**Rules for AN DB changes to avoid rollback**

**Why we need the rule?**

Sometimes we release failed and need rollback code to old version, but we have no way to rollback DB schema changes (or custom data changes), so we need follow the rules to make sure the new version DB be compatible with the old version code.

**What’s the rule?**

1. **DB schema changes:**

|  |  |  |
| --- | --- | --- |
| **Rule** | **Example** | **Correct Action** |
| Remove table/column which was added in a **released** version | “tableA” added in 15.3 (released), remove “tableA” in 15.3.1 | 1. Keep the table/column 2. Remove related code. |
| Alter table name/column name which was added in a **released** version | “tableA” added in 15.3 (released), alter “tableA” to “tableB” in 15.3.1 | 1. Keep the table/column 2. Remove related code. |
| Decrease column data size | data type of “columnA” is VARCHAR, column size is 20, need change the column size to 10 | Don’t decrease it |
| Change column data type | data type of “columnA” is VARCHAR, need change the column data type to INT | 1. Keep the column 2. Add another column 3. Copy data to new column, process for data type convention (code in DBUpdateScript) |

**Notes:**

1. The rule will only apply for DB schema changes, not code. For example, you add a wrong column and need delete it. Please keep that column in DB (follow the rule), and delete it from code (remove relate sql script hard code, remove filed in Class… …)
2. **Custom Update changes (DBUpdateScript.java):**

|  |  |
| --- | --- |
| **Rule** | **Example** |
| When we branch the code and begin work on a new major release, we leave a gap of 10 to the first custom update of the next major release. This allows space for one or more database updates in maintenance releases on the branch, such as for data repair. | When start 15.1.1, we should level 10 version for 15.1 for maintenance  // Leave a gap for maintenance branch  **if** (current\_level >= 706 && current\_level < 716) {  *setDBLevel*(dbc, current\_level = 716, "Gap for 15.1 maintenance");  } |
| when any custom updates are done in the branched code, it is necessary to analyze the custom update to determine if it is also necessary to do the same operation  at a different database level in the trunk |  |
| Custom Updates **must be safely rerunnable** | // Go from 347 to 348  **if** (current\_level == 347) {  // Update fields in facilities table  String sql = "update " + dbc.table(Facility.***sql\_table\_name***)  + " set allow\_use\_by\_public=" + dbc.put(**true**)  + " where item\_type=" + dbc.put(Facility.***enm\_item\_type\_equipment***);  dbc.executeUpdate(sql);  // We're now at level 348  *setDBLevel*(dbc, ++current\_level, "initialize allow\_use\_by\_public for facilities table");  } |
| If an update is expected to take a long time, it should be implemented as an asynchronous update, which are processed on a background thread in the class RaiseDBLevelThread |  |