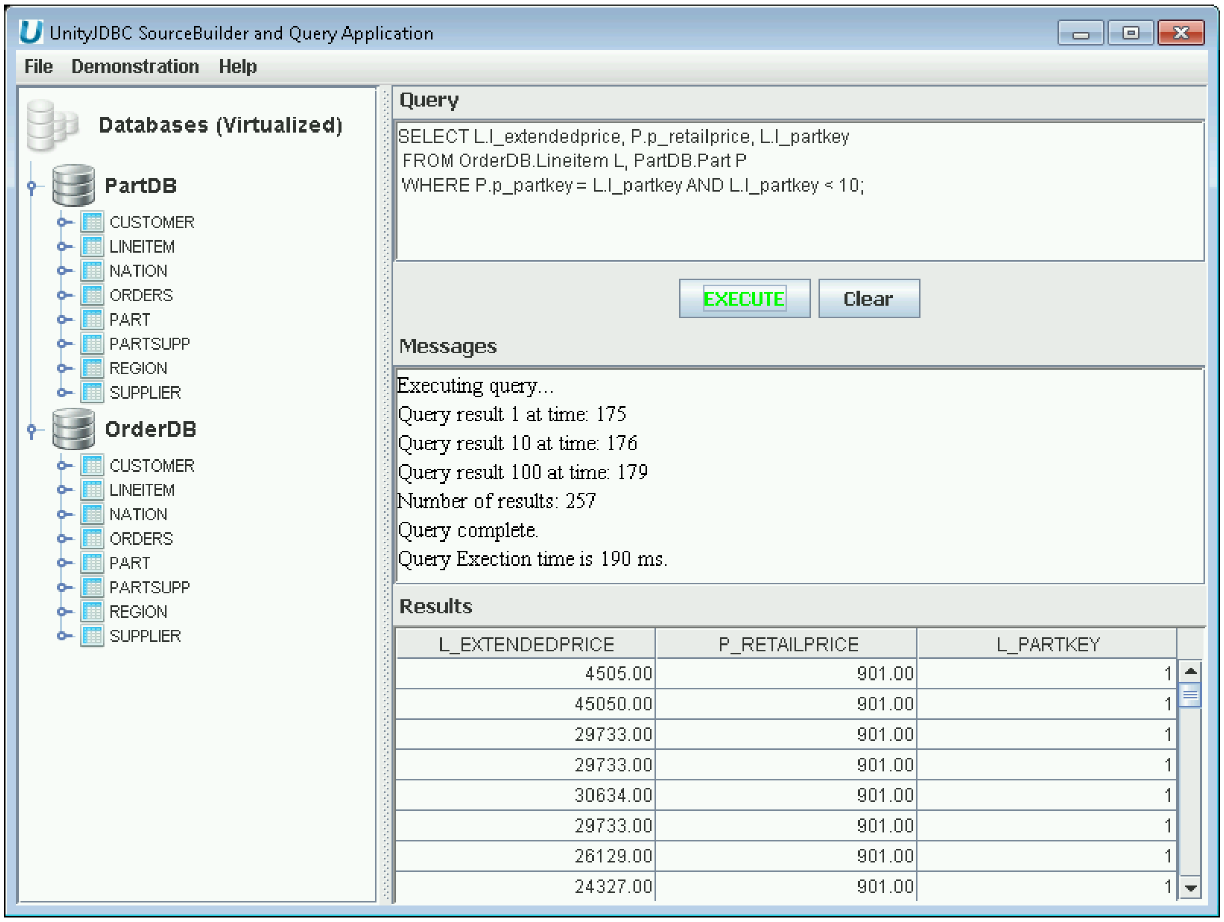
Quick Setup and Installation

UnityJDBC can be downloaded, installed, tested, and configured for your environment in less than 10 minutes. Here are the easy steps:

1. **Downloading** - First, download the UnityJDBC package to your computer from http://www.unityjdbc.com/download.php.

2. **Installation** - The UnityJDBC installation package contains the driver, a simple query GUI, and some test programs. The `UnityJDBC_Trial_Install.jar` can be run by typing the command: `java -jar UnityJDBC_Trial_Install.jar`. Follow the prompts to install UnityJDBC on your machine.

3. **Quick Start** - The installation creates a shortcut to the UnityJDBC SourceBuilder which is a simple graphical query editor and configuration tool. You can also run it directly using `initsources.bat`or `initsources.sh` in the installation directory. To try the sample queries, you will also want to click on the Start Database shortcut or run the script `startDB.bat` or `startDB.sh` in the directory `sampleDB/hsqldb`. Below is a screenshot of running a sample query that joins across two databases.



```
SELECT L.l_extendedprice, P.p_retailprice, L.l_partkey
 FROM OrderDB.Lineitem L, PartDB.Part P
 WHERE P.p_partkey = L.l_partkey AND L.l_partkey < 10;
```

```
Executing query...
Query result 1 at time: 175
Query result 10 at time: 176
Query result 100 at time: 179
Number of results: 257
Query complete.
Query Exection time is 190 ms.
```

L_EXTENDEDPRICE	P_RETAILPRICE	L_PARTKEY
4505.00	901.00	1
45050.00	901.00	1
29733.00	901.00	1
29733.00	901.00	1
30634.00	901.00	1
29733.00	901.00	1
26129.00	901.00	1
24327.00	901.00	1

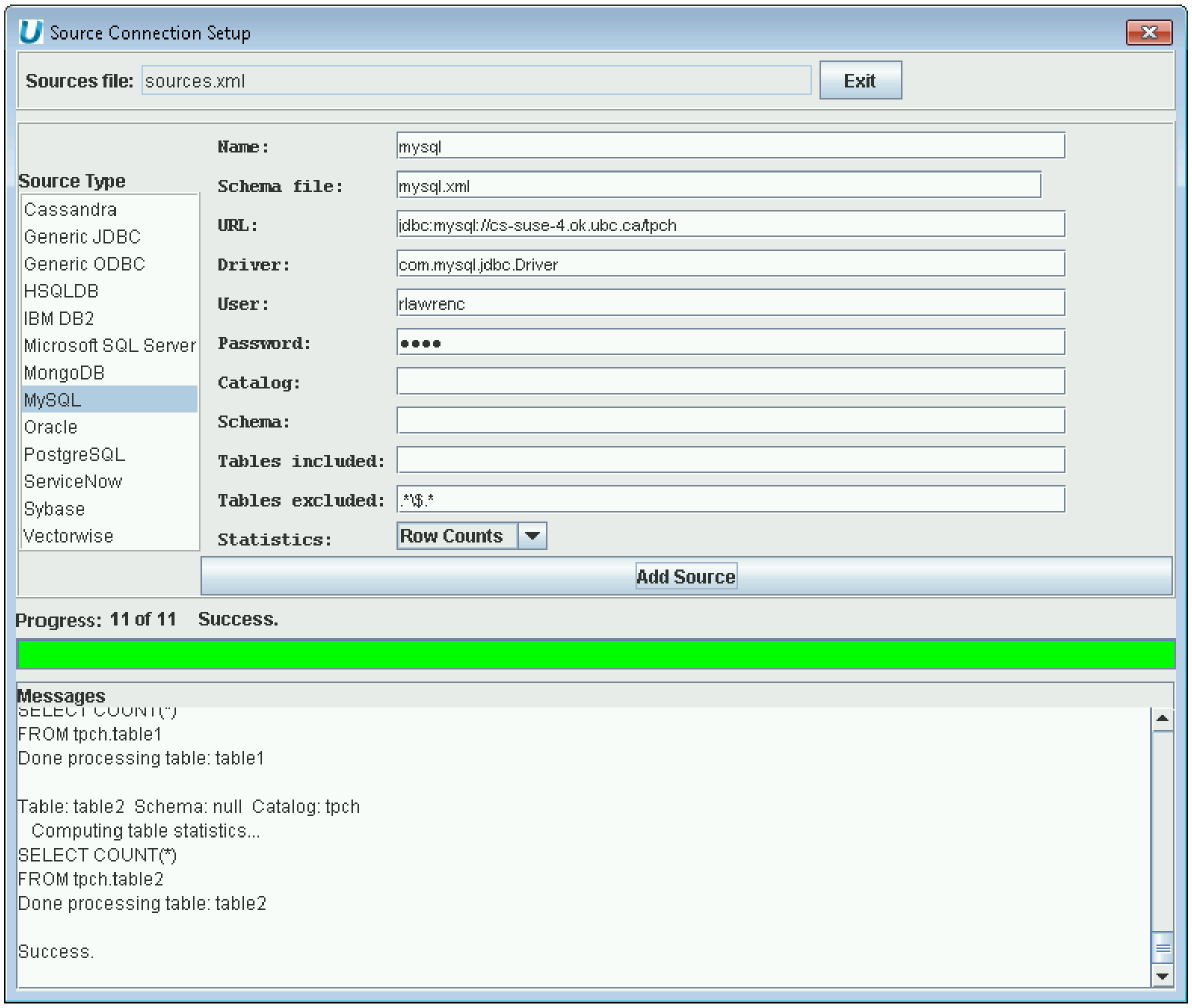
Multiple Database Query Example

4. **Writing queries** - UnityJDBC uses the standard SQL language. You can include as many databases in the virtual database that you wish. To reference a table in a given database, prefix it with the database name. In the example above, OrderDB.Lineitem refers to the Lineitem table in OrderDB (a user-assigned name for the sample HSQL database), and PartDB.Part refers to the Part table in the PartDB database. **Writing queries is as easy as prefixing the table name with the database name!**

5. **Creating Your Own Virtual Database** - A virtual database is defined using a sources XML file, often with the default filename of `sources.xml`. Each database also has a schema file storing information on its tables and fields. These XML files are automatically built by SourceBuilder, but can also be edited directly at any time. A few quick steps to produce your own virtual database:

   a. Select `File->New Source Group` and use the default `sources.xml` file.

   b. Select `File->New Source` and provide the JDBC connection information for your source. The screenshots below show MySQL and PostgreSQL.



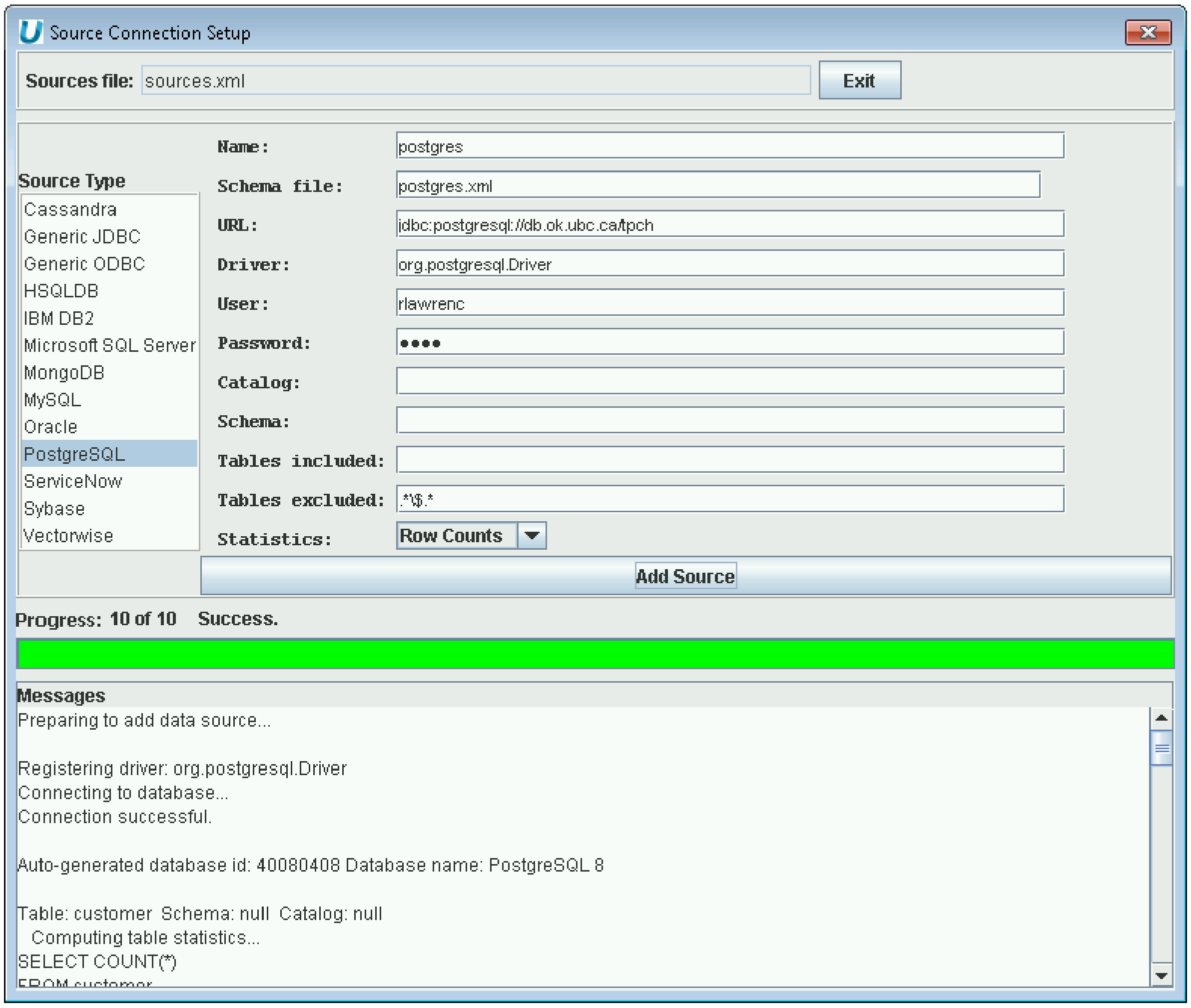
Sources file: sources.xml

Name: mysql
Schema file: mysql.xml
URL: jdbc:mysql://cs-suse-4.ok.ubc.ca/tpch
Driver: com.mysql.jdbc.Driver
User: rlawrenc
Password: ••••
Catalog:
Schema:
Tables included:
Tables excluded: .*\$.*
Statistics: Row Counts

Progress: 11 of 11 Success.

```
SELECT COUNT(*)
FROM tpch.table1
Done processing table: table1

Table: table2  Schema: null  Catalog: tpch
  Computing table statistics...
SELECT COUNT(*)
FROM tpch.table2
Done processing table: table2

Success.
```

Multiple Database Virtualization - Adding MySQL Database



Sources file: sources.xml

Source Type: Cassandra, Generic JDBC, Generic ODBC, HSQLDB, IBM DB2, Microsoft SQL Server, MongoDB, MySQL, Oracle, PostgreSQL, ServiceNow, Sybase, Vectorwise

Name:	postgres
Schema file:	postgres.xml
URL:	jdbc:postgresql://db.ok.ubc.ca/tpch
Driver:	org.postgresql.Driver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

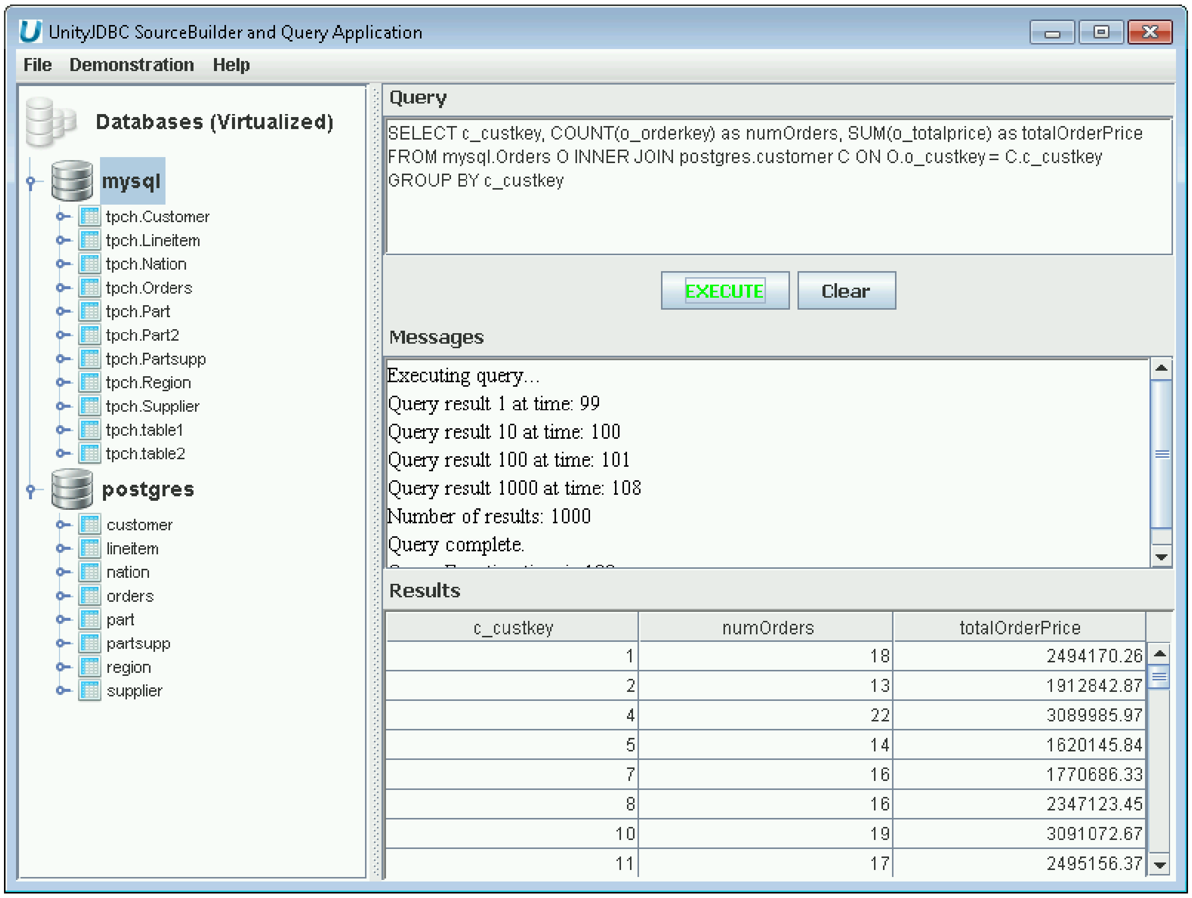
Progress: 10 of 10 Success.

Messages

```
Preparing to add data source...

Registering driver: org.postgresql.Driver
Connecting to database...
Connection successful.

Auto-generated database id: 40080408 Database name: PostgreSQL 8

Table: customer  Schema: null  Catalog: null
   Computing table statistics...
SELECT COUNT(*)
FROM customer
```

Multiple Database Virtualization - Adding PostgreSQL Database

c. Write a query that can join across multiple databases. The only difference from standard SQL is that you prefix the table name with the database name.



```
SELECT c_custkey, COUNT(o_orderkey) as numOrders, SUM(o_totalprice) as totalOrderPrice
FROM mysql.Orders O INNER JOIN postgres.customer C ON O.o_custkey = C.c_custkey
GROUP BY c_custkey
```

```
Executing query...
Query result 1 at time: 99
Query result 10 at time: 100
Query result 100 at time: 101
Query result 1000 at time: 108
Number of results: 1000
Query complete.
```

c_custkey	numOrders	totalOrderPrice
1	18	2494170.26
2	13	1912842.87
4	22	3089985.97
5	14	1620145.84
7	16	1770686.33
8	16	2347123.45
10	19	3091072.67
11	17	2495156.37

Multiple Database Query Example that Joins Table in MySQL with a Table in PostgreSQL

d. After creating your virtual database, you will have one `sources.xml` and an XML schema file for each database. These files can be moved to any location and are used to configure UnityJDBC when using it with other applications.

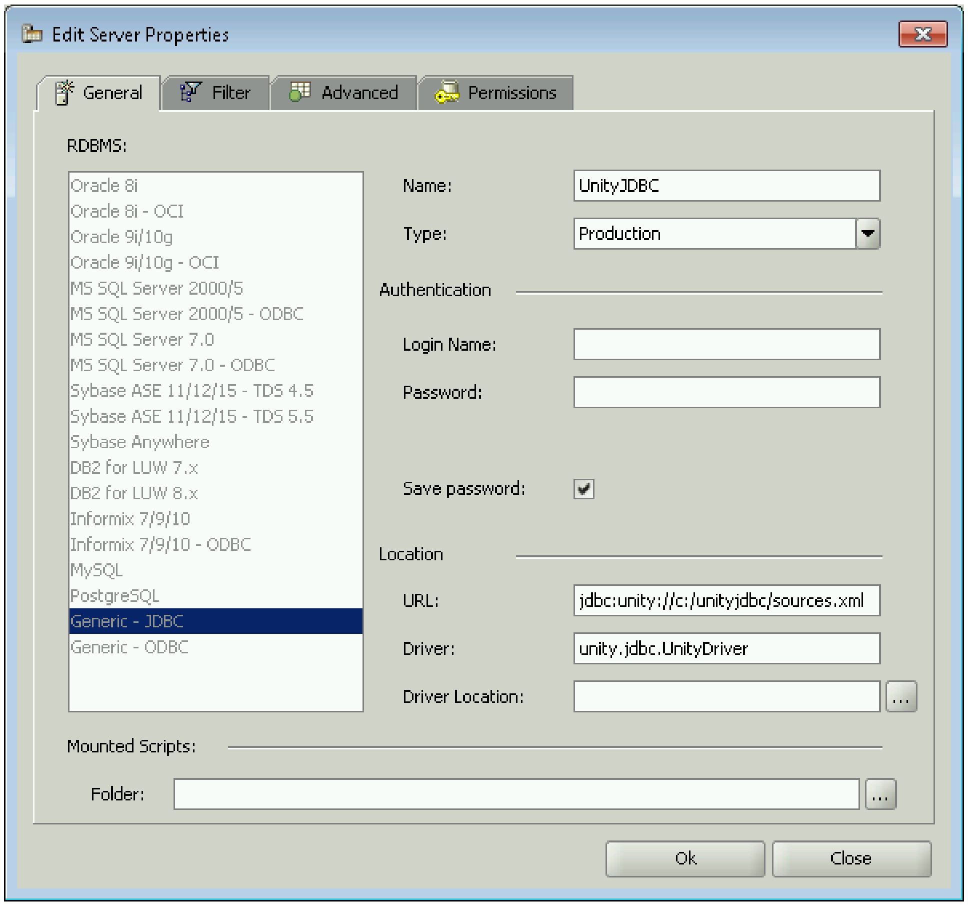
6. **Using UnityJDBC** - Now that a virtual database is created, there are three general steps to using UnityJDBC common in all cases.

   a. Move the `sources.xml` and the XML schema files for the databases into a desired directory. For this example, the directory is `/unityjdbc`.

   b. Put the `unityjdbc.jar` into your `CLASSPATH` as well as the drivers for each database. (Some drivers are included with the UnityJDBC distribution in the directory `drivers` in the installation folder.) A common location is in `<JAVA_HOME>/jre/lib/ext.`

   c. The connection information is the UnityJDBC driver class `unity.jdbc.UnityDriver` and the URL is `jdbc:unity://<relative or absolute path to sources file>` such as `jdbc:unity://unityjdbc/sources.xml`.

   The UnityJDBC driver can be used with any software that supports JDBC. A screenshot of configuring it in Aqua Data Studio is below. The same connection information applies to all JDBC-based software.



RDBMS:

Oracle 8i
Oracle 8i - OCI
Oracle 9i/10g
Oracle 9i/10g - OCI
MS SQL Server 2000/5
MS SQL Server 2000/5 - ODBC
MS SQL Server 7.0
MS SQL Server 7.0 - ODBC
Sybase ASE 11/12/15 - TDS 4.5
Sybase ASE 11/12/15 - TDS 5.5
Sybase Anywhere
DB2 for LUW 7.x
DB2 for LUW 8.x
Informix 7/9/10
Informix 7/9/10 - ODBC
MySQL
PostgreSQL
Generic - JDBC
Generic - ODBC

Name: UnityJDBC

Type: Production

Authentication

Login Name:

Password:

Save password: ✓

Location

URL: jdbc:unity://c:/unityjdbc/sources.xml

Driver: unity.jdbc.UnityDriver

Driver Location:

Mounted Scripts:

Folder:

Registering the UnityJDBC Driver in Aqua Data Studio

UnityJDBC can also be used with your own Java programs. There is sample code in the `code` directory in the installation folder. Here are two commands (executed from the `code` directory) to compile and run the sample code:

```
javac test/ExampleQuery.java
```

```
java test.ExampleQuery
```

If you have issues with compiling or running, try to explicitly indicate the location of the UnityJDBC JAR:

```
javac -cp .;../unityjdbc.jar;../sampleDB/hsqldb/hsqldb.jar test/
ExampleQuery.java
```

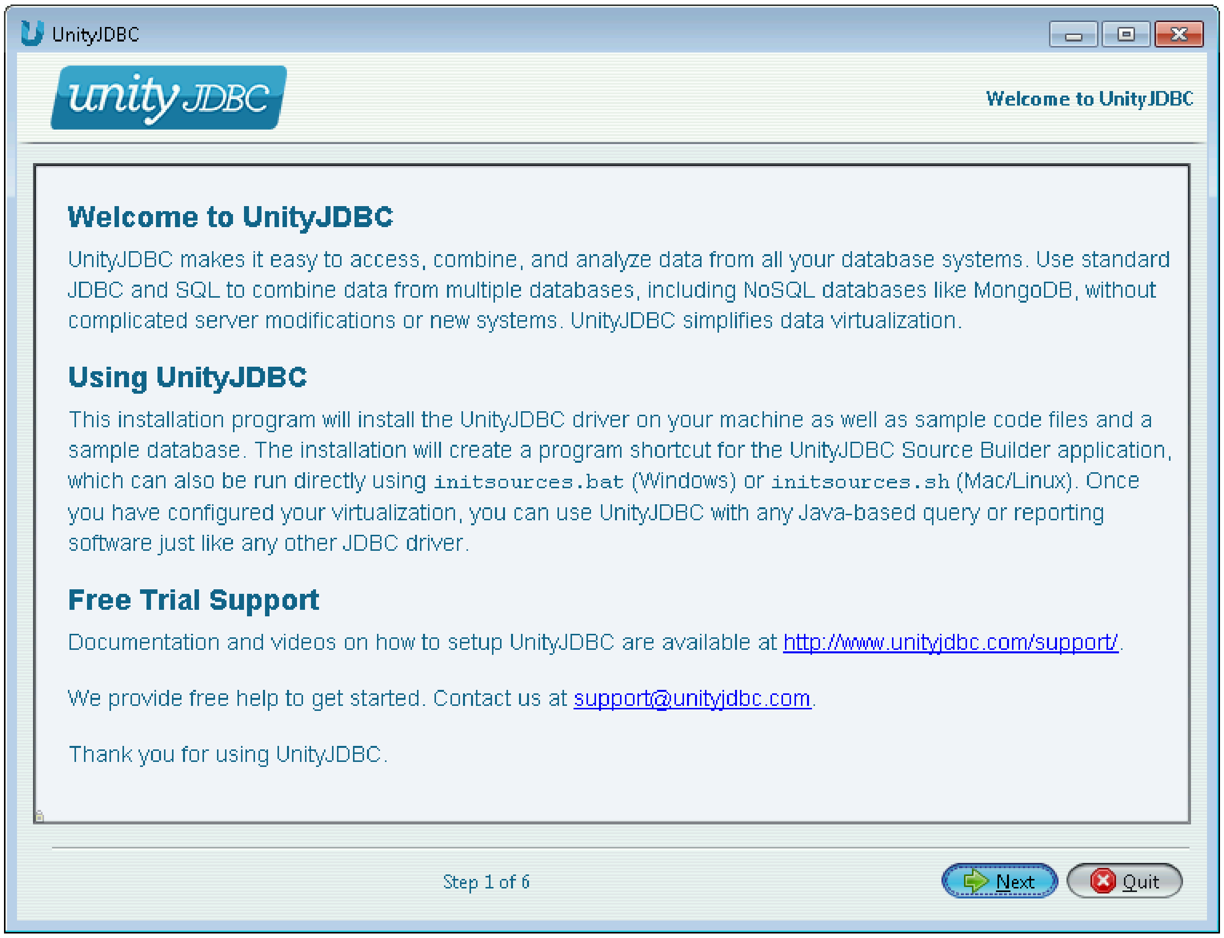
```
java -cp .;../unityjdbc.jar;../sampleDB/hsqldb/hsqldb.jar test.ExampleQuery
```

To create your own Java program, copy the file `ExampleQuery.java` to `MyQuery.java`. There are 2 lines that you must modify. The first line indicates where your new sources XML file is located on your machine. You may specify an absolute or relative path from the current directory. The second line you must modify is to change the SQL query to reference fields and tables in your data source(s). Compile and run the program. Queries can reference any table or field in any data source in your XML sources file as long as you prefix a table or field with the database name such as `MyDB.MyTable.MyField`.

Installation Walkthrough

Once you have downloaded UnityJDBC from http://www.unityjdbc.com/download.php, you will have downloaded a JAR file called `UnityJDBC_Trial_Install.jar`. **First, you need to have Java previously installed in order to install UnityJDBC.** Then, you can install UnityJDBC by either double-clicking the JAR file or running the following command: `java -jar UnityJDBC_Trial_Install.jar`. The installation steps are below.

1. **Welcome Screen** - Provides some background on UnityJDBC. Click Next.



Welcome to UnityJDBC

UnityJDBC makes it easy to access, combine, and analyze data from all your database systems. Use standard JDBC and SQL to combine data from multiple databases, including NoSQL databases like MongoDB, without complicated server modifications or new systems. UnityJDBC simplifies data virtualization.

Using UnityJDBC

This installation program will install the UnityJDBC driver on your machine as well as sample code files and a sample database. The installation will create a program shortcut for the UnityJDBC Source Builder application, which can also be run directly using `initsources.bat` (Windows) or `initsources.sh` (Mac/Linux). Once you have configured your virtualization, you can use UnityJDBC with any Java-based query or reporting software just like any other JDBC driver.

Free Trial Support

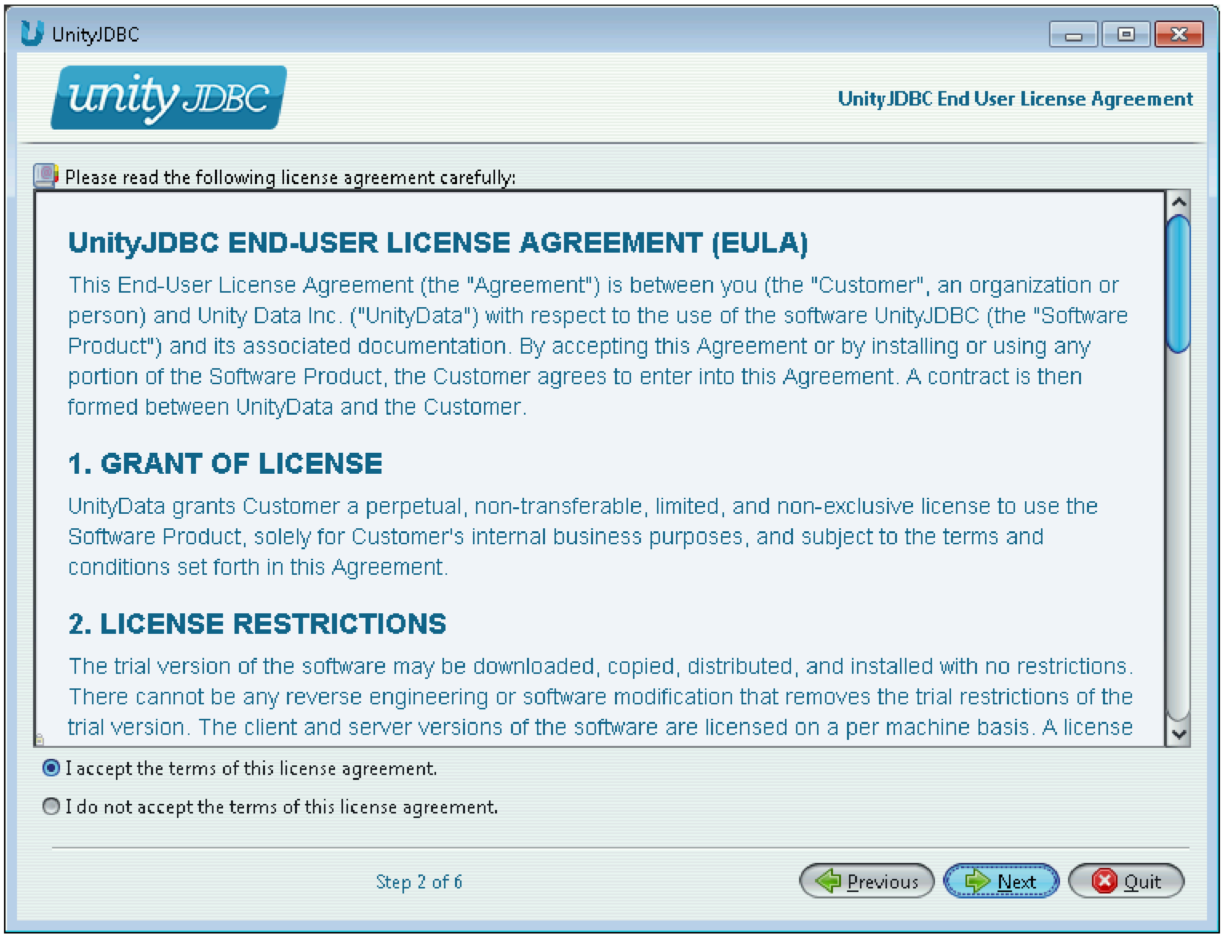
Documentation and videos on how to setup UnityJDBC are available at http://www.unityjdbc.com/support/.

We provide free help to get started. Contact us at support@unityjdbc.com.

Thank you for using UnityJDBC.

UnityJDBC Installation Part #1 - Welcome Screen

2. **End-User License Agreement** - Read the EULA, select Accept, and click Next.



Please read the following license agreement carefully:

UnityJDBC END-USER LICENSE AGREEMENT (EULA)

This End-User License Agreement (the "Agreement") is between you (the "Customer", an organization or person) and Unity Data Inc. ("UnityData") with respect to the use of the software UnityJDBC (the "Software Product") and its associated documentation. By accepting this Agreement or by installing or using any portion of the Software Product, the Customer agrees to enter into this Agreement. A contract is then formed between UnityData and the Customer.

1. GRANT OF LICENSE

UnityData grants Customer a perpetual, non-transferable, limited, and non-exclusive license to use the Software Product, solely for Customer's internal business purposes, and subject to the terms and conditions set forth in this Agreement.

2. LICENSE RESTRICTIONS

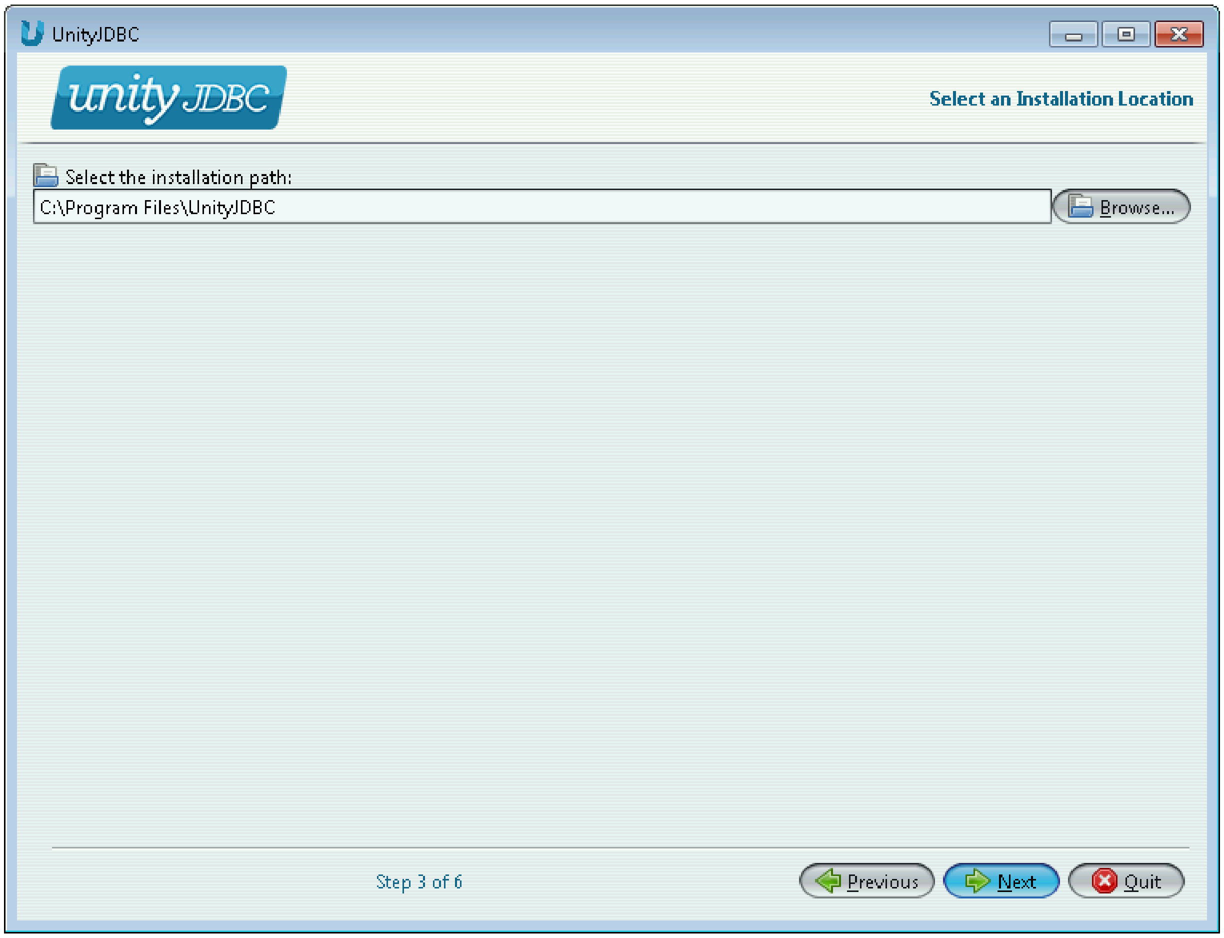
The trial version of the software may be downloaded, copied, distributed, and installed with no restrictions. There cannot be any reverse engineering or software modification that removes the trial restrictions of the trial version. The client and server versions of the software are licensed on a per machine basis. A license

[✓ I accept the terms of this license agreement.]

I do not accept the terms of this license agreement.

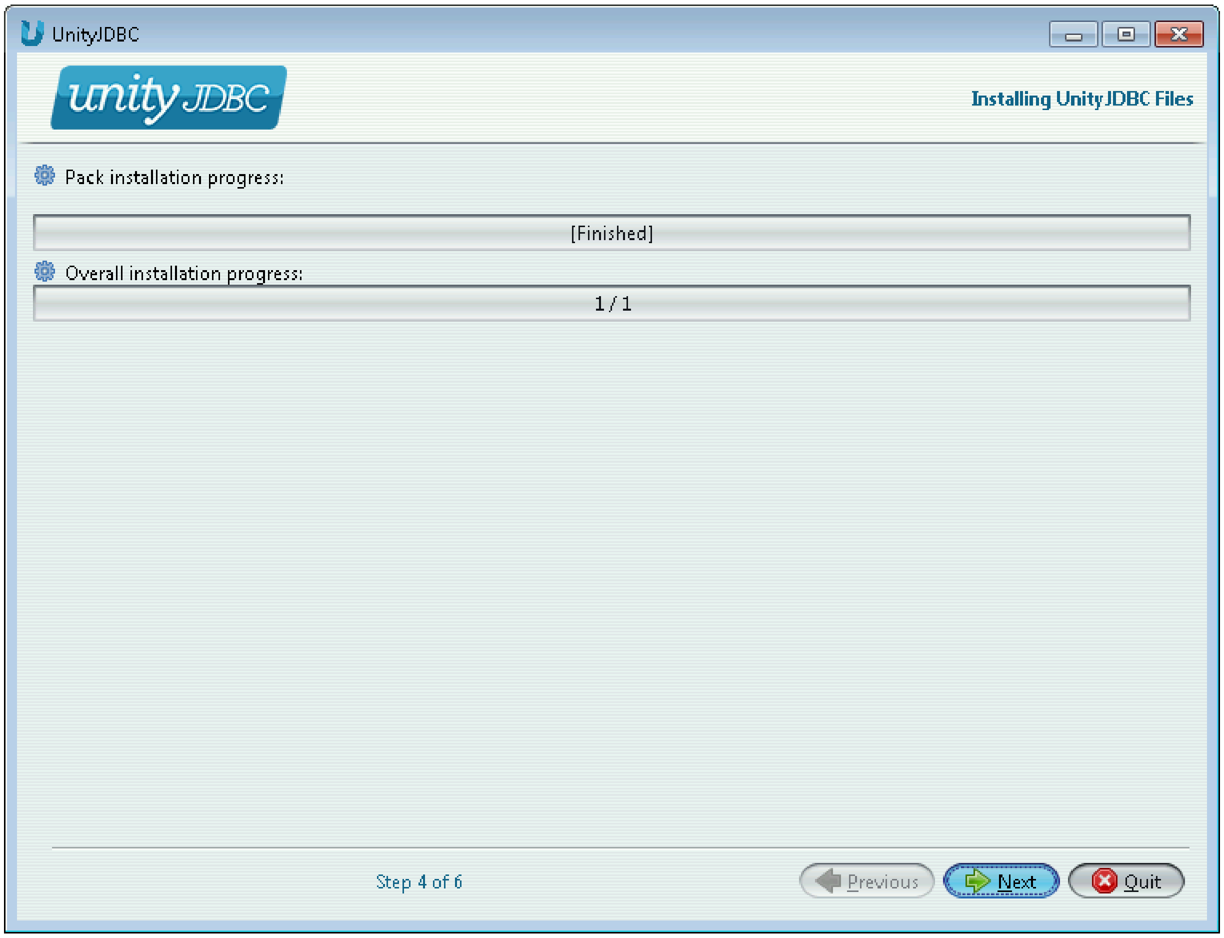
UnityJDBC Installation Part #2 - End-User License Agreement

3. **Install Path** - Select the installation path for UnityJDBC and then click Next.


UnityJDBC
unity JDBC
Select an Installation Location
Select the installation path:
C:\Program Files\UnityJDBC
Browse...
Step 3 of 6
Previous
Next
Quit

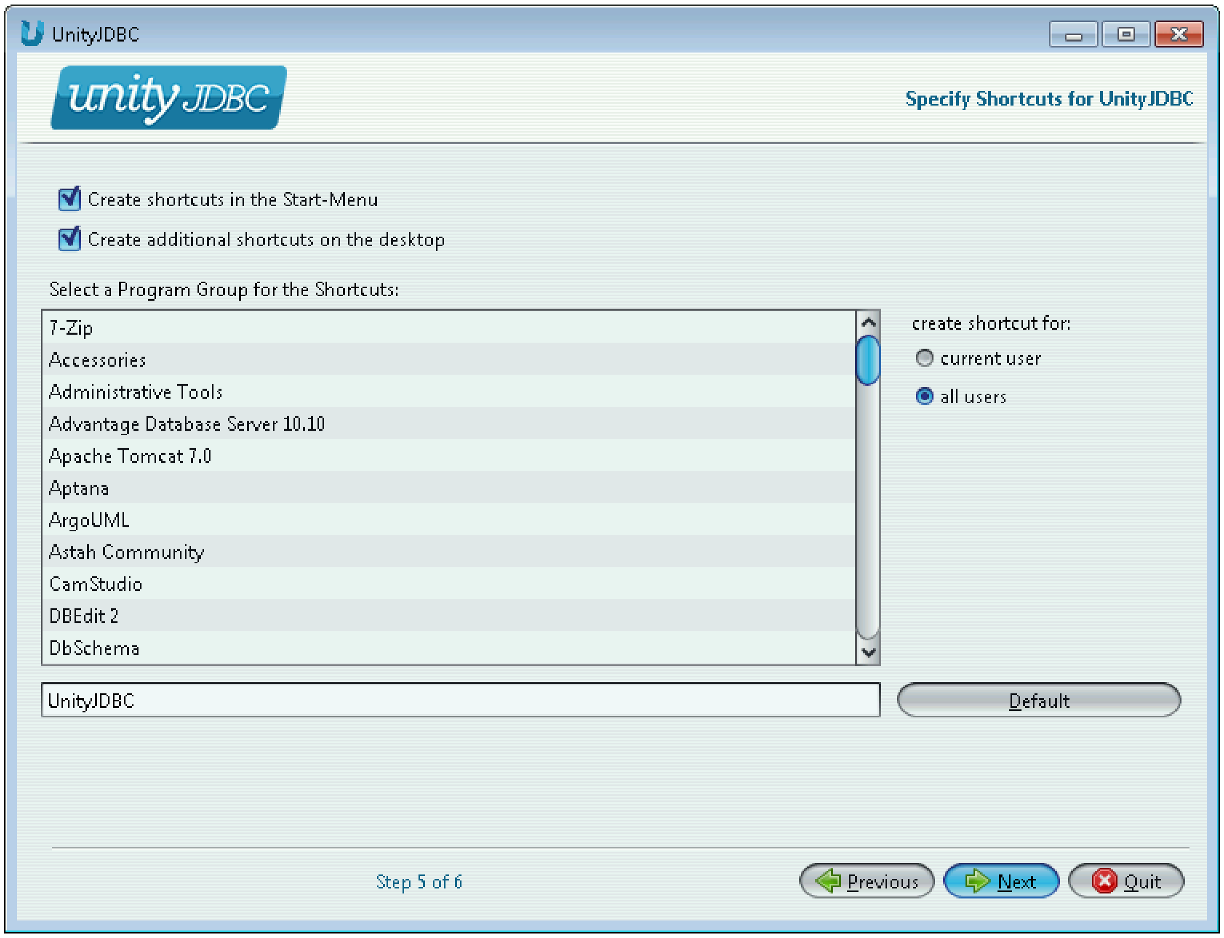
UnityJDBC Installation Part #3 - Select Installation Path

4. **Install Progress** - The installation will be performed and progress shown. Click Next when complete.


UnityJDBC
unity JDBC
Installing UnityJDBC Files
Pack installation progress:
[Finished]
Overall installation progress:
1 / 1
Step 4 of 6
Previous
Next
Quit

UnityJDBC Installation Part #4 - Installation Progress

5. **Install Shortcuts** - The installation will install shortcuts to start SourceBuilder and a sample database. This screen allows you to control if shortcuts are created and their location. When finished, click Next.


UnityJDBC

Specify Shortcuts for UnityJDBC

[✓] Create shortcuts in the Start-Menu

[✓] Create additional shortcuts on the desktop

Select a Program Group for the Shortcuts:

- 7-Zip
- Accessories
- Administrative Tools
- Advantage Database Server 10.10
- Apache Tomcat 7.0
- Aptana
- ArgoUML
- Astah Community
- CamStudio
- DBEdit 2
- DbSchema

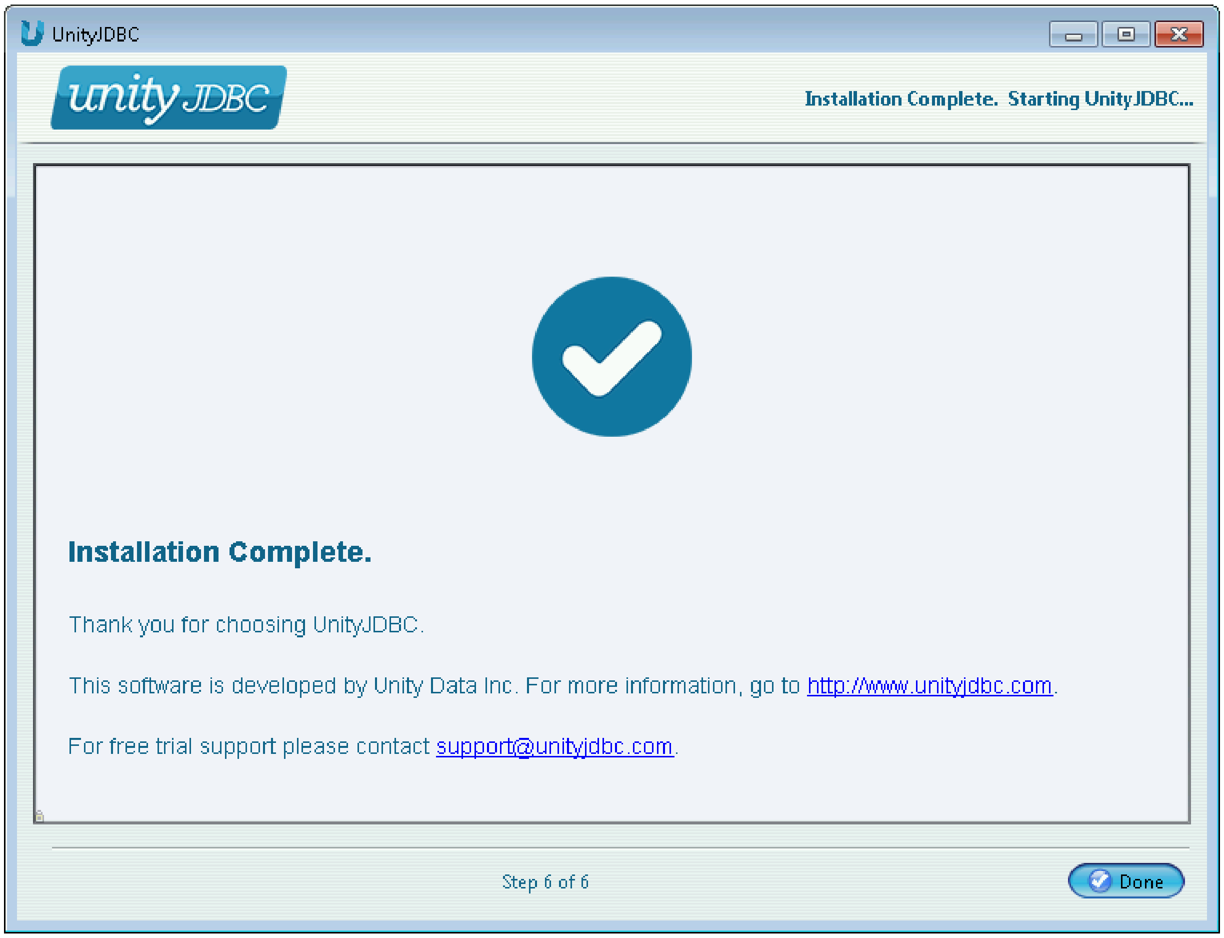
UnityJDBC

create shortcut for:

current user [✓ all users]

UnityJDBC Installation Part #5 - Select Shortcuts

6. **Installation Complete** - The installation complete screen will be displayed if the installation is successful. UnityJDBC SourceBuilder will be automatically started. Click Done to close the install window.



Installation Complete.

Thank you for choosing UnityJDBC.

This software is developed by Unity Data Inc. For more information, go to http://www.unityjdbc.com.

For free trial support please contact support@unityjdbc.com.

UnityJDBC Installation Part #6 - Installation Complete

UnityJDBC will auto-start at the completion of the installation. If it does not, click on the shortcut created. If there is no shortcut, on Windows run `initsources.bat` and on Linux/Mac run `initsources.sh` in the installation directory.

Configuring Data Sources

UnityJDBC requires information about the data sources being queried in order to validate, optimize, and execute queries against those data sources. All source information is stored in XML files. There are two types of source information files: the *sources file* and *schema files*. A *sources file* with default name `sources.xml` provides information on all the sources that could be potentially queried. The location of this file is provided via the URL when initializing the driver. Inside the file is information on each source including its connection URL and parameters, JDBC driver, and schema file location. The sample sources file `code\test\xspec\UnityDemo.xml` is provided in the distribution package. Each data source requires a schema file. The *schema file* is an XML encoding of the schema information including table and field names, keys, joins, and relation sizes. It is used for validating queries and optimization. Two schema files provided in the distribution are: `code\test\xspec\UnityDemoOrder.xml` and `code\test\xspec\UnityDemoPart.xml`.

There are two ways to create your own sources file and associated schema files:

1. The easiest way is to use the SourceBuilder GUI that can be started using the installed shortcut or running `initsources.bat` or `initsources.sh`. This GUI will automatically extract source information and build

the necessary files. If you are using UnityJDBC as the multisource plugin in SQuirreL SQL, all features of the SourceBuilder are integrated into SQuirreL, and the XML files are generated automatically.

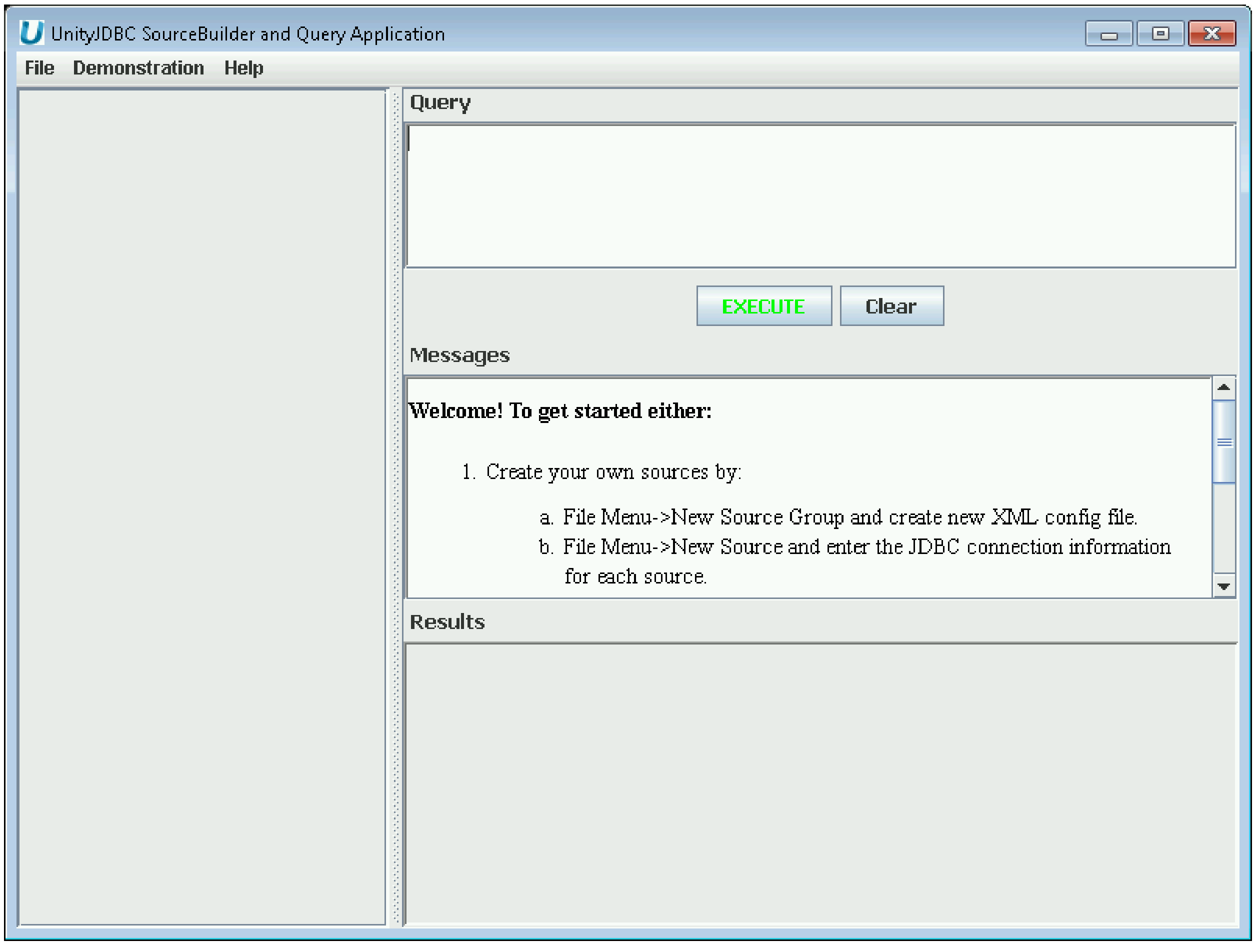
2. You can manually build a sources file using a text editor. To produce a schema file for a source, open up the program called `com/unityjdbc/SourceBuilder/SchemaExtractor.java` in the `code` directory. Modify the JDBC URL, driver path, and output directory accordingly and run the program. The account that you connect with must have read access to the database and associated tables that you want to access. After the XML schema file has been produced, move it to the directory where you want it and update the sources file to reference the correct location. In almost all cases, using the SourceBuilder utility will be faster and easier.

Chapter 3. Using SourceBuilder - Tutorial on Multiple Database Querying

Using the Sample Databases

SourceBuilder is a graphical query tool that allows you to define a virtual database consisting of multiple databases including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Sybase, MongoDB, and others. SourceBuilder can be started using the shortcut produced during the installation, by running `initsources.bat` or `initsources.sh` in the installation folder, or directly from the driver using the command `java -jar unityjdbc.jar`. It is required that the `unityjdbc.jar` and all JDBC driver JAR files for databases used be in the Java classpath.

When SourceBuilder is first started, a welcome screen is displayed.

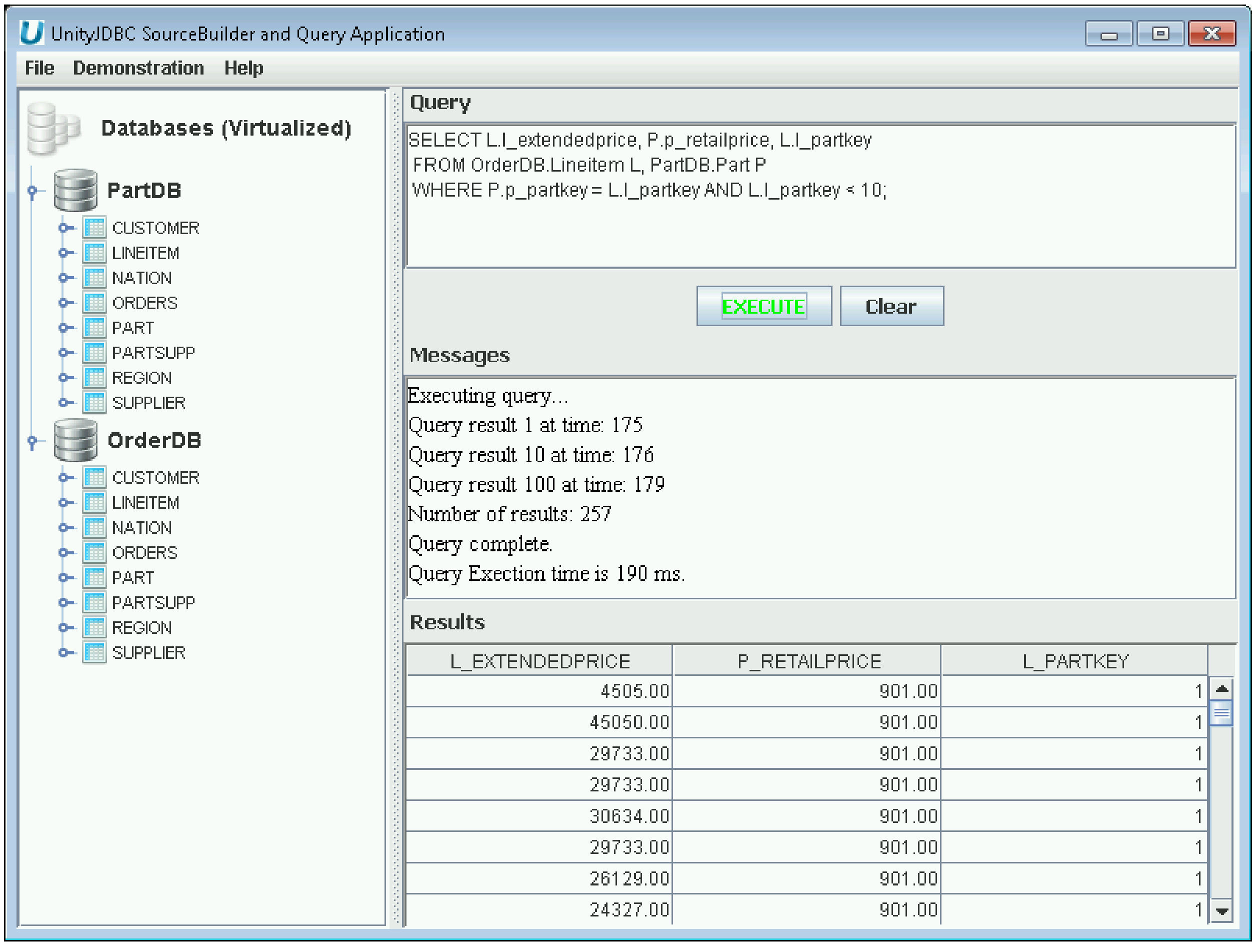


Welcome! To get started either:

1. Create your own sources by:
   a. File Menu->New Source Group and create new XML config file.
   b. File Menu->New Source and enter the JDBC connection information for each source.

SourceBuilder Welcome Screen

To use the sample database and queries included, make sure to start the HSQLDB using the Start Database shortcut or run the script `startDB.bat` or `startDB.sh` in the directory `sampleDB/hsqldb`. Then, select `Demonstration->Cross-database Join Test`. Click the `EXECUTE` button to run the query and get results. If an error occurs, verify that the sample database is started and the `hsqldb.jar` file is in your classpath. You can try several other of the sample queries or write your own.



```
SELECT L.l_extendedprice, P.p_retailprice, L.l_partkey
 FROM OrderDB.Lineitem L, PartDB.Part P
 WHERE P.p_partkey = L.l_partkey AND L.l_partkey < 10;
```

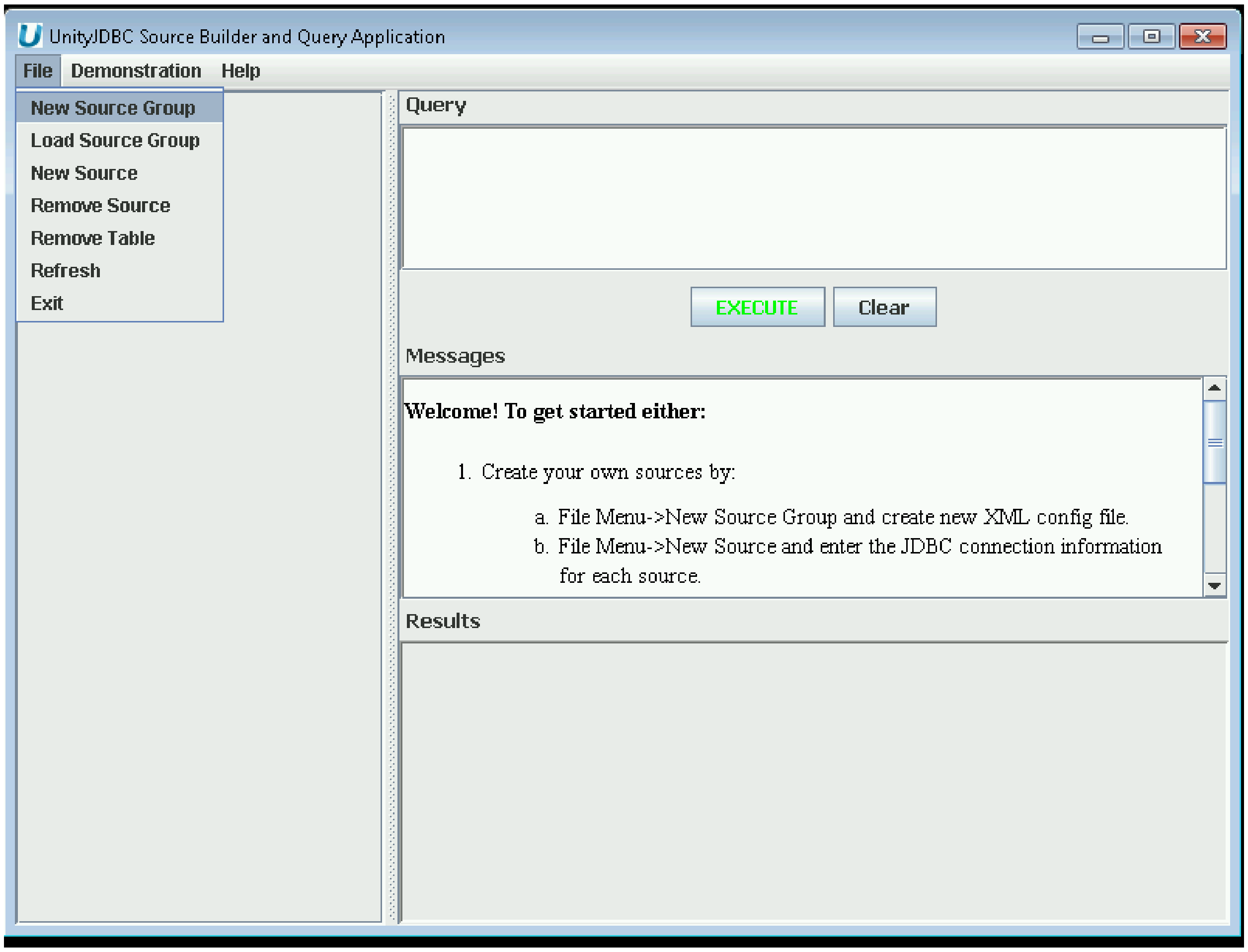
```
Executing query...
Query result 1 at time: 175
Query result 10 at time: 176
Query result 100 at time: 179
Number of results: 257
Query complete.
Query Exection time is 190 ms.
```

L_EXTENDEDPRICE	P_RETAILPRICE	L_PARTKEY
4505.00	901.00	1
45050.00	901.00	1
29733.00	901.00	1
29733.00	901.00	1
30634.00	901.00	1
29733.00	901.00	1
26129.00	901.00	1
24327.00	901.00	1

SourceBuilder Multiple Database Query Example with a Cross-Database Join

Data Virtualization Using SourceBuilder

To build your own data virtualization of multiple data sources, create a new sources file by selecting `File->New Source Group` and providing a file name (the default is `sources.xml`).



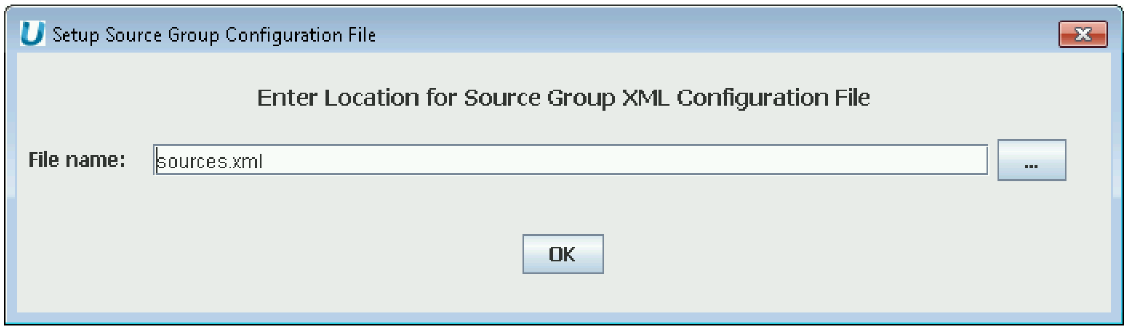
Welcome! To get started either:

1. Create your own sources by:

    a. File Menu->New Source Group and create new XML config file.

    b. File Menu->New Source and enter the JDBC connection information for each source.

Creating a New Source Group with SourceBuilder



Enter Location for Source Group XML Configuration File

File name: sources.xml

Prompt for Name of Sources File

Then, you will add each one of your sources by providing its JDBC connection information. To add a source you need the following information:

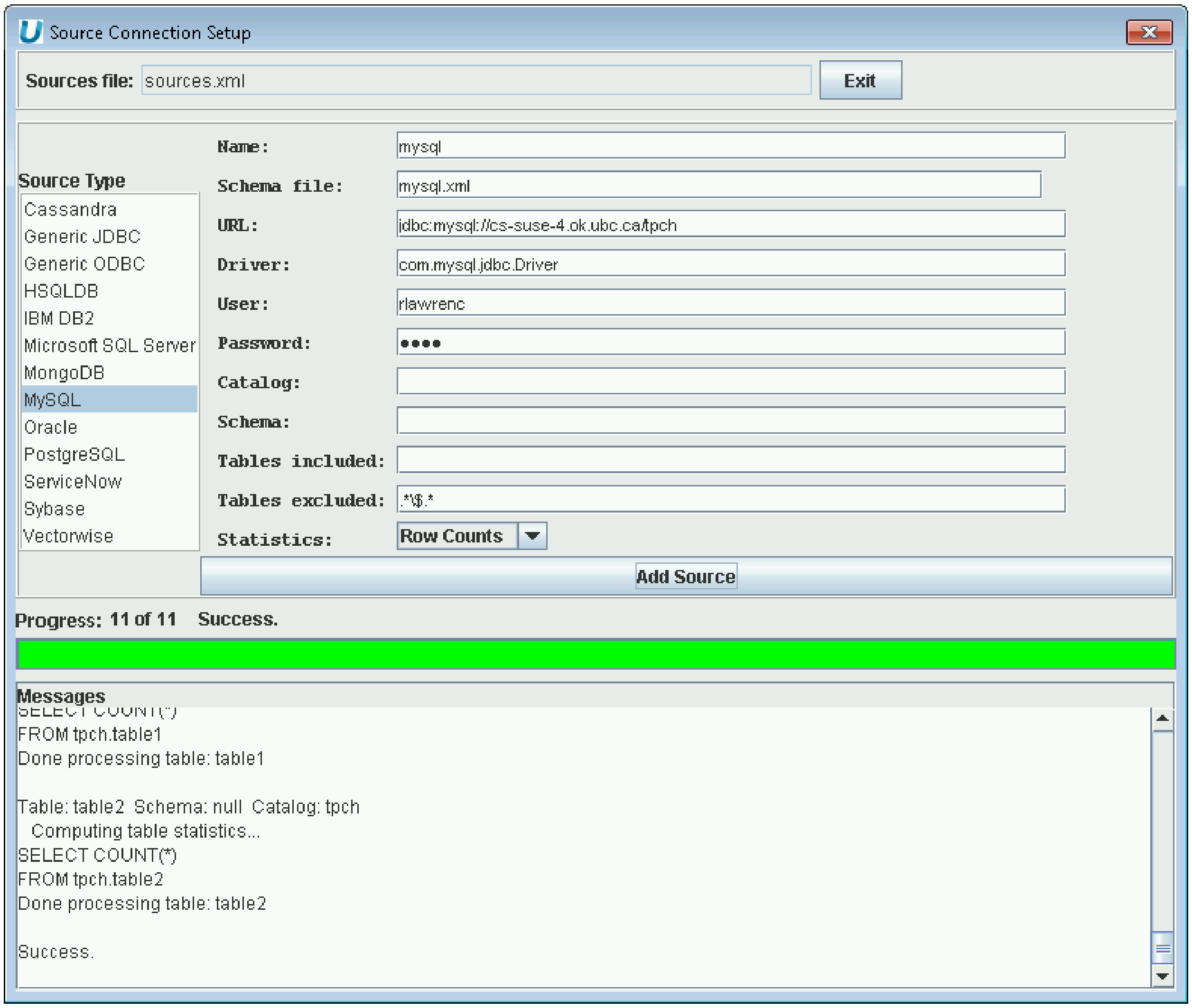
1. The JDBC Driver class name (e.g. `com.mysql.jdbc.Driver`).

2. The JDBC URL to connect to the source (e.g. `jdbc:mysql://localhost/mydb`).

3. User and password information if not specified in the JDBC URL.

4. A unique database name (does not have to be the same as the system database name) to refer to the data source in your data virtualization.

An example of adding a MySQL source to the virtual database is below.



Sources file: sources.xml

Name:	mysql
Schema file:	mysql.xml
URL:	jdbc:mysql://cs-suse-4.ok.ubc.ca/tpch
Driver:	com.mysql.jdbc.Driver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 11 of 11 Success.

```
SELECT COUNT(*)
FROM tpch.table1
Done processing table: table1

Table: table2  Schema: null  Catalog: tpch
   Computing table statistics...
SELECT COUNT(*)
FROM tpch.table2
Done processing table: table2

Success.
```

Adding a MySQL Data Source to the Data Virtualization

There are also several additional features that can be used to control the extraction.

1. **Schema** - Specify a schema name or pattern (JDBC API) to only retrieve tables in the given schema. This is especially important for Oracle as by default tables from all schemas will be added to the virtualization.

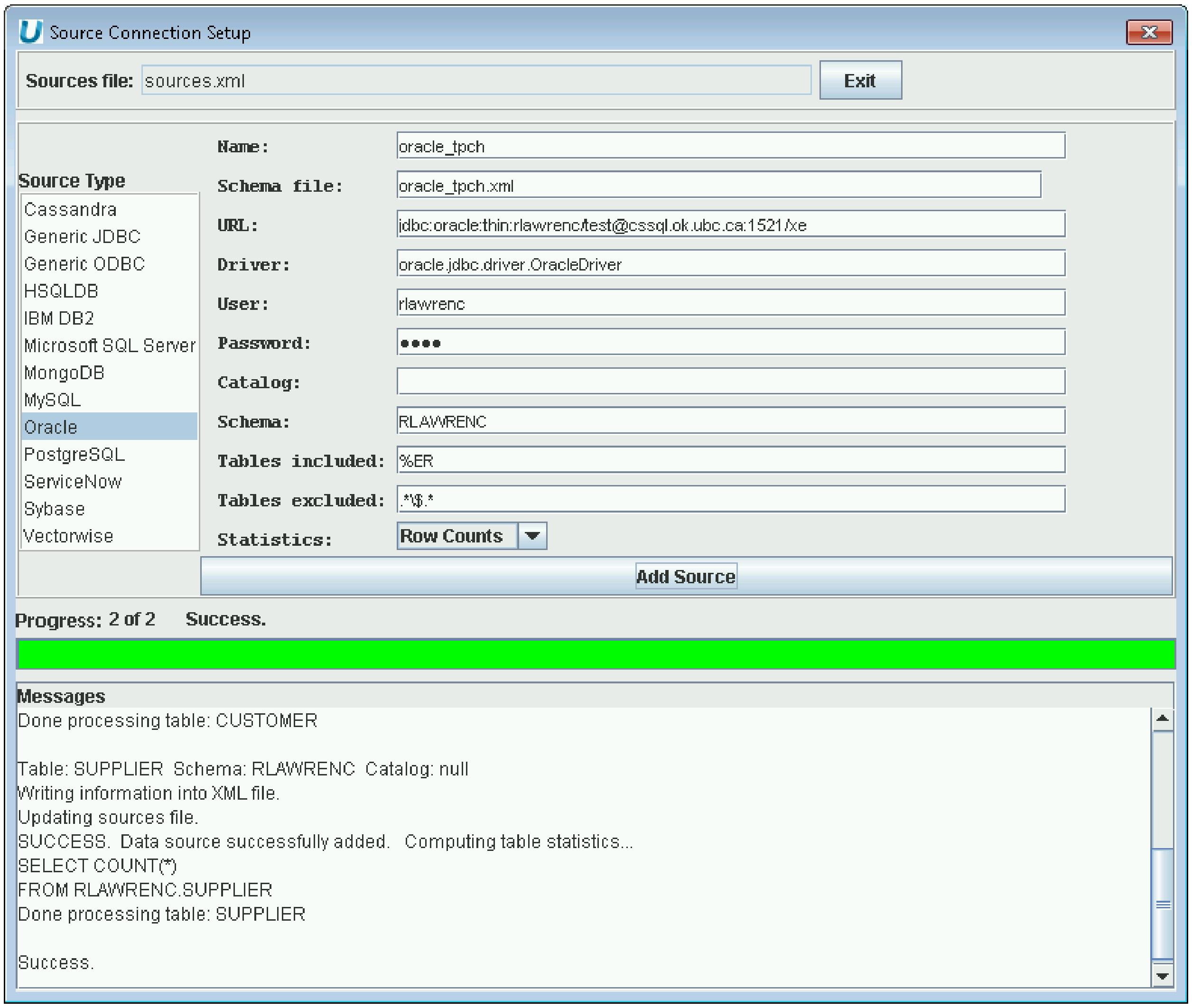
2. **Tables included** - Specify a JDBC API pattern (use % for wild card character match) to indicate which tables should be added to the extraction. For example, a pattern of R% will only add tables that begin with R.

3. **Tables excluded** - Specify a Java string pattern (use .* for matching any sequence of characters) to indicate what tables to exclude from extraction. Each database has certain default exclusion patterns to avoid extracting system tables. It is recommended to modify the exclusion pattern if there are any issues extracting too many tables, especially system tables for your data source.

4. **Statistics** - Statistics collection helps the UnityJDBC optimizer perform more efficient data virtualization queries. The default statistics collection is `Row Counts` which will calculate the number of rows for each extracted table.

The `All` statistics setting collects rows counts as well as value distribution information for each field. Some sources do not support collecting field distribution information in which case the `Row Counts` setting should be used. A setting of `None` will collect no statistics and is the fastest when performing extraction. UnityJDBC will execute queries with no statistics perfectly fine, although statistics do help in query planning for complex queries involving many tables.

Below is an example of extracting only the tables that end in `'ER'` from an Oracle database. Note how the schema is also supplied as `RLAWRENC`.



Sources file:	sources.xml
Name:	oracle_tpch
Schema file:	oracle_tpch.xml
URL:	jdbc:oracle:thin:rlawrenc/test@cssql.ok.ubc.ca:1521/xe
Driver:	oracle.jdbc.driver.OracleDriver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	RLAWRENC
Tables included:	%ER
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 2 of 2 Success.

```
Messages
Done processing table: CUSTOMER

Table: SUPPLIER  Schema: RLAWRENC  Catalog: null
Writing information into XML file.
Updating sources file.
SUCCESS.  Data source successfully added.   Computing table statistics...
SELECT COUNT(*)
FROM RLAWRENC.SUPPLIER
Done processing table: SUPPLIER

Success.
```

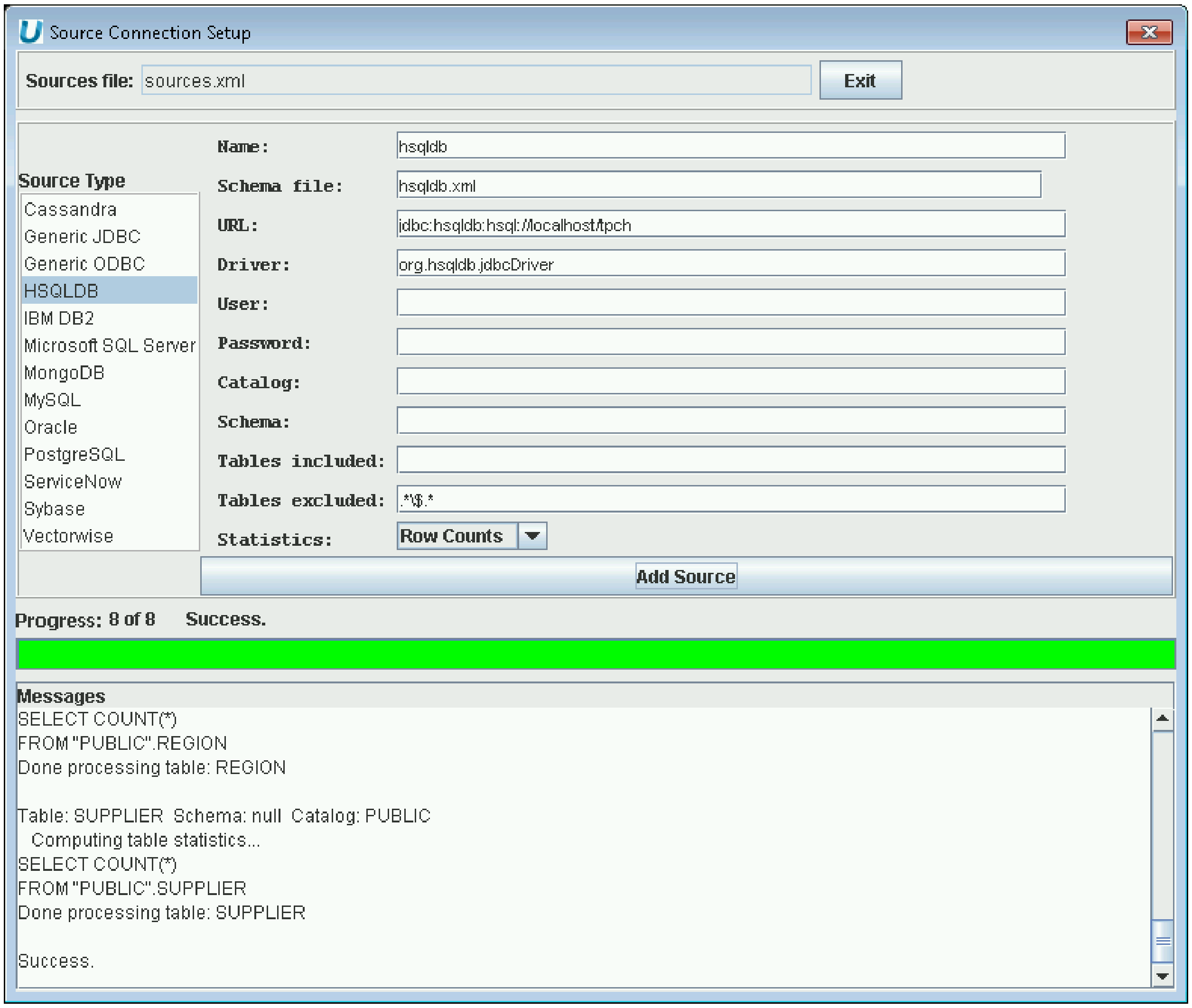
Adding an Oracle Data Source to the Data Virtualization (note use of schema name)

Multiple Database Virtualization and Querying for HSQLDB

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more HSQLDB databases with any other database system. To add a HSQLDB data source, the following information is used:

1. JDBC Driver class name: `org.hsqldb.jdbcDriver`.
2. JDBC URL: `jdbc:hsqldb:hsql://<host address>/<database name>`.

An example of adding a HSQLDB source to a data virtualzation is below.



Sources file: sources.xml

Name: hsqldb
Schema file: hsqldb.xml
URL: jdbc:hsqldb:hsql://localhost/tpch
Driver: org.hsqldb.jdbcDriver
User:
Password:
Catalog:
Schema:
Tables included:
Tables excluded: .*\$.*
Statistics: Row Counts

Progress: 8 of 8 Success.

```
SELECT COUNT(*)
FROM "PUBLIC".REGION
Done processing table: REGION

Table: SUPPLIER  Schema: null  Catalog: PUBLIC
  Computing table statistics...
SELECT COUNT(*)
FROM "PUBLIC".SUPPLIER
Done processing table: SUPPLIER

Success.
```

Adding a HSQLDB Data Source to the Data Virtualization

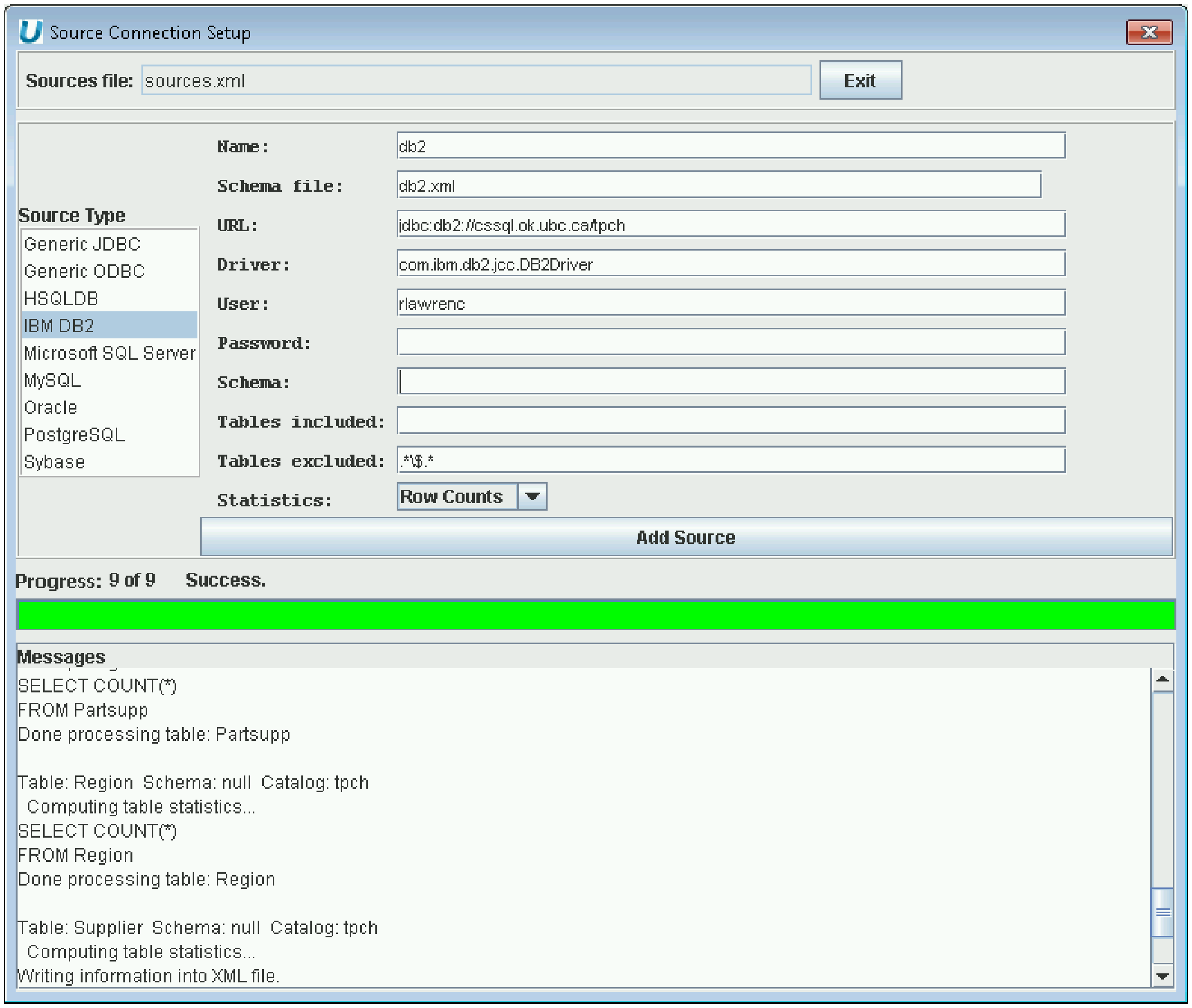
Multiple Database Virtualization and Querying for IBM DB2

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more IBM DB2 databases with any other database system. To add an IBM DB2 data source, the following information is used:

1. JDBC Driver class name: `com.ibm.db2.jcc.DB2Driver`.

2. JDBC URL: `jdbc:db2://<host name>/<database name>`.

An example of adding an IBM DB2 source to a data virtualzation is below.



Sources file: sources.xml

Name:	db2
Schema file:	db2.xml
URL:	jdbc:db2://cssql.ok.ubc.ca/tpch
Driver:	com.ibm.db2.jcc.DB2Driver
User:	rlawrenc
Password:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 9 of 9 Success.

```
SELECT COUNT(*)
FROM Partsupp
Done processing table: Partsupp

Table: Region  Schema: null  Catalog: tpch
 Computing table statistics...
SELECT COUNT(*)
FROM Region
Done processing table: Region

Table: Supplier  Schema: null  Catalog: tpch
 Computing table statistics...
Writing information into XML file.
```

Adding an IBM DB2 Data Source to the Data Virtualization

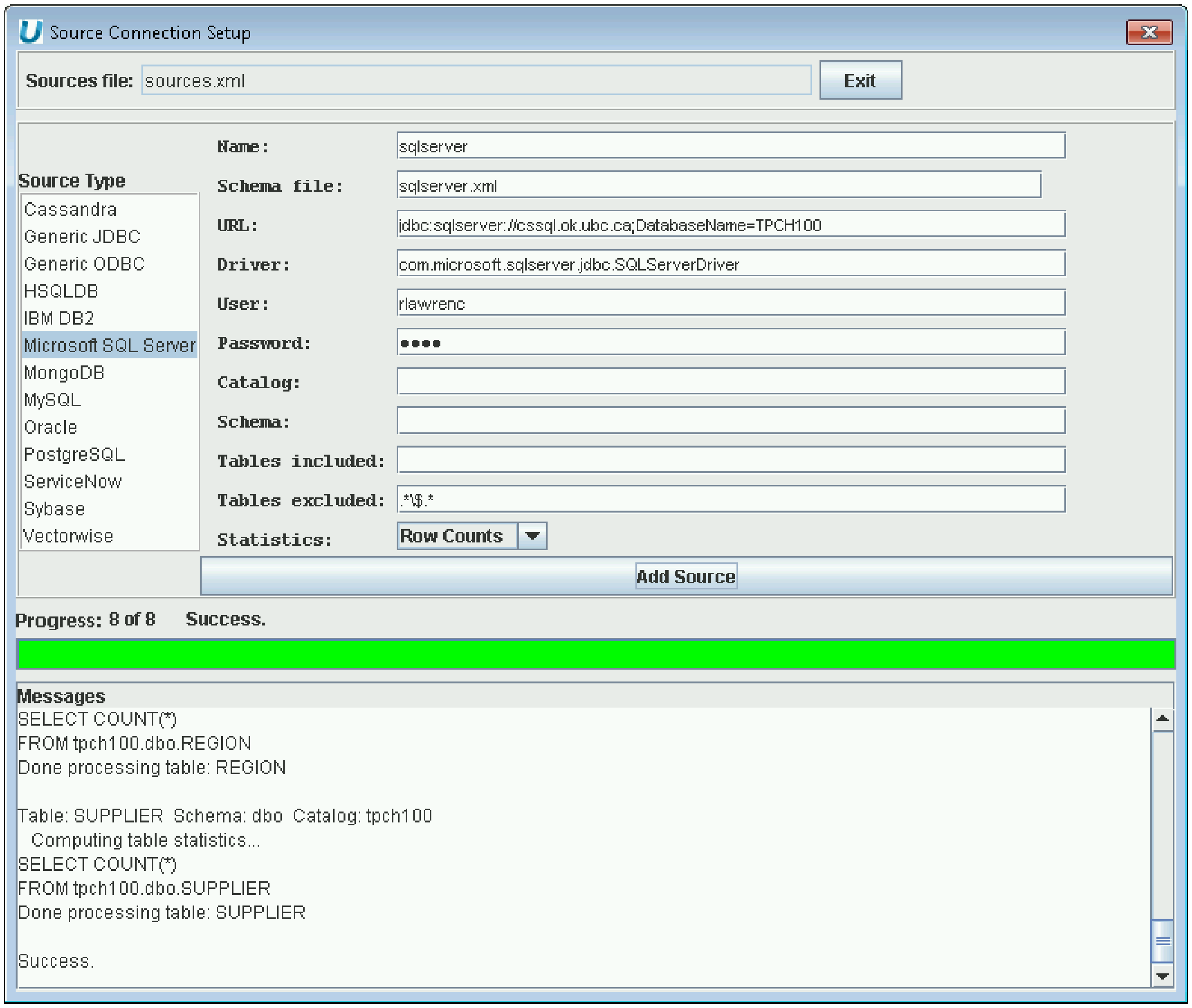
Multiple Database Virtualization and Querying for Microsoft SQL Server

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more Microsoft SQL Server databases with any other database system. To add a Microsoft SQL Server data source, the following information is used:

1. JDBC Driver class name: `com.microsoft.sqlserver.jdbc.SQLServerDriver`.

2. JDBC URL: `jdbc:sqlserver://<host>;DatabaseName=<database name>`.

An example of adding a Microsoft SQL Server data source to a data virtualzation is below.



Sources file: sources.xml

Source Type: Cassandra, Generic JDBC, Generic ODBC, HSQLDB, IBM DB2, Microsoft SQL Server, MongoDB, MySQL, Oracle, PostgreSQL, ServiceNow, Sybase, Vectorwise

Name:	sqlserver
Schema file:	sqlserver.xml
URL:	jdbc:sqlserver://cssql.ok.ubc.ca;DatabaseName=TPCH100
Driver:	com.microsoft.sqlserver.jdbc.SQLServerDriver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 8 of 8 Success.

Messages

```
SELECT COUNT(*)
FROM tpch100.dbo.REGION
Done processing table: REGION

Table: SUPPLIER  Schema: dbo  Catalog: tpch100
   Computing table statistics...
SELECT COUNT(*)
FROM tpch100.dbo.SUPPLIER
Done processing table: SUPPLIER

Success.
```

Adding a Microsoft SQL Server Data Source to the Data Virtualization

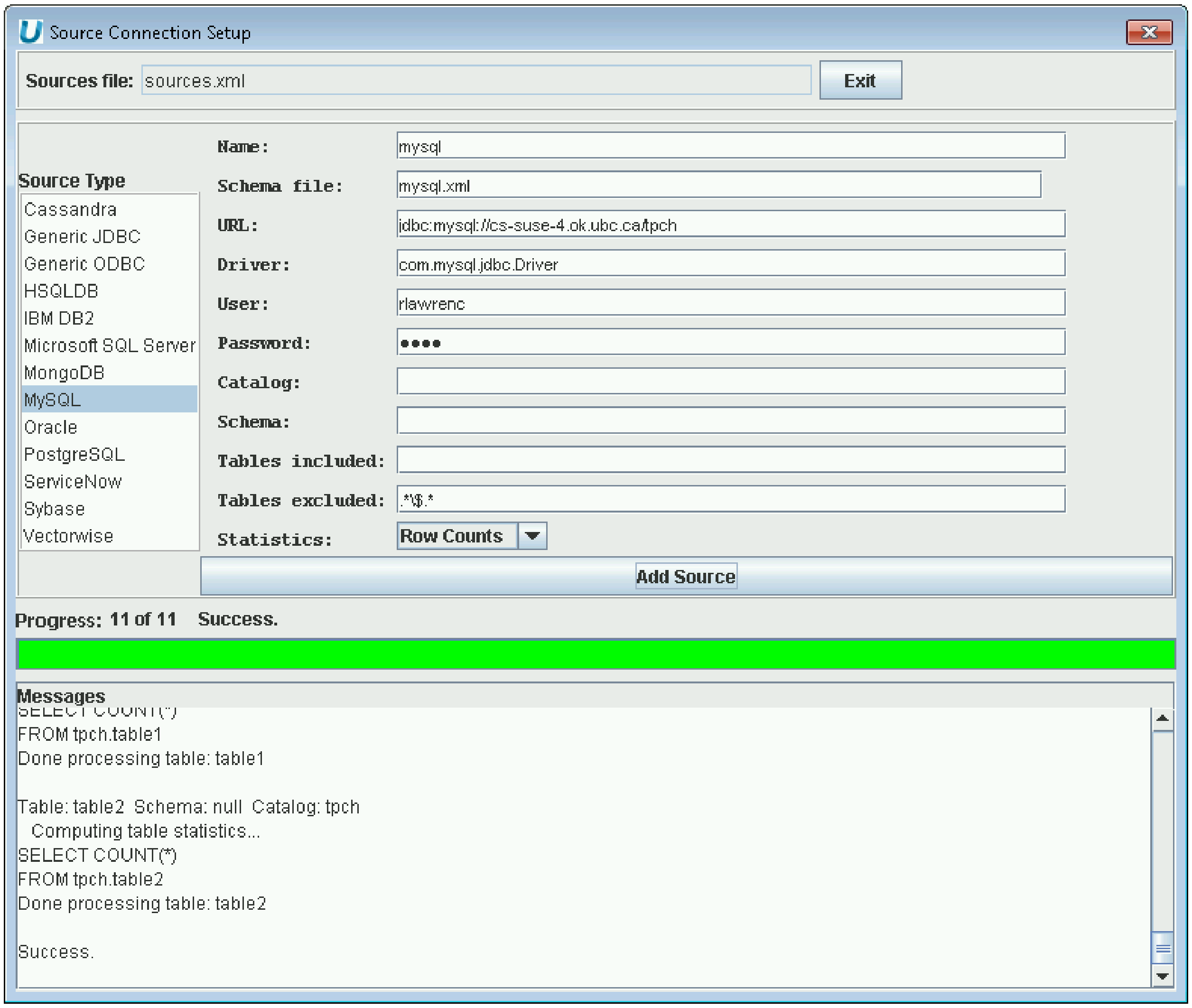
Multiple Database Virtualization and Querying for MySQL

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more MySQL databases with any other database system. To add a MySQL data source, the following information is used:

1. JDBC Driver class name: `com.mysql.jdbc.Driver`.

2. JDBC URL: `jdbc:mysql://<host address>/<database name>`.

An example of adding a MySQL source to a data virtualzation is below.



Sources file: sources.xml

Name:	mysql
Schema file:	mysql.xml
URL:	jdbc:mysql://cs-suse-4.ok.ubc.ca/tpch
Driver:	com.mysql.jdbc.Driver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 11 of 11 Success.

```
SELECT COUNT(*)
FROM tpch.table1
Done processing table: table1

Table: table2  Schema: null  Catalog: tpch
  Computing table statistics...
SELECT COUNT(*)
FROM tpch.table2
Done processing table: table2

Success.
```

Adding a MySQL Data Source to the Data Virtualization

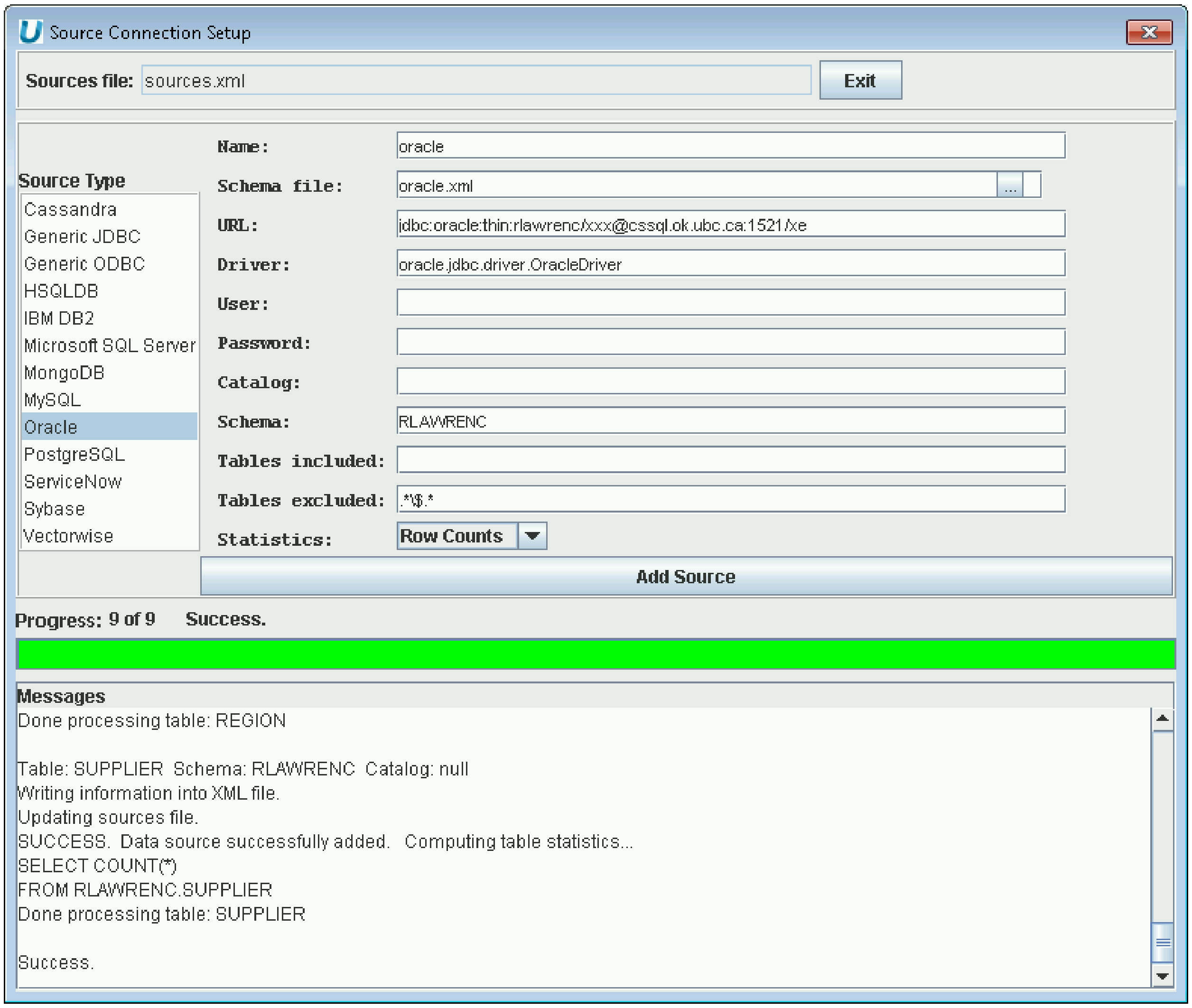
Multiple Database Virtualization and Querying for Oracle

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more Oracle databases with any other database system. To add an Oracle data source, the following information is used:

1. JDBC Driver class name: `oracle.jdbc.driver.OracleDriver`.

2. JDBC URL: `jdbc:oracle:thin:<user>/<password>@<server>:1521/<service>`.

An example of adding an Oracle data source to a data virtualzation is below.



Sources file: sources.xml

Name: oracle
Schema file: oracle.xml
URL: jdbc:oracle:thin:rlawrenc/xxx@cssql.ok.ubc.ca:1521/xe
Driver: oracle.jdbc.driver.OracleDriver
User:
Password:
Catalog:
Schema: RLAWRENC
Tables included:
Tables excluded: .*\$.*
Statistics: Row Counts

Progress: 9 of 9 Success.

```
Done processing table: REGION

Table: SUPPLIER  Schema: RLAWRENC  Catalog: null
Writing information into XML file.
Updating sources file.
SUCCESS.  Data source successfully added.   Computing table statistics...
SELECT COUNT(*)
FROM RLAWRENC.SUPPLIER
Done processing table: SUPPLIER

Success.
```

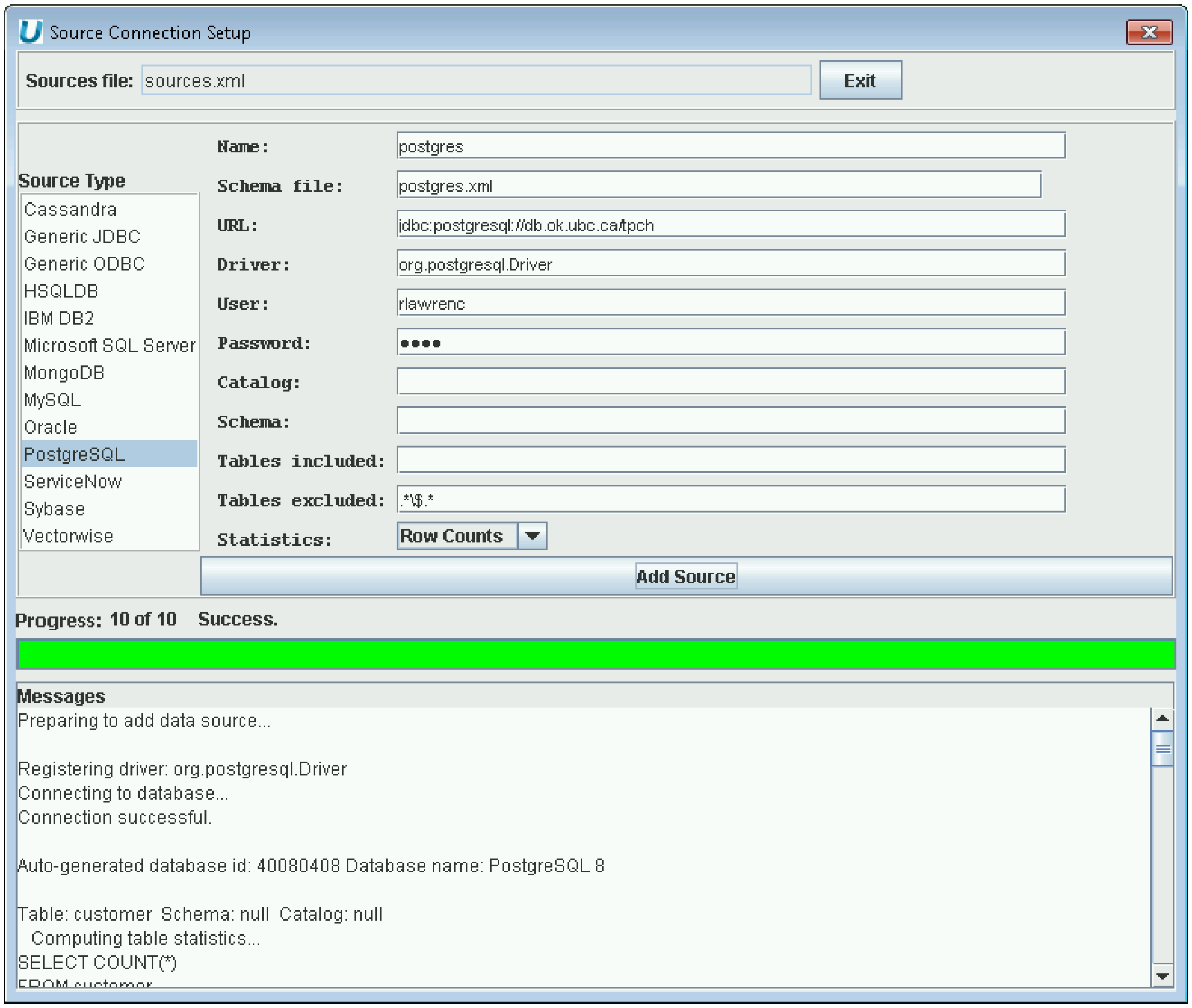
Adding an Oracle Data Source to the Data Virtualization

Multiple Database Virtualization and Querying for PostgreSQL

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more PostgreSQL databases with any other database system. To add a PostgreSQL data source, the following information is used:

1. JDBC Driver class name: `org.postgresql.Driver`.
2. JDBC URL: `jdbc:postgresql://<server>/<database>?user=<userId>&password=<password>`.

An example of adding a PostgreSQL source to a data virtualzation is below.



Sources file:	sources.xml
Name:	postgres
Schema file:	postgres.xml
URL:	jdbc:postgresql://db.ok.ubc.ca/tpch
Driver:	org.postgresql.Driver
User:	rlawrenc
Password:	••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 10 of 10 Success.

Messages

```
Preparing to add data source...

Registering driver: org.postgresql.Driver
Connecting to database...
Connection successful.

Auto-generated database id: 40080408 Database name: PostgreSQL 8

Table: customer  Schema: null  Catalog: null
  Computing table statistics...
SELECT COUNT(*)
```

Adding a PostgreSQL Data Source to the Data Virtualization

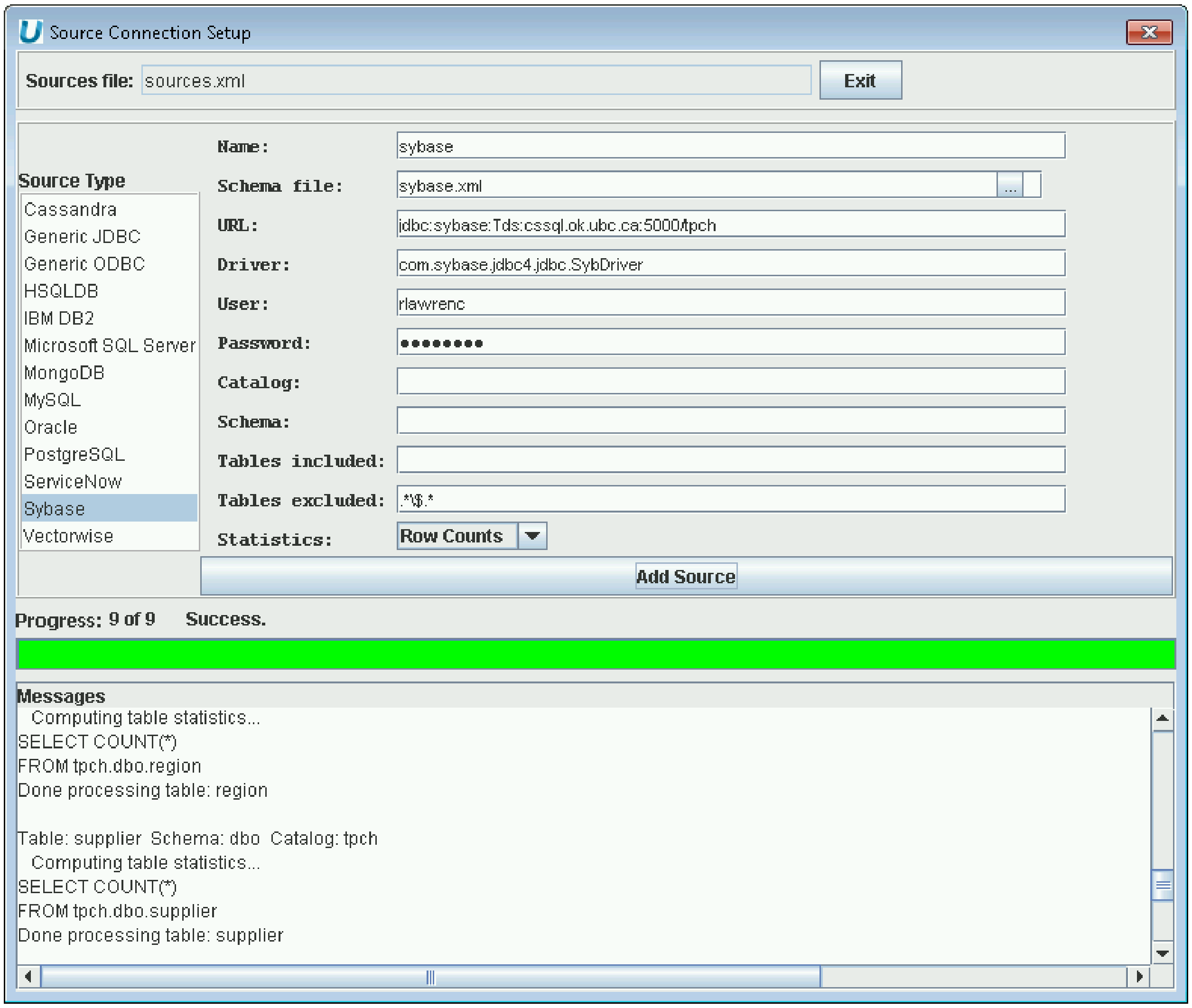
Multiple Database Virtualization and Querying for Sybase

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more Sybase databases with any other database system. To add a Sybase data source, the following information is used:

1. JDBC Driver class name: `com.sybase.jdbc4.jdbc.SybDriver`.

2. JDBC URL: `jdbc:sybase:Tds:<server>:<port>/<database>?user=<userid>&password=<password>`.

An example of adding a Sybase source to a data virtualzation is below.



Sources file:	sources.xml
Name:	sybase
Schema file:	sybase.xml
URL:	jdbc:sybase:Tds:cssql.ok.ubc.ca:5000/tpch
Driver:	com.sybase.jdbc4.jdbc.SybDriver
User:	rlawrenc
Password:	••••••••
Catalog:	
Schema:	
Tables included:	
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 9 of 9 Success.

```
   Computing table statistics...
SELECT COUNT(*)
FROM tpch.dbo.region
Done processing table: region

Table: supplier  Schema: dbo  Catalog: tpch
   Computing table statistics...
SELECT COUNT(*)
FROM tpch.dbo.supplier
Done processing table: supplier
```

Adding a Sybase Data Source to the Data Virtualization

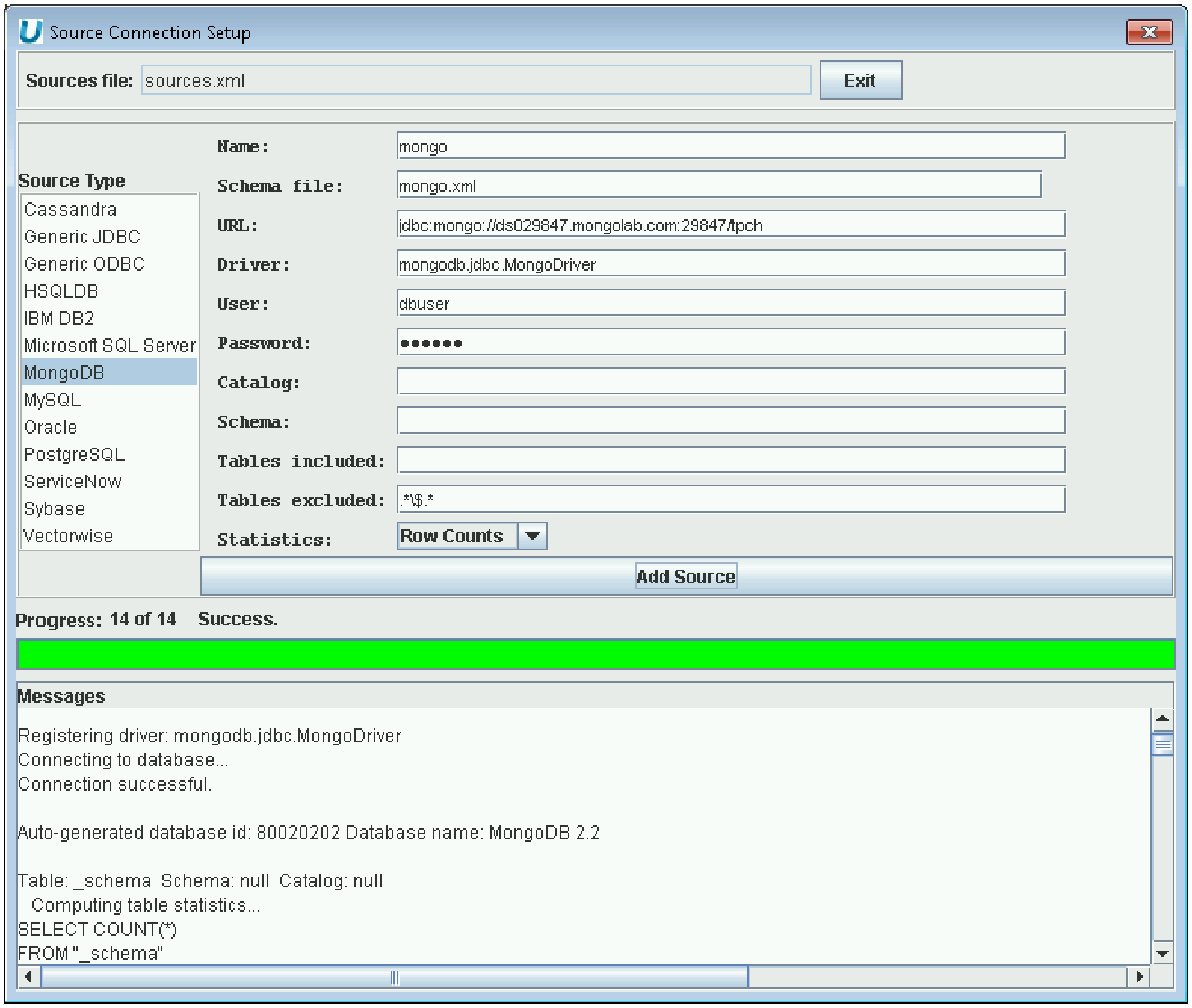
Multiple Database Virtualization and Querying for MongoDB

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization of one or more MongoDB databases (collections) with any other database system. Note that a MongoDB JDBC driver is built into UnityJDBC and can be used separately from UnityJDBC directly. For more information go to http://www.unityjdbc.com/mongojdbc To add a MongoDB data source, the following information is used:

1. JDBC Driver class name: `mongodb.jdbc.MongoDriver`.

2. JDBC URL: `jdbc:mongo://<server>:1521/<database>`.

An example of adding a MongoDB data source to a data virtualzation is below.



Sources file: sources.xml

Name: mongo
Schema file: mongo.xml
URL: jdbc:mongo://ds029847.mongolab.com:29847/tpch
Driver: mongodb.jdbc.MongoDriver
User: dbuser
Password: ••••••
Catalog:
Schema:
Tables included:
Tables excluded: .*\$.*
Statistics: Row Counts

Progress: 14 of 14 Success.

```
Registering driver: mongodb.jdbc.MongoDriver
Connecting to database...
Connection successful.

Auto-generated database id: 80020202 Database name: MongoDB 2.2

Table: _schema  Schema: null  Catalog: null
  Computing table statistics...
SELECT COUNT(*)
FROM "_schema"
```

Adding a MongoDB Data Source to the Data Virtualization

Multiple Database Virtualization and Querying for Other JDBC/ODBC Sources

Using UnityJDBC SourceBuilder, it is possible to build a data virtualization for any data source that supports JDBC or ODBC including Microsoft Access databases, Excel files, text files, and any data source that has a JDBC driver.

Chapter 4. Multiple Database Programming with UnityJDBC

Using the Sample Programs

UnityJDBC is a universal query translator. It allows you to develop your programs without worrying about the underlying database. All SQL statements executed with UnityJDBC are translated for the database used. Even if you do not need multiple database queries, data virtualization, or queries that span multiple different databases, UnityJDBC simplifies your development by handling all the issues with SQL dialects. If a function is not supported by your database, UnityJDBC will execute it internally. If you forget a function name for a database, it will translate to the correct function call for the particular source. This allows you the freedom to write your SQL code in a database independent way. You no longer have to make major changes to your code if you change database systems. If you want full control, you can use the UnityJDBC database engine directly in your code to join ResultSets with each other regardless of their source, and perform dynamic filtering, ordering, and analysis.

Sample programs are provided in the directory `code`. Here is a list of the programs and the features they demonstrate:

1. `ExampleQuery.java` - a query example that joins data across two databases
2. `ExampleUpdate.java` - demonstrates `INSERT/UPDATE/DELETE` and how to store a cross-database query result into a table
3. `ExampleMetaData.java` - query example showing how to extract metadata information
4. `ExampleEngine.java` - an advanced example that shows how users can use the UnityJDBC database engine directly
5. `ExampleNoFileConnection.java` - example showing how to configure UnityJDBC in code without using XML files

All of these examples use a local HSQL database that can be started using the script `startDB.bat` or `startDB.sh` in the directory `sampleDB/hsqldb` in the installation folder.

To compile and run any of these sample programs make sure you are in the `code` directory and execute the following commands:

```
javac test/ExampleQuery.java
```

```
java test.ExampleQuery
```

If you have CLASSPATH issues, you can explicitly indicate the location of the HSQL JDBC driver and the UnityJDBC driver by:

```
javac        -cp.;../UnityJDBC.jar;../sampleDB/hsqldb/hsqldb.jar        test/
ExampleQuery.java
```

```
java -cp.;../UnityJDBC.jar;../sampleDB/hsqldb/hsqldb.jar test.ExampleQuery
```

Using ExampleQuery.java

The `ExampleQuery.java` demonstrates the basic features of the UnityJDBC driver. The code is below.

```
import java.sql.*;

public class ExampleQuery
{
// URL for sources.xml file specifying what databases to integrate.
// This file must be locally accessible or available via http URL.
static String url="jdbc:unity://test/xspec/UnityDemo.xml";

public static void main(String [] args) throws Exception
{
Connection con = null;
Statement stmt = null;
ResultSet rst;

try {
   // Create new instance of UnityDriver and make connection
   System.out.println("\nRegistering driver.");
   Class.forName("unity.jdbc.UnityDriver");

   System.out.println("\nGetting connection:  "+url);
   con = DriverManager.getConnection(url);
   System.out.println("\nConnection successful for "+ url);

   System.out.println("\nCreating statement.");
   stmt = con.createStatement();
   // Unity supports scrollable ResultSets,
   //  but better performance with FORWARD_ONLY
   // stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
   //                            ResultSet.CONCUR_READ_ONLY);

   // A query is exactly like SQL.
   // Attributes should be FULLY qualified: database.table.field
   // Statement must end with a semi-colon ;
   // This query performs cross-database join on the client-side
   String sql =
     "SELECT PartDB.Part.P_NAME, OrderDB.LineItem.L_QUANTITY,"
   + " OrderDB.Customer.C_Name, PartDB.Supplier.s_name"
   + " FROM OrderDB.CUSTOMER, OrderDB.LINEITEM, OrderDB.ORDERS,"
   + "      PartDB.PART, PartDB.Supplier"
   + " WHERE OrderDB.LINEITEM.L_PARTKEY = PartDB.PART.P_PARTKEY AND"
   + "       OrderDB.LINEITEM.L_ORDERKEY = OrderDB.ORDERS.O_ORDERKEY"
   + "       AND OrderDB.ORDERS.O_CUSTKEY = OrderDB.CUSTOMER.C_CUSTKEY"
   + "       AND PartDB.supplier.s_suppkey = OrderDB.lineitem.l_suppkey"
   + " AND OrderDB.Customer.C_Name = 'Customer#000000025';";

   // Note: Client's local JVM is used to process some operations.
   // For large queries, this may require setting a large heap space.
   // JVM command line parameters: 0 -Xms500m -Xmx500m
   // These parameters set heap space to 500 MB.
   rst = stmt.executeQuery(sql);

   System.out.println("\n\nTHE RESULTS:");
   int i=0;
   long timeStart = System.currentTimeMillis();
   long timeEnd;
   ResultSetMetaData meta = rst.getMetaData();

   System.out.println("Total columns: " + meta.getColumnCount());
   System.out.print(meta.getColumnName(1));
   for (int j = 2; j <= meta.getColumnCount(); j++)
      System.out.print(", " + meta.getColumnName(j));
   System.out.println();

   while (rst.next()) {
      System.out.print(rst.getObject(1));
      for (int j = 2; j <= meta.getColumnCount(); j++)
         System.out.print(", " + rst.getObject(j));
      System.out.println();
      i++;
   }

   timeEnd = System.currentTimeMillis();
   System.out.println("Query took: "+
                     ((timeEnd-timeStart)/1000)+" seconds");
   System.out.println("Number of results printed: "+i);
   stmt.close();
   System.out.println("\nOPERATION COMPLETED SUCCESSFULLY!");
}
catch (SQLException ex)
{   System.out.println("SQLException: " + ex);
}
finally
{
   if (con != null)
   try{   con.close();   }
   catch (SQLException ex)
   {   System.out.println("SQLException: " + ex); }
}
}
```

The UnityJDBC driver behaves exactly like other JDBC drivers. The basic steps for querying a database with a JDBC driver are:

1. **Load the driver (optional)** - This is done by **`Class.forName("unity.jdbc.UnityDriver");`**

2. **Make a connection** - A connection is made to a database by providing the database URL and other properties including user id and password. This example is using the `DriverManager` to make the connection (**`con = DriverManager.getConnection(url);`**). Note that the URL is of the form `jdbc:unity://<path_to_sources_file>`. In this case, the URL is `jdbc:unity://test/xspec/UnityDemo.xml`.This path may be an absolute or relative path on the machine. It is also possible to retrieve encrypted XML files from a network source. The sources file provides the connection information for the individual data sources for use by UnityJDBC.

3. **Execute a statement** - UnityJDBC follows the JDBC API for creating statements and executing queries and updates. There are some methods unique to UnityJDBC which are covered in a later section. Standard SQL syntax is supported. The major difference is that tables in different databases can be referenced in the same query. This is accomplished using the syntax `database.table` to refer to tables and `database.table.field` to refer to fields. (Note that aliasing using AS is supported.) *If full names are not provided, UnityJDBC will attempt to match as appropriate, but it will generate errors if the provided names are not unique.*

This file is a good one to modify to start your own program. Simply change the class and file name, the URL to the location of your source list file, and the query executed, and you are done.

Using ExampleUpdate.java

UnityJDBC natively supports `INSERT`, `UPDATE`, and `DELETE` statements on a single database. These statements can be executed in by-pass mode in which case UnityJDBC does not parse or validate the statement and passes it straight to the JDBC driver for the corresponding database. In native mode, UnityJDBC will parse and validate the statement before passing it to the data source. Note that the basic `INSERT`, `UPDATE`, and `DELETE` statements operate only on a single table in SQL, so no cross-database query processing is necessary. A sample of the code in `ExampleUpdate.java` is below.

```
Class.forName("unity.jdbc.UnityDriver");
con = DriverManager.getConnection(url);
stmt = con.createStatement();

// Example #1: Basic query
String sql = "SELECT * FROM mydb.Customer;";
rst = stmt.executeQuery(sql);
printResult(rst);

// Example #2: DELETE using native parsing
String databaseName = "mydb";
sql = "DELETE FROM mydb.customer WHERE id = 51 or id=52;";
stmt.executeUpdate(sql);

// Example #3: INSERT (by-pass method)
sql = "INSERT INTO Customer (id,firstname,lastname,street,city) "
     + " VALUES (51,'Joe','Smith','Drury Lane', 'Detroit')";
((UnityStatement) stmt).executeByPassQuery(databaseName,sql);

// Example #4: INSERT - Unity Parsed
sql = "INSERT INTO mydb.Customer (id, firstname, "
          + " lastname, street, city) "
          + " VALUES (52,'Fred','Jones','Smith Lane', 'Chicago');";
stmt.executeUpdate(sql);

// Example #5: INSERT INTO (SELECT...) across databases
sql = "INSERT INTO emptydb.customer (SELECT * FROM mydb.customer);";
stmt.executeUpdate(sql);

// Prove that we transferred the data
sql = "SELECT * FROM emptydb.Customer;";
rst = stmt.executeQuery(sql);
printResult(rst);
```

Note that you can use the by-pass feature to execute any statement on a source database that UnityJDBC does not natively support. Experimental results show that the by-pass features adds insignificant overhead compared to calling the source JDBC driver directly. Thus, client code only needs to load and use the UnityJDBC driver directly. This results in more portable code that can be more easily moved between database systems.

When UnityJDBC parses the SQL, you can use table and field references that are prefixed with the database name. This is optional if the table and field names are unique across all databases, otherwise the database name is required.

The database name is assigned in the schema file describing the source and does not have to be the same as the system name used by the database system itself. That is, the name can be set by the developer using UnityJDBC.

Multiple source UnityJDBC queries can be used with an INSERT INTO statement to populate a table in the database. This allows a user to write a cross-database query to collect information from multiple sources and then insert the result back into a table in any one of the sources. Currently, the only restriction is that the table that will be inserted into must exist and must be present in the schema file describing the source.

Using ExampleMetadata.java

`ExampleMetadata.java` demonstrates UnityJDBC's support for the `DatabaseMetaData` interface. This interface functions exactly according to the standard with the major difference that metadata is returned for all databases in the data virtualization rather than from a single database. That is, all your "integrated" databases really do appear as a single database to your application.

Using ExampleEngine.java

Embedded in the UnityJDBC driver is a complete relational engine. This is required to process cross-database join queries. Most users will not interact with the engine directly, and their only contact with the engine may be to increase the JVM heap sizes for processing large cross-database queries. However, all of the relational operators of selection, projection, and join are available for direct use in your programs. The join algorithms support sources larger than main memory, and allow you the full power of combining ResultSets from multiple databases. It is also possible to explicitly track global query progress on a per operator basis or perform your own optimization of queries after the UnityJDBC optimizer has built an execution tree.

Using ExampleNoFileConnection.java

`ExampleNoFileConnection.java` demonstrates UnityJDBC's ability to be dynamically configured at run-time including adding, removing, or updating sources. It is possible to dynamically build a virtual database without using XML configuration files by interacting with UnityJDBC through its metadata API.

Chapter 5. Using UnityJDBC with Query and Reporting Software

The UnityJDBC SourceBuilder is a simple query utility for cross-database joins and data virtualization. In most cases, you will use UnityJDBC to perform data virtualization with reporting software such as JasperReports and Splunk or query software such as SQuirreL SQL or Aqua Data Studio. Once you have used SourceBuilder to build the virtual source and schema XML configuration files, you use UnityJDBC like any other JDBC driver. This section contains examples on how to install and use UnityJDBC data virtualization in popular software systems.

UnityJDBC Data Virtualization with SQuirreL SQL

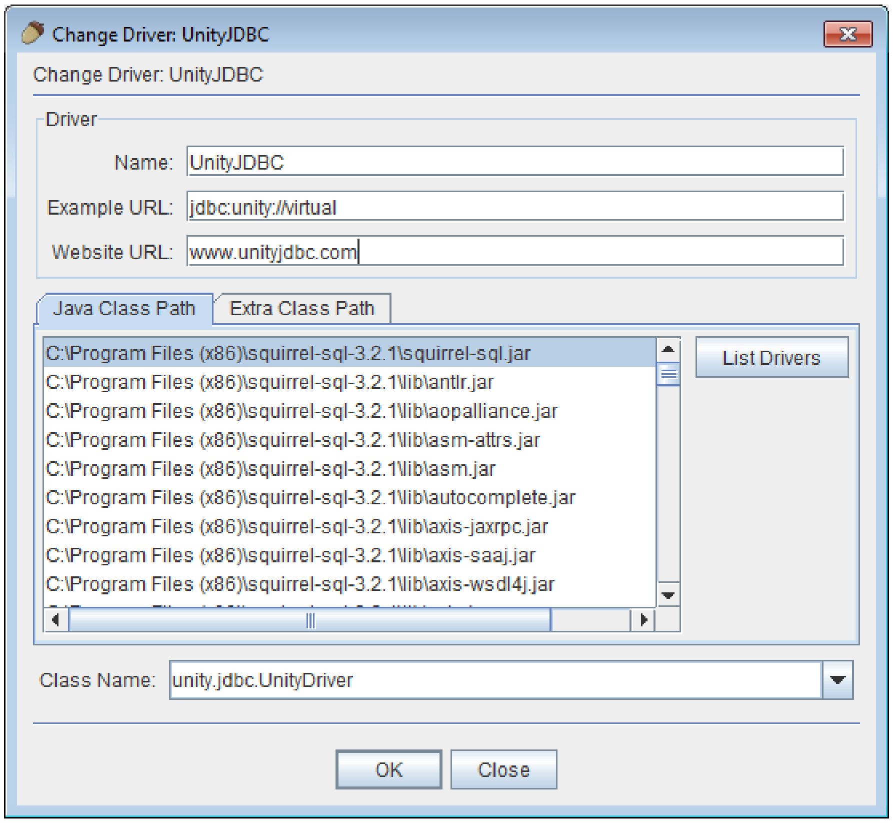
UnityJDBC is integrated into SQuirreL SQL allowing users to build SQL queries that join data from multiple sources directly within SQuirreL. The multiple source query plugin allows SQuirreL users to create a virtual data source that may consist of multiple data sources on different servers and platforms. The user can enter one SQL query to combine and join information from multiple sources.

Benefits:

- The multisource plugin powered by UnityJDBC allows SQuirreL SQL to support multiple source queries.
- No data source or server changes are required.
- The plugin supports standard SQL including joins, group by, aggregation, LIMIT, and ordering where tables may come from one or more sources.
- The plugin will perform function translation where a user requests a function that is not supported on a certain source.

UnityJDBC can be installed directly as a plug-in through the SQuirreL SQL plug-in interface. It is also possible to download the UnityJDBC distribution and replace the `unityjdbc.jar` in the plug-in with the latest from UnityJDBC.

1. **Registering the UnityJDBC Driver** - By adding a driver.


Change Driver: UnityJDBC

Change Driver: UnityJDBC

Driver

Name: UnityJDBC

Example URL: jdbc:unity://virtual

Website URL: www.unityjdbc.com

Java Class Path | Extra Class Path

```
C:\Program Files (x86)\squirrel-sql-3.2.1\squirrel-sql.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\antlr.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\aopalliance.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\asm-attrs.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\asm.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\autocomplete.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\axis-jaxrpc.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\axis-saaj.jar
C:\Program Files (x86)\squirrel-sql-3.2.1\lib\axis-wsdl4j.jar
```

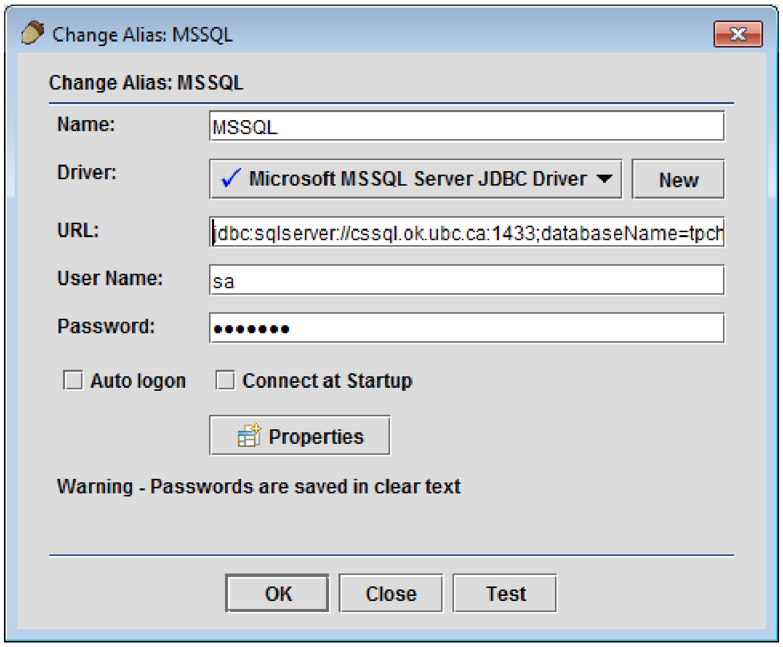
List Drivers

Class Name: unity.jdbc.UnityDriver

OK Close

Adding the UnityJDBC to SQuirreL SQL

2. **Registering your Data Sources** - Register your data sources as usual. In this example, we will perform data virtualization of multiple databases on Microsoft SQL Server, MySQL, Oracle, and PostgreSQL. Any database with a JDBC driver is supported including those accessible using the JDBC-ODBC bridge.


Change Alias: MSSQL

Change Alias: MSSQL

Name: MSSQL

Driver: ✓ Microsoft MSSQL Server JDBC Driver | New

URL: jdbc:sqlserver://cssql.ok.ubc.ca:1433;databaseName=tpch

User Name: sa

Password: •••••••

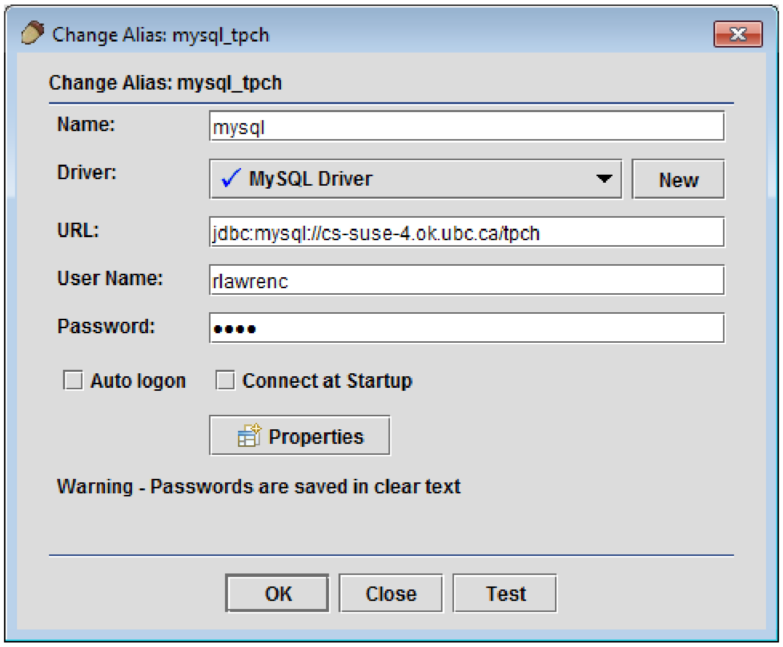
☐ Auto logon ☐ Connect at Startup

Properties

Warning - Passwords are saved in clear text

OK Close Test

Registering a Microsoft SQL Server Source in SQuirreL SQL



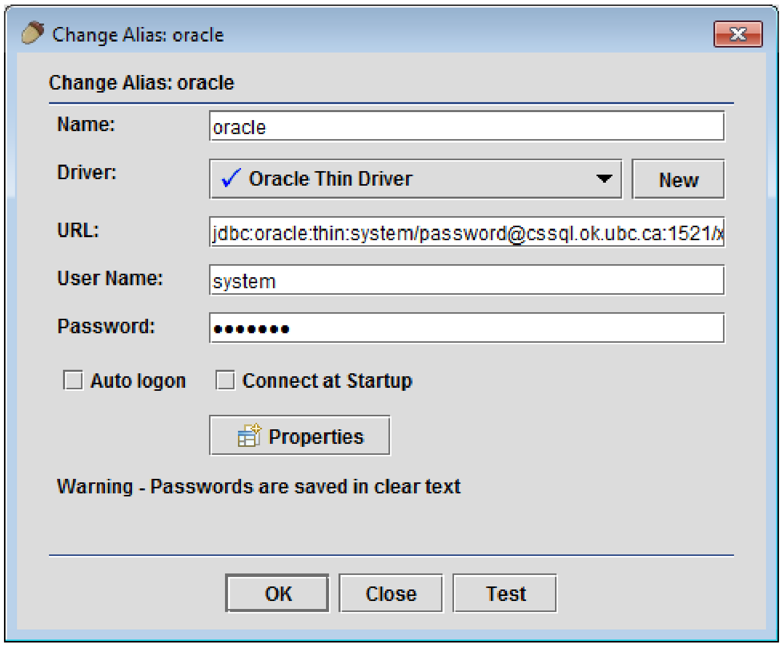
Change Alias: mysql_tpch

Name:	mysql
Driver:	✓ MySQL Driver
URL:	jdbc:mysql://cs-suse-4.ok.ubc.ca/tpch
User Name:	rlawrenc
Password:	●●●●

☐ Auto logon ☐ Connect at Startup

Warning - Passwords are saved in clear text

Registering a MySQL Source in SQuirreL SQL



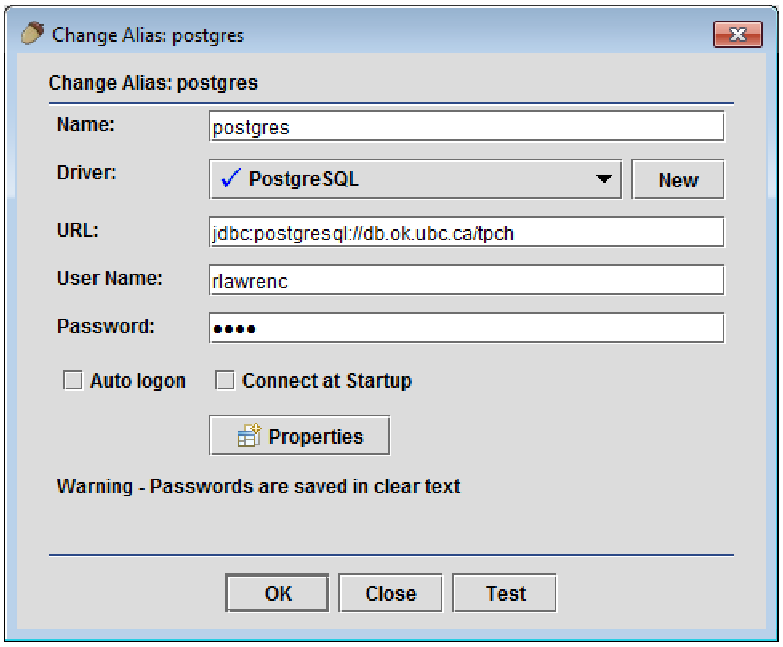
Change Alias: oracle

Name:	oracle
Driver:	✓ Oracle Thin Driver
URL:	jdbc:oracle:thin:system/password@cssql.ok.ubc.ca:1521/x
User Name:	system
Password:	●●●●●●●

☐ Auto logon ☐ Connect at Startup

Warning - Passwords are saved in clear text

Registering an Oracle Source in SQuirreL SQL


Change Alias: postgres

Change Alias: postgres

Name: postgres

Driver: PostgreSQL

URL: jdbc:postgresql://db.ok.ubc.ca/tpch

User Name: rlawrenc

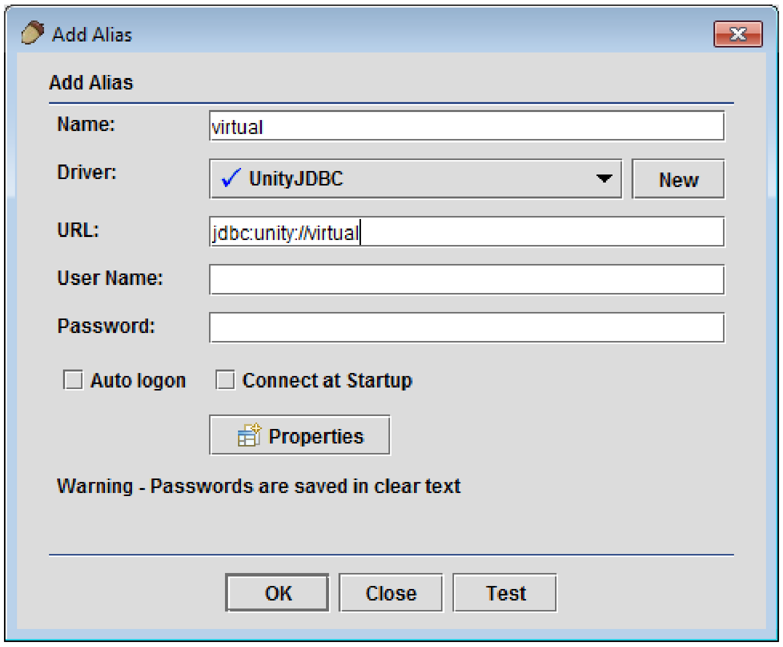
Password: ••••

☐ Auto logon ☐ Connect at Startup

Warning - Passwords are saved in clear text

Registering a PostgreSQL Source in SQuirreL SQL

3. **Create a Multiple Database Alias** - Make sure you have registered the UnityJDBC driver (during installation). Create an alias consisting of virtual sources. The name field can be any name. It does not have to be virtual. If you are using the virtualization embedded into the plugin, the URL is `jdbc:unity://virtual`. If you have previously created a data virtualization using the SourceBuilder utility, then the URL should be the file location of the sources file previously created.


Add Alias

Add Alias

Name: virtual

Driver: UnityJDBC

URL: jdbc:unity://virtual

User Name:

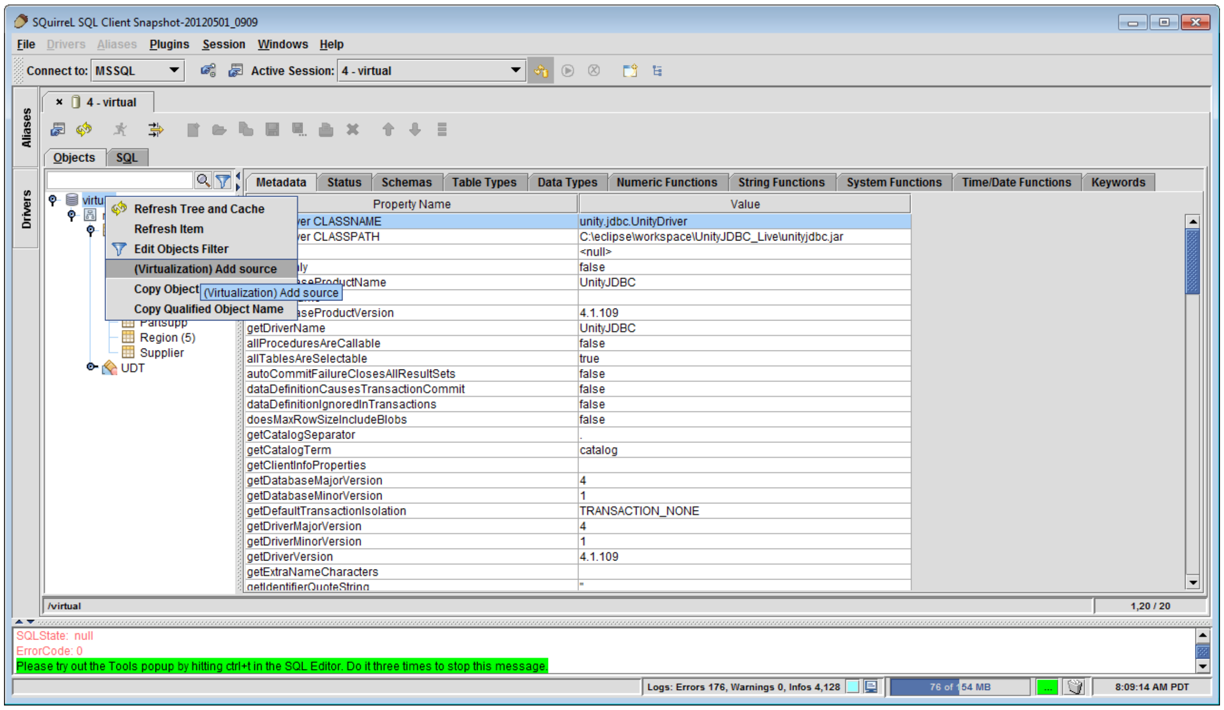
Password:

☐ Auto logon ☐ Connect at Startup

Warning - Passwords are saved in clear text

Add a Data Virtualization Alias for Multiple Sources in SQuirreL SQL

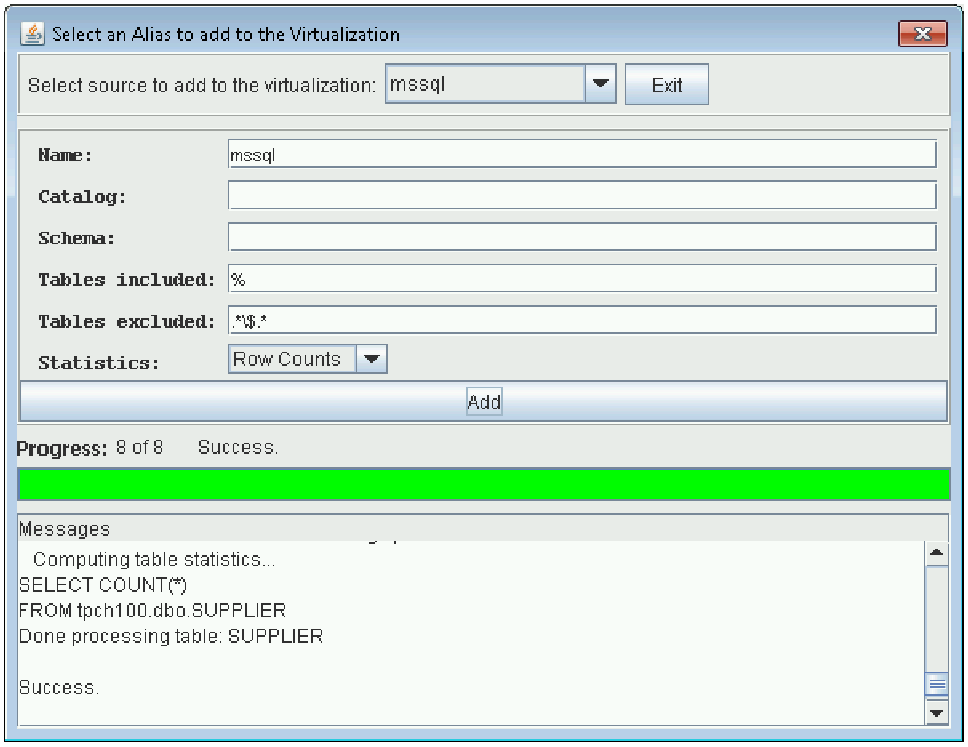
4. **Add Microsoft SQL Server to Data Virtualization** - Right-click on the root object in the object tree, and select `(Virtualization) Add Source`. User selects the Microsoft SQL Server source to add to the data virtualization.


SQuirreL SQL Client Snapshot-20120501_0909
File Drivers Aliases Plugins Session Windows Help
Connect to: MSSQL
Active Session: 4 - virtual
4 - virtual
Aliases
Drivers
Objects SQL
Metadata Status Schemas Table Types Data Types Numeric Functions String Functions System Functions Time/Date Functions Keywords
Refresh Tree and Cache
Refresh Item
Edit Objects Filter
(Virtualization) Add source
Copy Object
Copy Qualified Object Name
(Virtualization) Add source
virtu
Partsupp
Region (5)
Supplier
UDT
/virtual
1,20 / 20
Logs: Errors 176, Warnings 0, Infos 4,128
76 of 154 MB
8:09:14 AM PDT

Property Name	Value
ver CLASSNAME	unity.jdbc.UnityDriver
ver CLASSPATH	C:\eclipse\workspace\UnityJDBC_Live\unityjdbc.jar
	<null>
ly	false
aseProductName	UnityJDBC
aseProductVersion	4.1.109
getDriverName	UnityJDBC
allProceduresAreCallable	false
allTablesAreSelectable	true
autoCommitFailureClosesAllResultSets	false
dataDefinitionCausesTransactionCommit	false
dataDefinitionIgnoredInTransactions	false
doesMaxRowSizeIncludeBlobs	false
getCatalogSeparator	.
getCatalogTerm	catalog
getClientInfoProperties	
getDatabaseMajorVersion	4
getDatabaseMinorVersion	1
getDefaultTransactionIsolation	TRANSACTION_NONE
getDriverMajorVersion	4
getDriverMinorVersion	1
getDriverVersion	4.1.109
getExtraNameCharacters	
getIdentifierQuoteString	"

```
SQLState:  null
ErrorCode: 0
Please try out the Tools popup by hitting ctrl+t in the SQL Editor. Do it three times to stop this message.
```

Adding Microsoft SQL Server Source to the Data Virtualization in SQuirreL SQL



Select source to add to the virtualization: mssql

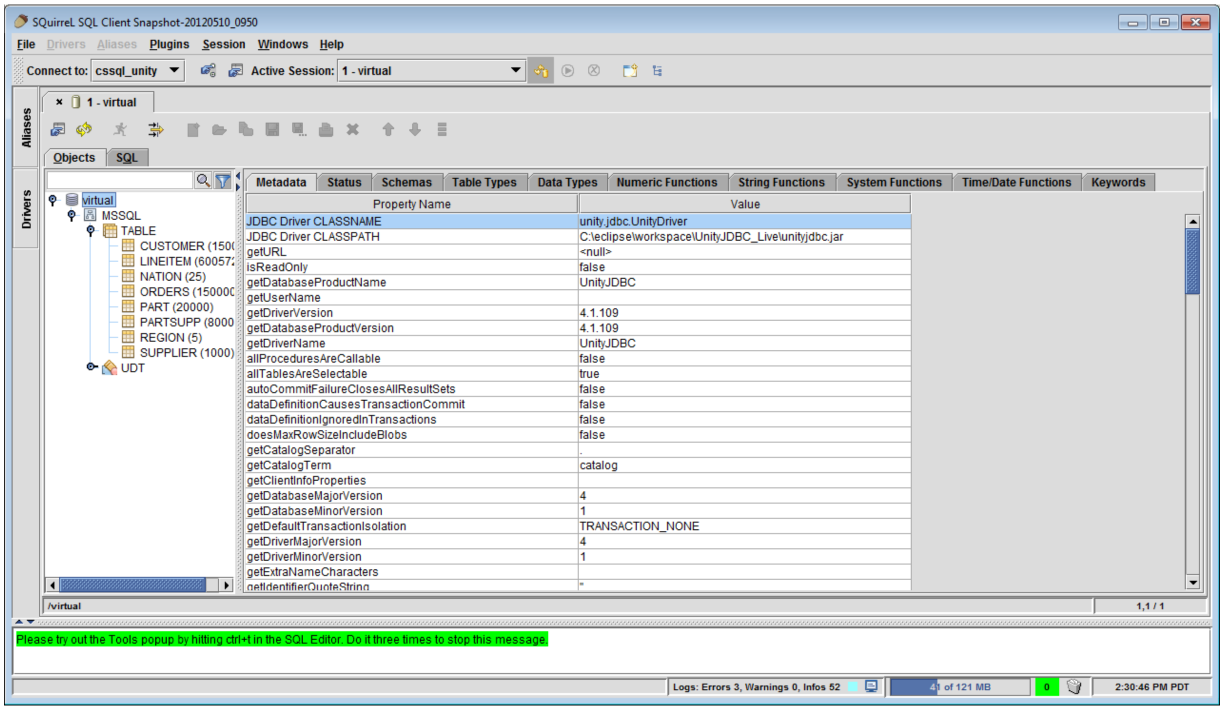
Name:	mssql
Catalog:	
Schema:	
Tables included:	%
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 8 of 8 Success.

Messages

```
  Computing table statistics...
SELECT COUNT(*)
FROM tpch100.dbo.SUPPLIER
Done processing table: SUPPLIER

Success.
```

Prompt to add Microsoft SQL Server Source to the Data Virtualization

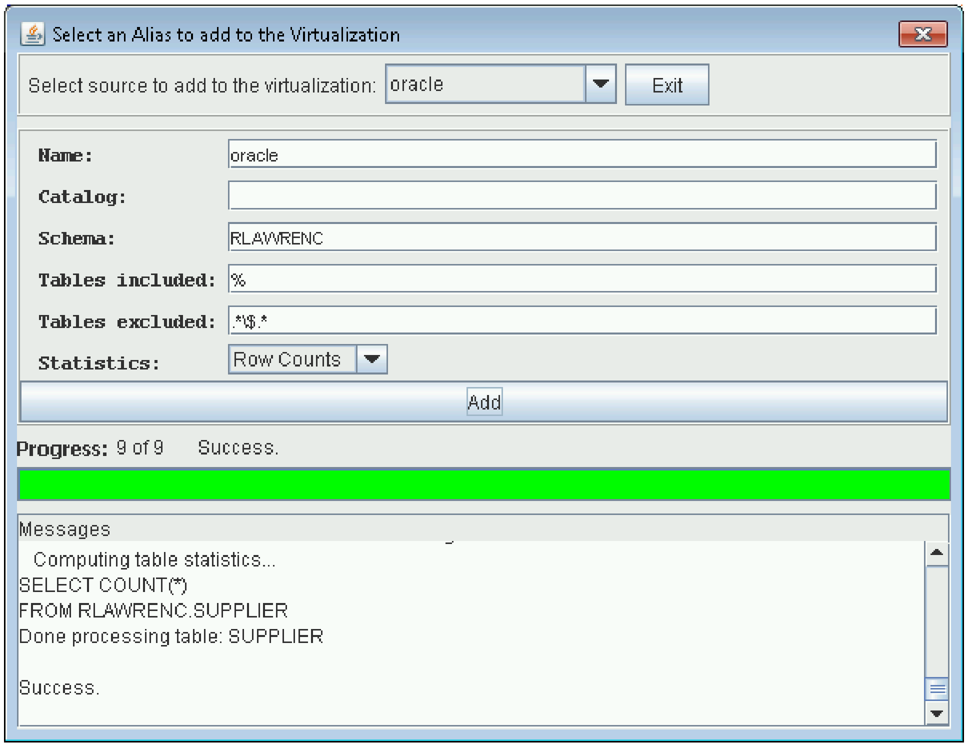

SQuirreL SQL Client Snapshot-20120510_0950
File Drivers Aliases Plugins Session Windows Help
Connect to: cssql_unity
Active Session: 1 - virtual
1 - virtual
Objects SQL
Aliases
Drivers
virtual
MSSQL
TABLE
CUSTOMER (150(
LINEITEM (60057:
NATION (25)
ORDERS (15000(
PART (20000)
PARTSUPP (8000
REGION (5)
SUPPLIER (1000)
UDT
Metadata Status Schemas Table Types Data Types Numeric Functions String Functions System Functions Time/Date Functions Keywords
/virtual
1,1 / 1
Logs: Errors 3, Warnings 0, Infos 52
41 of 121 MB
0
2:30:46 PM PDT

Property Name	Value
JDBC Driver CLASSNAME	unity.jdbc.UnityDriver
JDBC Driver CLASSPATH	C:\eclipse\workspace\UnityJDBC_Live\unityjdbc.jar
getURL	<null>
isReadOnly	false
getDatabaseProductName	UnityJDBC
getUserName	
getDriverVersion	4.1.109
getDatabaseProductVersion	4.1.109
getDriverName	UnityJDBC
allProceduresAreCallable	false
allTablesAreSelectable	true
autoCommitFailureClosesAllResultSets	false
dataDefinitionCausesTransactionCommit	false
dataDefinitionIgnoredInTransactions	false
doesMaxRowSizeIncludeBlobs	false
getCatalogSeparator	.
getCatalogTerm	catalog
getClientInfoProperties	
getDatabaseMajorVersion	4
getDatabaseMinorVersion	1
getDefaultTransactionIsolation	TRANSACTION_NONE
getDriverMajorVersion	4
getDriverMinorVersion	1
getExtraNameCharacters	
getIdentifierQuoteString	"

Please try out the Tools popup by hitting ctrl+t in the SQL Editor. Do it three times to stop this message.

Resulting Data Virtualization

5. **Add Oracle Database to Data Virtualization** - User can add as many sources as they wish. You can also rename the source in the virtual view. It does not have to be the same as the alias name used by SQuirreL. When adding Oracle sources, make sure to specify a schema so that system tables and tables from all schemas are not extracted.



Select source to add to the virtualization: oracle

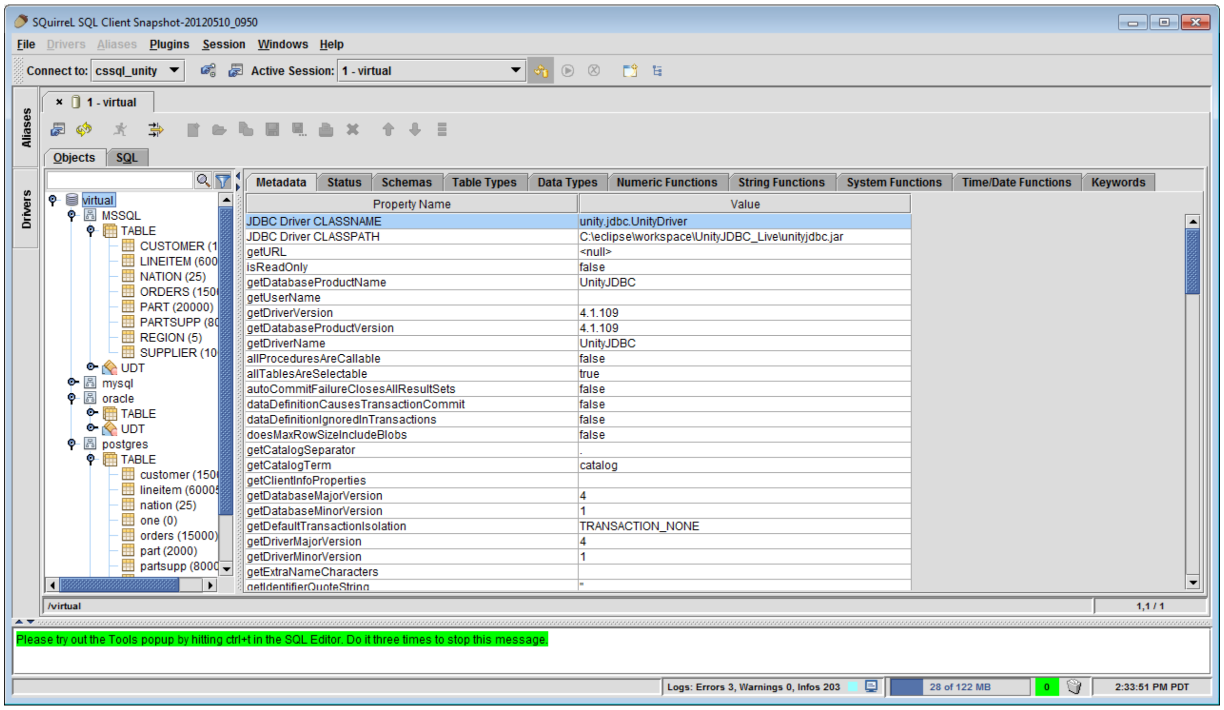
Name:	oracle
Catalog:	
Schema:	RLAWRENC
Tables included:	%
Tables excluded:	.*\$.*
Statistics:	Row Counts

Progress: 9 of 9 Success.

Messages

```
  Computing table statistics...
SELECT COUNT(*)
FROM RLAWRENC.SUPPLIER
Done processing table: SUPPLIER

Success.
```

Prompt to add Oracle Source to the Data Virtualization (Note use of SCHEMA).



Connect to: cssql_unity   Active Session: 1 - virtual

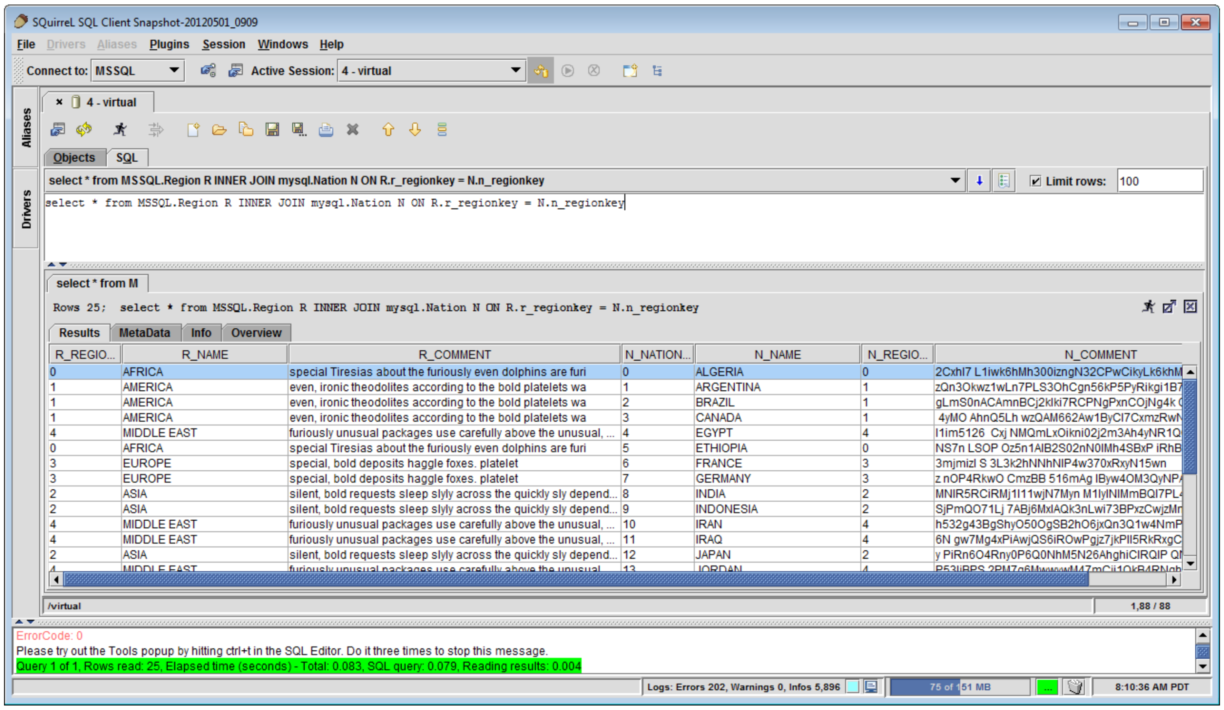
```
virtual
  MSSQL
    TABLE
      CUSTOMER (1
      LINEITEM (600
      NATION (25)
      ORDERS (150
      PART (20000)
      PARTSUPP (80
      REGION (5)
      SUPPLIER (10
    UDT
  mysql
  oracle
    TABLE
    UDT
  postgres
    TABLE
      customer (150
      lineitem (6000
      nation (25)
      one (0)
      orders (15000)
      part (2000)
      partsupp (8000
```

Property Name	Value
JDBC Driver CLASSNAME	unity.jdbc.UnityDriver
JDBC Driver CLASSPATH	C:\eclipse\workspace\UnityJDBC_Live\unityjdbc.jar
getURL	<null>
isReadOnly	false
getDatabaseProductName	UnityJDBC
getUserName	
getDriverVersion	4.1.109
getDatabaseProductVersion	4.1.109
getDriverName	UnityJDBC
allProceduresAreCallable	false
allTablesAreSelectable	true
autoCommitFailureClosesAllResultSets	false
dataDefinitionCausesTransactionCommit	false
dataDefinitionIgnoredInTransactions	false
doesMaxRowSizeIncludeBlobs	false
getCatalogSeparator	.
getCatalogTerm	catalog
getClientInfoProperties	
getDatabaseMajorVersion	4
getDatabaseMinorVersion	1
getDefaultTransactionIsolation	TRANSACTION_NONE
getDriverMajorVersion	4
getDriverMinorVersion	1
getExtraNameCharacters	
getIdentifierQuoteString	"

```
/virtual                                                   1,1 / 1
Please try out the Tools popup by hitting ctrl+t in the SQL Editor. Do it three times to stop this message.
Logs: Errors 3, Warnings 0, Infos 203    28 of 122 MB    0    2:33:51 PM PDT
```

A Data Virtualization in SQuirreL SQL with Databases MySQL, Oracle, PostgreSQL, and Microsoft SQL Server

6. **Execute a Multiple Database Query** - The user can execute an SQL query that spans multiple sources and get a single result. The virtualization is transparent to the user and SQuirreL. Below is an example of a query that joins two tables in different databases.



```
select * from MSSQL.Region R INNER JOIN mysql.Nation N ON R.r_regionkey = N.n_regionkey
```

Rows 25; select * from MSSQL.Region R INNER JOIN mysql.Nation N ON R.r_regionkey = N.n_regionkey

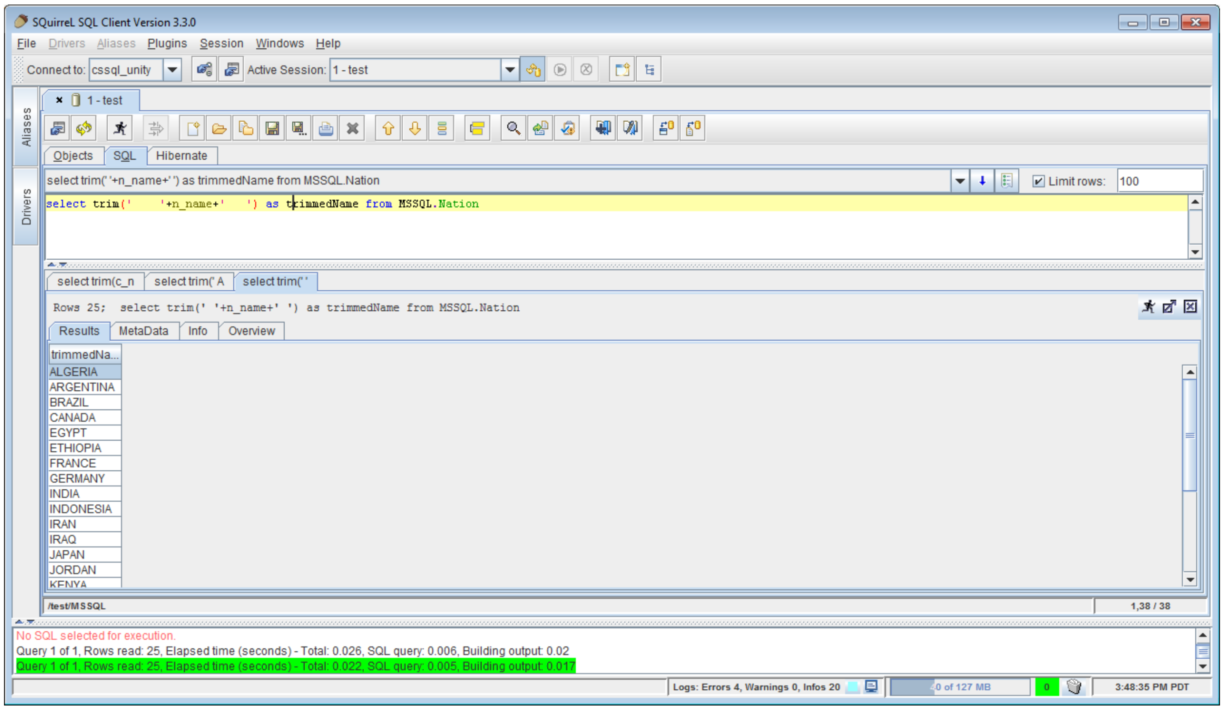
R_REGIO...	R_NAME	R_COMMENT	N_NATION...	N_NAME	N_REGIO...	N_COMMENT
0	AFRICA	special Tiresias about the furiously even dolphins are furi	0	ALGERIA	0	2Cxhl7 L1iwk6hMh300izngN32CPwCikyLk6khM
1	AMERICA	even, ironic theodolites according to the bold platelets wa	1	ARGENTINA	1	zQn3Okwz1wLn7PLS3OhCgn56kP5PyRikgi1B7
1	AMERICA	even, ironic theodolites according to the bold platelets wa	2	BRAZIL	1	gLmS0nACAmnBCj2klki7RCPNgPxnCOjNg4k
1	AMERICA	even, ironic theodolites according to the bold platelets wa	3	CANADA	1	4yMO AhnQ5Lh wzQAM662Aw1ByCl7CxmzRwN
4	MIDDLE EAST	furiously unusual packages use carefully above the unusual, ...	4	EGYPT	4	l1im5126 Cxj NMQmLxOikni02j2m3Ah4yNR1Q
0	AFRICA	special Tiresias about the furiously even dolphins are furi	5	ETHIOPIA	0	NS7n LSOP Oz5n1AlB2S02nN0lMh4SBxP iRhB
3	EUROPE	special, bold deposits haggle foxes. platelet	6	FRANCE	3	3mjmizl S 3L3k2hNNhNIP4w370xRxyN15wn
3	EUROPE	special, bold deposits haggle foxes. platelet	7	GERMANY	3	z nOP4RkwO CmzBB 516mAg lByw4OM3QyNPA
2	ASIA	silent, bold requests sleep slyly across the quickly sly depend...	8	INDIA	2	MNIR5RCiRMj1l11wjN7Myn M1lylNIMmBQI7PL4
2	ASIA	silent, bold requests sleep slyly across the quickly sly depend...	9	INDONESIA	2	SjPmQO71Lj 7ABj6MxlAQk3nLwi73BPxzCwjzMn
4	MIDDLE EAST	furiously unusual packages use carefully above the unusual, ...	10	IRAN	4	h532g43BgShyO50OgSB2hO6jxQn3Q1w4NmP
4	MIDDLE EAST	furiously unusual packages use carefully above the unusual, ...	11	IRAQ	4	6N gw7Mg4xPiAwjQS6iROwPgjz7jkPll5RkRxgC
2	ASIA	silent, bold requests sleep slyly across the quickly sly depend...	12	JAPAN	2	y PiRn6O4Rny0P6Q0NhM5N26AhghiClRQIP QI
4	MIDDLE EAST	furiously unusual packages use carefully above the unusual	13	JORDAN	4	P53liBPS 2PM7g6MwwwwM47mCij1OkB4RNgh

/virtual 1,88 / 88

```
ErrorCode: 0
Please try out the Tools popup by hitting ctrl+t in the SQL Editor. Do it three times to stop this message.
Query 1 of 1, Rows read: 25, Elapsed time (seconds) - Total: 0.083, SQL query: 0.079, Reading results: 0.004
```

A Multiple Database Query with Join Results Expressed on Previous Data Virtualization

7. **Perform SQL Query Translation** - The UnityJDBC driver used to perform the virtualization will also translate functions that are not implemented by certain sources. For example, Microsoft SQL Server does not support `TRIM()`, but you can do the same result using `RTRIM(LTRIM())`. Unity will automatically translate a `TRIM()` function specified in a MSSQL query to the correct syntax supported by the database.



```
select trim('     '+n_name+'    ') as trimmedName from MSSQL.Nation
```

```
Rows 25;  select trim(' '+n_name+' ') as trimmedName from MSSQL.Nation
```

trimmedNa...
ALGERIA
ARGENTINA
BRAZIL
CANADA
EGYPT
ETHIOPIA
FRANCE
GERMANY
INDIA
INDONESIA
IRAN
IRAQ
JAPAN
JORDAN
KENYA

```
No SQL selected for execution.
Query 1 of 1, Rows read: 25, Elapsed time (seconds) - Total: 0.026, SQL query: 0.006, Building output: 0.02
Query 1 of 1, Rows read: 25, Elapsed time (seconds) - Total: 0.022, SQL query: 0.005, Building output: 0.017
```

An example of SQL Query and Dialect Translation - Converting TRIM() function for Microsoft SQL Server

The plugin source code, like all of SQuirreL, is released under the GNU Lesser General Public License. The UnityJDBC virtualization driver is released under a commercial license. However, the UnityJDBC driver included in the plugin is fully functioning with no time limits allowing an unlimited number of sources and queries. The only limitation is the size of the result set is limited to the first 100 rows. (Note there is no limit on the number of rows extracted from each source. So select count(*) from table with a 1 million row table is fine as it only returns one result row.) Use LIMIT 100 to get the first 100 results of a query. A full version of the UnityJDBC driver can be purchased as www.unityjdbc.com [http://www.unityjdbc.com].

UnityJDBC Data Virtualization with JasperReports

The JasperReports library and community version of JasperReports server does not support data virtualization allowing one SQL query to extract data from multiple databases. UnityJDBC can be used with JasperReports to enable this data virtualization which simplifies the construction of reports, especially reports that would usually use subreports.

UnityJDBC Data Virtualization with Aqua Data Studio

Aqua Data Studio can query multiple databases with different SQL queries. However, you cannot query more than one database at the same time in one SQL query. UnityJDBC allows a user to write one query to join, aggregate, and summarize data across any number of databases. It also performs translation of SQL dialects and functions.

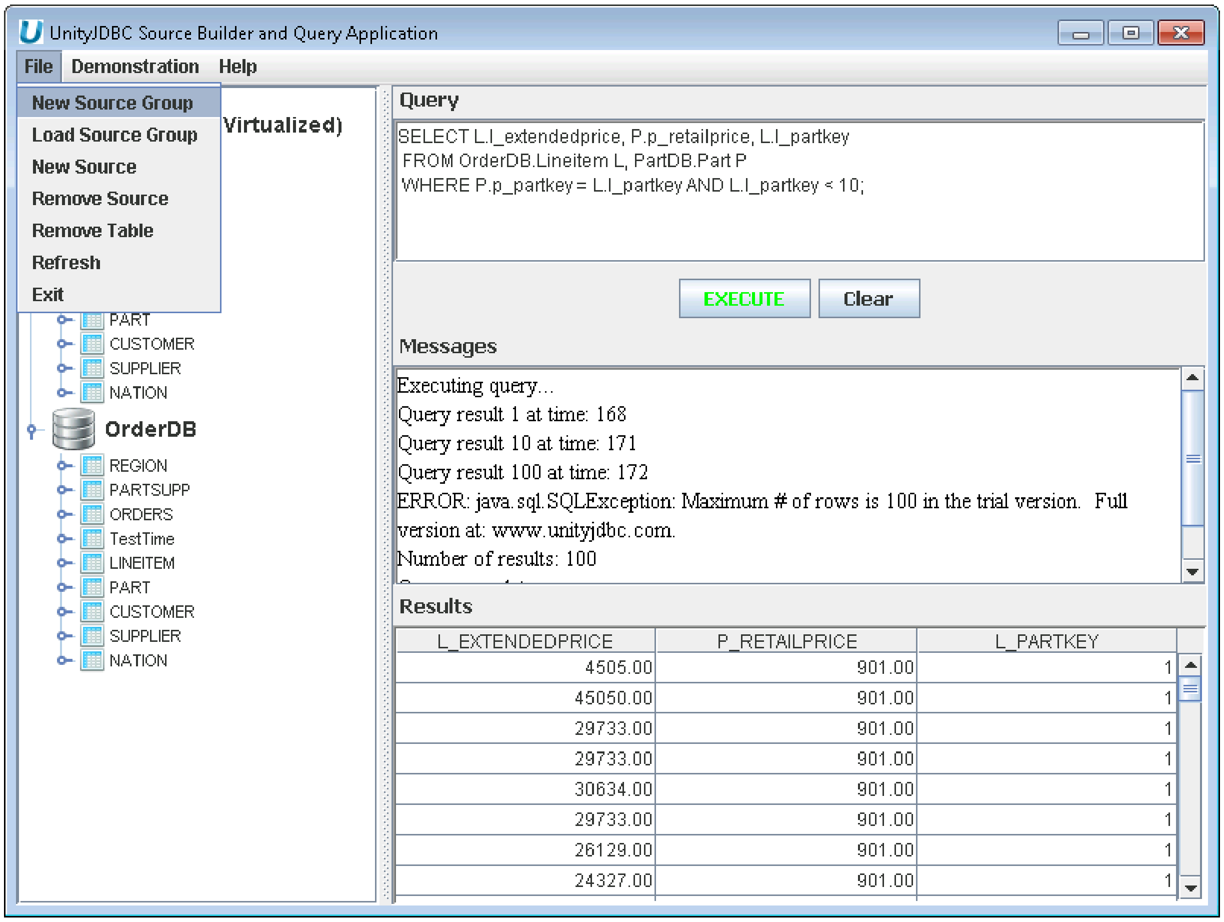
Benefits:

- UnityJDBC allows Aqua Data Studio to support multiple source queries.

- No data source or server changes are required.
- UnityJDBC supports standard SQL including joins, group by, aggregation, LIMIT, and ordering where tables may come from one or more sources.
- UnityJDBC will perform function translation where a user requests a function that is not supported on a certain source.

This example assumes that the installation of UnityJDBC has already been completed. The following is an example of creating sources and executing multiple database queries with Aqua Data Studio.

1. **Create a New Source Group** - Under File Menu, Select New Source Group.



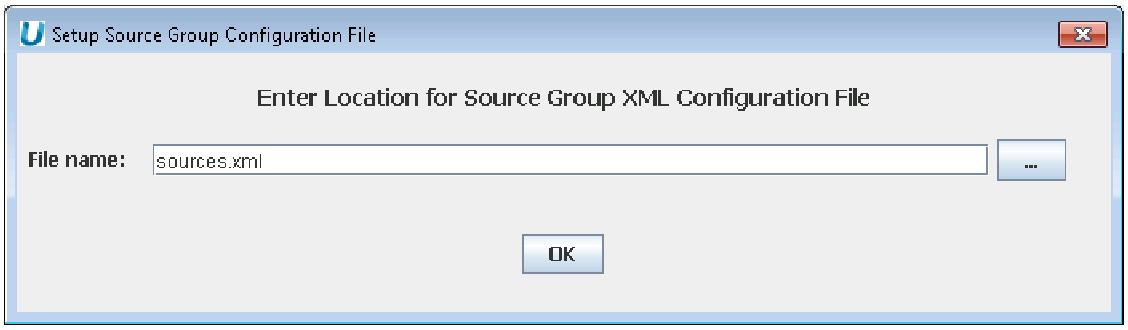
```
SELECT L.l_extendedprice, P.p_retailprice, L.l_partkey
 FROM OrderDB.Lineitem L, PartDB.Part P
 WHERE P.p_partkey = L.l_partkey AND L.l_partkey < 10;
```

```
Executing query...
Query result 1 at time: 168
Query result 10 at time: 171
Query result 100 at time: 172
ERROR: java.sql.SQLException: Maximum # of rows is 100 in the trial version.  Full
version at: www.unityjdbc.com.
Number of results: 100
```

L_EXTENDEDPRICE	P_RETAILPRICE	L_PARTKEY
4505.00	901.00	1
45050.00	901.00	1
29733.00	901.00	1
29733.00	901.00	1
30634.00	901.00	1
29733.00	901.00	1
26129.00	901.00	1
24327.00	901.00	1

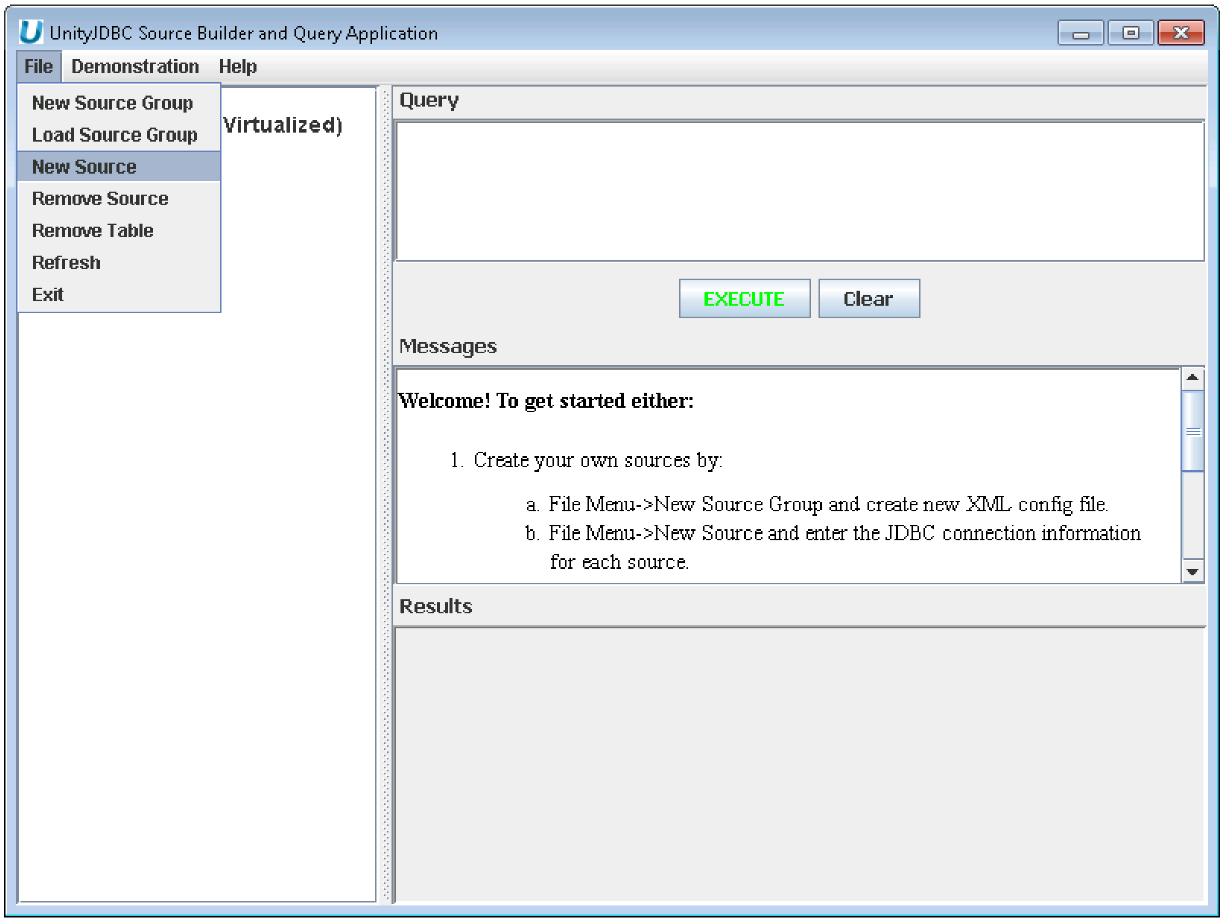
Aqua Example: Creating a New Source Group Data Virtualization

2. **Select Sources File Name and Location** - Use the default sources.xml or select a file location.


Setup Source Group Configuration File
Enter Location for Source Group XML Configuration File
File name:
sources.xml
...
OK

Aqua Example: Specifying a Source File Location

3. **Select New Source** - Under File Menu, Select New Source.

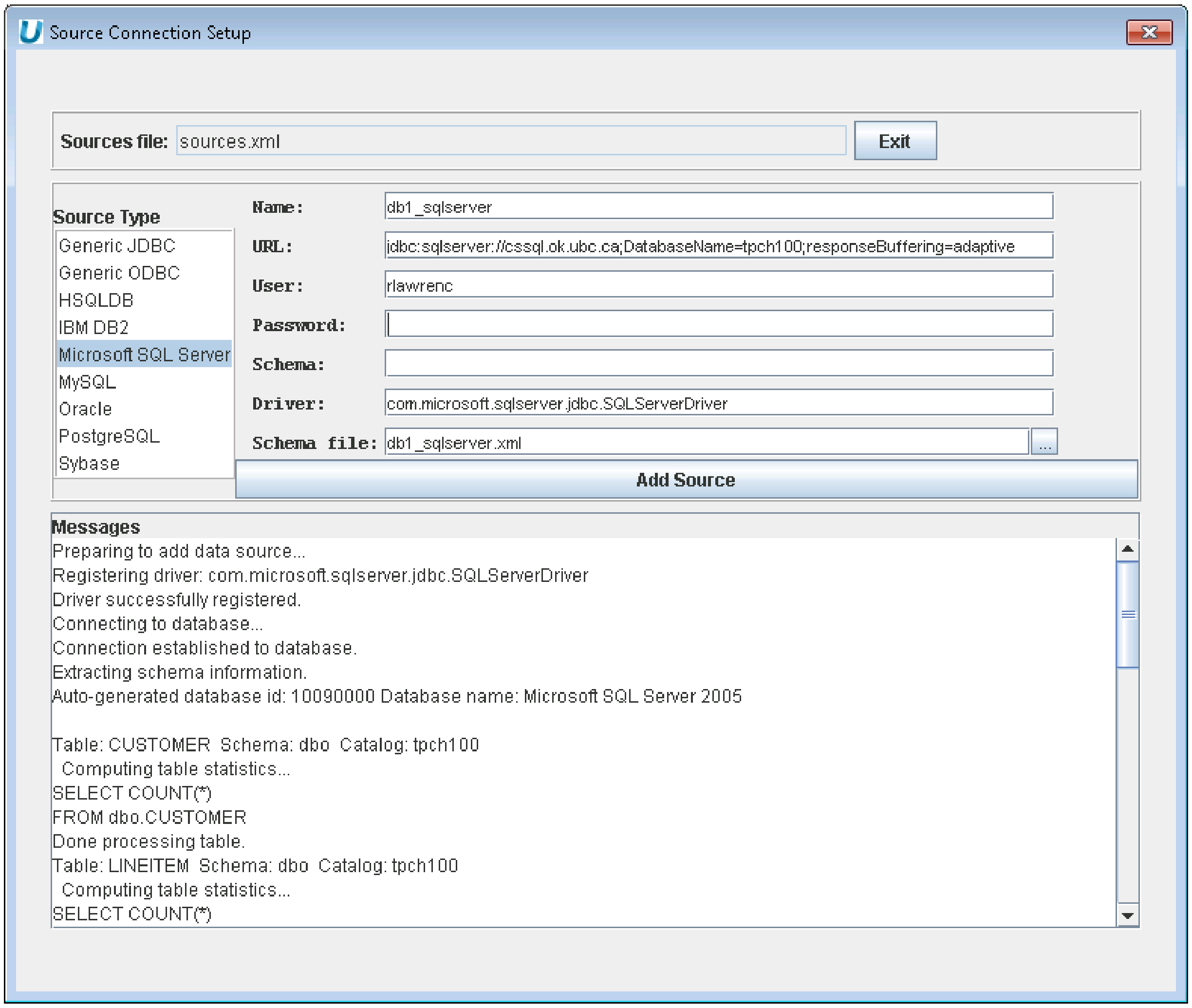


Welcome! To get started either:

1. Create your own sources by:
    a. File Menu->New Source Group and create new XML config file.
    b. File Menu->New Source and enter the JDBC connection information for each source.

Aqua Example: Creating a New Source

4. **Add a Microsoft Database** - Enter connection information for Microsoft SQL Server to add it to the data virtualization.



Sources file: sources.xml

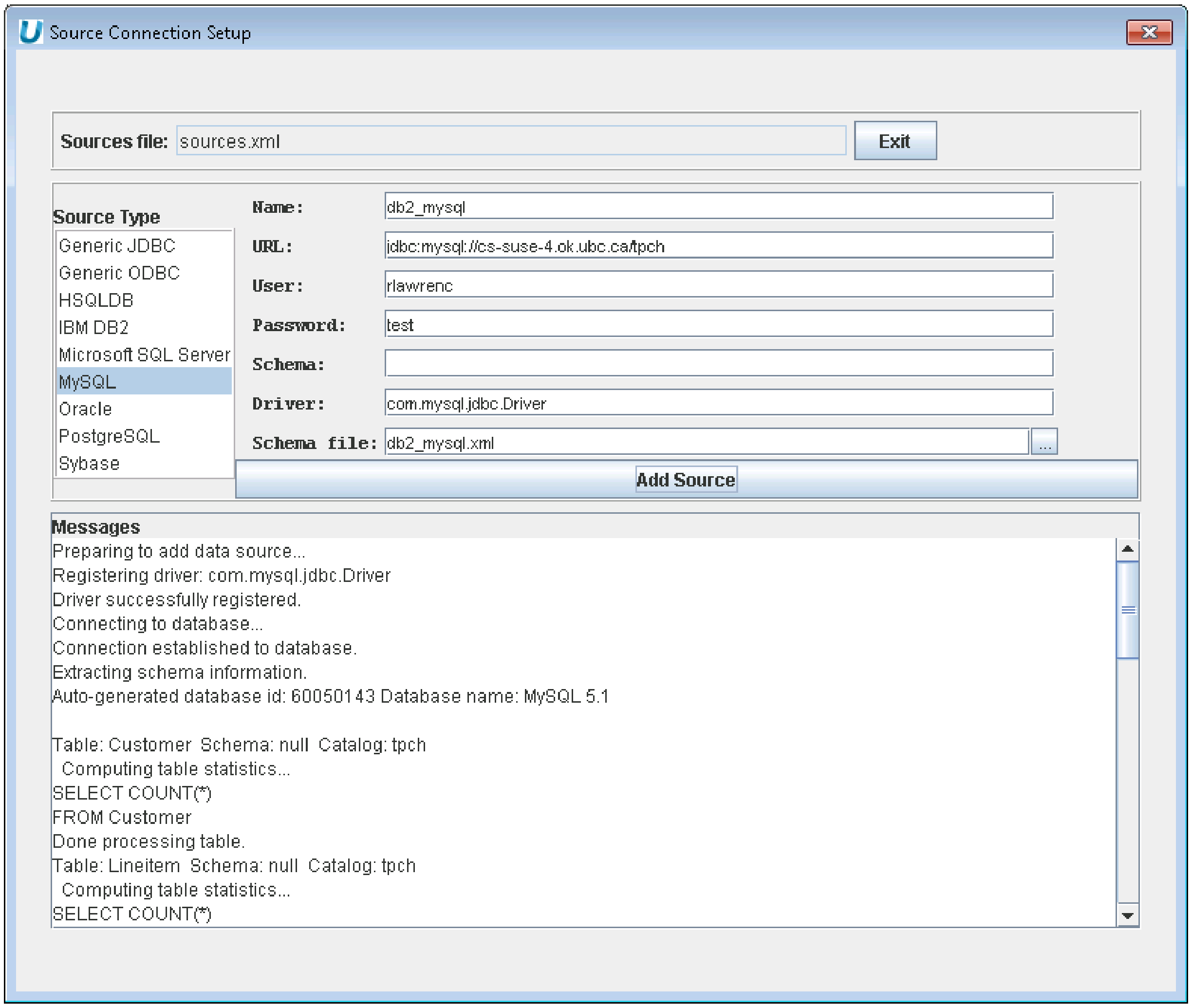
Name: db1_sqlserver
URL: jdbc:sqlserver://cssql.ok.ubc.ca;DatabaseName=tpch100;responseBuffering=adaptive
User: rlawrenc
Password:
Schema:
Driver: com.microsoft.sqlserver.jdbc.SQLServerDriver
Schema file: db1_sqlserver.xml

Messages

```
Preparing to add data source...
Registering driver: com.microsoft.sqlserver.jdbc.SQLServerDriver
Driver successfully registered.
Connecting to database...
Connection established to database.
Extracting schema information.
Auto-generated database id: 10090000 Database name: Microsoft SQL Server 2005

Table: CUSTOMER  Schema: dbo  Catalog: tpch100
  Computing table statistics...
SELECT COUNT(*)
FROM dbo.CUSTOMER
Done processing table.
Table: LINEITEM  Schema: dbo  Catalog: tpch100
  Computing table statistics...
SELECT COUNT(*)
```

Aqua Example: Adding a Microsoft SQL Server Database to the Data Virtualization

5. **Add a MySQL Database** - Enter connection information for MySQL database to add it to the data virtualization.



Sources file: sources.xml

Source Type: Generic JDBC, Generic ODBC, HSQLDB, IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase

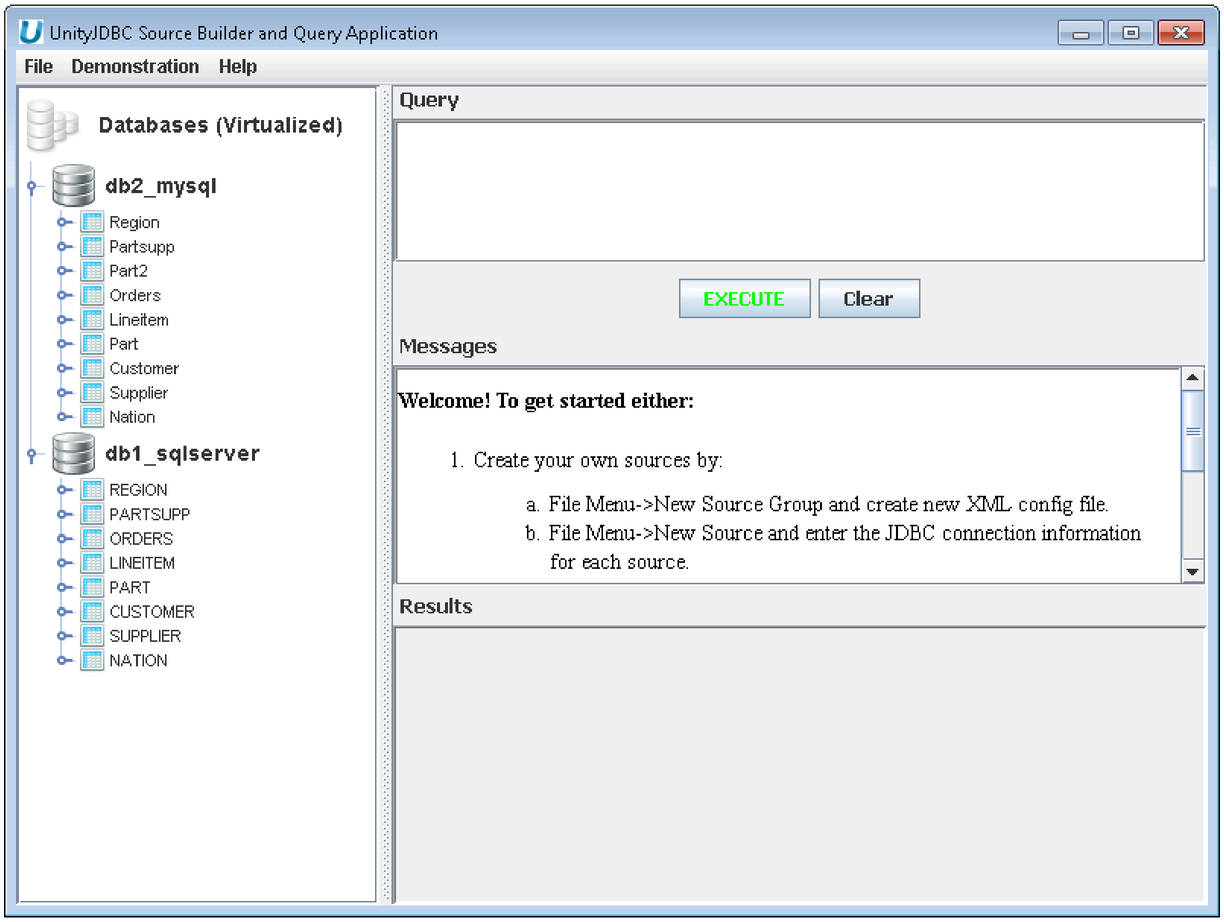
Name: db2_mysql
URL: jdbc:mysql://cs-suse-4.ok.ubc.ca/tpch
User: rlawrenc
Password: test
Schema:
Driver: com.mysql.jdbc.Driver
Schema file: db2_mysql.xml

Messages

```
Preparing to add data source...
Registering driver: com.mysql.jdbc.Driver
Driver successfully registered.
Connecting to database...
Connection established to database.
Extracting schema information.
Auto-generated database id: 60050143 Database name: MySQL 5.1

Table: Customer  Schema: null  Catalog: tpch
  Computing table statistics...
SELECT COUNT(*)
FROM Customer
Done processing table.
Table: Lineitem  Schema: null  Catalog: tpch
  Computing table statistics...
SELECT COUNT(*)
```

Aqua Example: Adding a MySQL Database to the Data Virtualization

6. **Viewing Data Virtualization of Sources** - Select Exit to return to the main screen and see data virtualization of MySQL and Microsoft SQL Server sources.

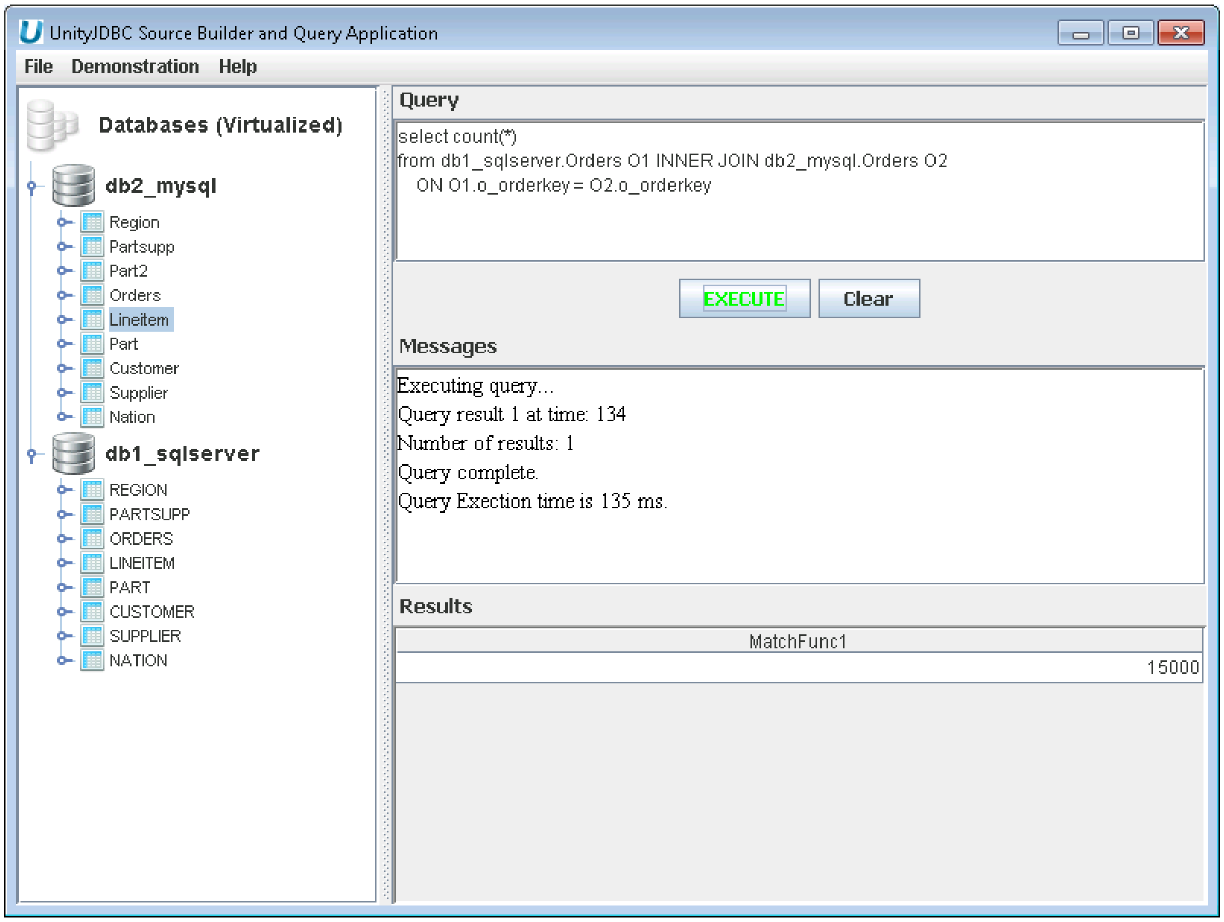


Welcome! To get started either:

1. Create your own sources by:
   a. File Menu->New Source Group and create new XML config file.
   b. File Menu->New Source and enter the JDBC connection information for each source.

Aqua Example: Resulting Data Virtualization of Two Database Sources

7. **Multiple Database Query and Cross-Database Join Example** - Type in a cross-database query and execute it to view results.



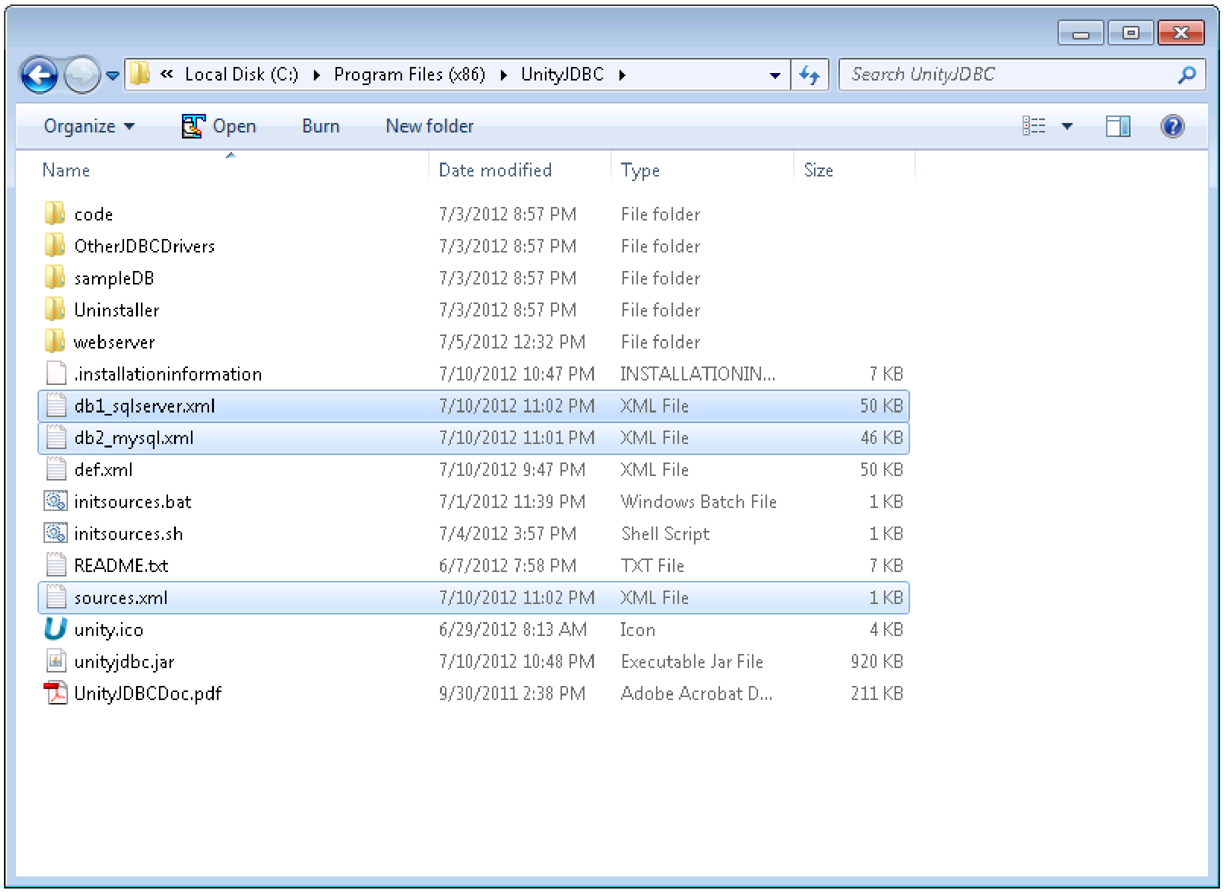
```
select count(*)
from db1_sqlserver.Orders O1 INNER JOIN db2_mysql.Orders O2
   ON O1.o_orderkey = O2.o_orderkey
```

```
Executing query...
Query result 1 at time: 134
Number of results: 1
Query complete.
Query Exection time is 135 ms.
```

MatchFunc1
15000

Aqua Example: Executing a Cross-Database Join of Two Databases

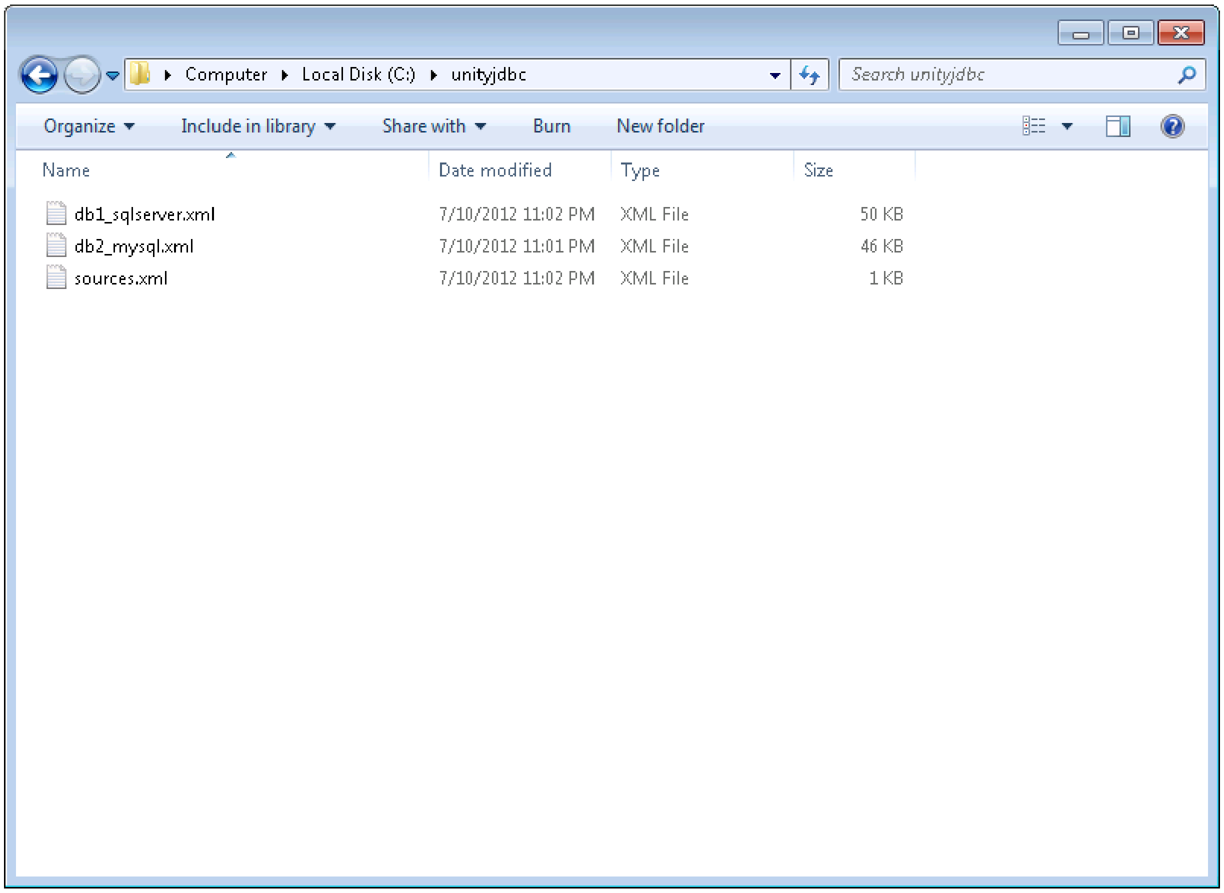
8. **Finding Sources and Schema Files** - Now that the data virtualization is complete, find the sources.xml file and schema files for your sources. The image below shows the default location which would be inside the UnityJDBC installation directory.



Name	Date modified	Type	Size
code	7/3/2012 8:57 PM	File folder	
OtherJDBCDrivers	7/3/2012 8:57 PM	File folder	
sampleDB	7/3/2012 8:57 PM	File folder	
Uninstaller	7/3/2012 8:57 PM	File folder	
webserver	7/5/2012 12:32 PM	File folder	
.installationinformation	7/10/2012 10:47 PM	INSTALLATIONIN...	7 KB
db1_sqlserver.xml	7/10/2012 11:02 PM	XML File	50 KB
db2_mysql.xml	7/10/2012 11:01 PM	XML File	46 KB
def.xml	7/10/2012 9:47 PM	XML File	50 KB
initsources.bat	7/1/2012 11:39 PM	Windows Batch File	1 KB
initsources.sh	7/4/2012 3:57 PM	Shell Script	1 KB
README.txt	6/7/2012 7:58 PM	TXT File	7 KB
sources.xml	7/10/2012 11:02 PM	XML File	1 KB
unity.ico	6/29/2012 8:13 AM	Icon	4 KB
unityjdbc.jar	7/10/2012 10:48 PM	Executable Jar File	920 KB
UnityJDBCDoc.pdf	9/30/2011 2:38 PM	Adobe Acrobat D...	211 KB

Aqua Example: Finding Source and Schema Files

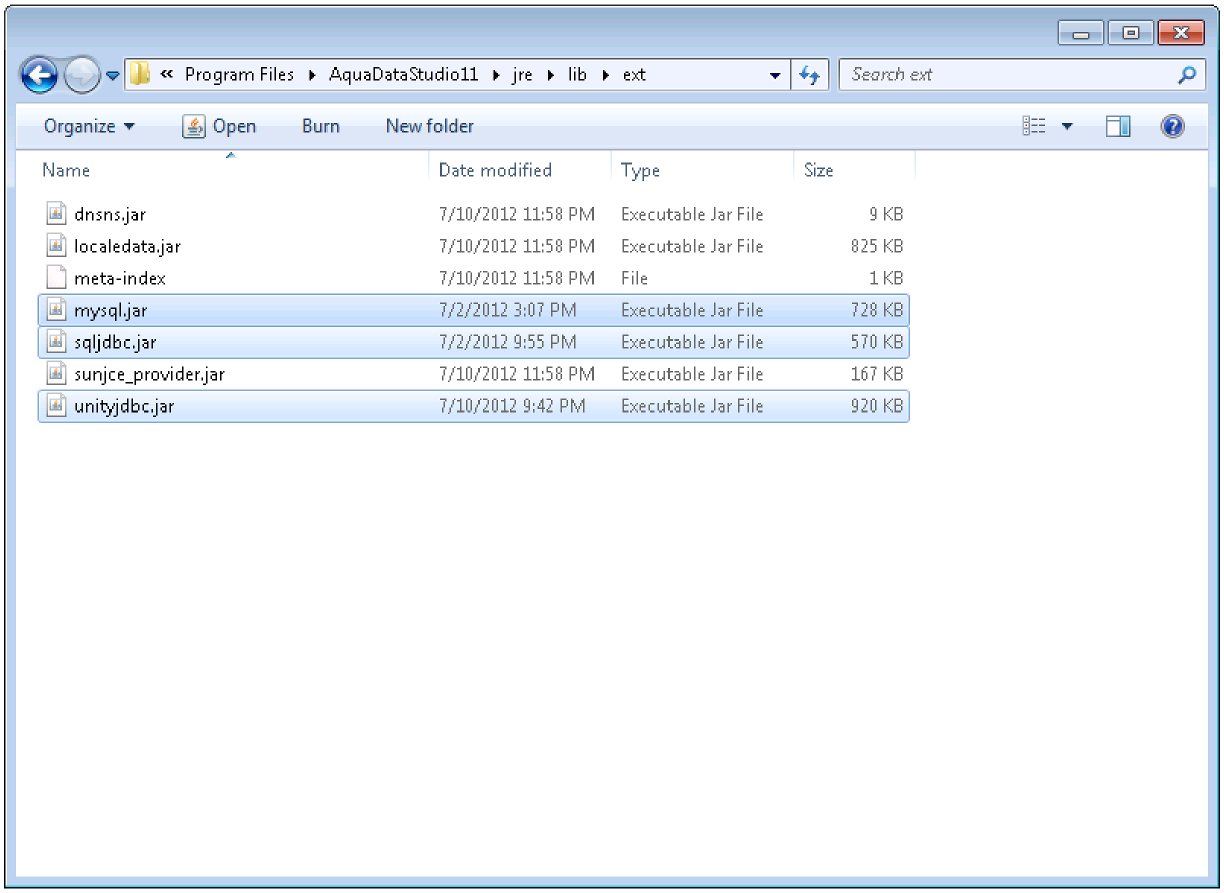
9. **Moving Sources and Schema Files** - Optionally, move the sources and schema files to a permanent location. In this example, they are moved to `C:/unityjdbc`.



Name	Date modified	Type	Size
db1_sqlserver.xml	7/10/2012 11:02 PM	XML File	50 KB
db2_mysql.xml	7/10/2012 11:01 PM	XML File	46 KB
sources.xml	7/10/2012 11:02 PM	XML File	1 KB

Aqua Example: Moving Source and Schema Files to Another Location

10.**Adding UnityJDBC data source to Aqua Data Studio** - Copy `unityjdbc.jar` and all JDBC drivers for databases into `jre/lib/ext` in Aqua Installation directory.

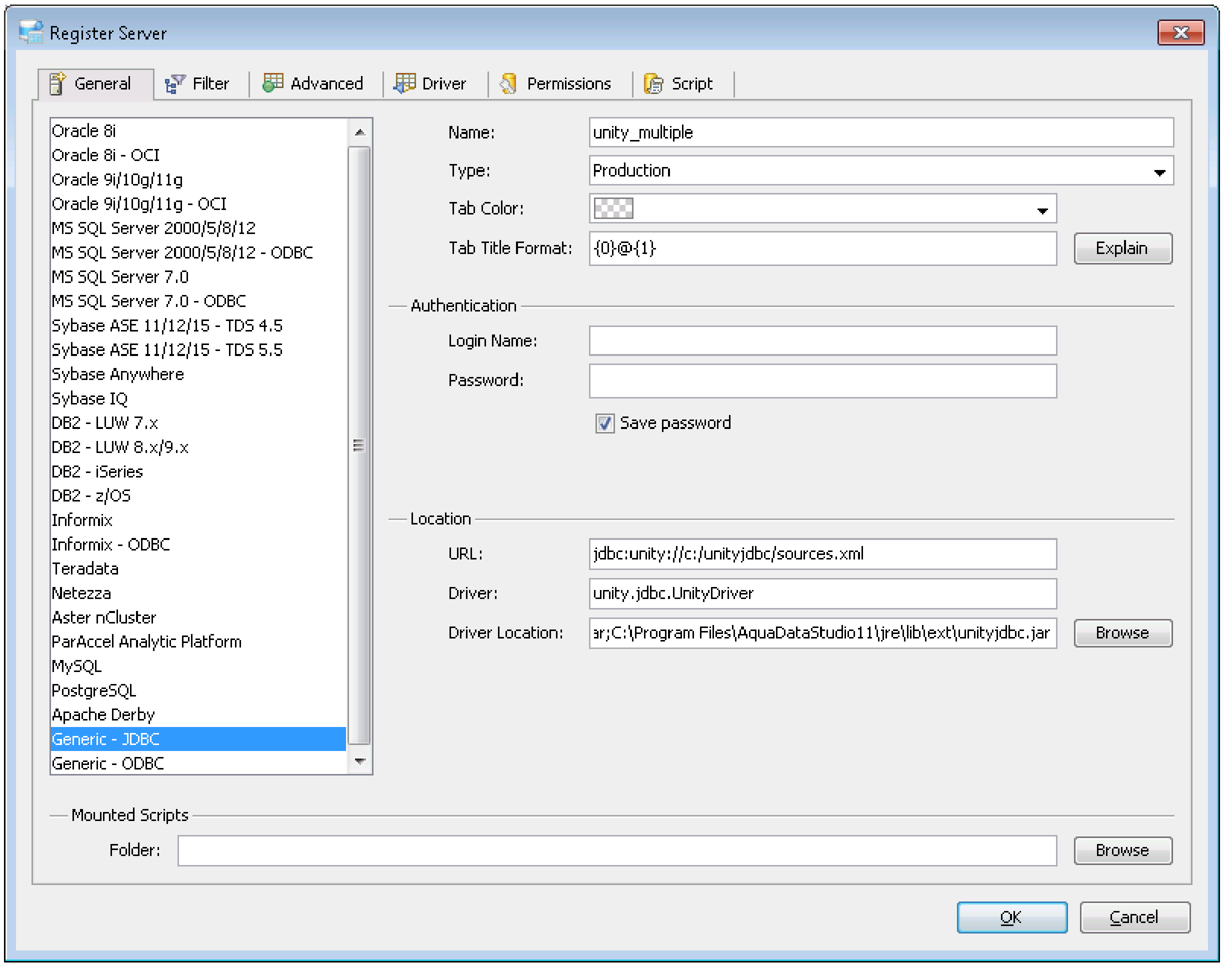


Name	Date modified	Type	Size
dnsns.jar	7/10/2012 11:58 PM	Executable Jar File	9 KB
localedata.jar	7/10/2012 11:58 PM	Executable Jar File	825 KB
meta-index	7/10/2012 11:58 PM	File	1 KB
mysql.jar	7/2/2012 3:07 PM	Executable Jar File	728 KB
sqljdbc.jar	7/2/2012 9:55 PM	Executable Jar File	570 KB
sunjce_provider.jar	7/10/2012 11:58 PM	Executable Jar File	167 KB
unityjdbc.jar	7/10/2012 9:42 PM	Executable Jar File	920 KB

Aqua Example: Installing unityjdbc.jar in Aqua Data Studio

11. **Start Aqua and Register a Generic JDBC Source** - The configuration information is:

- Driver: `unity.jdbc.UnityDriver`
- URL: `jdbc:unity://<path to sources.xml file>`



Name: unity_multiple
Type: Production
Tab Color:
Tab Title Format: {0}@{1}

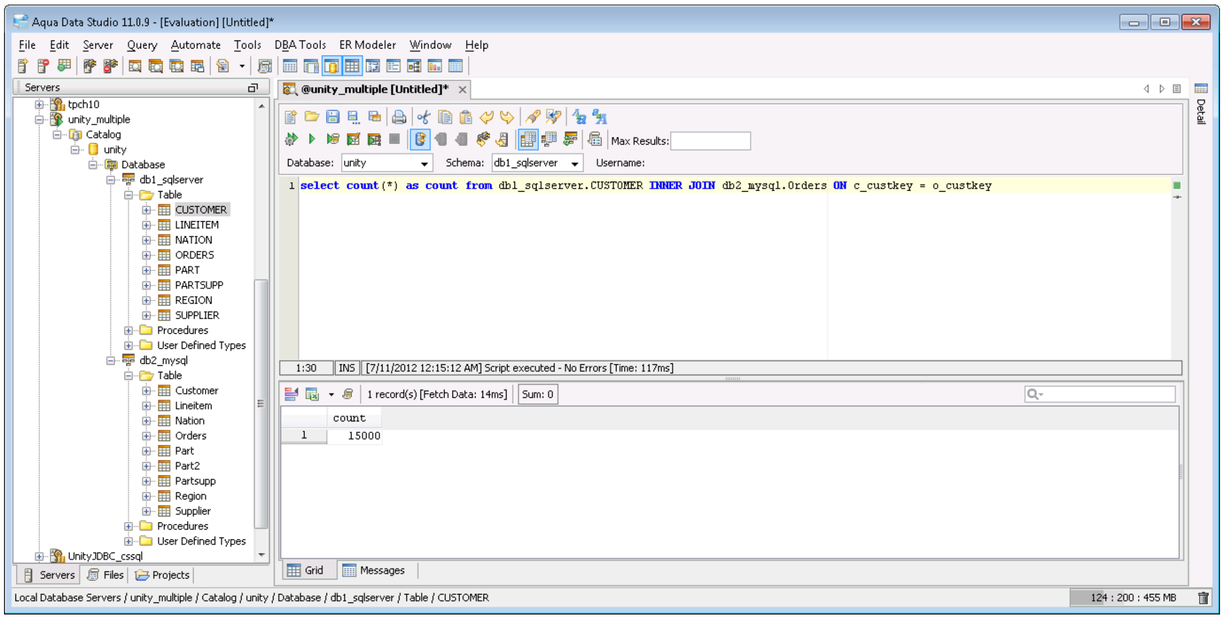
Authentication
Login Name:
Password:
[✓] Save password

Location
URL: jdbc:unity://c:/unityjdbc/sources.xml
Driver: unity.jdbc.UnityDriver
Driver Location: ar;C:\Program Files\AquaDataStudio11\jre\lib\ext\unityjdbc.jar

Mounted Scripts
Folder:

Aqua Example: Adding a UnityJDBC Virtual Data Source in Aqua Data Studio

12.**Multiple Database Query in Aqua Data Studio** - Create a query like usual except it can contain multiple databases.



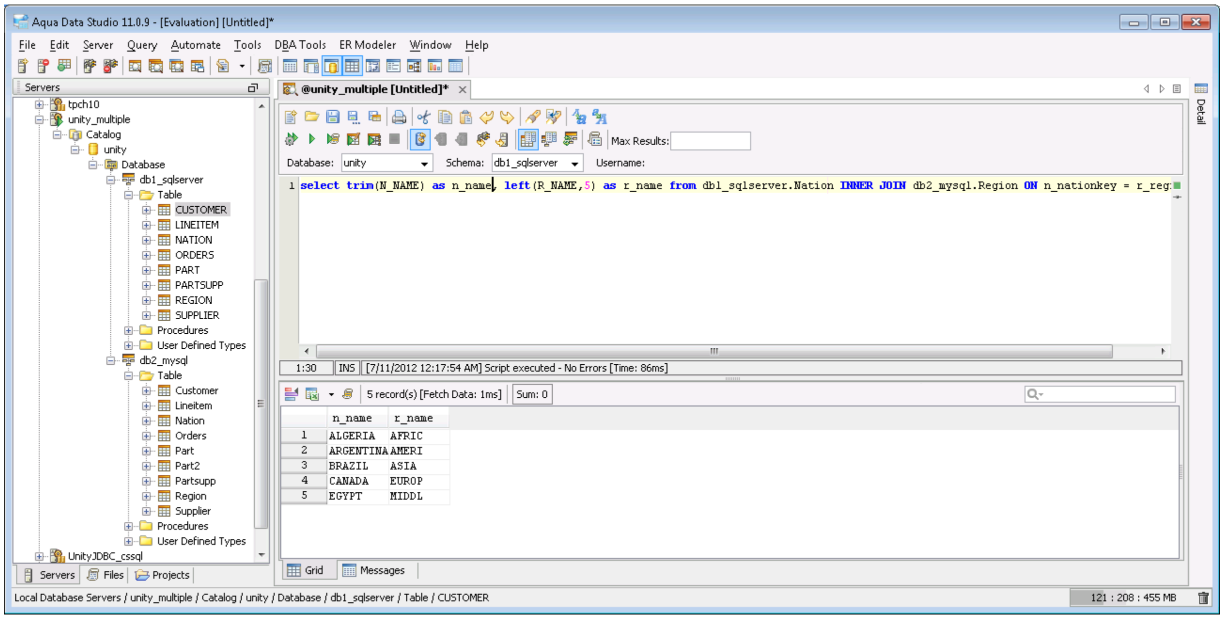
```
select count(*) as count from db1_sqlserver.CUSTOMER INNER JOIN db2_mysql.Orders ON c_custkey = o_custkey
```

```
1:30 | INS | [7/11/2012 12:15:12 AM] Script executed - No Errors [Time: 117ms]
```

```
1 record(s) [Fetch Data: 14ms] | Sum: 0

     count
1    15000
```

Aqua Example: Executing a Multiple Database Join Query and Displaying Results

13.**Function and SQL Dialect Translation with Aqua Data Studio** - Forget what functions you can use on each database? No problem – UnityJDBC will translate automatically. This translation is supported for common databases and can be freely extended by user-defined functions and translations for each database dialect.



```
select trim(N_NAME) as n_name, left(R_NAME,5) as r_name from db1_sqlserver.Nation INNER JOIN db2_mysql.Region ON n_nationkey = r_reg
```

```
1:30 | INS | [7/11/2012 12:17:54 AM] Script executed - No Errors [Time: 86ms]
```

```
5 record(s) [Fetch Data: 1ms] | Sum: 0

   n_name    r_name
1  ALGERIA   AFRIC
2  ARGENTINA AMERI
3  BRAZIL    ASIA
4  CANADA    EUROP
5  EGYPT     MIDDL
```

Aqua Example: Performing SQL Dialect and Function Translation

UnityJDBC Data Virtualization with RazorSQL

UnityJDBC allows RazorSQL users to create a virtual data source that may consist of multiple data sources on different servers and platforms. The user can enter one SQL query to combine and join information from multiple sources.

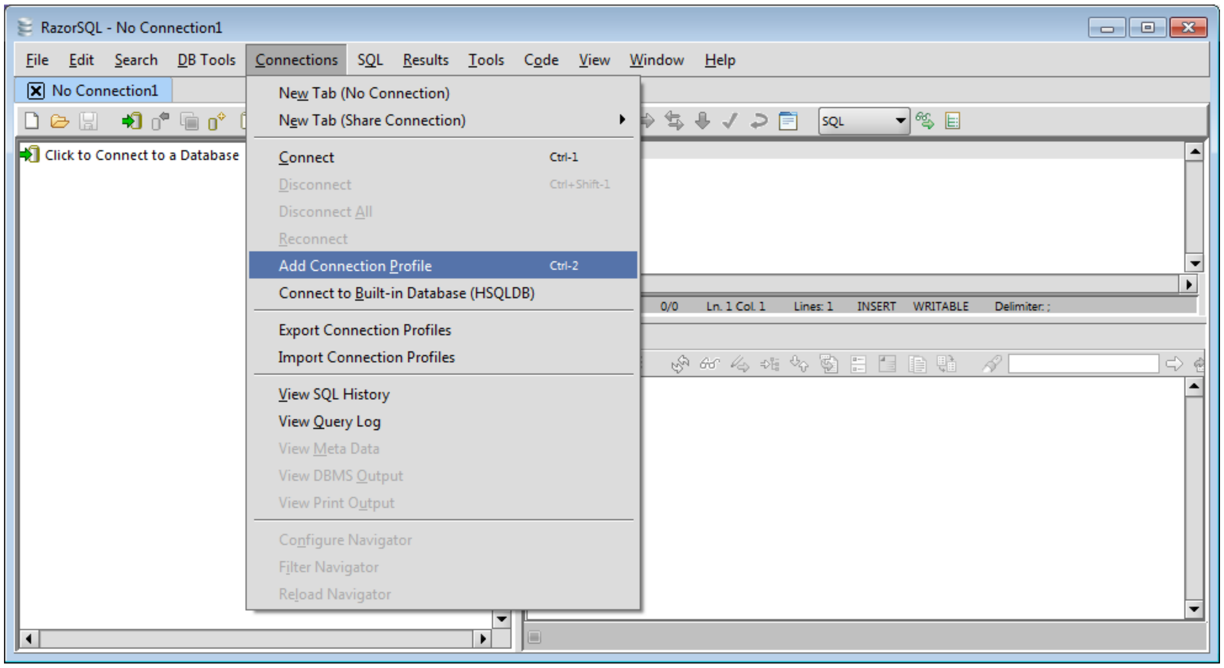
Benefits:

- UnityJDBC allows RazorSQL to support multiple database queries.
- No data source or server changes are required.
- UnityJDBC will perform function translation where a user requests a function that is not supported on a certain source.

This example assumes that the installation of UnityJDBC has already been completed, and that the user has already created a data virtualization using the UnityJDBC SourceBuilder. The following is an example of creating sources and executing multiple database queries with RazorSQL.

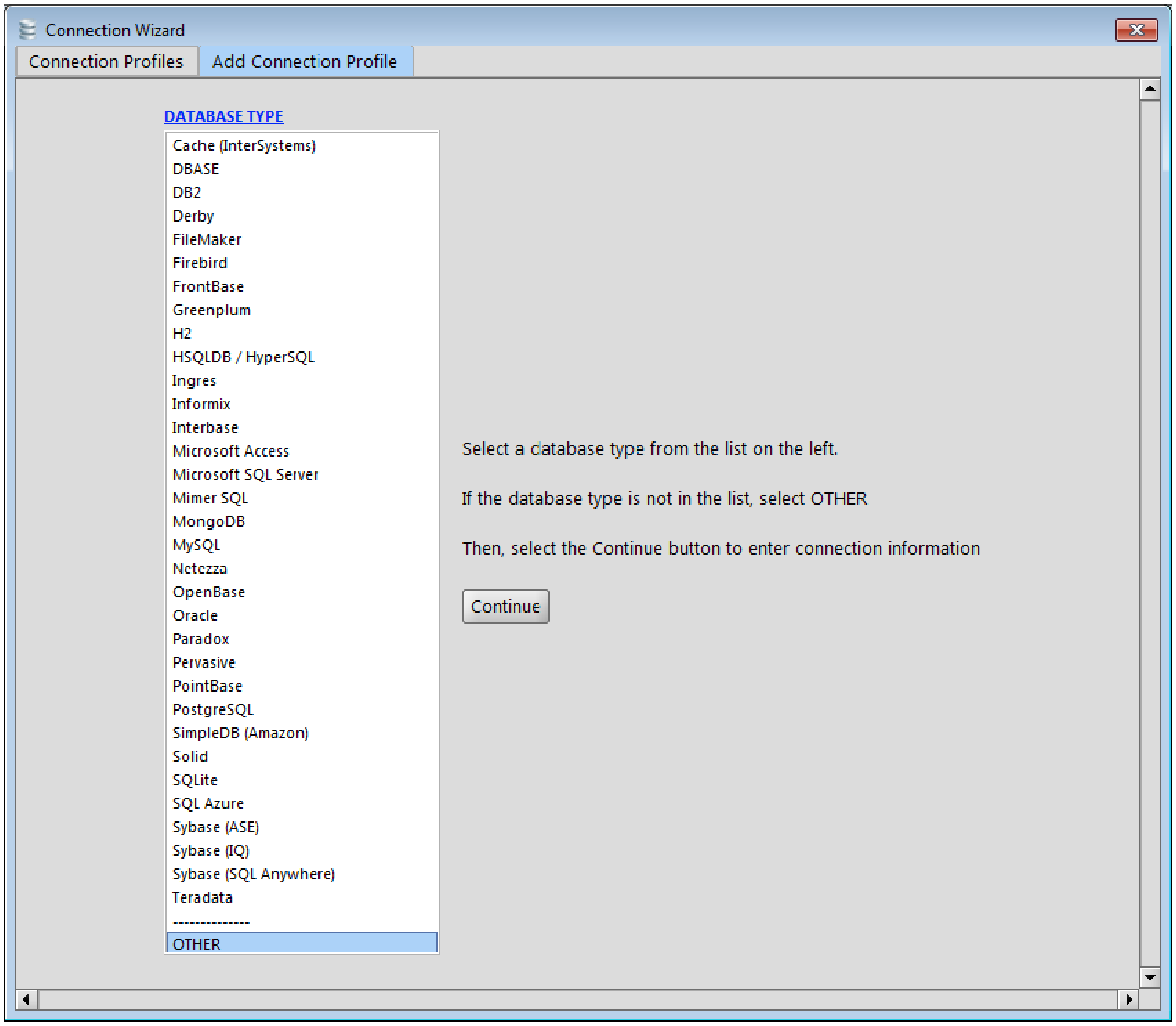
1. **Installing the Jars** - Copy the `unityjdbc.jar` into the RazorSQL JRE directory (e.g. `C:\Program Files (x86)\RazorSQL\jre\lib\ext`).

2. **Setup RazorSQL** - Start RazorSQL. Add a UnityJDBC connection profile under menu `Connection->Add Connection Profile`.



RazorSQL: Adding a New Connection Profile

3. **Setup RazorSQL - Part 2** - Select OTHER then press the CONTINUE button.



Cache (InterSystems)
DBASE
DB2
Derby
FileMaker
Firebird
FrontBase
Greenplum
H2
HSQLDB / HyperSQL
Ingres
Informix
Interbase
Microsoft Access
Microsoft SQL Server
Mimer SQL
MongoDB
MySQL
Netezza
OpenBase
Oracle
Paradox
Pervasive
PointBase
PostgreSQL
SimpleDB (Amazon)
Solid
SQLite
SQL Azure
Sybase (ASE)
Sybase (IQ)
Sybase (SQL Anywhere)
Teradata

OTHER

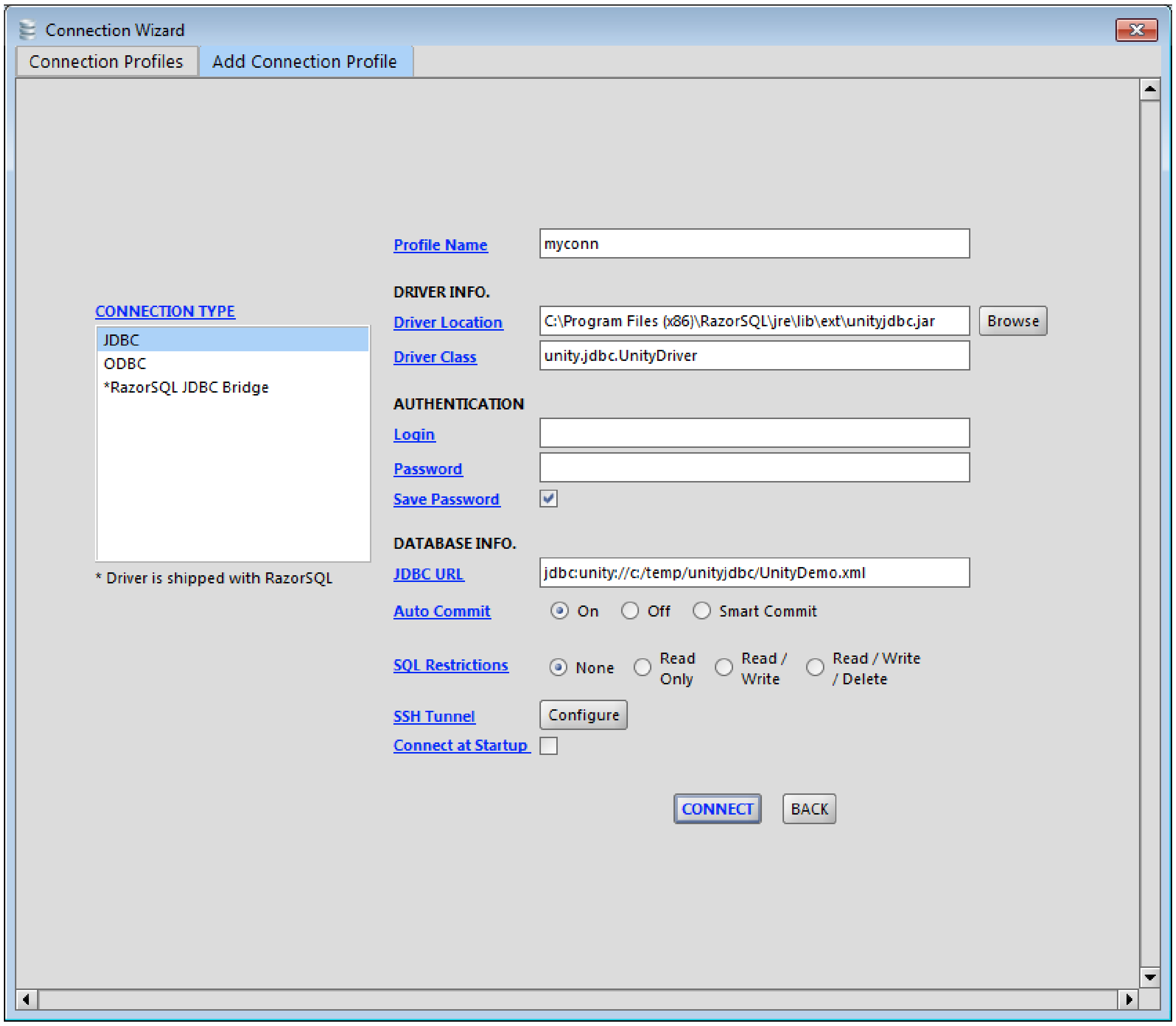
Select a database type from the list on the left.

If the database type is not in the list, select OTHER

Then, select the Continue button to enter connection information

RazorSQL: Adding a UnityJDBC Virtual Connection using OTHER Option

4. **Setup RazorSQL Connection Profile** - When setting up the connection profile, the profile name can be anything you wish. The driver location is where you just put the `unityjdbc.jar`. The driver class is `unity.jdbc.UnityDriver`. There is no login or password. The JDBC URL is the location of the configuration files for the virtual sources. In this example, we put the files `UnityDemo.xml`, `UnityDemoOrder.xml`, `UnityDemoPart.xml` that came in the distribution in the directory `c:/temp/unityjdbc`. You will change this to your source files that you built using the SourceBuilder utility.



CONNECTION TYPE

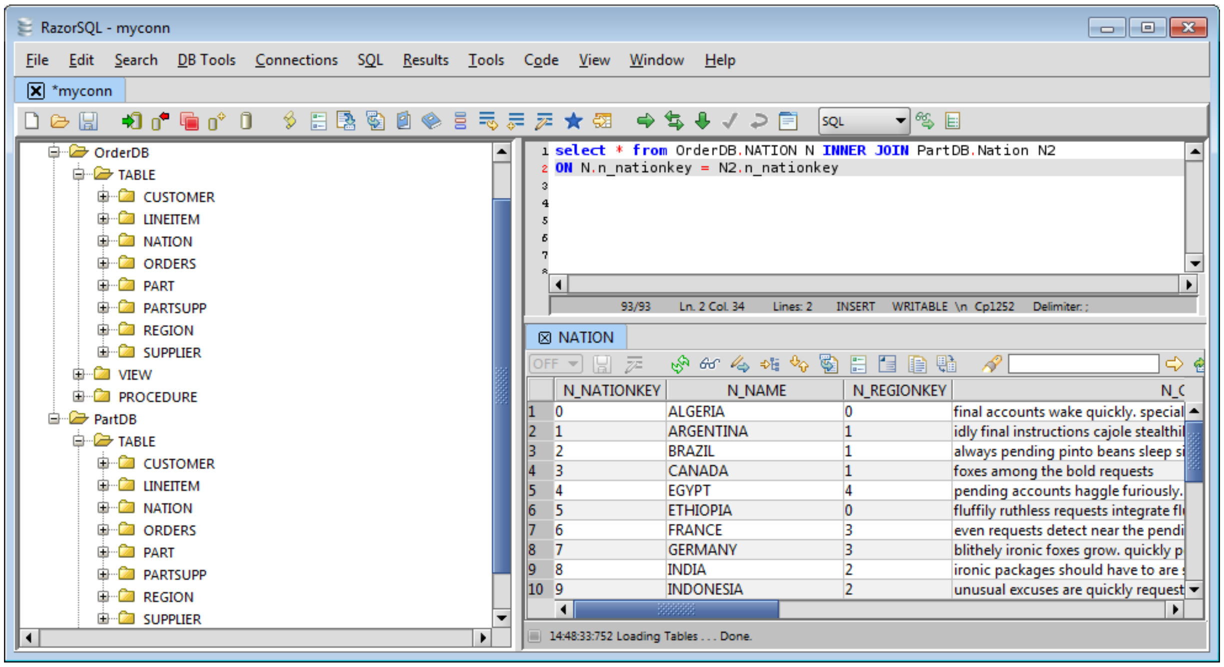
- JDBC
- ODBC
- *RazorSQL JDBC Bridge

* Driver is shipped with RazorSQL

Profile Name	myconn
DRIVER INFO.	
Driver Location	C:\Program Files (x86)\RazorSQL\jre\lib\ext\unityjdbc.jar
Driver Class	unity.jdbc.UnityDriver
AUTHENTICATION	
Login	
Password	
Save Password	✓
DATABASE INFO.	
JDBC URL	jdbc:unity://c:/temp/unityjdbc/UnityDemo.xml
Auto Commit	[✓ On] Off Smart Commit
SQL Restrictions	[✓ None] Read Only Read / Write Read / Write / Delete
SSH Tunnel	Configure
Connect at Startup	☐

RazorSQL: Creating a UnityJDBC Virtual Connection

5. **RazorSQL Multiple Database Connection and Query** - Connect to the connection profile. You can then see tables from all your virtual sources and built a multiple database query.



```
select * from OrderDB.NATION N INNER JOIN PartDB.Nation N2
ON N.n_nationkey = N2.n_nationkey
```

```
93/93     Ln. 2 Col. 34     Lines: 2     INSERT     WRITABLE  \n  Cp1252     Delimiter: ;
```

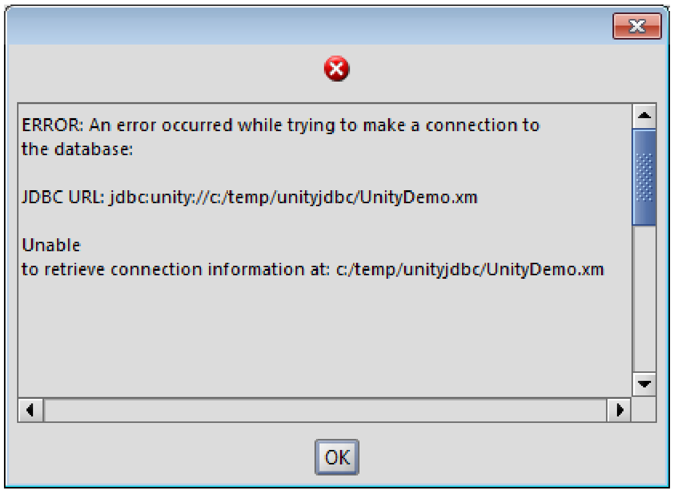
	N_NATIONKEY	N_NAME	N_REGIONKEY	N_C
1	0	ALGERIA	0	final accounts wake quickly. special
2	1	ARGENTINA	1	idly final instructions cajole stealthi
3	2	BRAZIL	1	always pending pinto beans sleep s
4	3	CANADA	1	foxes among the bold requests
5	4	EGYPT	4	pending accounts haggle furiously.
6	5	ETHIOPIA	0	fluffily ruthless requests integrate fl
7	6	FRANCE	3	even requests detect near the pendi
8	7	GERMANY	3	blithely ironic foxes grow. quickly p
9	8	INDIA	2	ironic packages should have to are
10	9	INDONESIA	2	unusual excuses are quickly request

```
14:48:33:752 Loading Tables . . . Done.
```

RazorSQL: Executing a Multiple Database Join with UnityJDBC

A few possible setup errors and their resolution:

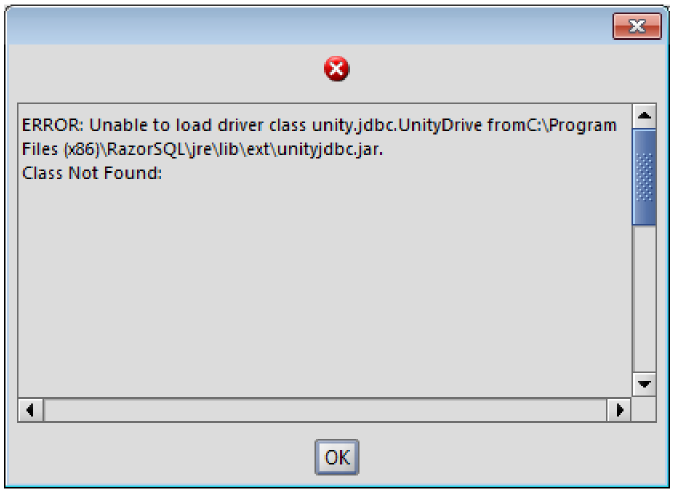
1. **Unable to make a connection** - If you do not type the connection string correctly or you have not put the XML files in the right location, you will get the error below. To fix, verify the location of the files.



```
ERROR: An error occurred while trying to make a connection to
the database:

JDBC URL: jdbc:unity://c:/temp/unityjdbc/UnityDemo.xm

Unable
to retrieve connection information at: c:/temp/unityjdbc/UnityDemo.xm
```

RazorSQL: Error When Unable to Find Sources File

2. **Unable to find driver** - If you did not put the correct class name or location of the `unityjdbc.jar`, you will get the following error. Verify the class name and jar location to resolve.



```
ERROR: Unable to load driver class unity.jdbc.UnityDrive fromC:\Program
Files (x86)\RazorSQL\jre\lib\ext\unityjdbc.jar.
Class Not Found:
```

RazorSQL: Error When Unable to Find Driver or Incorrect Driver Class Name

Chapter 6. Supported SQL Syntax

Overview

UnityJDBC supports a cross-database `SELECT` statement. The `SELECT` statement has the standard SQL-92 syntax and supports `WHERE`, `ORDER BY`, `GROUP BY`, and `HAVING`. UnityJDBC supports subqueries on a single database and cross-database including subqueries in the `FROM` clause and `WHERE` clause. entire query is on a single database. SQL functions are supported using a function syntax with parameters rather than using SQL keywords and syntax. Table and fields often should be prefixed with the database name they originate from. This database name is provided in the schema file for the data source.

Data Types

The standard SQL data types are supported. Since UnityJDBC uses the JDBC drivers provided by database vendors, non-standard data types may not be universally supported.

Identifiers

An *identifier* is a string used to reference a database, table, or field. Identifiers follow the standard SQL rules. Since a UnityJDBC query may span multiple databases, table and field identifiers defined in a data source may not be unique across all data sources. In which case, the database name should be added to the identifier to create a unique system-wide identifier. For instance, consider an order database given the name `OrderDB` with a table called `Orders` and fields `id` and `orderDate`. The `Orders` table may be referred to using only `Orders` or `OrderDB.Orders`. Similarly, the field `id` may be referred to as `Orders.id` or `OrderDB.Orders.id`. Standard aliasing using `AS` in the `FROM` and `SELECT` clauses is supported. Delimited identifiers are supported by enclosing in double quotes (e.g. `"from"` or `"my field with spaces"`). Delimited identifiers must be used for SQL reserved words.

Functions and Operators

Arithmetic operators +, -, /, %, * are supported as well as generic expressions. Functions are not specified according to SQL keyword syntax but rather as a function identifier with parameters similar to programming languages. The format of functions is: `function (param1, param2, ...).`

Logical Operators

The logical operators of `AND`, `OR`, `NOT`, and `XOR` are available.

Comparison Operators

The following comparison operators are available:

Table 6.1. Comparison Operators

Operator	Description
<	less than
>	greater than
<=	less than or equal to

Operator	Description
>=	greater than or equal to
=	equal
!=	not equal
IS [NOT] NULL	tests if value is NULL
IS [NOT] [TRUE | FALSE]	tests if value is true or false

Arithmetic Functions and Operators

The following mathematical operators are supported:

Table 6.2. Mathematical Operators

Operator	Description
+	addition (and string concatenation for strings)
-	subtraction
/	division
%	modulus (remainder of integer division)
*	multiplication

The following are a few of the mathematical functions supported. A complete list of functions is available on the web site.

Table 6.3. Mathematical Functions

Function	Return Type	Example	Result	Description
abs(x)	Same as x	abs(-17.4)	17.4	Absolute value
ceil(x)	Same as input	ceil(-42.8)	-42	Smallest integer not less than argument
exp(x)	Same as input	exp(1.0)	2.718	exponential
floor(x)	Same as input	ln(2.0)	0.69314	natural logarithm
log(x)	Same as input	log(100.0)	2	base 10 logarithm
power(a, b)	double precision	power(9,3)	729	a raised to the power of b
random()	double precision	random()		random value between 0.0 and 1.0
sqrt(x)	double precision	sqrt(2.0)	1.4142	square root

String Functions

The following are a few of the string functions supported. A complete list of functions is available on the web site.

Table 6.4. String Functions

Function	Return	Example	Result	Description
<str> + <str>	String	'Unity' + 'JDBC'	UnityJDBC	String concatenation
ascii(string)	int	ascii('xyz')	120	ASCII code of the first character of the input string
length(string)	int	length('UnityJDBC')	9	Length of string in characters
lower(string)	String	lower('JDBC')	jdbc	Convert string to lower case
position(search, target)	int	position('J','UnityJDBC')	5	Location of search in target (indexed from 1)
replace(source, search, replace)	String	replace('abUnityabJDBC', 'ab', 'XX')	XXUnityXXJDBC	Replace all occurrences of search string in source string with replace string
substring(string, start)	String	substring('UnityJDBC',6)	JDBC	substring starting at position start
substring(string, start, count)	String	substring('UnityJDBC,6,2)	JD	substring starting at position start and taking count characters
trim(string)	String	trim(' UnityJDBC ')	UnityJDBC	remove leading and trailing spaces from string
ltrim(string) OR trim(string, 'LEADING')	String	trim(' UnityJDBC ')	'UnityJDBC '	remove leading spaces from string
rtrim(string) OR trim(string, 'TRAILING')	String	trim(' UnityJDBC ')	' UnityJDBC'	remove trailing spaces from string
trim(string, ['BOTH', 'LEADING', 'TRAILING'], [<chars>])	String	trim('aaaUnityJDBCbbb', 'BOTH', 'ab')	UnityJDBC	remove leading, trailing or both from string where characters removed may be optionally specified in <chars>
upper(string)	String	upper('jdbc')	JDBC	Convert string to upper case

Pattern Matching Operators

Pattern matching is supported using the `LIKE` operator.

For example, `'abcdef' LIKE 'ab%'` is true. The '%' is used to match one or more characters, and '_' is used to match a single character.

Data Type Conversion Functions

Data type conversions are performed using the `CAST(x,y)` function. The `CAST` function takes any object as the first parameter and takes a string literal representation of the type to cast to as the second parameter. Note that the type must be put in single quotes as a string literal. Example:

`CAST(45, 'VARCHAR')` creates `'45'`

Possible type names are: `'VARCHAR'`, `'CHAR'`, `'INT'`, `'FLOAT'`, `'DOUBLE'`, `'DATE'`, `'TIMESTAMP'`, `'TIME'`.

Date/Time Functions and Operators

The following are a few of the date functions supported. A complete list is on the website.

Table 6.5. Date Functions

Function	Return Type	Example	Result	Description
CURRENT_TIMESTAMP	TIMESTAMP	CURRENT_TIMESTAMP	2011-07-06 12:53:45	Returns the current date. Format: "yyyy-MM-dd HH:mm:ss"
CURRENT_TIME	TIME	CURRENT_TIME	12:53:45	Returns the current time. Format: "HH:mm:ss"
CURRENT_DATE	DATE	CURRENT_DATE	2011-07-06	Returns the current date. Format: "yyyy-MM-dd"
YEAR	INT	YEAR('2011-07-06)	2011	Returns the year of the given date expression.
MONTH	INT	MONTH('2011-07-06)	7	Returns the month of the given date expression.
DAY	INT	DAY('2011-07-06)	6	Returns the day of the given date expression.
DATEADD	TIMESTAMP	DATEADD('2011-07-06', INTERVAL 3 days)	2006-07-06 12:53:45	Allows the addition of a given date field to a datetime expression. Intervals are supported and are translated as necessary for systems that do not support them.

Aggregate Functions

The following aggregate functions are supported:

Table 6.6. Aggregate Functions

Function	Argument Type	Return Type	Description
avg(x)	int, float, double precision type	int for integer types, double precision for float/double types	Average of all input values
count(*)	N/A	int	Count of number of input values
count(x)	any	int	Count of number of non-null input values
group_concat(x)	any	varchar	Returns a comma-separated list of all input values.
max(x)	any comparable type	same as input	Maximum of all input values

Function	Argument Type	Return Type	Description
min(x)	any comparable type	same as input	Minimum of all input values
sum(x)	int, float, double precision type	int for integer types, double precision for float/double types	Sum of all input values

User-Defined Functions and Support for Other Functions

For queries on a single database, UnityJDBC parses functions and passes them directly to the database engine for execution. Thus, all functions that can be executed at the source are available. UnityJDBC and user-defined functions are used only when applying functions to data **after** it is extracted from the sources. UnityJDBC will parse queries containing functions that it itself cannot process in its internal database engine. These functions are passed down to the database engine and executed locally. Only functions that require inputs from more than one database are processed in the UnityJDBC database engine. All other functions are passed down to the sources.

UnityJDBC supports user-defined functions (UDFs). Adding your own user-defined function is easy. There are two types of functions: row functions and aggregate functions. A row function operates on one row at a time for its data and includes functions like `SUBSTRING()` and `ABS()`. An aggregate function is used in `GROUP BY` queries and aggregates an expression (usually a column) across multiple rows in a group to produce a single value. Examples include `MAX()` and `COUNT()`.

To create a row function, you must create a Java class that extends the Function class. A template example is in the file `F_Function_Template.java`. This class must implement a constructor, an `evaluate()` method, and provide information on the parameters it requires. Once completed, as long as this function is available in the `CLASSPATH`, UnityJDBC will search for it when called. A similar template is available for aggregate functions, `A_Aggregrate_Template.java`. Sample code is provided in the directory `unity/functions`.

Function Translation

UnityJDBC has a database of known functions. This database contains information on what functions are supported on each data source. This is how UnityJDBC processes functions:

1. **UnityJDBC does not support function** - If a function is not in the UnityJDBC database, it is passed down as-is to the underlying source. If the source is able to execute it successfully, the query continues. If not, an error is thrown.

2. **UnityJDBC supports function, data source requires translation** - If the function requested in the query is not directly supported by the data source (different name, different parameters, etc.), but UnityJDBC contains a mapping in its database, the function is translated to the correct form on the data source and executed on the data source.

3. **UnityJDBC supports function, data source does not support function** - If UnityJDBC supports the function but not the data source, then the query is optimized to perform as much of the processing as possible on the source, but the function execution is performed internally in UnityJDBC. This way your query can execute on data sources with the help of UnityJDBC that do not support the required functions.

4. **UnityJDBC is running with local execution** - If the local execution flag is set for the UnityStatement object executing the query, all functions except aggregate functions are executed by UnityJDBC. This setting may be useful to reduce load on the source or to guarantee absolute consistency of function execution across different sources.

The UnityJDBC function database is encrypted and stored in the `unityjdbc.jar`. To add user-defined functions to the function database, create a `mapping.xml` file in the JRE classpath (execution directory, etc.) that stores the information on the function. An example is included in the release and more information is available on the web site.

Non-parsed Functions

UnityJDBC attempts to support most of the SQL standard. If there is a function or feature not supported, it is possible to use the `NP()` function to pass the query string directly to the data source by-passing UnityJDBC validation. This may be used to support non-standard functions or SQL syntax. Below are several examples.

```
Query:
SELECT N1.n_nationkey, NP('OrderDB','n_name','varchar')
FROM OrderDB.Nation N1 WHERE N1.n_nationkey = 1;

Result: (n_name is substituted directly into the query)
SELECT n_nationkey, n_name
FROM Nation N1 WHERE N1.n_nationkey = 1

Query:
SELECT N1.n_nationkey,  NP('OrderDB','(select n_name from nation n2
where N1.n_nationkey = N2.n_nationkey)','varchar') as name
FROM OrderDB.Nation N1 WHERE N1.n_nationkey = 1

Result:
SELECT N1.N_NATIONKEY,
(select n_name from nation n2 where N1.n_nationkey = N2.n_nationkey) name
FROM NATION N1 WHERE N1.N_NATIONKEY = 1

Query:
SELECT N2.*
FROM NP('OrderDB',
 '(select n_name,n_nationkey from nation)','n_name,n_nationkey') N1,
NP('PartDB',
 '(select n_name,n_nationkey from nation)','n_name,n_nationkey') as N2
where N2.n_nationkey < 2 and N1.n_nationkey = N2.n_nationkey;

Result:
// Substitutes subquery for each of the two data sources (OrderDB and PartDB).
// The result of the two subqueries is then joined at the UnityJDBC level.
// OrderDB:
SELECT N2.n_name, N2.n_nationkey
FROM (select n_name,n_nationkey from nation) N2\n WHERE N2.n_nationkey < 2
// PartDB:
SELECT N1.n_nationkey FROM (select n_name,n_nationkey from nation) N1
```

More information on non-parsed functions is available on the web site.

SELECT Statement

The SELECT statement supported by UnityJDBC has the following syntax.

```
SELECT [ALL | DISTINCT ] <exprList>
      [FROM <tableList>]
      [WHERE <condition>]
      [GROUP BY <exprList>
      [HAVING <condition>]
      [ORDER BY <expr> [ASC | DESC],...]
      [LIMIT <expr> [OFFSET <expr>]]
```

- An `<exprList>` is a list of expressions. Each individual expression `<expr>` may be a column identifier, a literal constant, or some expression consisting of operators, functions, constants, and column identifiers. Recall that a column identifier may often need to be prefixed by its database name and table name.
- A `<tableList>` is a list of table references. Each table reference can be aliased using the `AS` operator. A table reference may also be a named subquery such as `SELECT * FROM (SELECT * FROM T1) AS R WHERE R.val > 50`.
- A `<condition>` is a boolean condition that may contain multiple subconditions related using `AND`, `OR`, and `XOR`.
- If the `GROUP BY` clause is used, no attributes should be present in the `SELECT <exprList>` that are not in an aggregate function or are `GROUP BY` attributes.
- The `HAVING <condition>` filters groups and typically should contain only aggregate functions.
- The `ORDER BY` clause can order results on any number of attributes in either ascending or descending order.
- The `LIMIT` clause allows paging of results. The `OFFSET` clause determines the first row of the result with the first row numbered as 1.

Some examples using the TPC-H schema follow. The database name for these examples is '`OrderDB`'.

Return all nations with their key and name:

```
SELECT OrderDB.Nation.n_nationkey, OrderDB.Nation.n_name
FROM   OrderDB.Nation;
```

Return the nations and their regions. Only return nations in the region name of 'AMERICA'. Note the use of table aliasing using AS.

```
SELECT N.n_nationkey, N.n_name, R.r_regionkey, R.r_name
FROM OrderDB.Nation as N, OrderDB.Region as R
WHERE N.n_regionkey = R.r_regionkey AND R.r_name = 'AMERICA';
```

Calculate the number of countries in each region. Only return a region and its country count if it has more than 4 countries in it. Order by regions with most countries.

```
SELECT R.r_regionkey, R.r_name, COUNT(N.n_nationkey)
FROM OrderDB.Nation as N, OrderDB.Region as R
WHERE N.n_regionkey = R.r_regionkey
GROUP BY R.r_regionkey, R.r_name
HAVING COUNT(N.n_nationkey) > 4
ORDER BY COUNT(N.n_nationkey) DESC;
```

INSERT Statement

The `INSERT` statement supported by UnityJDBC has the following syntax:

```
INSERT INTO <tbl_name> [(<col_name>,...)] VALUES <exprList>;
```

Specifying column names is optional. An example is below:

```
INSERT INTO mydb.Customer (id,firstname,lastname,street,city)
       VALUES (52,'Fred','Jones','Smith Lane', 'Chicago');
```

UnityJDBC also supports `INSERT INTO ... SELECT` with the following syntax:

```
INSERT INTO <tbl_name> [(<col_name>,...)] VALUES <exprList>
(SELECT <query>);
```

This is useful for storing query results into another table. Note that this table and all its column must already exist or an error will be returned. Here is an example:

```
INSERT INTO emptydb.customer (SELECT * FROM mydb.customer);
```

UPDATE Statement

The `UPDATE` statement supported by UnityJDBC has the following syntax:

```
UPDATE <tbl_name> SET col1=expr1, col2=expr2, ... [WHERE <condition>];
```

An example is below:

```
UPDATE Employee SET salary=salary*1.10 WHERE age > 50;
```

DELETE Statement

The `DELETE` statement supported by UnityJDBC has the following syntax:

```
DELETE FROM <tbl_name> [WHERE <condition>];
```

An example is below:

```
DELETE FROM Employee WHERE salary > 100000;
```

EXPLAIN Statement

The `EXPLAIN` statement supported by UnityJDBC has the following syntax:

```
EXPLAIN <query>
```

The `EXPLAIN` statement provides an explanation of the how UnityJDBC will execute a given query include the translated queries to be executed on each source, the operations performed by UnityJDBC, and the expected cost of each query operation. Using EXPLAIN is a great way to determine the performance of queries and improve their execution speed.

By-Pass Statement

You can use methods to by-pass or flow through the driver to execute an untranslated query directly on a single source. In the `UnityStatement` class are these two methods:

```
ResultSet executeByPassQuery(String dbName, String sql)
int executeByPassUpdate(String dbName, String sql)
```

These methods will execute a query or update on a single source (given by name). The SQL statement provided is not parsed or validated and passed directly to the source driver. There is no overhead in this type of query as it is equivalent to invoking the source's JDBC driver directly.

Chapter 7. Supported JDBC Methods

Overview

UnityJDBC supports the majority of the methods in the `Driver`, `Connection`, `Statement`, `ResultSet`, and `ResultSetMetaData` interfaces. UnityJDBC supports the `PreparedStatement` interface but not the `CallableStatement` interface. UnityJDBC supports native updates using `INSERT`, `DELETE`, and `UPDATE`. It is also possible to use `INSERT INTO` to insert query results into another table. UnityJDBC does not support transactions across databases. Support for other JDBC methods is also limited by the underlying support of the JDBC driver for each data source. UnityJDBC requires a JDK of 1.6 or higher.

Chapter 8. UnityJDBC Driver Internals

Overview

UnityJDBC contains an embedded database engine to join the results produced by executing queries on other JDBC-accessible sources. It requires a JDBC driver for each source to be accessed. The UnityJDBC architecture is the result of years of research and development and has been published in numerous technical and research publications.

Embedded Relational Database Engine

Embedded in UnityJDBC is a relational database engine and associated operators of selection, projection, grouping, ordering, and join. You can build your own global query spanning data sources by combining these operators into an execution tree. In the distribution is a file called `ExampleEngine.java` which demonstrates how to use the engine to build an execution tree. Also in this file is an example on how you can have Unity parse but not execute a global query. UnityJDBC will return its global query and execution plan which you can later execute. This feature gives you the opportunity to modify the global execution plan before execution if desired. It also allows you to track the progress of a global query at the operator level.

Chapter 9. History and Planned Features

History of UnityJDBC

UnityJDBC is the product of over 10 years of research and development in database integration and virtualization. UnityJDBC was first released in 2006 and commercial support and development has been ongoing since 2011.

Table 9.1. UnityJDBC Release History

Release Version and Date	Major Features
UnityJDBC v1.0 - May 2006	Cross-database join support, match functions, full optimizer, query by-pass
UnityJDBC v2.0 - May 2007	Connection pools, DataSource connections, more functions
UnityJDBC v3.0 - May 2008	Native INSERT/UPDATE/DELETE, INSERT INTO...SELECT across databases, PreparedStatements, user-defined functions
UnityJDBC v4.0 - August 2011	Database dialect translation, paging using LIMIT/OFFSET, single database subqueries, result caching
UnityJDBC v4.1 - June 2012	Integration with SQuirreL SQL and JasperReports, BLOB support
UnityJDBC v4.2 - January 2013	Memory-optimized query execution engine, EXPLAIN for query plans, subqueries in FROM clause, improved SourceBuilder GUI
UnityJDBC v4.3 - June 2014	Full subqueries, EXCEPT/INTERSECT, implementation of PooledDataSource and ConnectionPoolDataSource, support for MongoDB, TokuMX, Cassandra, and ServiceNow.

Planned Features

The following features are planned in coming versions. Version 5.0 will be released in November 2014 with new versions released approximately every year. If you have any feature requests, please e-mail support@unityjdbc.com.

Table 9.2. Planned Features in Coming Versions

Version	Feature Description
5.0	Improved SourceBuilder user interface
6.0	Support for distributed transactions

Feature List

The following table summarizes the features of UnityJDBC and the version where they were first introduced.

Table 9.3. UnityJDBC Feature List

Version	Feature Description
1.0	Cross-database SQL queries for any JDBC source
1.0	Query by-pass

Version	Feature Description
1.0	MERGE feature with MATCH functions
1.0	Embedded relational database engine
1.0	Source and schema file encryption
1.0	Support for Applets
1.0	Support for query results/databases larger than main memory
2.0	DataSource connections
2.0	Pooled connections
3.0	Prepared Statements
3.0	User-defined Functions
3.0	INSERT, DELETE, UPDATE on a single source
3.0	INSERT, DELETE, UPDATE across sources
3.0	INSERT INTO across sources
4.0	Paging using LIMIT/OFFSET
4.0	Query and ResultSet caching
4.0	Universal dialect and function translation (support for sources missing functions)
4.0	Single source subqueries
4.1	Support for BLOBs/CLOBs.
4.2	Multiple source subqueries in FROM clause
4.2	EXPLAIN command for query execution information
4.2	Ability to control tables extracted into data virtualization by inclusion/exclusion patterns
4.3	Full subquery support including correlated subqueries
4.3	INTERSECT/EXCEPT
4.3	MongoDB, TokuMX, and Cassandra support

Contacts and Support

Please contact support@unityjdbc.com if you encounter any bugs, issues, or have feature requests.