JDBC URL Format For Different Databases

Last modified: December 5, 2021

by Kai Yuan (https://www.baeldung.com/author/kai-yuan/)

Persistence (https://www.baeldung.com/category/persistence/)

JDBC (https://www.baeldung.com/tag/jdbc/)

SQL (https://www.baeldung.com/tag/sql/)

1. Overview

When we work with a database in Java, usually we connect to the database with JDBC (/java-jdbc).

The JDBC URL is an important parameter to establish the connection between our Java application and the database. However, the JDBC URL format can be different for different database systems.

In this tutorial, we'll take a closer look at the JDBC URL formats of several widely used databases: Oracle (https://www.oracle.com/database/technologies/), MySQL (https://www.mysql.com/), Microsoft SQL Server (https://www.microsoft.com/en-us/sql-server/sql-server-2019), and PostgreSQL (https://www.postgresql.org/).

2. JDBC URL Formats for Oracle

Oracle database systems are widely used in enterprise Java applications. Before we can take a look at the format of the JDBC URL to connect Oracle databases, we should first make sure the Oracle Thin database driver is in our classpath.

For example, if our project is managed by Maven, we need to add the *ojdbc8* dependency (https://search.maven.org/search?q=g:com.oracle.database.jdbc%20AND%20a:ojdbc8) in our *pom.xml*:

```
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>21.1.0.0</version>
</dependency>
```

The Thin driver offers several kinds of JDBC URL formats:

- Connecting to **SID (https://docs.oracle.com/cd/E11882_01/network.112/e41945/glossary.htm#BGBFBBAI)**
- Connecting to Oracle **service name (https://docs.oracle.com/cd/E11882_01/network.112/e41945/glossary.htm#BGBGIHFG)**
- URL with ***tnsnames.ora* (https://docs.oracle.com/database/121/NETRF/tnsnames.htm#NETRF007)**entries

Next, we'll go through each of these formats.

2.1. Connect to Oracle Database SID

In some older versions of the Oracle database, the database is defined as a SID. Let's see the JDBC URL format for connecting to a SID:

```
jdbc:oracle:thin:[<user>/<password>]@<host>[:<port>]:<SID>
```

For example, assuming we have an Oracle database server host "*myoracle.db.server:1521*", and the name of the SID is "*my_sid*", we can follow the format above to build the connection URL and connect to the database:

```
@Test
public void givenOracleSID_thenCreateConnectionObject() {
    String oracleJdbcUrl =
"jdbc:oracle:thin:@myoracle.db.server:1521:my_sid";
    String username = "dbUser";
    String password = "1234567";
    try (Connection conn = DriverManager.getConnection(oracleJdbcUrl,
username, password)) {
        assertNotNull(conn);
    } catch (SQLException e) {
        System.err.format("SQL State: %s\n%s", e.getSQLState(),
e.getMessage());
    }
}
```

2.2. Connect to Oracle Database Service Name

The format of the JDBC URL to connect Oracle databases via service name is pretty similar to the one we used to connect via SID:

```
@Test
public void givenOracleServiceName_thenCreateConnectionObject() {
    String oracleJdbcUrl =
"jdbc:oracle:thin:@//myoracle.db.server:1521/my_servicename";
    ...
    try (Connection conn = DriverManager.getConnection(oracleJdbcUrl,
username, password)) {
        assertNotNull(conn);
        ...
    }
    ...
}
```

2.3. Connect to Oracle Database With *tnsnames.ora* Entries

We can also include *tnsnames.ora* entries in the JDBC URL to connect to Oracle databases:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<host>)
(PORT=<port>))(CONNECT_DATA=(SERVICE_NAME=<service>)))
```

Let's see how to connect to our "*my_servicename*" service using entries from the *tnsnames.ora* file:

```
@Test
public void givenOracleTnsnames_thenCreateConnectionObject() {
    String oracleJdbcUrl = "jdbc:oracle:thin:@" +
      "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)" +
      "(HOST=myoracle.db.server)(PORT=1521))" +
      "(CONNECT_DATA=(SERVICE_NAME=my_servicename)))";
    ...
    try (Connection conn = DriverManager.getConnection(oracleJdbcUrl,
username, password)) {
        assertNotNull(conn);
        ...
    }
    ...
}
```

In this section, let's discuss how to write the JDBC URL to connect to MySQL databases.

To connect to a MySQL database from our Java application, let's first add the JDBC driver *mysql-connector-java* dependency (https://search.maven.org/search?q=a:mysql-connector-java%20g:mysql) in our *pom.xml*:


```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.22</version>
</dependency>
```

Next, let's take a look at the generic format of the connection URL supported by the MySQL JDBC driver:

```
protocol//[hosts][/database][?properties]
```

Let's see an example of connecting to the MySQL database "*my_database*" on the host "*mysql.db.server*":

```
@Test
public void givenMysqlDb_thenCreateConnectionObject() {
    String jdbcUrl = "jdbc:mysql://mysql.db.server:3306/my_database?
useSSL=false&serverTimezone=UTC";
    String username = "dbUser";
    String password = "1234567";
    try (Connection conn = DriverManager.getConnection(jdbcUrl,
username, password)) {
        assertNotNull(conn);
    } catch (SQLException e) {
        System.err.format("SQL State: %s\n%s", e.getSQLState(),
e.getMessage());
    }
}
```

The JDBC URL in the example above looks straightforward. It has four building blocks:

- *protocol* – *jdbc:mysql:*
- *host* – *mysql.db.server:3306*
- *database* – *my_database*
- *properties* – *useSSL=false&serverTimezone=UTC*

However, sometimes, we may face more complex situations, such as different types of connections or multiple MySQL hosts, and so on.

Next, we'll take a closer look at each building block.

3.1. Protocol

Except for the ordinary "*jdbc:mysql:*" protocol, the *connector-java* JDBC driver still supports protocols for some special connections:

- **Load-balancing JDBC connections (https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-usagenotes-j2ee-concepts-managing-load-balanced-connections.html)** – *jdbc:mysql:loadbalance:*
- **JDBC replication connections (https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-source-replica-replication-connection.html)** – *jdbc:mysql:replication:*

Next, let's check out the details of another part of the connection URL — *hosts*.

3.2. Hosts

We've seen the JDBC URL example of defining a single host in a previous section — for example, *mysql.db.server:3306.*

However, **if we need to handle multiple hosts, we can list hosts in a comma-separated list: *host1, host2,…,hostN*.**

We can also enclose the comma-separated host list by square brackets: *[host1, host2,…,hostN]*.

Let's see several JDBC URL examples of connecting to multiple MySQL servers:

- *jdbc:mysql://myhost1:3306,myhost2:3307/db_name*
- *jdbc:mysql://[myhost1:3306,myhost2:3307]/db_name*
- *jdbc:mysql:loadbalance://myhost1:3306,myhost2:3307/db_name?user=dbUser&password=1234567&loadBalanceConnectionGroup=group_name&ha.enableJMX=true*

If we look at the last example above closely, we'll see that after the database name, there are some definitions of properties and user credentials. We'll look at these next.

Valid global properties will be applied to all hosts. Properties are preceded by a question mark "*?*" and written as *key=value* pairs separated by the "*&*" symbol:

```
jdbc:mysql://myhost1:3306/db_name?prop1=value1&prop2=value2
```

We can put user credentials in the properties list as well:

```
jdbc:mysql://myhost1:3306/db_name?user=root&password=mypass
```

Also, **we can prefix each host with the user credentials in the format "*user:password@host*"**:


```
jdbc:mysql://root:mypass@myhost1:3306/db_name
```

Further, **if our JDBC URL contains a list of hosts and all hosts use the same user credentials, we can prefix the host list**:

```
jdbc:mysql://root:mypass[myhost1:3306,myhost2:3307]/db_name
```

After all, **it is also possible to provide the user credentials outside the JDBC URL**.

We can pass the username and password to the *DriverManager.getConnection(String url, String user, String password)*

ql/DriverManager.html#getConnection(java.lang.String,java.lang.String,java.lang.String)) method when we call the method to obtain a connection.

4. JDBC URL Format for Microsoft SQL Server

Microsoft SQL Server is another popular database system. To connect an MS SQL Server database from a Java application, we need to add the *mssql-jdbc* dependency (https://search.maven.org/search?q=g:com.microsoft.sqlserver%20a:mssql-jdbc) into our *pom.xml*:

```
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>8.4.1.jre11</version>
</dependency>
```

Next, let's look at how to build the JDBC URL to obtain a connection to MS SQL Server.

The general format of the JDBC URL for connection to the MS SQL Server database is:

```
jdbc:sqlserver://[serverName[\instanceName][:portNumber]]
[;property=value[;property=value]]
```

Let's have a closer look at each part of the format.

- *serverName* – the address of the server we'll connect to; this could be a domain name or IP address pointing to the server
- *instanceName* – the instance to connect to on *serverName*; it's an optional field, and the default instance will be chosen if the field isn't specified
- *portNumber* – this is the port to connect to on *serverName* (default port is *1433*)
- *properties* – can contain one or more optional connection properties

Now, let's say we have an MS SQL Server database running on host "*mssql.db.server*", the *instanceName* on the server is "*mssql_instance*", and the name of the database we want to connect is "*my_database*".

Let's try to obtain the connection to this database:

```
@Test
public void givenMssqlDb_thenCreateConnectionObject() {
    String jdbcUrl =
"jdbc:sqlserver://mssql.db.server\\mssql_instance;databaseName=my_datab
ase";
    String username = "dbUser";
    String password = "1234567";
    try (Connection conn = DriverManager.getConnection(jdbcUrl,
username, password)) {
        assertNotNull(conn);
    } catch (SQLException e) {
        System.err.format("SQL State: %s\n%s", e.getSQLState(),
e.getMessage());
    }
}
```

5. JDBC URL Format for PostgreSQL

PostgreSQL is a popular, open-source database system. To work with PostgreSQL, the JDBC driver *postgresql* (https://search.maven.org/search?

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.18</version>
</dependency>
```

The general form of the JDBC URL to connect to PostgreSQL is:

```
jdbc:postgresql://host:port/database?properties
```

Now, let's look into each part in the above JDBC URL format.

The *host* parameter is the domain name or IP address of the database server.

If we want to specify an IPv6 address, the *host* parameter must be enclosed by square brackets, for example, *jdbc:postgresql://[::1]:5740/my_database*.mysql

The *port* parameter specifies the port number PostgreSQL is listening on. **The port parameter is optional, and the default port number is *5432*.**

As its name implies, the *database* parameter defines the name of the database we want to connect to.

```
@Test
public void givenPostgreSqlDb_thenCreateConnectionObject() {
    String jdbcUrl =
"jdbc:postgresql://postgresql.db.server:5430/my_database?
ssl=true&loglevel=2";
    String username = "dbUser";
    String password = "1234567";
    try (Connection conn = DriverManager.getConnection(jdbcUrl,
username, password)) {
        assertNotNull(conn);
    } catch (SQLException e) {
        System.err.format("SQL State: %s\n%s", e.getSQLState(),
e.getMessage());
    }
}
```

In the example above, we connect to a PostgreSQL database with:

- *host:port* – *postgresql.db.server:5430*
- *database* – *my_database*
- properties – *ssl=true&loglevel=2*

6. Conclusion

This article discussed the JDBC URL formats of four widely used database systems: Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.

We've also seen different examples of building the JDBC URL string to obtain connections to those databases.

As always, the full source code of the article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/persistence-modules/core-java-persistence-2).

**An intro to Spring Data, JPA
and Transaction Semantics Details with JPA**

Get Persistence Right with Spring

Download the E-book (/persistence-with-spring)

2 COMMENTS

Oldest

View Comments

Comments are closed on this article!

'freestar.com/?
ɔ&utm_source=baeldung.com&utm_content=baeldung_adhesion)

(https://freestar.com/?

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)

’freestar.com/?
o&utm_source=baeldung.com&utm_content=baeldung_adhesion)