 **Talend User Component tPostgresqlConnectionPool\***

**Purpose**

This component provides a data source to normal jobs as well as to services made in DI technology.

The idea is to have a connection pool wth following advantages:

1. The pool can be used in normal DI jobs, especcially in use cases where worker job needs a database connection and these jobs are called very frequently
2. The pool can be configured with normal context variables and can therefore can be configured with the same configuration file as the DI batch jobs.

**Talend-Integration**

You find these components in the palette under Databases/PostgreSQL

**Basic settings**

|  |  |
| --- | --- |
| **Property** | **Content** |
| Operational Mode | **Create Connection Pool**: Creates a connection pool for the PostgreSQL database  **Provide pooled DataSource in child job**: This causes every database component which is configured to use the data source gets its own connection.  Otherwise ALL components works with the SAME connection.  **Close Connection Pool**: Closes the pool. |
| Host | Database host |
| Port | Database port |
| Database | Database instance |
| Schema | Database schema |
| User | Database user |
| Password | Database users password |
| DataSource alias | The alias of the connection pool (JNDI name) |
| Auto Commit | Set the new connctions auto commit |

**Advanced settings**

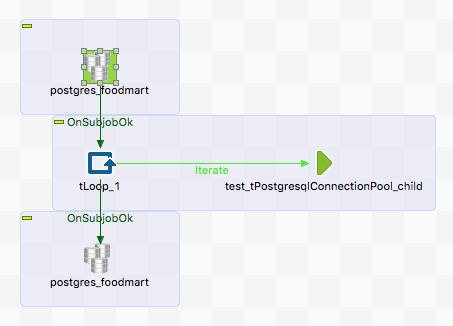
|  |  |
| --- | --- |
| **Property** | **Content** |
| Addional JDBC Parameters | Set here the addional JDBC properties. The parameters are key=value pairs separated by a semikolon or & |
| Test on borrow | If true all connction requested by the job will be checked before delivered |
| Test while idle | These settings are for the a background check for idle connections |
| Validation SQL | Set here a cheap SQL to check the connection. Avoid time consuming SQL here! |
| Time between checks | The time between 2 checks automatically performed in ms |
| Max idle time | The maximum milliseconds a connection can be idle before it will be removed from the pool |
| Number checked connection | The number of the connections checked whithin a background check cycle |
| Initial Size | The initial number of connection made at the start of the pool |
| Maximum Pool Size | The maximum number of connecxtion within the pool. If this size is reached all next attempts to get a connection will wait for the next free connection |
| Max. time to wait for a connection | The max. time a request have to wait until releasing the thread. The return connection is null – this will in most cases cause an exception. |
| Intial SQL | For some reason it is necessary to run special SQL code just at the time when the connction is newly created. |
| Debug | Cause debug output |

**Return values**

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | Last error message. Unfortunately, this is not the error message from the actually running job. This message is build from the tRunTask component. The current TAC web service does not provide this message. |
| DATABASE | The name of the connected database |

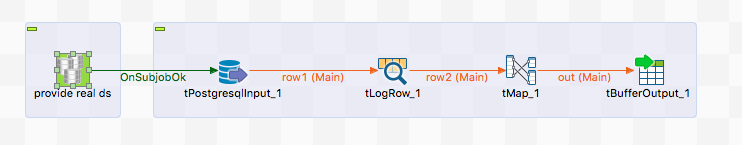
**Scenario 1: Using the pool in a batch DI job**

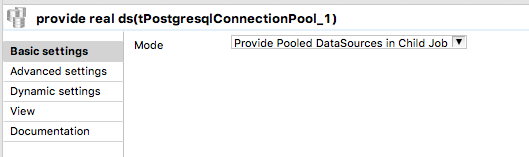
Data-Integration Job which calls very rapidly a job using database connctions.



The loop simulates the call trigger for the embedded job.

In the embedded job this pool can be used as usual. If you want to habe a standard conform behavior of the DataSource (unfortunately Talend in release up to 6.1.1 provides) you can improve the embedded job by adding the component here again with the mode: “Provide pooled DataSource to child job”

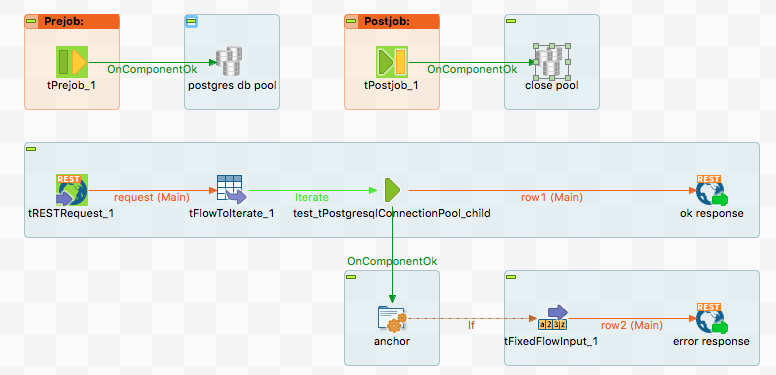




This way all database components using the DataSource settings gets their own connection – exactly what the standard recommends. If you need exactly one connection, leaf out this component or if you wish to steer it use reight at the start of the job a tPostgresqlConnection and configure here the DataSource. In all other database components choose the using of a separate connection -> your tPostgresqlConnection. Do not forget to close this connction again (it means in this case to put the connction back into the pool.

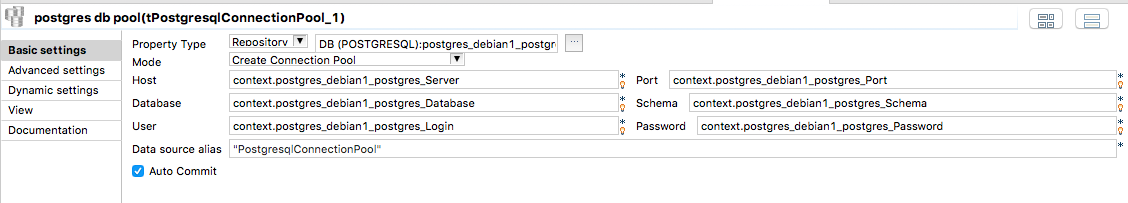
**Scenario 2: Using the pool in a service job**

This established the pool at the start of the service. The advantage is, you can use the normal context variables to control the pool and use e.g. the implicit context load. The disadvantage of a pool dedicated to one service is the pooled connections are dedicated to this service. Because of the idle eviction this does not lead to a unwanted high number of connections.



The service design. If the service will shut down the pool will be closed this way.

Here the settings:



The job is right the same as in Scenario 1.