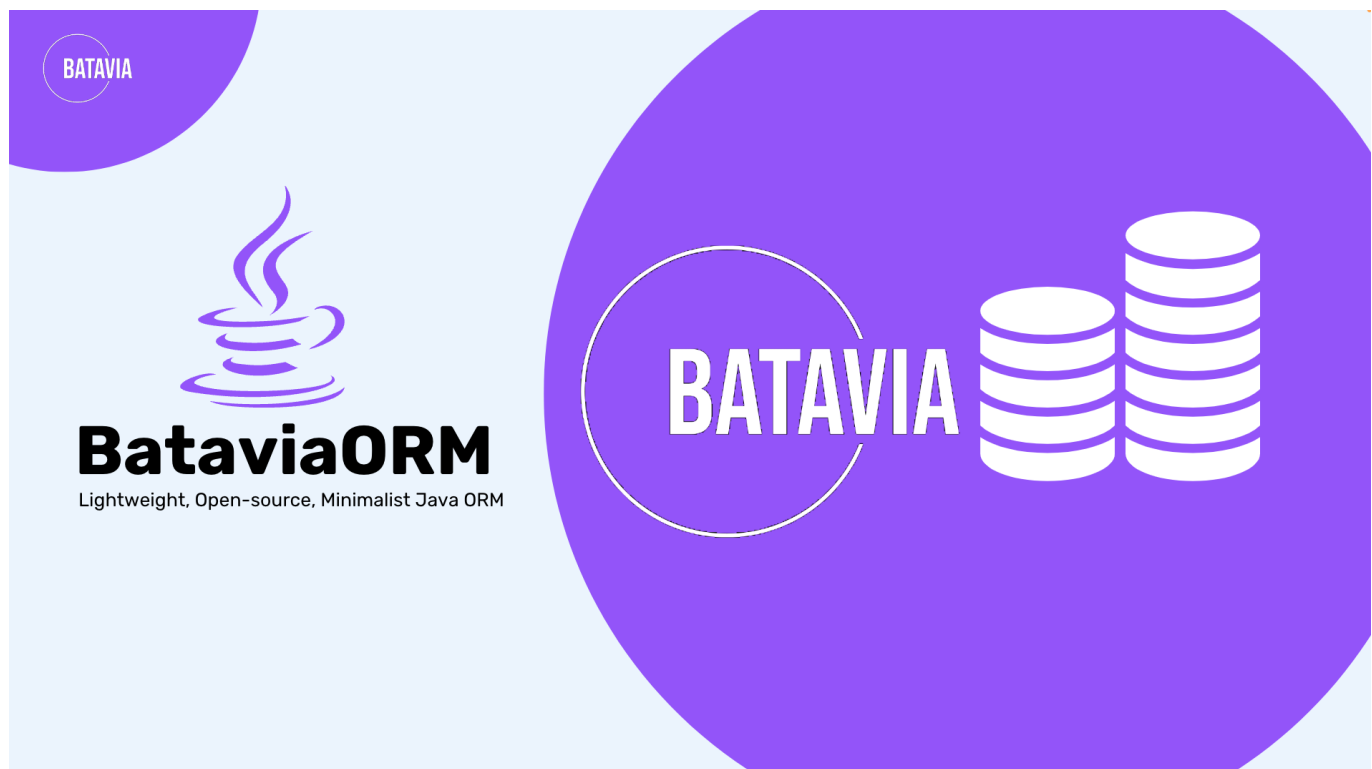


BataviaORM



BataviaORM is an innovative lightweight Java-to-SQL ORM (Object-Relational Mapper) framework, designed to revolutionize the way developers interact with databases. BataviaORM embraces a minimalist design philosophy, allowing developers to seamlessly map Java classes to SQL tables with minimal setup and configuration, and apply database schema changes without writing a single SQL code yourself. It includes 3 core functionalities; migration generator, migration runner, migration reverter.

- [Product Demo](#)
- [Installation](#)
- [Setup](#)
- [Usage](#)

Product Demo

Part 1:

<https://www.loom.com/share/69d60ff82224406fae9dab4c1d94b477?sid=074c83a8-25c6-4624-bdcb-875f8ea509bc>

Part 2:

<https://www.loom.com/share/13db5e2e4e784e778e1e6c12bcbe4dd0?sid=be6ec119-64a1-4de2-b02a-8b4c1a793296>

Installation

Install the intended Batavia ORM version JAR files from the [releases](#) folder in this repository or the [release page](#) (release summary on the bottom of this README). The installation will include 2 JAR files with the name:

`batavia-{version_number}-exec.jar` which is the executable JAR file for running the CLI

`batavia-{version_number}-package.jar` which is the package JAR file to be imported in your Java project

Setup

Copy / move JAR files to your project directory

Add both JAR files to the root directory of your Java project

```
├── src
│   ├── main
│   └── test
├── .env
├── batavia-{version_number}-exec.jar
├── batavia-{version_number}-package.jar
└── README.md
```

Set environment variables

In the `.env` file, set 3 necessary variables

- **DATABASE_URL**: the full JDBC Database URL (connection string).

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

- **MIGRATIONS_DIR**: the path to the migration directory
- **DATASOURCE_DIR**: the path to your Java schema/classes directory

For example, for the following directory structure:

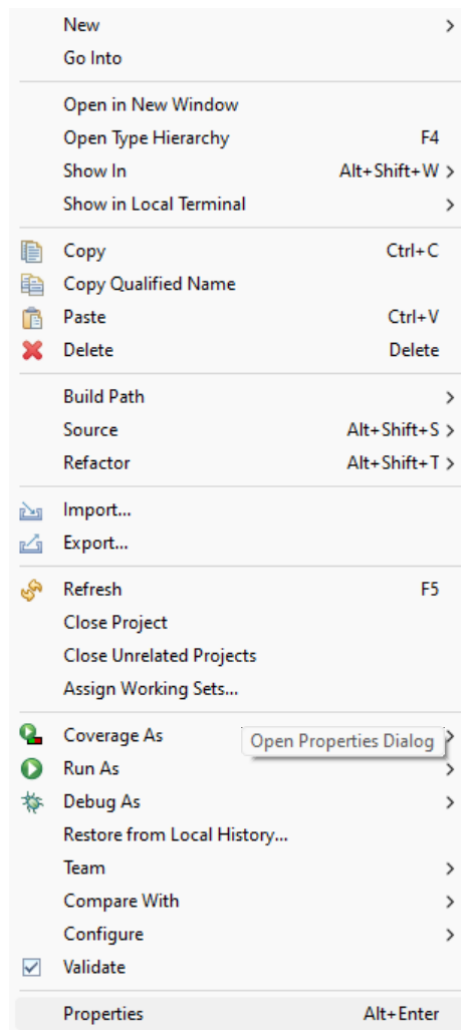
```
├── bin
│   └── pokerProject
│       └── Employee.class
├── migrations
│   ├── 2023-12-01_171223_automatic.down.sql
│   └── 2023-12-01_171223_automatic.sql
├── src
│   └── pokerProject
│       └── Employee.java
```

The complete `.env` file would look like the following:

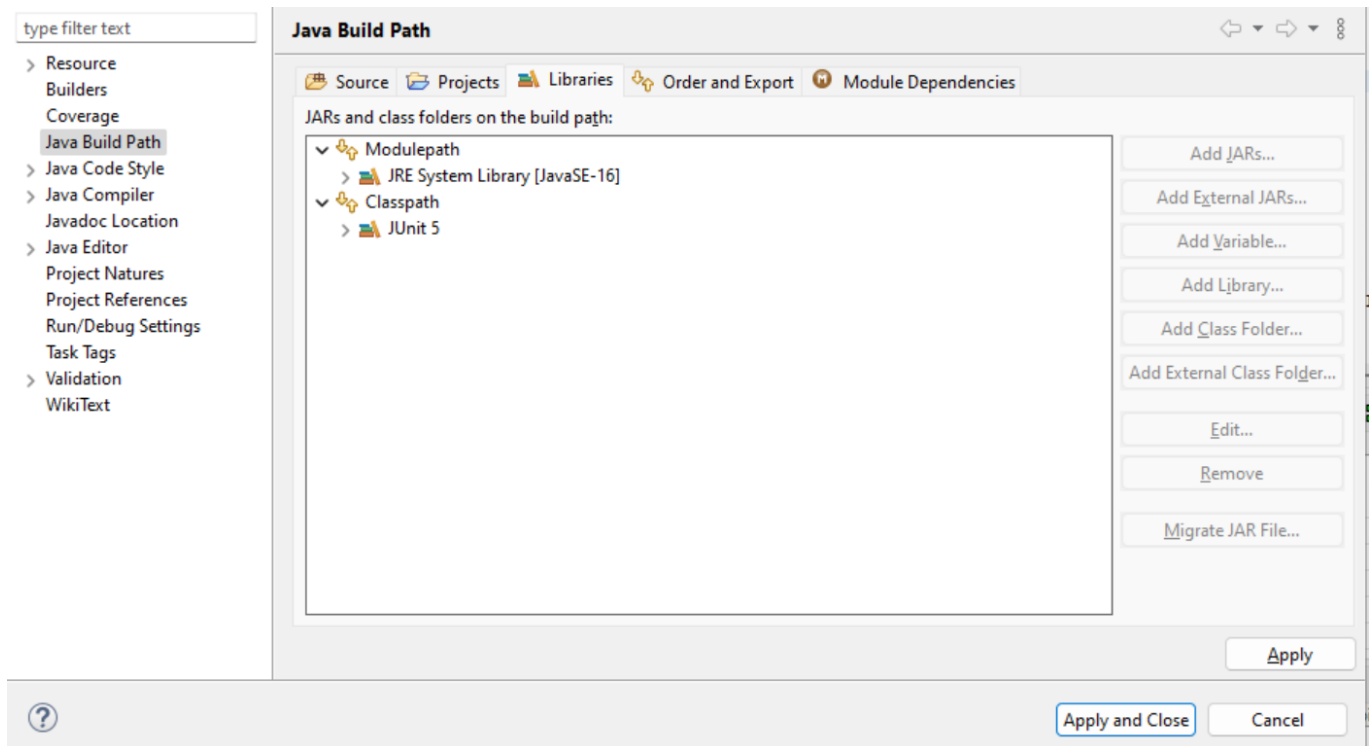
```
DATABASE_URL=jdbc:postgresql://host:port/database?properties
MIGRATIONS_DIR=migrations
DATASOURCE_DIR=src
```

Set up batavia build path in your Java project

In the eclipse package explorer, right click on your java project and click properties



Then, click on the **Java Build Path** tab and choose **Libraries** in the internal tab. Click on **modulepath** then click **Add JARs** and choose the **batavia-{version_number}-package.jar**. Last, click the **apply** and **close**.



You can now import the annotation classes from batavia as such to annotate your Java schema (explanation in the next section)

```
import com.batavia.orm.annotations.Entity;  
import com.batavia.orm.annotations.EntityColumn;  
import com.batavia.orm.annotations.PrimaryColumn;
```

Usage

Schema Annotations

There are 4 annotations that can be used to denote the properties of the object:

- `@Entity`: to denote SQL table
- `@EntityColumn`: to denote SQL table column
- `@PrimaryColumn`: to denote the primary column of a table
- `@Unique`: to denote a unique content in column

Complete Java schema file example:

```
package employeeProject;  
  
import com.batavia.orm.annotations.Entity;  
import com.batavia.orm.annotations.EntityColumn;  
import com.batavia.orm.annotations.PrimaryColumn;  
  
@Entity  
class Employee {
```

```
@EntityColumn
@PrimaryColumn
private String name;

@EntityColumn
private Integer age;

@EntityColumn
private Boolean married;

@EntityColumn
private Boolean religious;
}
```

CLI

There are 2 ways to run the CLI commands:

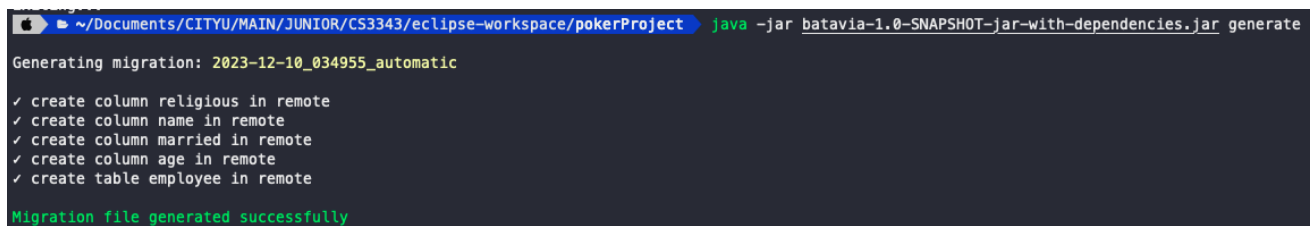
```
java -jar batavia-{version_number}-exec.jar
```

which will run the program in loop mode, accepting and executing the commands, or

```
java -jar batavia-{version_number}-exec.jar {COMMAND}
```

which will directly execute the inputted command. The commands include the following:

Generate



```
~/Documents/CITYU/MAIN/JUNIOR/CS3343/eclipse-workspace/pokerProject java -jar batavia-1.0-SNAPSHOT-jar-with-dependencies.jar generate
Generating migration: 2023-12-10_034955_automatic
✓ create column religious in remote
✓ create column name in remote
✓ create column married in remote
✓ create column age in remote
✓ create table employee in remote
Migration file generated successfully
```

Once you have made the changes to your Java schema/classes and are ready to capture the changes and generate the migration, there are 2 ways to run the generate command:

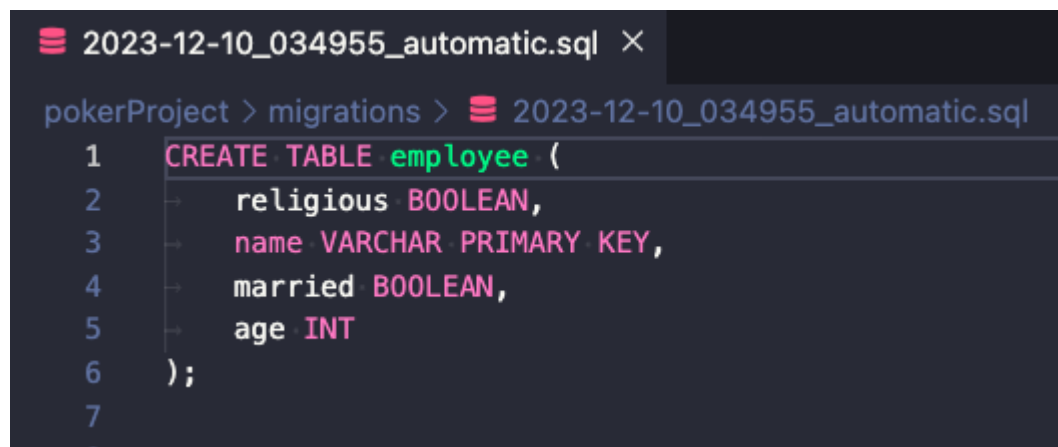
```
java -jar batavia-{version_number}-exec.jar generate
```

which will create a migration file with a time-stamped auto-generated file name (e.g. **2023-12-01_171223_automatic.sql**), or

```
java -jar batavia-{version_number}-exec.jar generate {file_name}
```

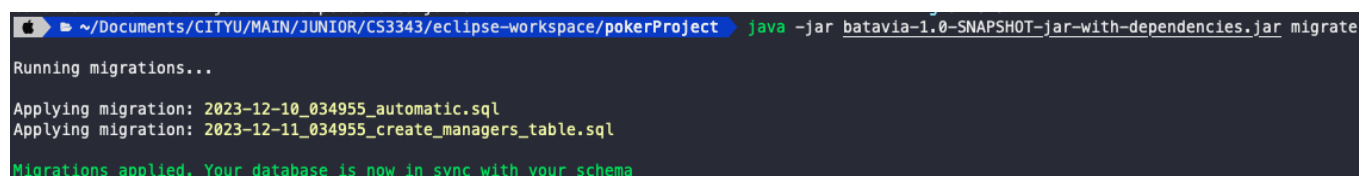
which will create a time-stamped migration file with the given file name (e.g. `2023-12-01_171223_create_employee_table.sql`)

Both commands will generate time-stamped filenames to maintain file uniqueness and sortedness in the migration folder. The generated files will include the down migration file that will be needed in case of reverts. The content of the migration files will be the necessary SQL statements (automatically generated) to apply the schema changes. Example:



```
2023-12-10_034955_automatic.sql X
pokerProject > migrations > 2023-12-10_034955_automatic.sql
1 CREATE TABLE employee (
2     religious BOOLEAN,
3     name VARCHAR PRIMARY KEY,
4     married BOOLEAN,
5     age INT
6 );
7
8
```

Migrate



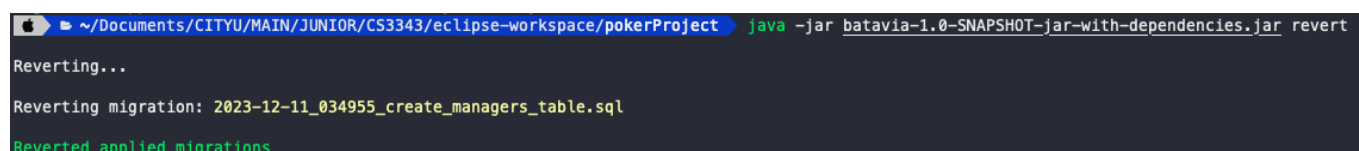
```
~/Documents/CITYU/MAIN/JUNIOR/CS3343/eclipse-workspace/pokerProject java -jar batavia-1.0-SNAPSHOT-jar-with-dependencies.jar migrate
Running migrations...
Applying migration: 2023-12-10_034955_automatic.sql
Applying migration: 2023-12-11_034955_create_managers_table.sql
Migrations applied. Your database is now in sync with your schema
```

Once the migration files are generated and you are ready to run the migration against the database, you can run the following:

```
java -jar batavia-{version_number}-exec.jar migrate
```

which will execute the migration and apply the schema changes to the remote database

Revert



```
~/Documents/CITYU/MAIN/JUNIOR/CS3343/eclipse-workspace/pokerProject java -jar batavia-1.0-SNAPSHOT-jar-with-dependencies.jar revert
Reverting...
Reverting migration: 2023-12-11_034955_create_managers_table.sql
Reverted applied migrations
```

If you intend to rollback / undo any changes applied by your previous migration(s) to the remote database, you can revert the migrations through the following commands:

```
java -jar batavia-{version_number}-exec.jar revert
```

which will revert the last migration applied to the database, you can also do:

```
java -jar batavia-{version_number}-exec.jar revert
{previous_migration_filename}
```

which will return the database schema state back to the specified migration by reverting all migrations up to the specified `previous_migration_filename`. Say, for instance, the migrations folder in your app is something like below with migration `0012_latest_migration.sql` being applied the most recently.

```
0010_previous_migration.sql
0011_next_migration.sql
0012_latest_migration.sql
```

If you want to go back to `0010_previous_migration.sql`, you can run the following:

```
java -jar batavia-{version_number}-exec.jar revert
0010_previous_migration.sql
```

The reverter will then revert `0012_latest_migration.sql` by executing `0012_latest_migration.down.sql` against the database, and subsequently reverting `0011_next_migration.sql` the same way. You can then delete the local files of the migrations that got reverted if you'd like to further generate a different migration to ensure migration consistency.

Acknowledgements

We express gratitude to our amazing brilliant team, comprising the following members:

- Vannes Wijaya, as Project Manager, who mainly built our Migration Runner & Migration Reverter
- Enryl Einhard, as Software Engineer, who mainly built our Comparator & Datasource Scanner
- Dannel Mulja, as Software Engineer, who mainly built our Comparator & Database Scanner
- Alvin Thosatria, as Software Engineer, who mainly built our Script Generator
- Cindy Falencia Irawan, as Software Engineer, who mainly built the CLI / interface of our app

Release Summary

Release	Date
Beta	20/11/2023
1.0	09/12/2023
1.1	10/12/2023

LTS release (1.1) (recommended version)

What's new :

- New colorful CLI UI
- Support for primary column annotations
- Revert with `{previous_migration_filename}` feature
- Type changes are now reflected in migration files

Bug fixes :

- Fixed revert migration FileNotFound error
- Resolved java.lang.NoClassDefFoundError when running .jar file
- Addressed Java to SQL data type mapping issue