# CRUD Operations on Tables

This example shows the basic out-of-the-box CRUD (create, read, update, delete) operations on a table of the database. This example shows how to operate on single rows identifying them by the PK (primary key) of the table; the PK is auto-generated as an identity key.

However, if the table did not have a PK, then only the insert operation would be available and the options to perform select, update, and delete would be more limited. We could still use the out-of-the-box select, update, and delete “by example” operations described on examples further on.

## DAO/VO class and its properties

For each table Hotrod generates a DAO class that combines the DAO pattern (persistence methods) with the VO pattern (value object that represents a row of a table).

This example uses the VEHICLE table to demonstrate the basic CRUD operations. HotRod created a DAO class VehicleDAO for this table. Its name is configurable but defaults to the table name with the DAO suffix.

For each table column this class includes a property with a name and a specific Java type. Both are configurable: the name defaults to the column name in Java format, an the Java type defaults to a suitable Java type for each column. This Java type heavily depends on the specifics of the RDBMS.

## How to Run this example

To run this example please refer to the section How to Run the Examples above.

## Insert

In this example, the section below shows how to insert a new row in the table VEHICLE:

VehicleDAO skoda = **new** VehicleDAO();

skoda.setBrand("Skoda");

skoda.setModel("Octavia");

skoda.setUsed(**false**);

skoda.setCurrentMileage(7);

skoda.setPurchasedOn(**new** Date(System.*currentTimeMillis*()));

**int** rows = skoda.insert();

System.***out***.println("New vehicle Skoda added. New ID=" + skoda.getId()

+ ". Rows inserted=" + rows);

The insert() method is always available, regardless if the table has or has not a PK: you can always insert.

It first creates an object of the class VehicleDAO, then it set the property values for the row we want to insert, and finally it performs the insert() operation.

There are a few things to note here:

* This example does not include any transaction demarcation (begin transaction and commit transaction). When no transaction demarcation is specified, each database operation commits automatically.
* The PK property id was not set before inserting the row. Since HotRod was configured to treat this column as auto-generated its value is ignored while inserting and is populated right after the insert. In this case, the PK id has the value of the newly inserted PK.
* The insert() method returns the number of affected rows.
* If the insert() operation fails by any reason an Exception is thrown. By default the checked SQLException is thrown, but can be also configured to throw the unchecked Mybatis PersistenceException. See the MyBatis Generator Options section.

## Select by PK

In this example, the section below shows how to select a row using the PK.

VehicleDAO delorean = VehicleDAO.*select*(3);

If the table has a PK, the DAO includes the *select*() method that retrieves a row by a PK value. This method is not present if the table does not have a PK.

There are a few things to note here:

* The *select*() method is static, so it's used from the DAO class itself.
* The *select*() method parameters are the PK values. For a composite PK, multiple parameters are used as needed.
* The *select*() method returns a single DAO object. If no row is found for the specified PK a null is returned.
* If the *select*() operation fails by any reason an Exception is thrown. By default the checked SQLException is thrown, but can be also configured to throw the unchecked Mybatis PersistenceException. See the MyBatis Generator Options section.

## Update by PK

In this example, the section below shows how to update a row using the PK.

VehicleDAO delorean = VehicleDAO.*select*(3);

// Update by PK: set the mileage of the DeLorean to 270500

**if** (delorean != **null**) {

delorean.setCurrentMileage(270500);

rows = delorean.update();

System.***out***.println("DeLorean updated. Rows updated=" + rows);

} **else** {

System.***out***.println("Could not update DeLorean. Car not found.");

}

If the table has a PK, the DAO includes the update() method that updates a row by the PK value. This method is not present if the table does not have a PK. The key section of this example are the two lines:

delorean.setCurrentMileage(270500);

rows = delorean.update();

This example retrieved a row using a Select by PK method (example above), then modified one property, and finally updated the row.

There are a few things to note here:

* The update() method is not static. It operates on an object, not the class itself. All the non-PK columns (properties) are updated into the database row, including the unmodified properties.
* In this example, since some time passes between the *select*() and update() operations, it's technically possible that the table row could have been updated in the mean time by another process or thread. If so, the update() operation will override the other process' values, effectively losing that information. To deal with this problem, you can use the pessimistic locking strategy (row locks or table locks), or use optimistic locking strategy (Row Version Control). The latter is implemented by HotRod and is described later in the section Optimistic Locking (Row Version Control).
* This example does not include any transaction demarcation (begin transaction and commit transaction). When no transaction demarcation is specified, each database operation commits automatically.
* The update() method returns the number of affected rows. If there's no row for the specified PK values, it returns a zero.
* If the update() operation fails by any reason an Exception is thrown. By default the checked SQLException is thrown, but can be also configured to throw the unchecked Mybatis PersistenceException. See the MyBatis Generator Options section.

## Delete By PK

In this example, the section below shows how to delete a row using the PK.

VehicleDAO toyota = **new** VehicleDAO();

toyota.setId(2);

rows = toyota.delete();

If the table has a PK, the DAO includes the delete() method that deleted a row by the PK value. This method is not present if the table does not have a PK.

It first uses an object of the class VehicleDAO, the sets the PK columns (one or more), and finally deletes the row.

There are a few things to note here:

* The delete() method is not static. It operates on an object, not the class itself. Only the PK columns are used. All other columns are ignored.
* This example does not include any transaction demarcation (begin transaction and commit transaction). When no transaction demarcation is specified, each database operation commits automatically.
* The delete() method returns the number of affected rows. If there's no row for the specified PK values, it returns a zero.
* If the delete() operation fails by any reason an Exception is thrown. By default the checked SQLException is thrown, but can be also configured to throw the unchecked Mybatis PersistenceException. See the MyBatis Generator Options section.