AUTOMATING Z/OS DB2 CHANGES WITH

REXX AND FLYWAY

René Vincent Jansen

28th International Rexx Language Symposium, April 2017

GIT
JENKINS
FLYWAY

Oracle
SQL Server
SQL Azure
DB2
DB2 z/OS
MySQL
MariaDB
PostgreSQL
Redshift
Vertica
EnterpriseDB
H2
Hsql
Derby
SQLite
SAP HANA
solidDB
Sybase ASE
Phoenix

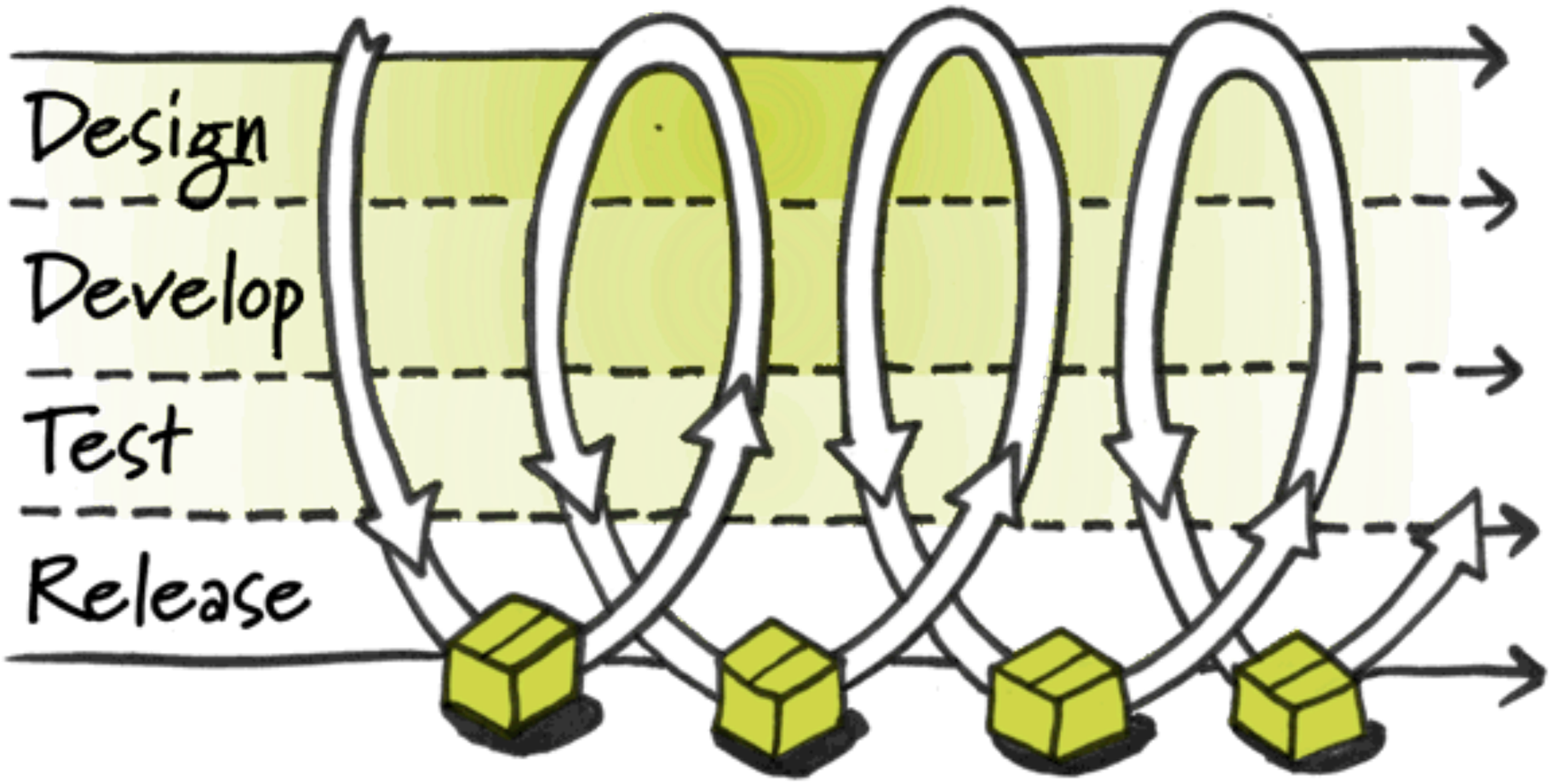
The Open Source Deployment Pipeline

AGENDA

- The deployment pipeline
- Version management for DB2 objects
- Using Flyway on the command line
- Automating the process with NetRexx
- Using SQL based conversions
- Using Java based conversions
- Using DB2 z/OS Utilities

THE DEPLOYMENT PIPELINE

THE DEPLOYMENT PIPELINE



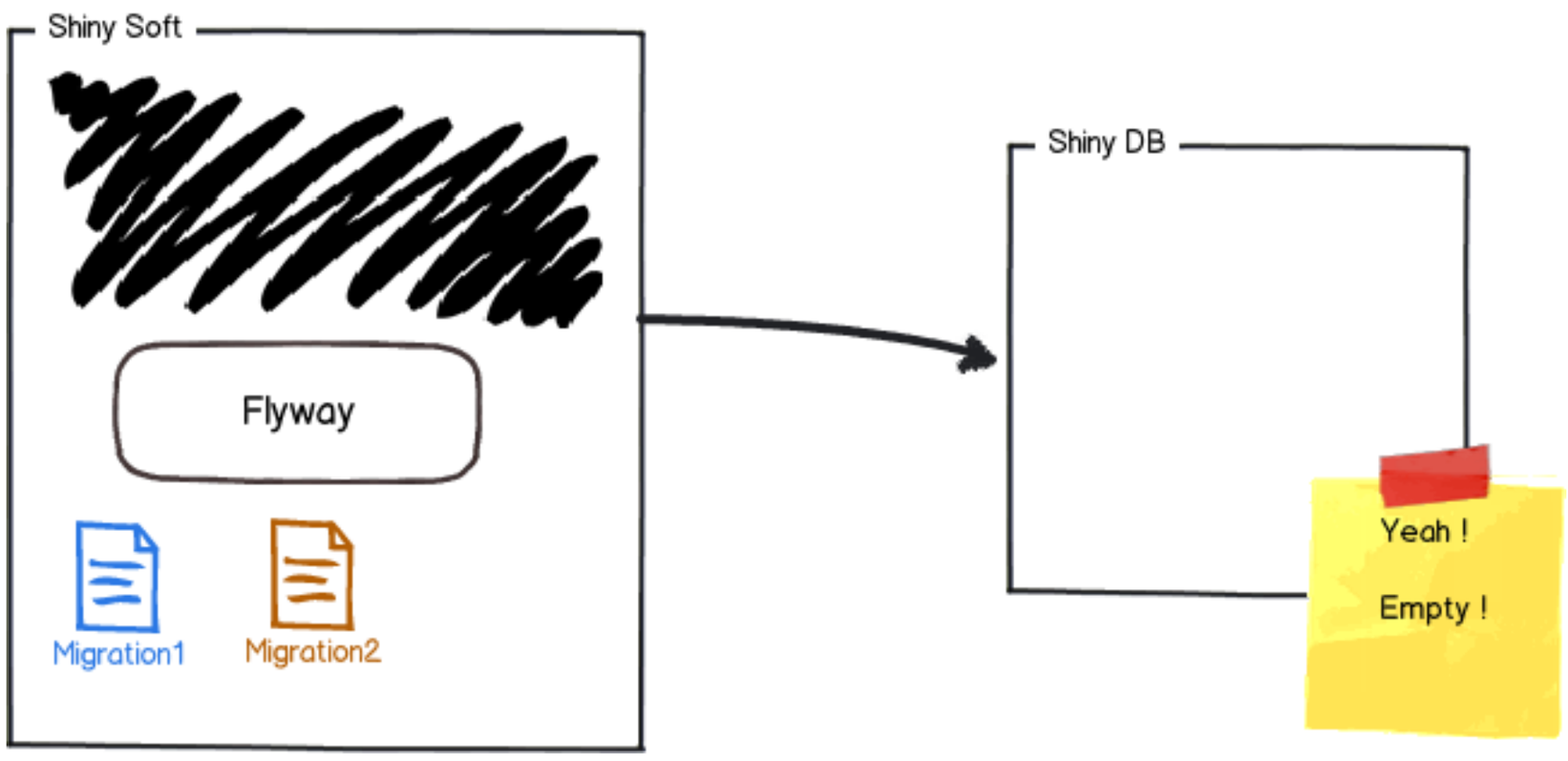
VERSION MANAGEMENT FOR DB2 OBJECTS

THE PROBLEM WITH DATABASES

- The problem with databases is that there is DATA in them
- In Dev, Test and Acc you can skirt this problem
- But in PROD you better keep the data and don't lose it
- If the table structure changes, you need to either:
  - **alter** and **reorg**
  - **unload**, **drop**, **define**, **reload**, put back **auth** and **FK**'s
  - a copy table, an "insert into … from select" and a copy back
  - the crossloader, that finally can handle most trouble

HOW DOES IT WORK

The easiest scenario is when you point Flyway to an empty database.



It will try to locate its metadata table. As the database is empty, Flyway won't find it and will create it instead.

WELL, NOT ALWAYS

- At this site, one may not use BP0 (only for DB2 catalog)
- BP0 (Buffer Pool Zero) is hardcoded in the definition
- No worries, we define it ourselves
- You cannot do this with flyway until you have modified flyway - this pays off the moment you have to do more databases, so invest in this small modification

CREATE TABLESPACE

Flyway will create this table when it does not find it. You only have to do this, once for every schema, if automatic creation fails

```
SET CURRENT SQLID="JANSR16";

CREATE TABLESPACE SFLYWAY
  IN JANSR16
  USING STOGROUP SGDB0O
  PRIQTY -1 SECQTY -1
  ERASE  NO
  FREEPAGE 0 PCTFREE 10
  GBPCACHE CHANGED
  TRACKMOD NO
MAXPARTITIONS 4
LOGGED
DSSIZE 4 G
SEGSIZE 32
BUFFERPOOL BP1
LOCKSIZE ANY
LOCKMAX 0
CLOSE YES
COMPRESS YES
CCSID      UNICODE
DEFINE YES
MAXROWS 255;
```

CREATE TABLE

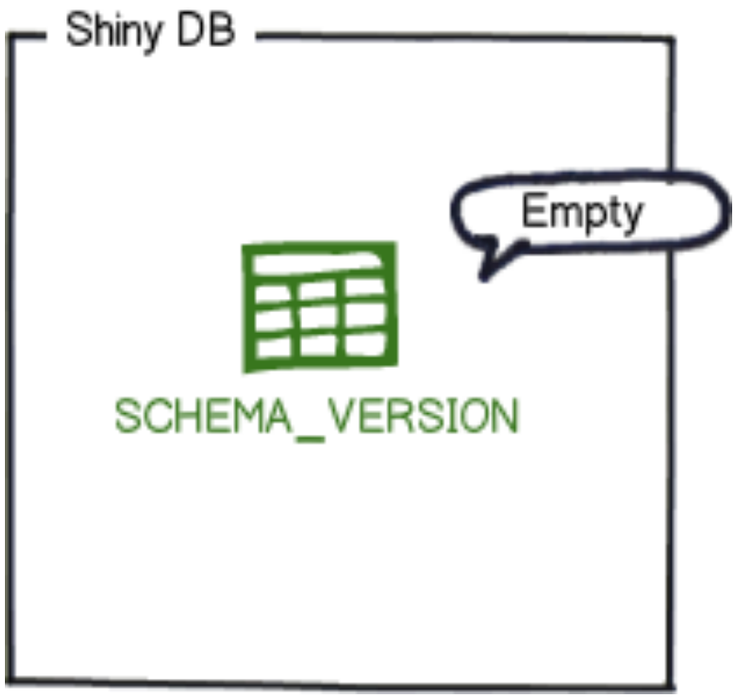
```
CREATE TABLE schema_version (
    "installed_rank" INT NOT NULL,
    "version" VARCHAR(50),
    "description" VARCHAR(200) NOT NULL,
    "type" VARCHAR(20) NOT NULL,
    "script" VARCHAR(1000) NOT NULL,
    "checksum" INT,
    "installed_by" VARCHAR(100) NOT NULL,
    "installed_on" TIMESTAMP NOT NULL WITH DEFAULT,
    "execution_time" INT NOT NULL,
    "success" SMALLINT NOT NULL
)
IN JANSR16.SFLYWAY;
```

YOU CAN ALSO ADAPT FLYWAY TO YOUR (CUSTOMERS) SITE

- It is open source
- Git clone it from
  - git clone https://github.com/flyway/flyway.git
- Build it with Maven (will download the internet first time)
- This definition file (for DB2 z/OS) is in the jar at:
  - org/flywaydb/core/internal/dbsupport/db2zos/createMetaDataTable.sql
- I was lazy/efficient and just used zip to add the mod to the jar

HOW DOES IT WORK (WITH SCHEMA_VERSION DEFINED)

You now have a database with a single empty table called SCHEMA_VERSION by default:

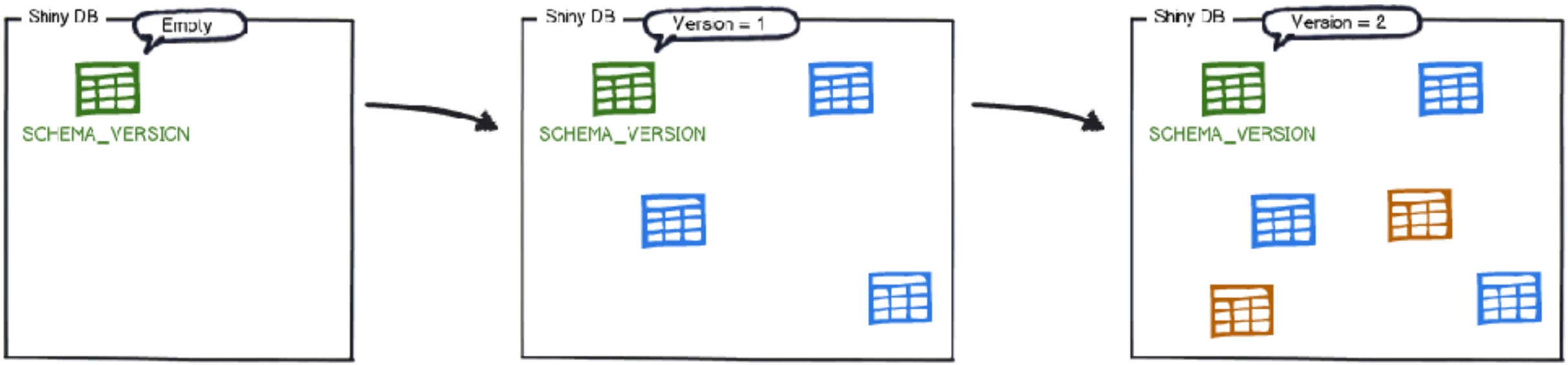


This table will be used to track the state of the database.

HOW DOES IT WORK

Immediately afterwards Flyway will begin scanning the filesystem or the classpath of the application for migrations. They can be written in either Sql or Java.

The migrations are then sorted based on their version number and applied in order:



As each migration gets applied, the metadata table is updated accordingly:

installed_rank	version	description	type	script	checksum	installed_by	installed_on	execution_time	success
1	1	Initial Setup	SQL	V1__Initial_Setup.sql	1996767037	axel	2016-02-04 22:23:00.0	546	TRUE
2	2	First Changes	SQL	V2__First_Changes.sql	1279644856	axel	2016-02-06 09:18:00.0	127	TRUE

THE NAMING SCHEME

- Out of the box, Flyways uses the convention
  - V1.0__TableDefinition.sql
  - V1.1__Add_index.sql
  - V1.2__Drop_Recreate_and_Reload.sql
  - V1.n__etcetera_ad_infinitum

V1.0__TABLE_DEFINITION.SQL

```
SET CURRENT SQLID='JANSR16';

CREATE TABLESPACE JANTST
  IN JANSR16
  USING STOGROUP SGDB00
  PRIQTY -1 SECQTY -1
  ERASE  NO
  FREEPAGE 0 PCTFREE 10
  GBPCACHE CHANGED
  TRACKMOD NO
  MAXPARTITIONS 4
  LOGGED
  DSSIZE 4 G
  SEGSIZE 32
  BUFFERPOOL BP1
  LOCKSIZE ANY
  LOCKMAX 0
  CLOSE YES
  COMPRESS YES
  CCSID      UNICODE
  DEFINE YES
  MAXROWS 255;
```

```
CREATE TABLE TSTFLYWAY
     (ID                    BIGINT NOT NULL,
      AUTHORIZED            VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
      DISABLED              VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
      ELEMENT               VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
      ENTEREDAT             TIMESTAMP (6) WITHOUT TIME ZONE
        WITH DEFAULT NULL,
      FORPROFILE_ID         BIGINT WITH DEFAULT NULL,
      FORUSER_ID            BIGINT WITH DEFAULT NULL,
      FUNCTIONALITY         VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
      LASTMODIFIEDAT        TIMESTAMP (6) WITHOUT TIME ZONE
        WITH DEFAULT NULL,
      NAME                  VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
        TARGET                VARCHAR(255) FOR MIXED DATA
        WITH DEFAULT NULL,
      CONSTRAINT DATAACCESS_PK
      PRIMARY KEY (ID))
IN JANSR16.JANTST
PARTITION BY SIZE
AUDIT NONE
DATA CAPTURE CHANGES
CCSID      UNICODE
NOT VOLATILE
APPEND NO  ;
```

V1.1__ADD_INDEX.SQL

```
SET CURRENT SQLID='JANSR16';

  CREATE UNIQUE INDEX X1TSTFLW
  ON TSTFLYWAY
   (ID                      ASC)
  USING STOGROUP SGDB0O
  PRIQTY -1 SECQTY -1
  ERASE  NO
  FREEPAGE 0 PCTFREE 10
  GBPCACHE CHANGED
  CLUSTER
  COMPRESS NO
  INCLUDE NULL KEYS
  BUFFERPOOL BP2
  CLOSE YES
  COPY NO
  DEFER NO
  DEFINE YES
  PIECESIZE 2 G;
```

Forgot the Primary Key index, you won't get that for free on z/OS DB2

So there we have our first update/migration

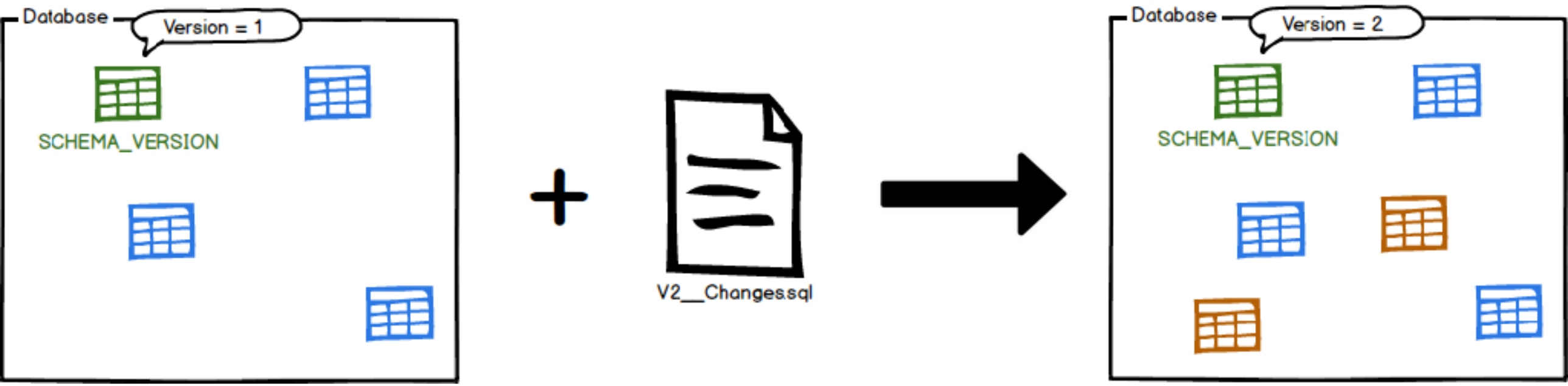
AUTO-MIGRATION ON STARTUP

- It is possible (and recommended) to have your application check at startup if it speaks to the right database level
- There is an API for that
- More about that later

USING FLYWAY ON THE COMMAND LINE

THE MIGRATE COMMAND

$ flyway migrate



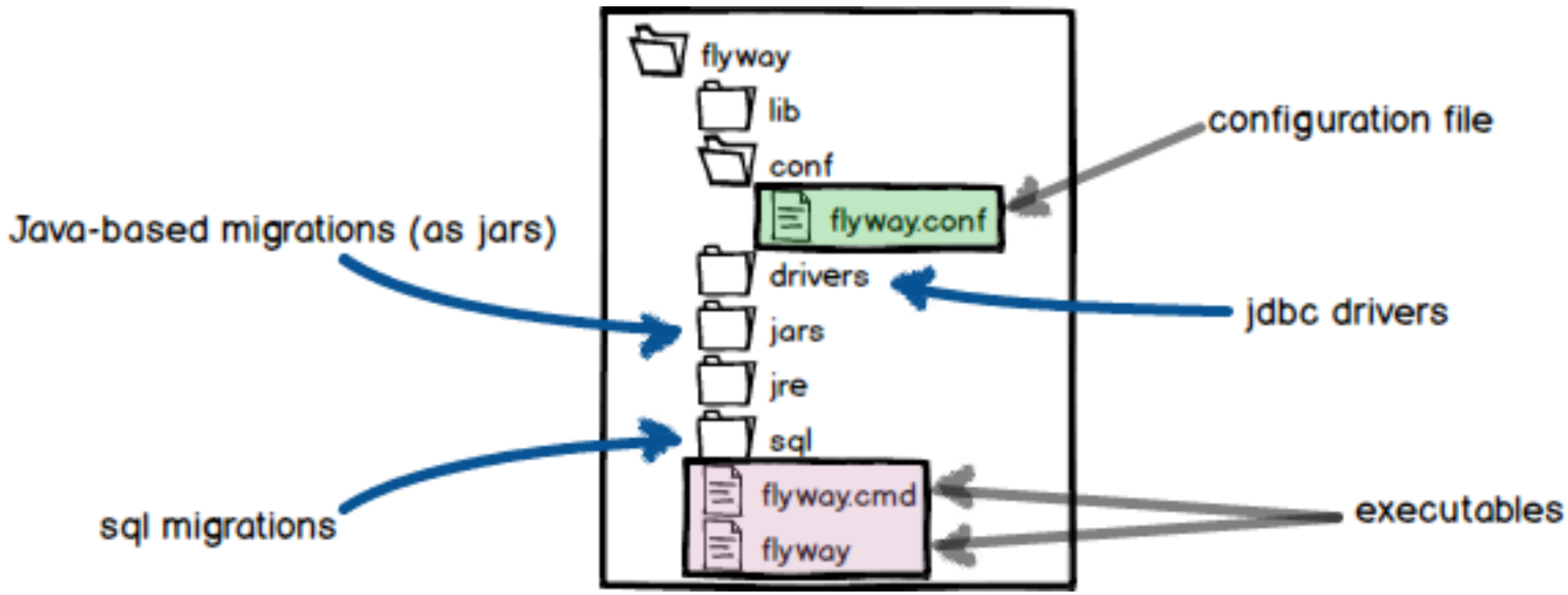
Example 1: We have migrations available up to version 9, and the database is at version 5.

Migrate will apply the migrations 6, 7, 8 and 9 in order.

Example 2: We have migrations available up to version 9, and the database is at version 9.

Migrate does nothing.

WHERE IT FINDS WHAT – FOR COMMAND LINE USAGE



PARAMETERS

- In the mainframe world, a DB2 subsystems contains numerous databases
- Generally, naming conventions are used to separate concerns: for database, stogroup, buffer pools, and authorisations
- Also, different DTAP environments have different dimensioning: PRIQTY, SEQTY, LOCKSIZE, LOCKMAX
- BUT you want to keep one copy of DDL, DCL, DML in version management
- yes, you do.

PARAMETERS

- The solution is a set of substitutable parameters
- Flyway supports these
- They can be specified on the command line
- Standard convention is ${parm} but configurable using API
  - for example <parm> works fine

CLEAN

- $ flyway clean
  - cleans out the schema (drops everything)
  - good for development
    - scary for other environments
- Limited usefulness: does not work when dropping a table in an explicitly defined tablespace with

-669 THE OBJECT CANNOT BE EXPLICITLY DROPPED. REASON 001

AUTOMATING THE PROCESS WITH NETREXX

WHY DO THIS

- have a look at the Flyway script and ask yourself if this is going to work on all shells that you are using (think of USS with ksh or tcsh in EBCDIC)
- The answer is probably: Nah
- Also, the script counts on a specific layout for the directory structure
- Instead of layout, .conf file, jars lookup, just one nrx script
- Why NetRexx: Flyway is a Java Jar. All methods can be seamlessly called by NetRexx
- We are using NetRexx scripting mode: no need to use the compiler
- You can use the generated Java for the customer

THIS IS ALL YOU NEED

```
import org.flywaydb.core.Flyway

fw = Flyway()
fw.setDataSource("jdbc:db2:xxxxxxx/LOCDB00","xxxxxx","xxxxxx", null)
fw.setTable("SCHEMA_VERSION")
fw.setBaselineOnMigrate(1)
fw.migrate()
```

OOREXX

```
fw = .bsf~new("org.flywaydb.core.Flyway")
fw~setDataSource("jdbc:db2:xxxxxxx/LOCDB0O","xxxxxx","xxxxxx", .nil)
fw~setTable("SCHEMA_VERSION")
fw~setBaselineOnMigrate(1)
fw~migrate

::requires "BSF.CLS"
```

WHEN USING PARAMETERS – PUT THEM IN A MAP AND TELL FLYWAY

```
import org.flywaydb.core.Flyway

parms = TreeMap()
parms.put(String "SQLID", String "JANSR16")
parms.put(String "DB2DBNAME", String "JANSR16")
parms.put(String "DB2TSSTOGROUP", String "SGDB0O")

fw = Flyway()
fw.setDataSource("jdbc:db2:xxxxxxx/LOCDB0O","xxxxxx","xxxxxx", null)
fw.setTable("SCHEMA_VERSION")
fw.setBaselineOnMigrate(1)

fw.setPlaceholderPrefix('<')
fw.setPlaceholderSuffix('>')
fw.setPlaceHolders(parms)

fw.migrate()
```

OOREXX

```
parms = .bsf~new("java.util.TreeMap")
parms~put("SQLID", "JANSR16")
parms~put("DB2DBNAME", "JANSR16")
parms~put("DB2TSSTOGROUP", "SGDB0O")

fw =  .bsf~new("org.flywaydb.core.Flyway")
fw~setDataSource("jdbc:db2:xxxxxxx/LOCDB0O","xxxxxx","xxxxxx", .nil)
fw~setTable("SCHEMA_VERSION")
fw~setBaselineOnMigrate(1)

fw~setPlaceholderPrefix('<')
fw~setPlaceholderSuffix('>')
fw~setPlaceHolders(parms)

fw~migrate

::requires "BSF.CLS"
```

CONVERSIONS: SQL BASED

SQL BASED CONVERSIONS

- The simplest way, and lots of people do it always like this, is to make a copy or rename the old table and insert the data back into the newly defined new table; then drop the old one
- If you cannot switch off logging this is not a good idea for those very large tables
- Also, you can alter tables, add or delete (novelty for DB2 V11) columns - but your tablespace enters Advisory Reorg status
- But an SQL-based conversion looks like this:

SQL BASED CONVERSION

```
SET CURRENT SQLID = '<schema>';

CREATE TABLESPACE SFLYWAY
       IN "<schema>"
       SEGSIZE 4
       BUFFERPOOL BP0
       LOCKSIZE PAGE
       LOCKMAX SYSTEM
       CLOSE YES
       COMPRESS YES
  ;


CREATE TABLE "<schema>"."TMP_<table>" (
    "installed_rank" INT NOT NULL,
    "version" VARCHAR(50),
    "description" VARCHAR(200) NOT NULL,
    "type" VARCHAR(20) NOT NULL,
    "script" VARCHAR(1000) NOT NULL,
    "checksum" INT,
    "installed_by" VARCHAR(100) NOT NULL,
    "installed_on" TIMESTAMP NOT NULL WITH DEFAULT,
    "execution_time" INT NOT NULL,
    "success" SMALLINT NOT NULL,
    CONSTRAINT "<table>_S" CHECK ("success" in(0,1))
)
IN "<schema>".SFLYWAY;
```

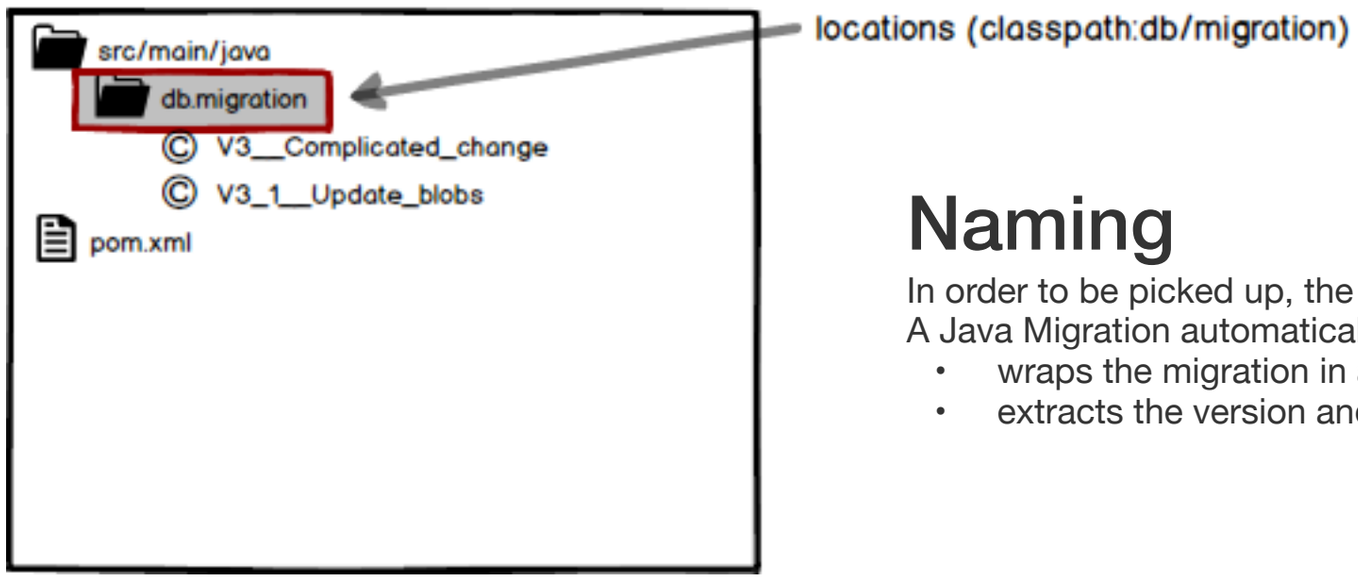
```
INSERT INTO "<schema>"."TMP_<table>"(
SELECT
    "installed_rank",
    "version",
    "description",
    "type",
    "script",
    "checksum",
    "installed_by",
    "installed_on",
    "execution_time",
    "success"
FROM "<schema>"."<table>");

--drop old tablespace
DROP TABLESPACE "<schema>".SDBVERS;

RENAME TABLE "<schema>"."TMP_<table>" TO
"<table>";

UPDATE "<schema>"."<table>" SET
"type"='BASELINE' WHERE "type"='INIT';

CREATE UNIQUE INDEX
"<schema>"."<table>_IR_IDX" ON
"<schema>"."<table>" ("installed_rank");
ALTER TABLE "<schema>"."<table>" ADD
CONSTRAINT "<table>_PK" PRIMARY KEY
("installed_rank");

CREATE INDEX "<schema>"."<table>_S_IDX"
ON "<schema>"."<table>" ("success");
```

CONVERSIONS: JAVA BASED

JAVA BASED CONVERSIONS

- Mostly used for BLOB or CLOB handling, Java based conversions give more freedom over the workflow
- You can open and close cursors, read and write files, insert (and validate) XML columns from files

LOCATION AND DISCOVERY

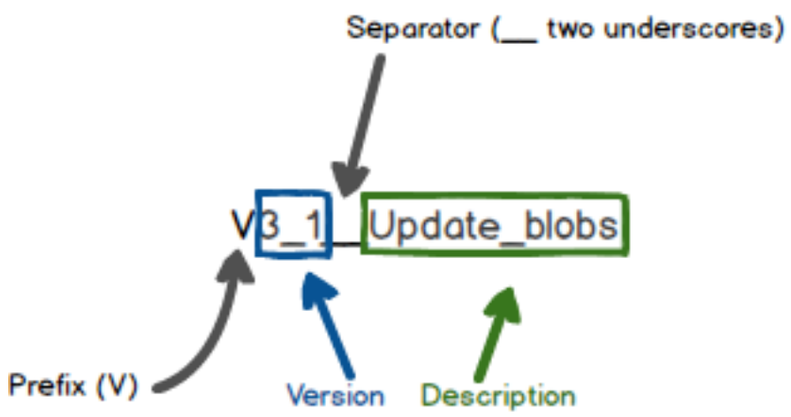


Naming

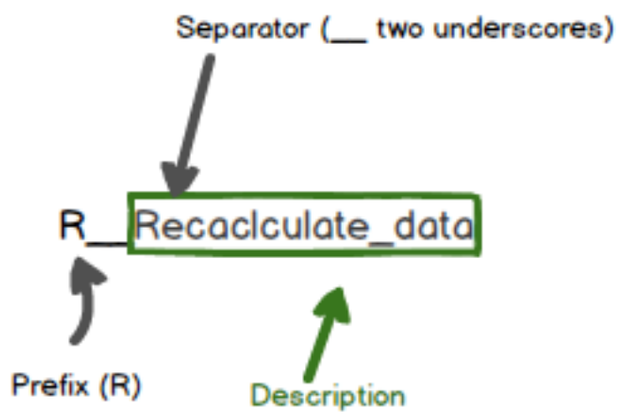
In order to be picked up, the Java Migrations must implement **JdbcMigration**.
A Java Migration automatically

- wraps the migration in a transaction
- extracts the version and the description from the class name

Versioned Migration



Repeatable Migration



Repeatable Migrations are always run

YOU CAN USE CALLBACKS

Name	Execution
beforeMigrate	Before Migrate runs
beforeEachMigrate	Before every single migration during Migrate
afterEachMigrate	After every single migration during Migrate
afterMigrate	After Migrate runs
beforeClean	Before Clean runs
afterClean	After Clean runs
beforeInfo	Before Info runs
afterInfo	After Info runs
beforeValidate	Before Validate runs
afterValidate	After Validate runs
beforeBaseline	Before Baseline runs
afterBaseline	After Baseline runs
beforeRepair	Before Repair runs
afterRepair	After Repair runs

CONVERSIONS:
DB2 Z/OS UTILITIES

USING DB2 UTILITIES

- A z/OS DB2 DBA will want to use DB2 utilities in a number of cases
  - LOAD LOG(NO) instead of SQL INSERT
  - LOAD Replace to clear out a partition
  - REORG and RUNSTATS
  - The Crossloader
  - IMAGECOPY for recoverability

DB2 UTILITIES

- But … don't you need JCL to start a DB2 utility?
  - You cannot make a Rexx exec to start pgm DSNUTILB
    - Because it runs in storage key 7
    - Believe me, it has been tried
- There are two stored procedures, however:
  - DSNUTILS (EBCDIC only, deprecated)
  - DSNUTILU (EBCDIC and Unicode, supported)

FIRST TRY IF THE SYSUTILU STORED PROCEDURE WORKS

```
import java.sql.
class.forName("com.ibm.db2.jcc.DB2Driver")

con = java.sql.Connection -
      java.sql.DriverManager.getConnection(-
      "jdbc:db2://xxx.xxxxx.xxxx.xxxx.xxx/xxxx", "xxxxxx", "xxxxxxxx")

cstmt = con.prepareCall("CALL DSNUTILU(?,?,?,?,?)")

cstmt.setString(1, "JANSR16");
cstmt.setString(2, "NO")
cstmt.setString(3, "TEMPLATE TEMPL01 "-
                   " DSN 'XXXXXX.&DB..&SN..P&PA(2,4)..T&TIME.'  " -
                   " UNIT SYSDA DISP(NEW,CATLG,DELETE) " -
                   " REORG TABLESPACE XXXXXX.XXXXX COPYDDN (TEMPL01) " -
                   " SHRLEVEL REFERENCE NOSYSREC SORTDEVT SYSDA SORTNUM 64 " -
                   " STATISTICS INDEX TABLE SAMPLE 25")
cstmt.setString(4, "")

cstmt.execute()

rs = cstmt.getResultSet()
loop while rs.next()
  say rs.getString(2)
end
cstmt.close()
```

OOREXX

```
call bsf.loadClass "com.ibm.db2.jcc.DB2Driver"

con = bsf.loadClass("java.sql.DriverManager") ~getConnection(-
    "jdbc:db2://xxx.xxxxx.xxxx.xxxx.xxx/xxxx", "xxxxxx", "xxxxxxxx")

cstmt = con~prepareCall("CALL DSNUTILU(?,?,?,?,?)")

cstmt~setString(1, "JANSR16")
cstmt~setString(2, "NO")
cstmt~setString(3, "TEMPLATE TEMPL01 "-
            " DSN 'XXXXXX.&DB..&SN..P&PA(2,4)..T&TIME.'  " -
            " UNIT SYSDA DISP(NEW,CATLG,DELETE) " -
            " REORG TABLESPACE XXXXXX.XXXXX COPYDDN (TEMPL01) " -
            " SHRLEVEL REFERENCE NOSYSREC SORTDEVT SYSDA SORTNUM 64 " -
            " STATISTICS INDEX TABLE SAMPLE 25")
cstmt~setString(4, "")

cstmt~execute

rs = cstmt~getResultSet
loop while rs~next
  say rs~getString(2)
end
cstmt~close

::requires “BSF.CLS"
```

DSNUTILU

- Note that DSNUTILU can reside in a package that has been bound with ENCODING(EBCDIC) or ENCODING(UNICODE)
- When a Unicode space (X'20') is recognised, the output for the SYSPRINT resultset is Unicode
  - So when you start the command with a quote, it goes terribly wrong. Well, not terribly, and not wrong, but you cannot read the output from the DB2 utility, IDCAMS, DFSORT, and the rest
- For debugging this, you need to convert EBCDIC strings to Unicode with String.getBytes("Cp1047") and String(var,"UTF-8")

DSNUTILU

- You don't have DDnames. So use the TEMPLATE utility that generates and dynamically allocates datasets for you
- When using GDG's, which is possible, you need to have a preallocated model DCB dataset cataloged - then you can use it from a TEMPLATE

TO A JAVA BASED CONVERSION

- When you have seen that DSNUTILU is working, it is time to have these Stored Procedure calls in your conversion scenario

- Make sure your class inherits from **JdbcConversion**
- Make sure to replace the '.' in the class name with a '_'
  - so **V1.2__Reorg.sql** becomes class **V1_2__Reorg**
- Put it in a package **db.migrations** and leave the .class file in the same directory next to the other, sql-based migration files

V1_2__REORG.NRX

```
package db.migration

import java.sql.
import org.flywaydb.core.

class V1_2__Reorg implements JdbcMigration

  method migrate(con=Connection) signals Exception

    cstmt = con.prepareCall("CALL DSNUTILU(?,?,?,?,?)")

    cstmt.setString(1, "<SQLID>REO");
    cstmt.setString(2, "NO")
    cstmt.setString(3, "TEMPLATE TEMPL01 "-
                    " DSN 'A21G089.<SQLID>.&DB..&SN..P&PA(2,4)..T&TIME.'  " -
                    " UNIT SYSDA DISP(NEW,CATLG,DELETE) " -
                    " REORG TABLESPACE <DB2DBNAME>.JANTST COPYDDN (TEMPL01) " -
                    " SHRLEVEL REFERENCE NOSYSREC SORTDEVT SYSDA SORTNUM 64 " -
                    " STATISTICS INDEX TABLE SAMPLE 25")
    cstmt.setString(4, "")

    cstmt.execute()

    rs = cstmt.getResultSet()
    loop while rs.next()
      say rs.getString(2)
    end
    cstmt.close()
```

OOREXX

```
return BSFCreateRexxProxy(.V1_2__Reorg~new, ,"db.migration.JdbcMigration")

::requires "BSF.CLS"

::class V1_2__Reorg

::method migrate
  use arg con

  cstmt = con~prepareCall("CALL DSNUTILU(?,?,?,?,?)")
  cstmt~setString(1, "<SQLID>REO")
  cstmt~setString(2, "NO")
  cstmt~setString(3, "TEMPLATE TEMPL01 "-
            " DSN 'A21G089.<SQLID>.&DB..&SN..P&PA(2,4)..T&TIME.'  " -
            " UNIT SYSDA DISP(NEW,CATLG,DELETE) " -
            " REORG TABLESPACE <DB2DBNAME>.JANTST COPYDDN (TEMPL01) " -
            " SHRLEVEL REFERENCE NOSYSREC SORTDEVT SYSDA SORTNUM 64 " -
            " STATISTICS INDEX TABLE SAMPLE 25")
  cstmt~setString(4, "")
  cstmt~execute

  rs = cstmt~getResultSet
  loop while rs~next
    say rs~getString(2)
  end
  cstmt~close
```

AUTO-MIGRATION ON STARTUP

THE EASY CASE

- Have your code and migrations in Git, and deploy always the corresponding migrations (as the highest level) with the application
- Start the application with a
  - fw.migrate()
- It will migrate, but only the first time
- Subsequently, it will check and do nothing
- Always cater for quick fallback - you never know

THE NOT-SO-EASY CASE

- In testing and acceptance/certification environments, you need repeatable conversions, and not always to the highest available database level

ROLL-BACK AND RECOVERY

- On z/OS DB2, DDL updates are atomic within a transaction
- Failed migrations are properly rolled back
- (this works for DB2, PostgreSQL, Derby, EnterpriseDB and to a certain extent SQL Server); Oracle surreptitiously sneaks in commits between DDL statements, invalidating the transaction concept

THANK YOU

Q?

RVJANSEN@XS4ALL.NL