Connection pools in OutSystems Platform

This document applies to OutSystems 10.

OutSystems Platform uses connection pools to ensure your applications perform well as they scale. With connection pools, an application doesn't open and close database connections. Instead the application requests a connection, and returns the connection to the pool when it's done.

This avoids the performance overhead of opening and closing database connections. This way, a new connection is only opened if there is no available connection on the pool. Connections are closed after they're idle for a given period.

For each database connection you define for a database server or schema, OutSystems Platform creates a connection pool to manage that connection.

Connection pooling under the hood

OutSystems Platform uses standard connection polling mechanisms:

- .NET stack - ADO.NET;
- Java stack - Oracle UCP.

Controlling the number of connections

You can fine-tune the minimum and maximum number of connections available on the pool.

.NET Stack

In the Configuration Tool you can customize the connection string used by the applications at runtime. For this, go to the 'Database' tab, and choose 'Advanced'. Check the documentation of
your database server to learn what parameters to send in the connection string.

Java Stack

In the Configuration Tool, you can control the number of connections in the 'Advanced Configurations'.

```
Skip Advanced configurations? [y]: n

==== Advanced Database Connection Settings =====
Default query timeout [30]:
Database update query timeout [600]:

==== Application Server =====
JDBC Connection Pool Inactivity Timeout [300]:
JDBC Connection Pool Minimum Size [10]:
JDBC Connection Pool Maximum Size [100]:
```

The values in square brackets are the default for that setting.

Minimum connections used at runtime

.NET Stack

On the .NET stack there is one connection pool per database server per application module. At runtime you typically have

(minimum number of connections configured) x (application modules) x (front-end servers)

As an example, if you have:

- 1 connection configured by default;
- 10 application modules deployed;
- 2 front-end servers.

You'll typically have 20 connections to the main database, and 20 connections to the sessions
database.

Java Stack

On the Java stack you have a single connection pool for all application modules. So you typically have

\((\text{minimum number of connections configured}) \times (\text{front-end servers})\)

As an example, if you have:

- 10 connection configured by default;
- 10 application modules deployed;
- 2 front-end servers

You'll typically have 20 connections to the main database, and 20 connections to the sessions database.

Maximum connections used at runtime

To make your applications run smoothly under heavy load, OutSystems Platform increases the number of database connections used. The maximum number of connections can be calculated as above, but replacing the minimum number of connections by the maximum number.

Real-world example

To better understand how OutSystems Platform handles database connections, analyze what happens on a typical production environment with the following configurations. We'll be using the default values for the connection pool, and assume the Sessions and Main database schemas are stored on the same database instance.

<table>
<thead>
<tr>
<th></th>
<th>.NET</th>
<th>JAVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front-end servers</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Updated: Tue, 24 Sep 2019 10:17:15 GMT
<table>
<thead>
<tr>
<th>Application modules</th>
<th>200</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. Connections</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Max. Connections</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

**.NET Stack**

On the .NET stack the application modules are not automatically loaded. Instead, they are loaded on their first access. This means that initially you only have a handful of application modules loading up, and therefore very few database connections.

Assuming that all application modules are eventually loaded:

- You have 200 application modules on each front-end;
- Each of these has one connection for each database the application module connects to. As a minimum it connects to the OutSystems Platform session and main databases.

This gives a total of 800 connections.

On a high-demand scenario, if there are 50 concurrent user accesses to an application module on each front-end:

- 50 concurrent users require 50 connections to the session database;
- We need other 50 connections to the main database to satisfy the user request. One connection was already established on each front-end for each database, so we'll need additional 49 connections per front-end per database.

This gives a total of 996 connections.

If that application module becomes even more popular and now receives 150 concurrent requests on each front-end:

- This value exceeds the maximum number of connections configured;
- Only the first 100 requests are answered concurrently;
- The other 50 requests on each front-end server will have to wait for a connection to become available. If this takes too long, the request will timeout.

Throughout this peak, the database will have:

- The minimum 800 connections open;
- 99 connections per database per front-end server. On a minimum there's the session and
main databases, giving 396 connections.

This gives a total of 1196 connections.

### JAVA Stack

In both the JBoss and WebLogic application servers all application modules are loaded when the application server starts. After the initialization there are 10 connections per database per front-end. At a minimum there are the session and main databases. This gives a total of 40 database connections.

On a high-demand scenario, if an application module receives 50 concurrent requests on each front-end:

- Each of the connection pools to the session and main databases is shared between all application modules. It's likely that some of 10 initial connections are being used by other application modules;
- If only 5 of those connections are free, we'll need 45 connections per database per front-end.

This gives a total of 220 connections.

If that application module becomes even more popular and now receives 150 concurrent requests on each front-end:

- Assuming we still have 5 free connections;
- We can create another 95 connections before reaching the maximum configured. This creates a total of 190 connections per front-end server;
- The remaining 55 requests have to wait for an available connection.

During this high-usage peak, there's a total of 400 connections.

### More Information:

This information was initially posted on the Community Forums.