Database Tables 2

Defining a database table

Business object instances are typically java object representations of rows of a database table.

The addition of following columns to each database table is ***strongly*** *suggested*:

- Object ID
- Version number

Object ID

The Object ID is used as a globally unique identifier of each row across all database tables (see caveat below). That is, every row in every table should have a different object value. It is typically defined as a VARCHAR field of 36 characters, and should be named "OBJ_ID". A unique constraint should be applied to the object ID column, but **must NOT** be part of the primary key.

The object ID value is automatically stored by the framework and/or the database layer.

About the object ID
The value of the object ID is a Globally Unique Identifier (GUID). It is extremely likely that each row will have a unique value, and for practical purposes, we assume that each row will have a unique object ID.

Version number

KFS/Rice uses optimistic locking to provide concurrency control. Optimistic locking requires the use of a version number field, named "VER_NBR". On Oracle, the field is defined as a `NUMBER(8,0)`. On MySQL, the field is defined as a `DECIMAL(8)`. This column should **NOT** be part of the primary key.

About optimistic locking
Optimistic locking helps to prevent updates to stale data and consists of two steps:

1. Retrieval of a row from a database, including the value of the version number column
2. Updating/deleting a row from the database with the same primary key and version number criteria. If updating the table, the version number will be incremented by one.

The following series of steps demonstrates how optimistic locking works:

1. User A retrieves the row for chart code "BL". Assume that the row has version number of 3.
2. User A performs a update of the "BL" record. The SQL query that updates the record would read something like "UPDATE CA_CHART_T SET <some updates>, VER_NBR = 4 WHERE FIN_COA_CD = "BL" and VER_NBR = 3. (The "4" refers to the incremented version number.)
3. User B retrieves the row for chart code "BL". The version number is now 4.
4. User B performs a update of the "BL" record. The SQL query that updates the record would read something like "UPDATE CA_CHART_T SET <some updates>, VER_NBR = 5 WHERE FIN_COA_CD = "BL" and VER_NBR = 4. (The "5" refers to the incremented version number.)

The following series of steps demonstrates how optimistic locking prevents concurrency problems.

1. User A retrieves the row for chart code "BL". Assume that the row has version number of 3.
2. User B retrieves the row for chart code "BL". Like user A, the version number is 3.
3. User A performs a update of the "BL" record. The SQL query that updates the record would read something like "UPDATE CA_CHART_T SET <some updates>, VER_NBR = 4 WHERE FIN_COA_CD = "BL" and VER_NBR = 3. (The "4" refers to the incremented version number.)
4. User B performs a update of the "BL" record. The SQL query that updates the record would read something like what User A executed above (notice the version numbers). However, the previous step already updated the version number to 4 from 3, so this update does nothing (i.e. update row count = 0) because it was trying to update the BL chart with a version number of **3**. The system detects the 0 update row count, and throws an OptimisticLockingException. This exception indicates that the system tried to update stale data.

Example Account table definition

Below is the definition of the simplified account table.

```
CREATE TABLE CA_ACCOUNT_T
(
      FIN_COA_CD VARCHAR(2),
      ACCOUNT_NBR VARCHAR(7),
      OBJ_ID VARCHAR(36) NOT NULL,
      VER_NBR DECIMAL(8) default 1 NOT NULL,
      ACCOUNT_NM VARCHAR(40),
      ACCT_FSC_OFC_UID VARCHAR(10),
      CONSTRAINT CA_ACCOUNT_TP1 PRIMARY KEY(FIN_COA_CD,ACCOUNT_NBR),
      CONSTRAINT CA_ACCOUNT_TC0 UNIQUE (OBJ_ID)
)

-- the chart code in the account table comes from the chart table
-- each account row refers to exactly one row in the chart table
ALTER TABLE CA_ACCOUNT_T
    ADD CONSTRAINT CA_ACCOUNT_TR1
    FOREIGN KEY (FIN_COA_CD)
    REFERENCES CA_CHART_T (FIN_COA_CD)

-- this is one of the foreign keys for the sub account table.
-- Each account may have 0 to many sub-accounts
ALTER TABLE CA_SUB_ACCT_T
    ADD CONSTRAINT CA_SUB_ACCT_TR1
    FOREIGN KEY (FIN_COA_CD, ACCOUNT_NBR)
    REFERENCES CA_ACCOUNT_T (FIN_COA_CD, ACCOUNT_NBR)

-- note that even though there is a column for a universal user ID field in the
account table (i.e. ACCT_FSC_OFC_UID),
-- there is no foreign key to FP_UNIVERSAL_USR_T.  This relationship will be mapped in
a data dictionary based relationship.
```