

Smooth Transition to PostgreSQL for Java Applications Using a JDBC Proxy

A seamless solution for database migration

Thorsten Reimers, SoftQuadrat GmbH



The Challenge

Common obstacles when transitioning database systems:

- Transaction management differs significantly (e.g., compared to Oracle)
- SQL dialects vary even with ANSI SQL compliance
- Essential features like CLOB support might be missing
- Direct driver replacement rarely results in a working application



Our Solution

- JDBC proxy driver as an intermediate layer between application and PostgreSQL
- Maps transaction management from various databases (MS SQL Server, Oracle, Exasol) to PostgreSQL
- Translates SQL dialects to PostgreSQL-compliant syntax
- Adds missing functionality like CLOB support



Implementation: Step 1

- Modify JDBC URL in your application
- Switch to the proxy driver format
- Example: jdbc:dsjdbc:postgresql://localhost:5432/
- No changes to your application code required

		Connection "Datasqill Jdbc Proxy" configuration	
	Connection settings Datasqill Postgres connection sett	tings	PostgreSC
Example with DBeaver	 Connection settings Initialization Shell Commands Client identification Transactions General Metadata Errors and timeouts Data Editor SQL Editor 	Main Datasqill Postgres Driver properties SSH SSL Server Connect by: Host URL URL: jdbc:dsjdbc:postgresql://localhost:5432/ Host: localhost Database: Username is used if not specified Authentication Authentication: Authentication: Database Native Password: ••••••••• Session role: Local Client:	+ Network configurations Port: Show all databases
	Test Connection		Cancel OK



Implementation: Step 2

- Add dsjdbcproxy-1.0.jar to your Java classpath
- Compatible with existing connection management tools
- Works with tools like DBeaver (as shown in example)
- Seamless integration with your existing code

Add Folder Add Artifact
Add Folder Add Artifact
Add Artifact
Edit
Delete
ownload/Upda
Information
Classpath
own Inf

Example with DBeaver



Under the Hood

```
@Override
```

```
public ResultSet executeQuery(String sql) throws SQLException {
  boolean error = true;
  Savepoint savepoint = null;
  if (!connection.getIsAutoCommit()) {
    savepoint = physicalConnection.setSavepoint("datasqill");
  }
 try {
   ResultSet resultSet = physicalStatement.executeQuery(sql);
   error = false;
    if (!connection.getIsAutoCommit())
      try {
        physicalConnection.releaseSavepoint(savepoint);
      } catch (Exception e) {}
        return createResultSet(resultSet);
 } finally {
    if (error) {
      if (!connection.getIsAutoCommit())
        try {
          physicalConnection.rollback(savepoint);
        } catch (Exception e) {}
    }
  }
```

- Transparent
 transaction handling
- Automatic savepoint management
- Error recovery mechanisms

}



Contact Us

Interested in our solution? Email: contact@softquadrat.de We would be glad to share more information and support your transition.