HotRod Cheat Sheet

Select by Primary Key

```
UserVO u = this.userDAO.select(17);
```

Available in tables with a PK

Insert

```
UserVO u = new UserVO();
u.setUsername("jsmith");
u.setFirstName("John");
u.setLastName("Smith");
u.setGroupId(101);
u.setState(1);
this.userDAO.insert(u);
```

Available in tables and views

Update by Primary Key

```
UserVO u = UserVO.select(17);
u.setFirstName("Jamie");
this.userDAO.update(u);
```

Available on tables with a PK

Delete by Primary Key

```
UserVO u = new UserVO();
u.setId(17);
this.userDAO.delete(u);
```

Available in tables with a PK

Select by Unique Constraint

```
UserVO u = this.userDAO
  .selectByUIUsername("jsmith");
```

Available in tables with unique constraints

Insert with Auto-generated PK

```
// Sequence
GroupVO g = new GroupVO();
g.setName("admin");
this.groupDAO.insert(g);
System.out.println("id=" + g.getId());
// shows the new id value
```

```
// Identity Generated Always
GroupVO g = new GroupVO();
g.setName("admin");
this.groupDAO.insert(g);
System.out.println("id=" + g.getId());
// shows the new id value
```

```
// Identity Generated By Default
GroupVO g = new GroupVO();
g.setId(123); // explicit PK
g.setName("admin");
this.groupDAO.insert(g);
System.out.println("id=" + g.getId());
// shows the value 123
```

Available in tables with PK auto-generation

Select by Example

```
UserVO example = new UserVO();
example.setState(1); // Get active users
List<UserVO> users = this.userDAO
  .selectByExample(example);
```

Available in tables and views

Select by Example with Cursor

```
UserVO example = new UserVO();
example.setState(1); // Get active users
Cursor<UserVO> users = this.userDAO
  .selectByExampleCursor(example);
```

Available in tables and views

Select by Example with Ordering

```
UserVO example = new UserVO();
example.setState(1); // Get active users
List<UserVO> users = this.userDAO
  .selectByExample(example,
     UserOrderBy.LAST_NAME,
     UserOrderBy.FIRST_NAME);
```

Available in tables and views

Update by Example

```
// Deactivate all active users of group 78
UserVO example = new UserVO();
example.setState(1); // active
example.setGroup(78);
UserVO newValues = new UserVO();
newValues.setState(0); // inactive
this.userDAO.updateByExample(
  example, newValues);
```

Available in tables and views

Delete by Example

```
// Delete all inactive users
UserVO example = new UserVO();
example.setState(0); // inactive
this.userDAO.deleteByExample(example);
```

Available in tables and views

Select Sequence Value

```
// Get the value of sequence user_seq
```

- In the configuration file:

```
<table name="user">
  <sequence method="getUserSeq"
            name="user_seq" />
</table>
```

- In the java application:

```
long value = this.userDAO
  .getUserSeq();
```

Can be added to table, view, and dao tags.

Select Parent Row by FK

```
UserVO u = this.userDAO.select(17);
GroupVO g = this.userDAO
  .selectParentGroup(u)
  .fromGroupId()
  .toId();
```

Available in tables with imported FKs

Select Children Rows by FK

```
UserVO u = this.userDAO.select(17);
List<PrivilegeVO> privs = this.userDAO
  .selectChildrenPrivilege(u)
  .fromId();
  .toUserId();
```

Available in tables with exported FKs

Reflexive Select Parent by FK

```
UserVO u = this.userDAO.select(17);
UserVO creator = this.userDAO
  .selectParentUser(u)
  .fromCreatedBy()
  .toId();
```

Available in tables with reflexive foreign keys

Reflexive Select Children by FK

```
UserVO u = this.userDAO.select(17);
List<UserVO> createdUsers = this.userDAO
  .selectChildrenUser(u)
  .fromId()
  .toCreatedBy();
```

Available in tables with reflexive foreign keys

Update with Optimistic Lock

```
UserVO u = this.userDAO.select(17); // #1
u.setGroupId(102);
try {
  this.userDAO.update(u); // #2
  // Successfully updated
} catch (SQLException e) {
  // Row had been updated/deleted by
  // other process between steps #1 and #2
}
```

Available in tables with optimistic locking enabled

Delete with Optimistic Lock

```
UserVO u = this.userDAO.select(17); // #1
u.setGroupId(102);
try {
  this.userDAO.delete(u); // #2
  // Successfully deleted
} catch (SQLException e) {
  // Row had been updated/deleted by
  // other process between steps #1 and #2
}
```

Available in tables with optimistic locking enabled

Nitro Select Query

```
// Get all privileges set by a user for all
// active users on a specific user group.
```

- In the configuration file:

```
<dao name="PrivilegesDAO">
  <select method="getGroupPrivileges"
          vo="ActivePrivilege">
    <parameter name="group"
               java-type="Integer" />
    <parameter name="createdBy"
               java-type="Long" />
    select u.id as uid, u.username, p.*
    from privilege p
    join user u on u.id = p.user_id
    where u.state = 1
      and u.group = #{group}
      <if name="createdBy != null" />
        and p.created_by = #{createdBy}
      </if/>
  </select>
</dao>
```

- In the java application:

```
List<ActivePrivilege> privs =
  this.privilegesDAO
    .getGroupPrivileges(123, 5);
```

Defines a new DAO method. Native SQL and Dynamic SQL can be combined in the query

Nitro Select Query with Cursor

- In the configuration file:

```
<dao name="PrivilegesDAO">
  <select method="getGroupPrivileges"
          vo="ActivePrivilege"
          mode="cursor">
    ... query definition...
  </select>
</dao>
```

- In the java application:

```
Cursor<ActivePrivilege> privs =
  this.privilegesDAO
    .getGroupPrivileges(123, 5);
```

Defines a new DAO method. Native SQL and Dynamic SQL can be combined in the query

Nitro General Query

```
// Mark as non-outstanding all invoices
// with payments that exceed the balance
```

- In the configuration file:

```
<dao name="InvoicesDAO">
  <query method="closePaidInvoices">
    <parameter name="branchId"
               java-type="Integer" />
    update invoice
    set outstanding = 0
    where payment >= balance
    <if name="branchId != null" />
      and branch_id = #{branchId}
    </if/>
  </query>
</dao>
```

- In the java application:

```
int rows = this.invoicesDAO
  .closePaidInvoices(5072);
```

Defines a new DAO method. Native SQL and Dynamic SQL can be combined in the query

Example Database