# Example 15 - Database Transactions

This example shows how to use database transactions.

The DAOs for MyBatis operate by default in auto-commit mode. That is, every single insert, delete, and update commits its execution right away without need to specify it.

However, when transactions are required to group multiple database operations the DAOs provide a simple TxManager object that allows the code to initiate, commit, rollback, and free transaction resources.

Since behind the scenes MyBatis is controlling all transactions, many features of it are available such as locally controlled transactions, or participation in external transactions using JTA.

Transactions can also be linear or interleaved, the latter only available if supported by the database and JDBC driver.

## How to Run this example

The **Example 15** is included in the download package. To run this example please refer to the section How to Run the Examples above.

## Case #1: No transaction

This case implements a cash transfer between two branches, that requires to decrement the cash on one and increment it on the other one.

Both SQL updates are executed without any transaction declaration. Since by default HotRod works in auto-commit mode, these operations are independent and are committed immediately and separately. A system crash could produce inconsistencies in the database state.

## Case #2: Standard transaction

This case implements a cash transfer between two branches, that requires to decrement the cash on one and increment it on the other one.

Both SQL updates are executed within a transaction. These operations are not independent and are committed at once by the commit() method. A system crash will not produce inconsistencies in the database state.

## Case #3: Custom transaction

This case implements a cash transfer between two branches, that requires to decrement the cash on one and increment it on the other one.

The transaction raises the isolation level to SERIALIZABLE to prevent some level of concurrency in the database.

Both SQL updates are executed within a transaction. These operations are not independent and are committed at once by the commit() method. A system crash will not produce inconsistencies in the database state.

**Note**: Custom options on transaction may not be supported by all databases.

## Case #4: Interlaced transactions

This case implements TWO cash transfers between four branches IN PARALLEL, that requires to decrement the cash on two of them and increment it on the other two.

**Note**: Interlaced transactions are not common. You may want to use them on specialized cases when you really want some fine grained control over them. Even there, it's usually better to leave these intricacies to the data source pooling solution or application server.

**Note**: Interlaced transactions may not be supported by all databases.

All four SQL updates are executed intertwined in two SEPARATE independent transactions. A system crash will not produce inconsistencies in the database state. Also, a commit or rollback on one of the transactions does not produce a commit or rollback on the other one (they are fully independent).