Best Practices for Migrating MySQL Sakila Database to DBMaker

December 11, 2018

凌羣電腦
THE SYSCOM GROUP

Contents

1. Overview

This document demonstrates how to migrate a local version of the MySQL **Sakila** sample database to a DBMaker database.

Sakila is the traditional MySQL example database, we're going to fully migrate it over to DBMaker database.

To migrate the data, we use the Migration Wizard included with DBMaker. To migrate schema, store procedures, user-defined functions, and triggers, we perform some modifications to the SQL code to ensure the correct behavior is achieved.

Required Software:

- DBMaker 5.4 Windows x64 or later
- MySQL Community Server 8.0 or later
- MySQL connector ODBC 8.0 Driver (included in MySQL Community Server)
- MySQL Sakila sample database (included in MySQL Community Server)

This document is written for the Windows 7 x64 operating system.

2. Migration Preparation

This chapter discusses the following topics:

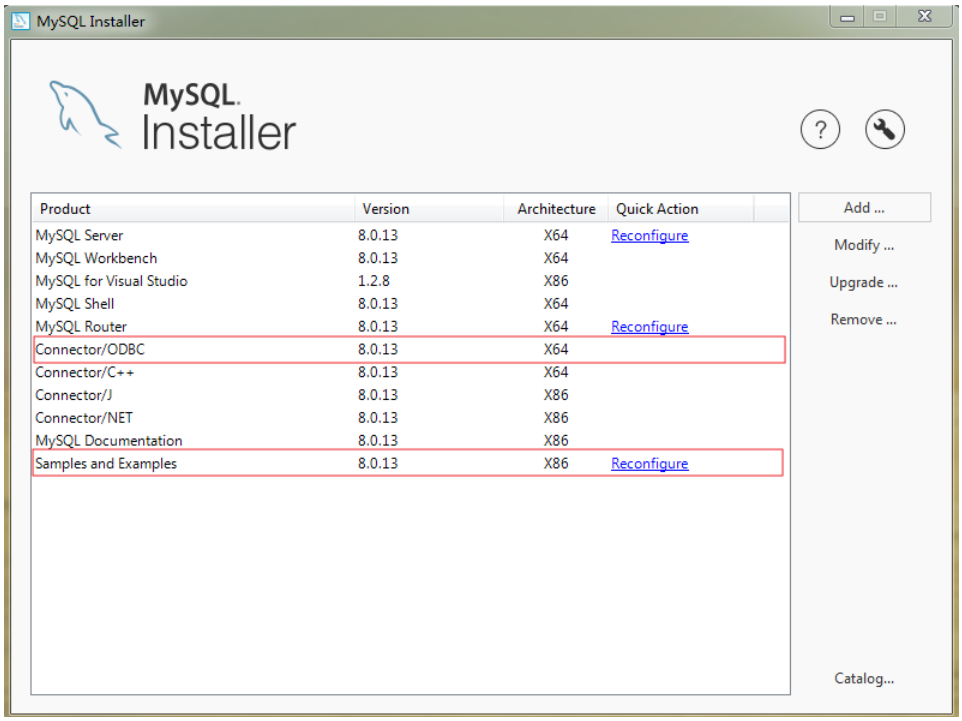
- Installing and Configuring MySQL Database
- Installing and Configuring DBMaker Database
- Creating a New database
- Creating System DSN
- Migration Assistance tool

2.1 Installing and Configuring MySQL database

- Download and install the MySQL Installer for Windows

  https://dev.mysql.com/downloads/mysql/

  If you already have MySQL installed, then you can skip the above step.



Note: Ensure that Connector/ODBC and Samples and Examples are installed

By default, Sakila database comes with the MySQL installation. If you do not have the sample database, then you can download the database scripts at

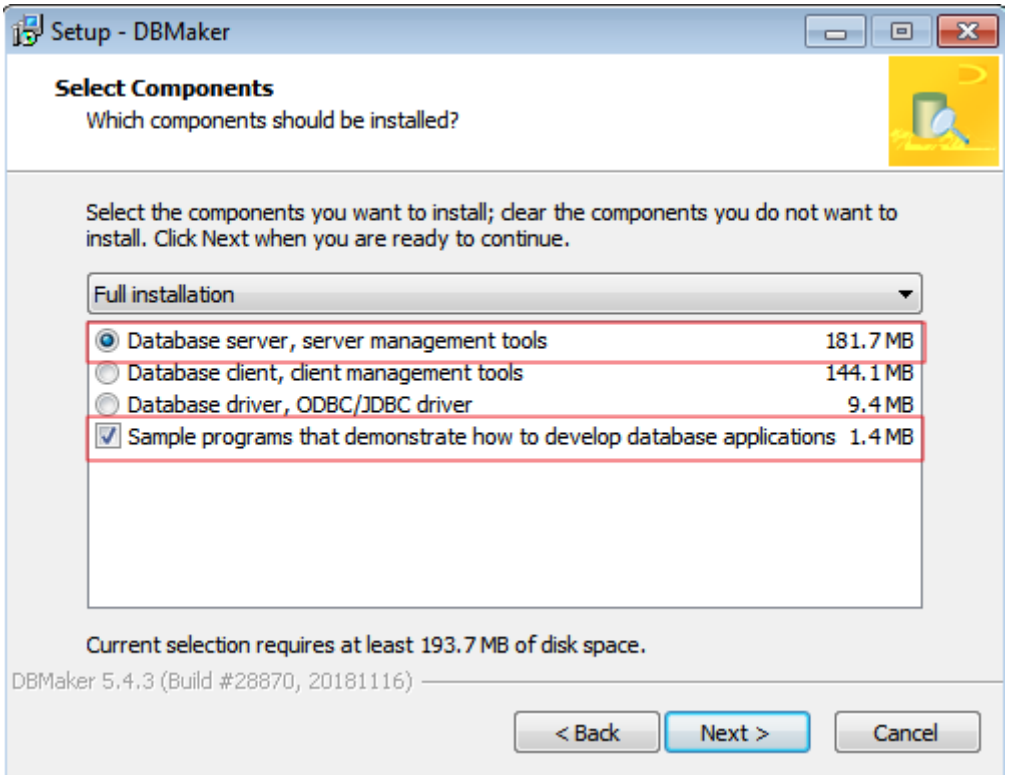
http://downloads.mysql.com/docs/sakila-db.zip.

2.2 Installing and Configuring DBMaker database

- Download and install DBMaker Installer for Windows

http://www.dbmaker.com.tw/downloads542.html

If you already have DBMaker installed, then you can skip the step.



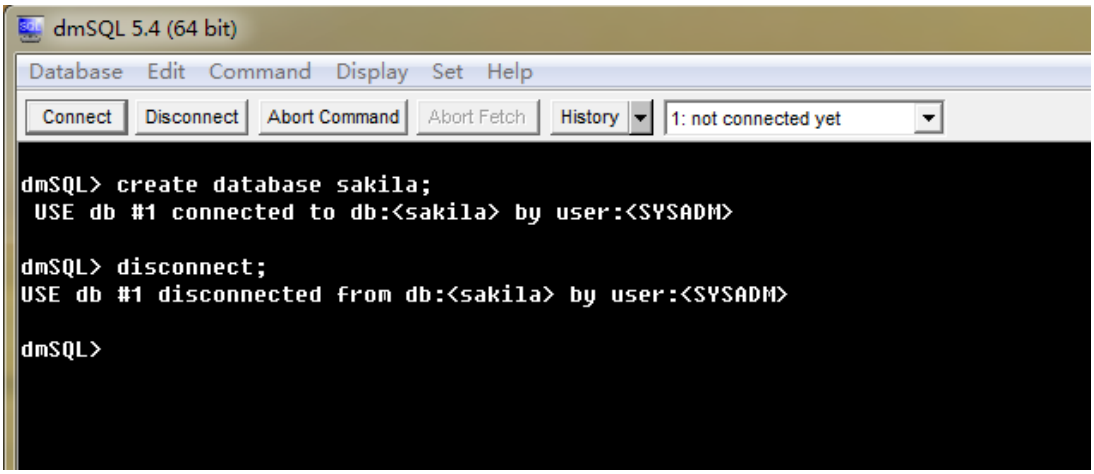
2.3 Create a New DBMaker Database

You can find the dmconfig.ini in the installation directory of DBMaker, and edit it and add new section [sakila]:

```
[sakila]
DB_DBDIR = C:\DBMaker\5.4\rdb
DB_FODIR = C:\DBMaker\5.4\rdb\fo
DB_SVADR = 127.0.0.1
DB_PTNUM = 12453
DB_USRID = SYSADM
DB_JNLSZ = 1000
DB_LCODE = 10
```

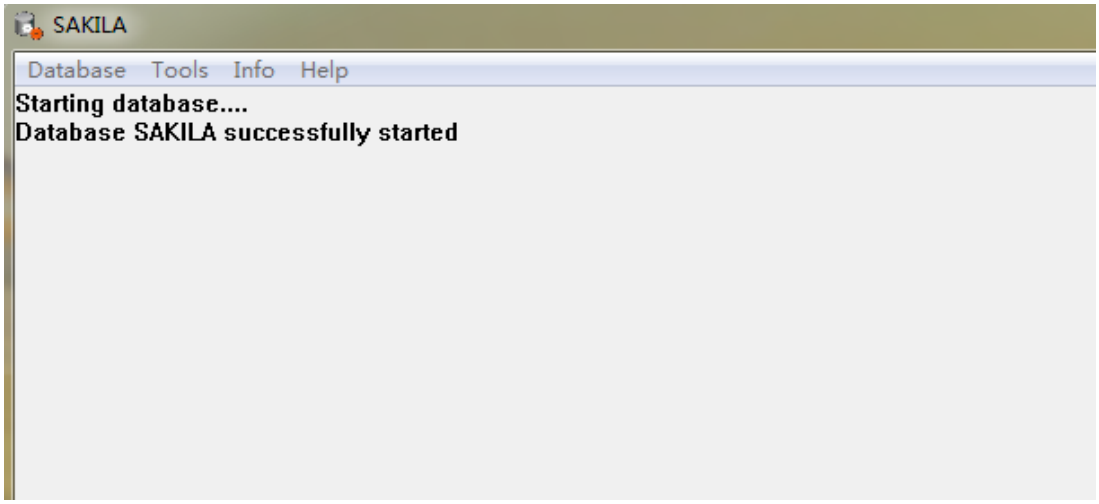
On the Start menu, click DBMaker 5.4 and dmSQL.

In dmSQL (64bit) console, click Connect and input user name sysadm and password empty.



On the Start menu, click DBMaker 5.4 and DBMaker Server.

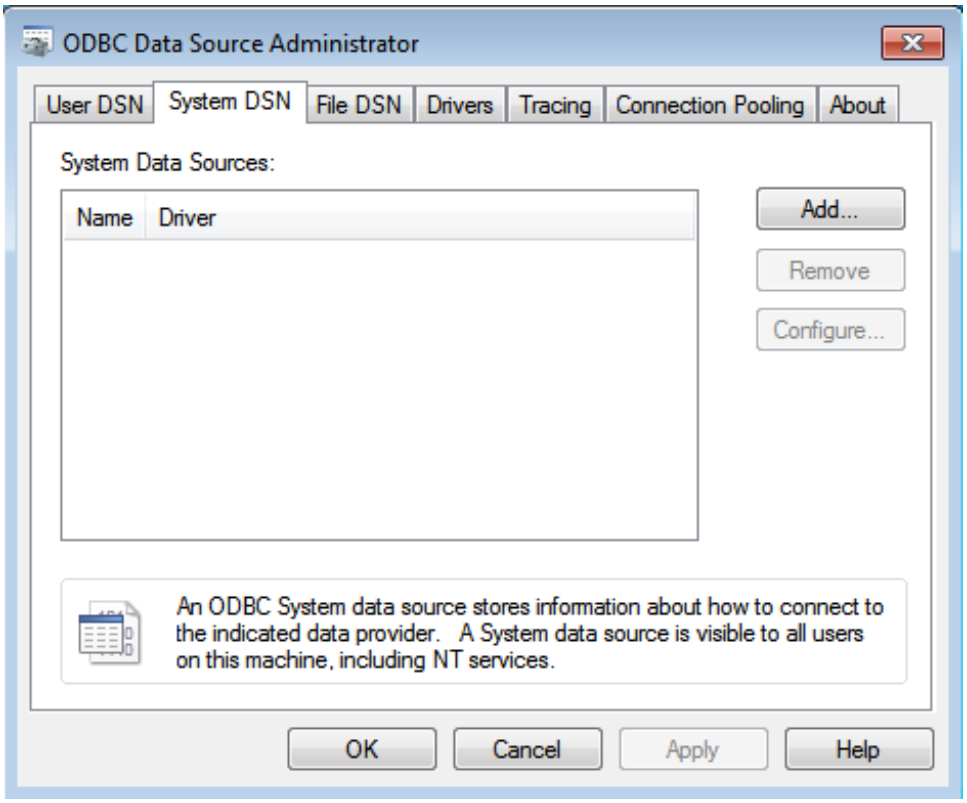
In DBMaker Server Panel, select Database Name -- sakila



2.4 Creating System DSN

To open the ODBC Data Source Administrator in Windows 7

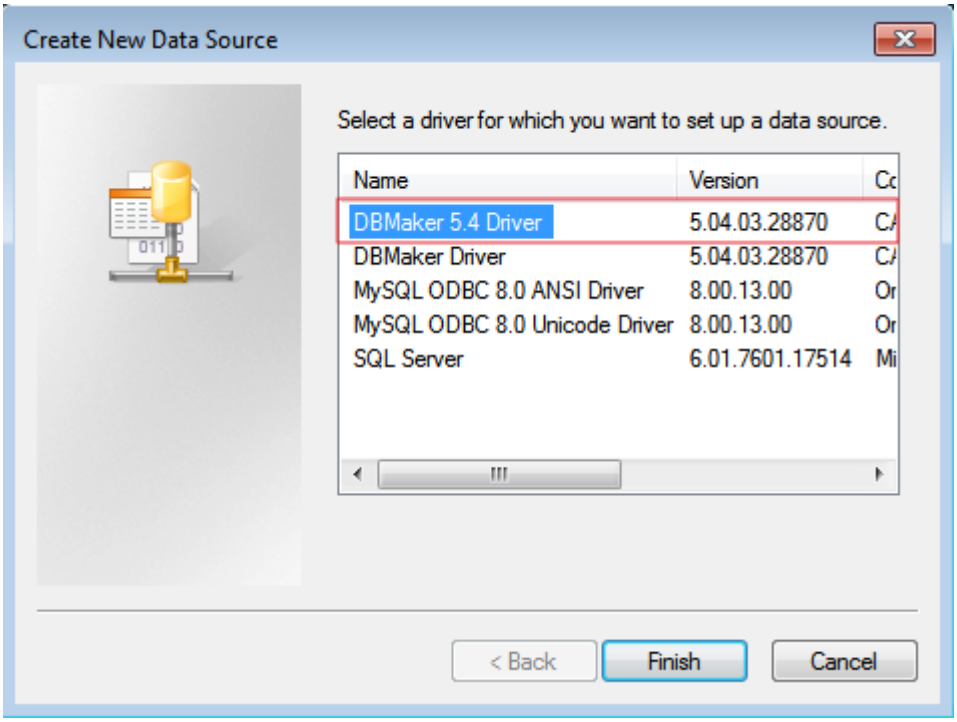
- On the Start menu, click Control Panel.
- In Control Panel, click Administrative Tools.
- In Administrative Tools, click Data Sources (ODBC).



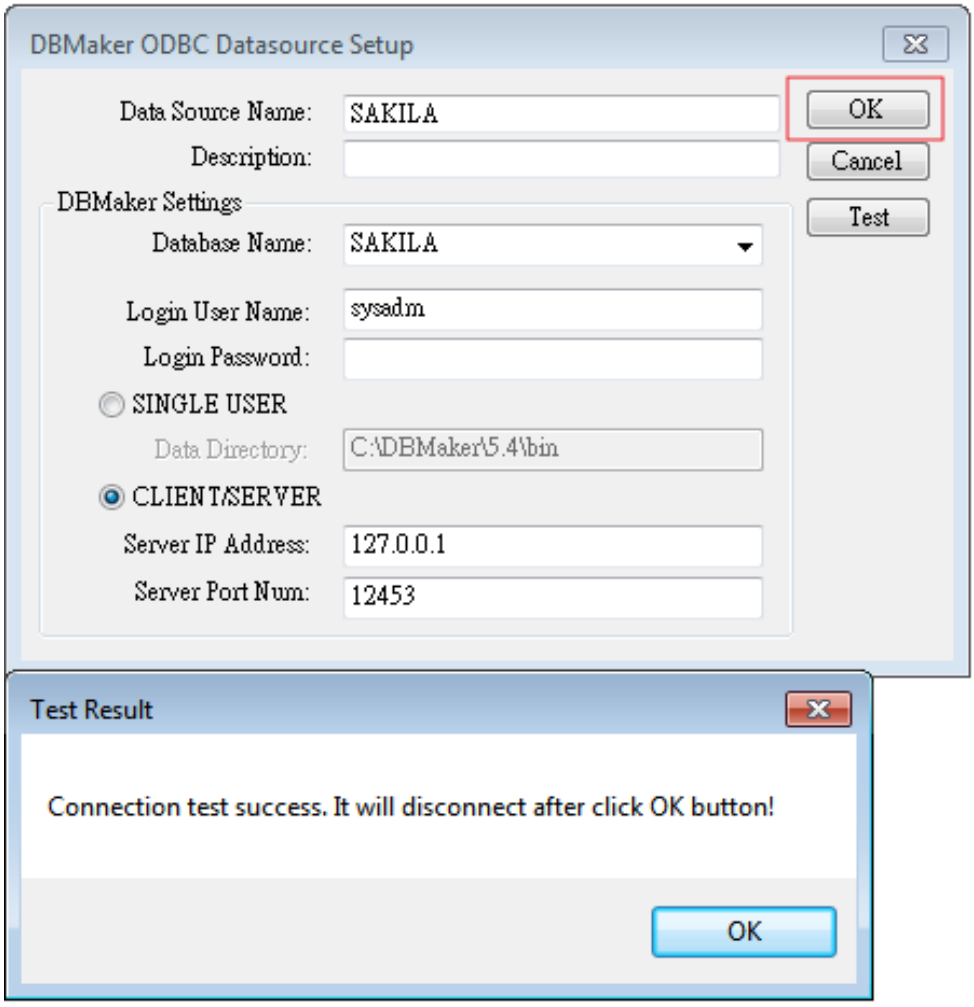
2.4.1 CREATE SYSTEM DSN FOR DBMAKER

In Windows, in the System DSN tab of the ODBC Data Source Administrator, create an ODBC connection to DBMaker database using a DBMaker 5.4 driver or DBMaker Driver.

Test the connections to make sure they work.



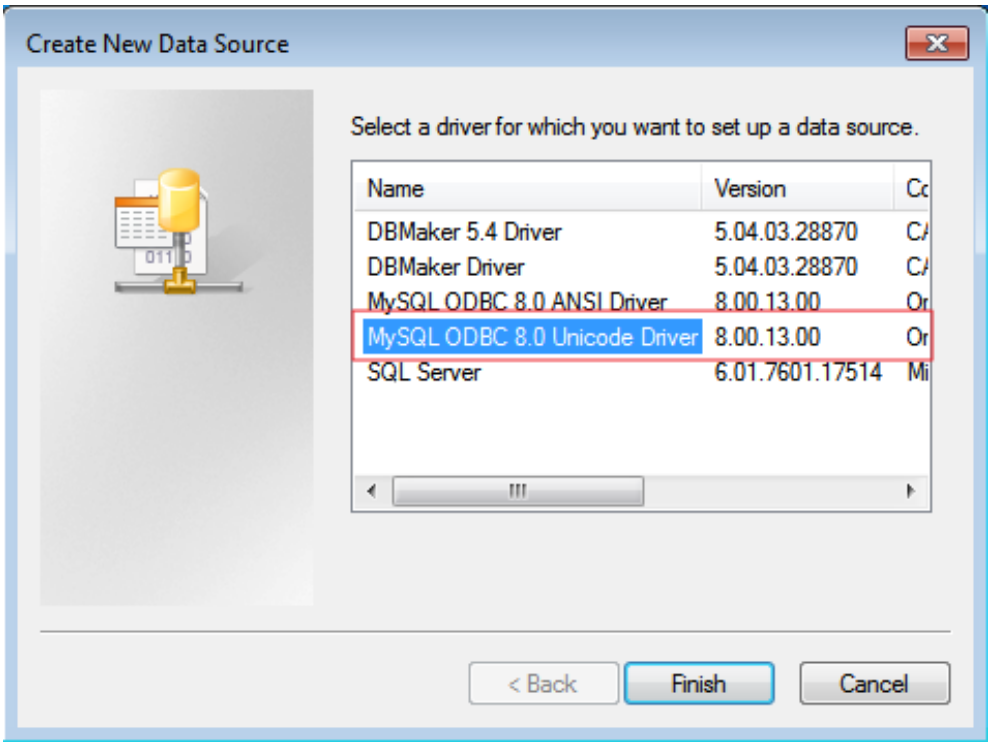
On the DBMaker ODBC Datasource Setup Window, enter your connection string settings. If you installed DBMaker locally, enter:



2.4.2 CREATE SYSTEM DSN FOR MYSQL

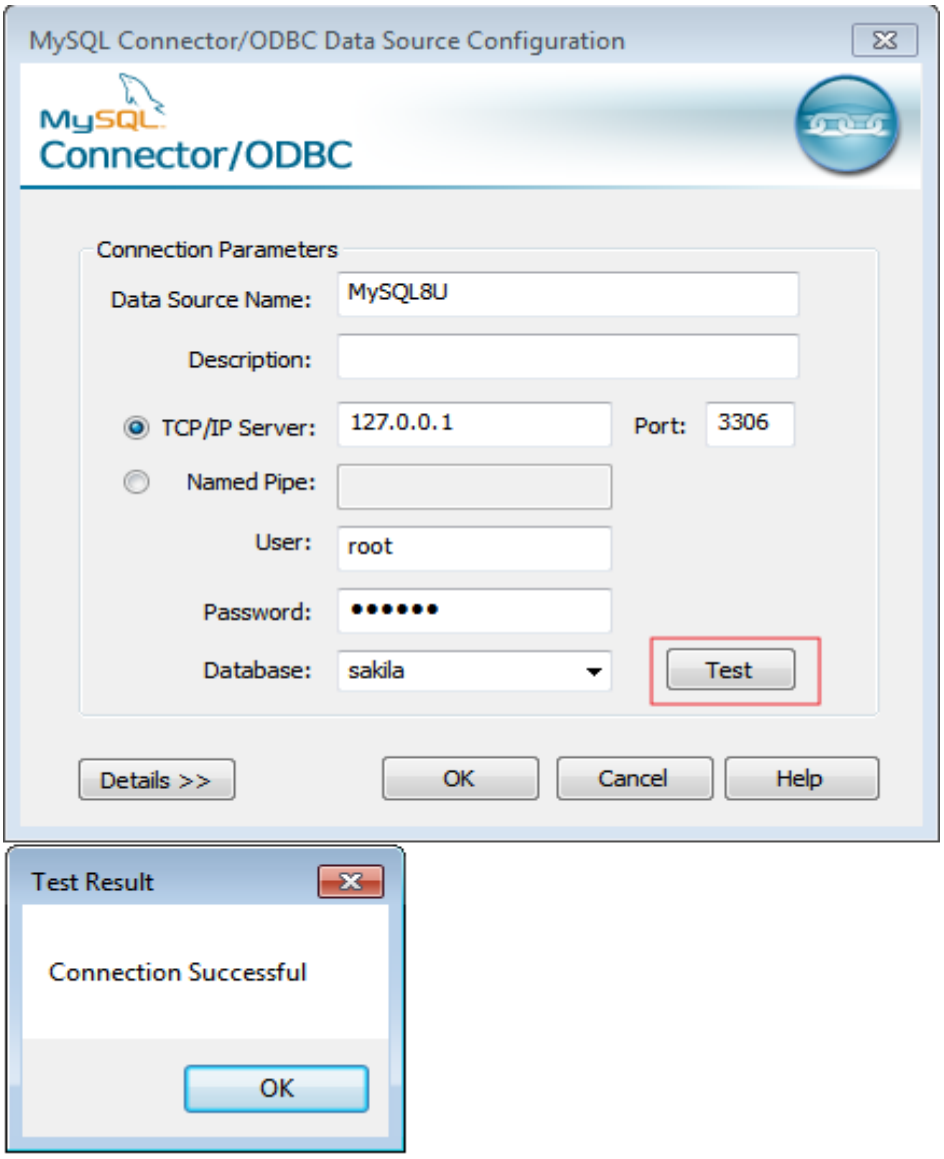
In Windows, in the System DSN tab of the ODBC Data Source Administrator, create an ODBC connection to MySQL database using a MySQL ODBC 8.0 Unicode driver.

Test the connections to make sure they work.

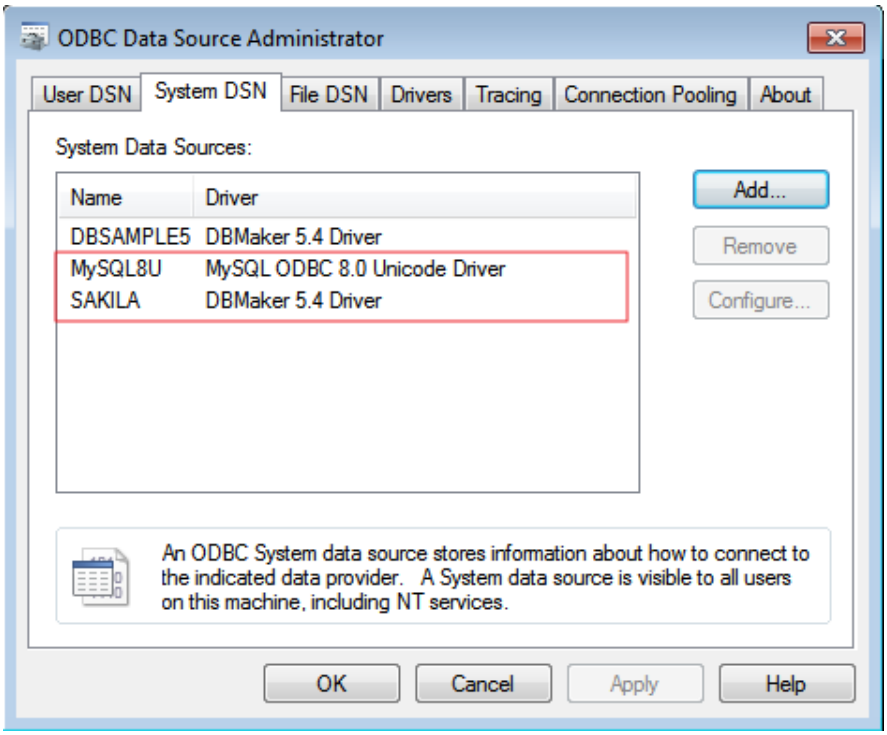


Note: Because DBMaker Java client uses Unicode encoding, the data source driver must select the MySQL ODBC 8.0 Unicode driver.

On the MySQL Connector/ODBC Data Source Configuration Window, enter your connection string settings. If you installed MySQL locally, enter:



Once you have finished configuring your data source, you should see your new DSN in the list shown in the ODBC Data Source Administrator System DSN tab.



2.5 Migration Assistance tool

Before migrating MySQL sample database "sakila" to DBMaker Database, You must be familiar with the DBMaker migration Tool.

JDATATRANSFER TOOL

DBMaker offers a tool - **JDATA Transfer Tool** for migrating from a third-party database to DBMaker. The **Data Transfer Tool** provides a user-friendly interface for transferring data in and out of the database.

The **Data Transfer Tool** is a separate application which can be started as GUI.

Start>**programs**>**DBMaker 5.4**>J**DataTransfer**, or start within **JDBA Tool**.

3. Migration Process

This chapter contains the following sections:

- Performing the Migration
- Migration Validating

3.1 Performing the Migration

Once the pre-requisites had been performed the following steps were required to complete the migration:

- Step 1: Migrate Table Objects
- Step 2: Migrate Data
- Step 3: Fixing the Migration wizard
- Step 4: Migrate Constraints Objects
- Step 5: Migrate View Objects
- Step 6: Migrate Trigger Objects
- Step 7: Migrate Function Objects
- Step 8: Migrate Stored Procedure Objects

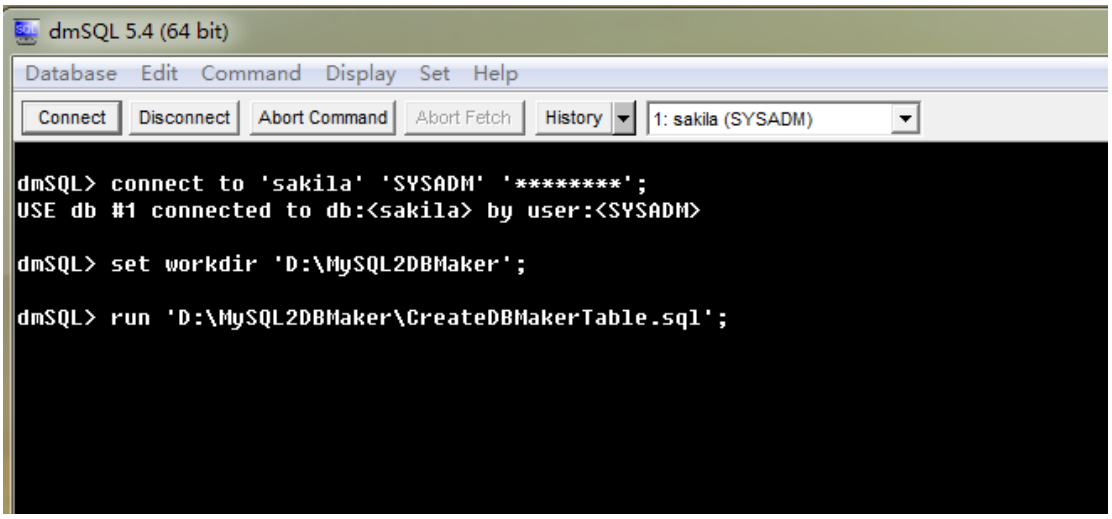
Follow the step-by-step instructions in this section to subscribe to start the migration.

3.1.1 MIGRATE TABLE OBJECTS

There are sixteen tables defined in the MySQL Sakila database, the syntax and behavior of table are very similar, data types mapping only requiring a few modifications.

This script is launched with the following command in dmsql tool. To run the script the database must be started.

See Chapter 7.2.1 for CreateDBMakerTable.sql

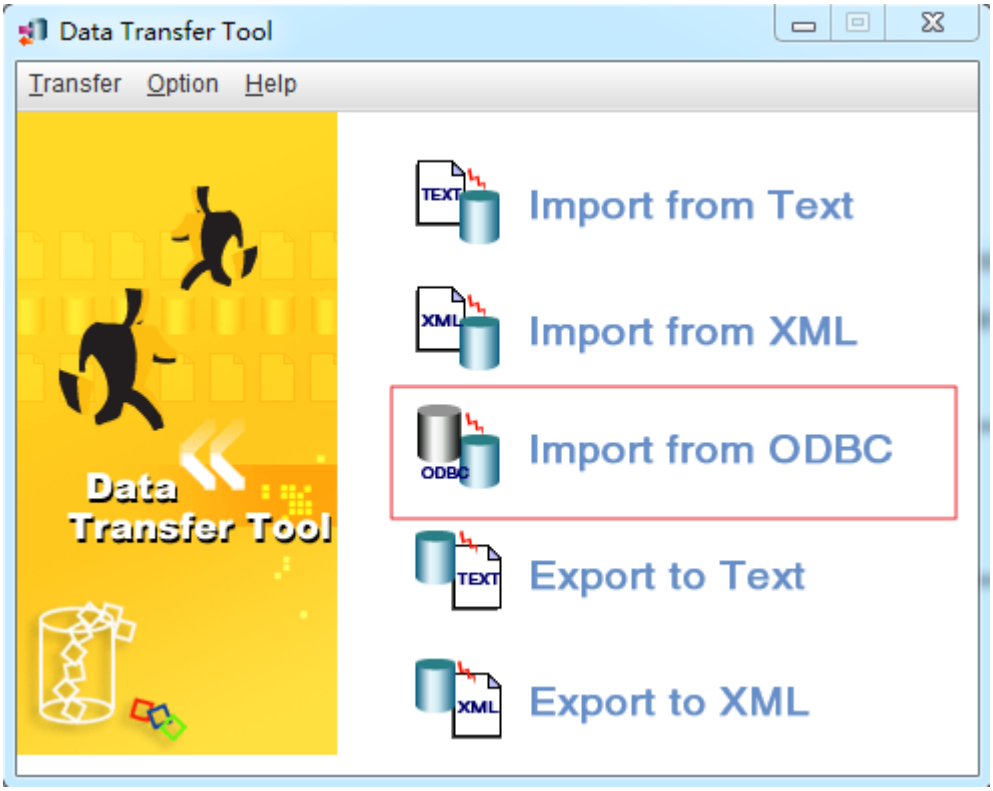


Note: All tables belong to the user "sysadm" and do not include indexes.

3.1.2 MIGRATE DATA

To migrate data from the MySQL sakila database to the newly created DBMaker database, we use the Migrate Database Wizard. To launch the Migration wizard, do the following:

- Step 1: Open the data transter tool and click the Import from ODBC option



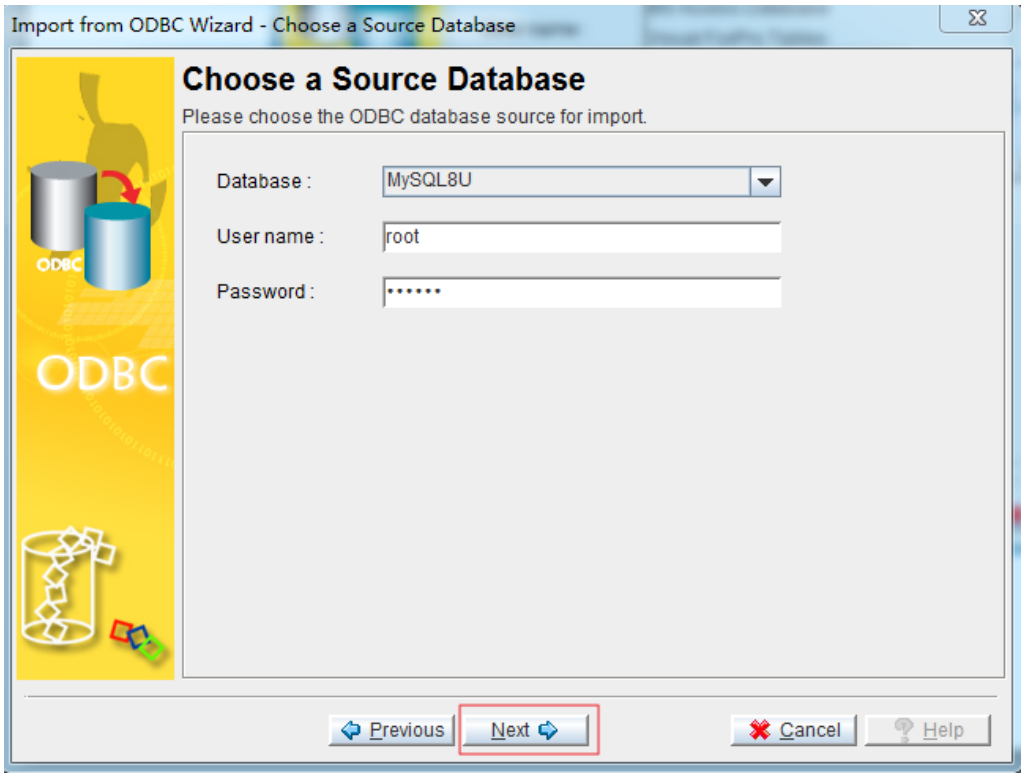
- Step 2: Click the **Next** button.

- Step 3: Specify a Source Database

Complete the following fields:

- Database name — MySQL ODBC Data Source Name
- User Name — MySQL user
- Password — MySQL password

Click the **Next** button.

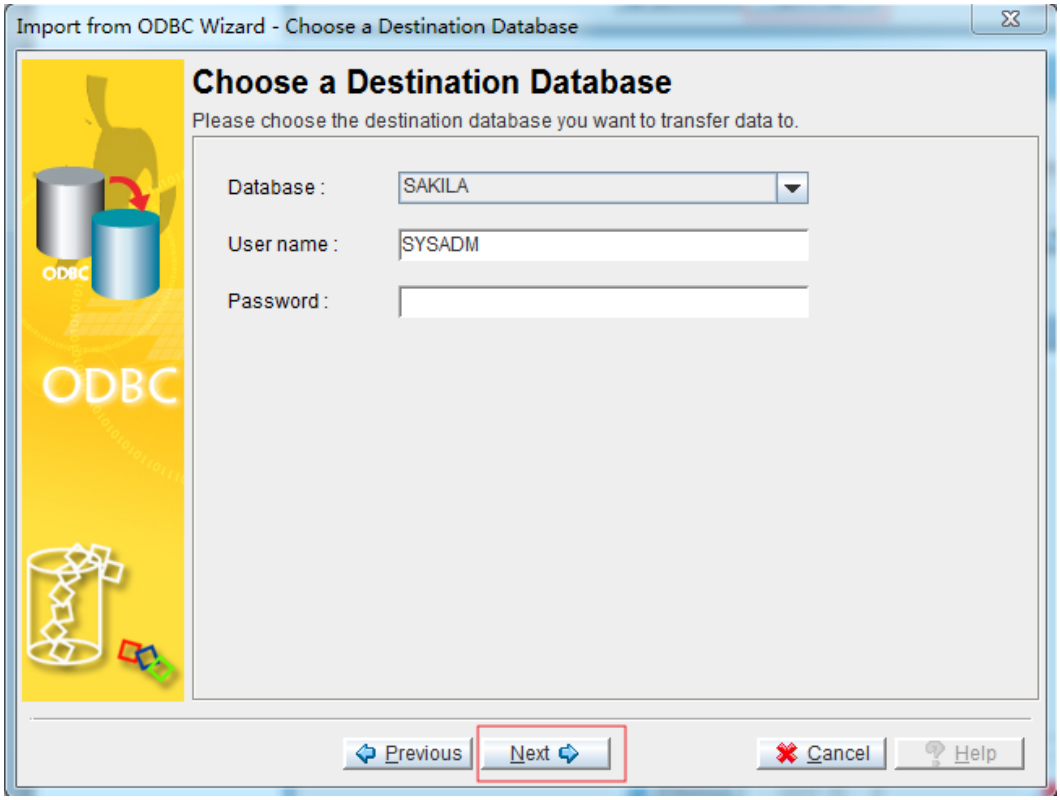


- Step 4: Specify a Destination database

Complete the following fields:

- Database — DBMaker ODBC Data Source Name
- User Name — DBMaker user
- Password — DBMaker password

Click the **Next** button.

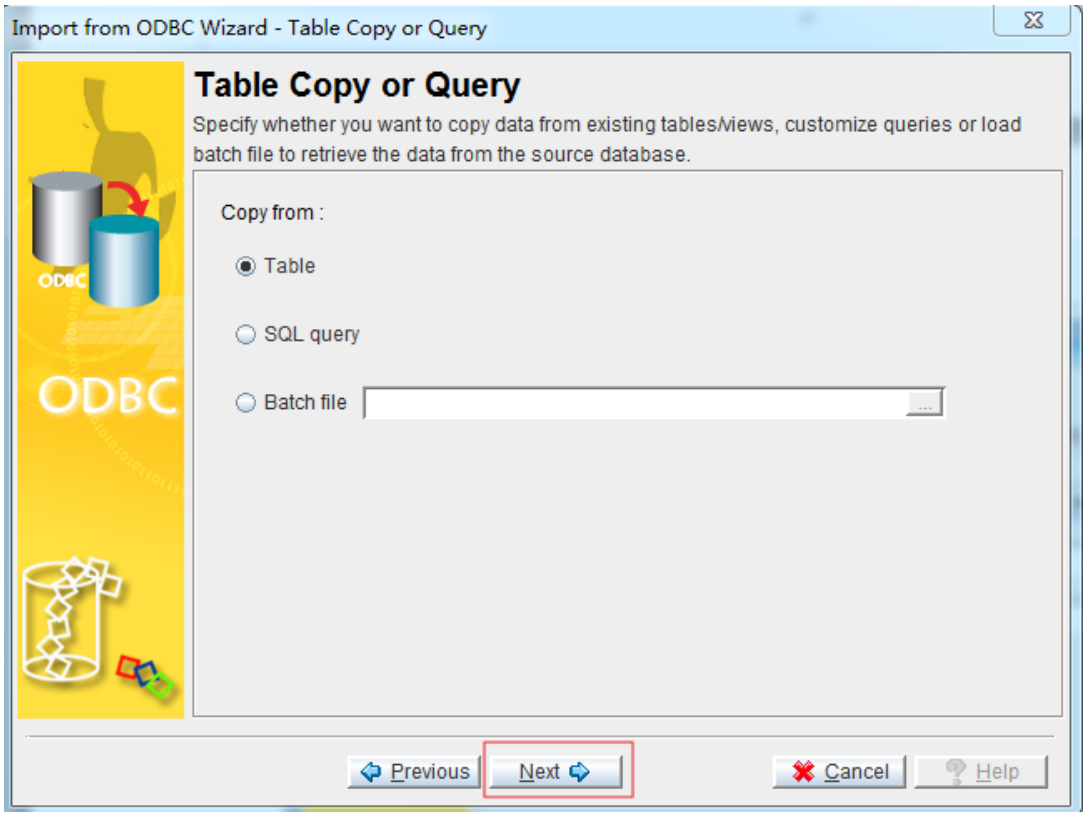


- Step 5: Specify the Table copy or query

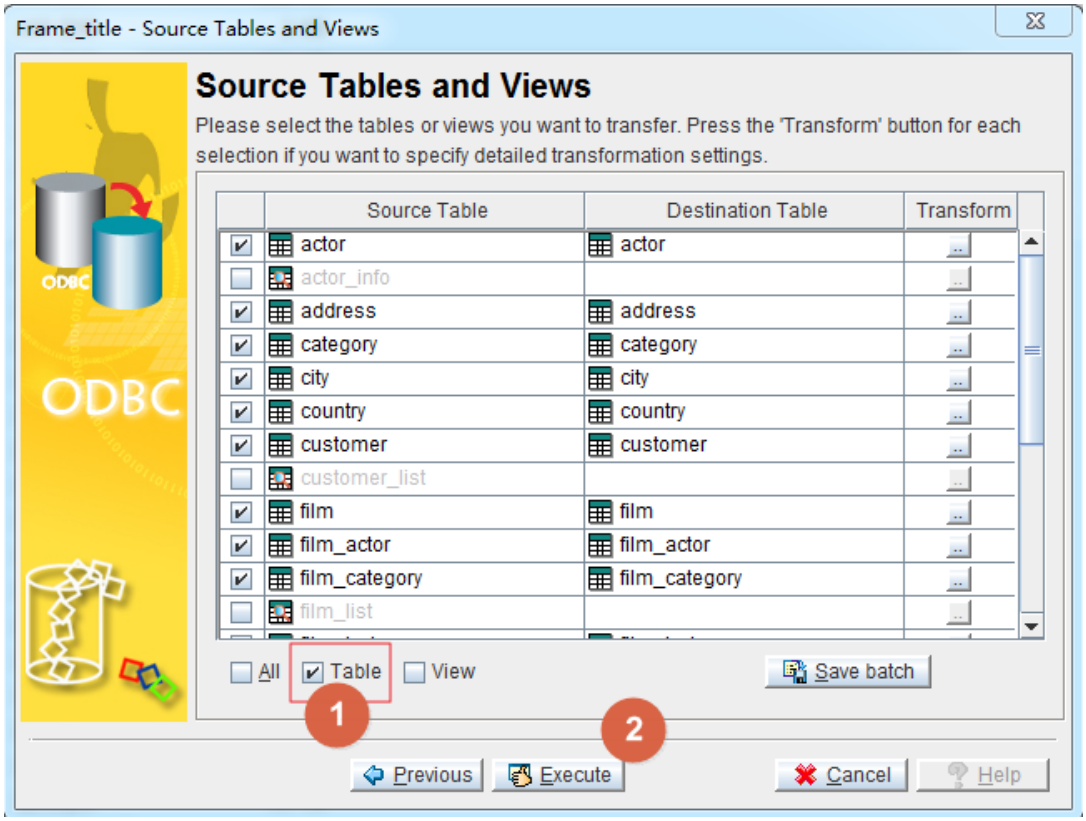
Complete the following fields:

- Table — MySQL database schema, includes tables and views
- SQL Query — customize queries by user
- Batch File— retrieve the data from MySQL

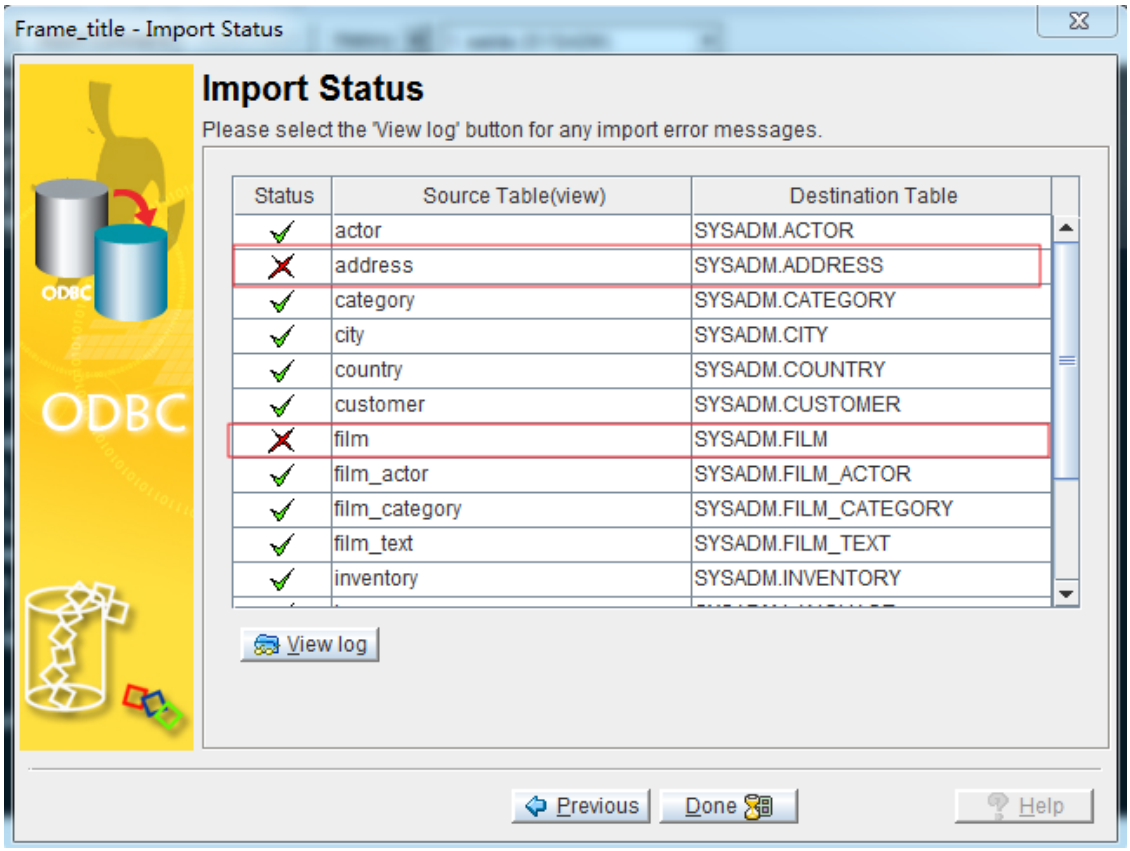
Click the **Next** button.

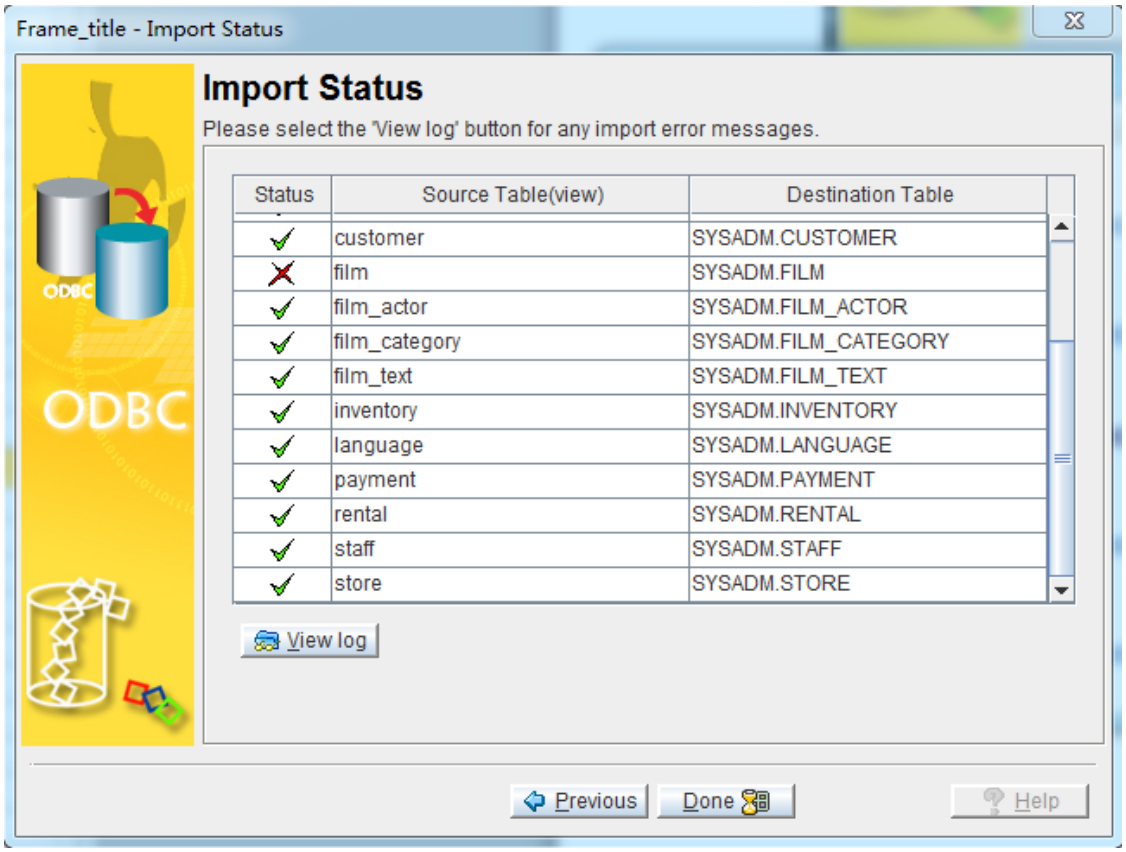


- Step 6: Select Table checkbox and Click **Execute** to begin the migration



- Step 7: Click **Done** to close the wizard and end the migration process.





Note: Reasons Address and Film table Migrations Failed

During migration, DBMaker cannot automatically convert MySQL's geometry and year(4) data types.

Click **View log** for import error and this error message appears:

```
Error (5328) : [DBMaker] the driver not capable of doing conversion
insert into "SYSADM"."ADDRESS"
(ADDRESS_ID,ADDRESS,ADDRESS2,DISTRICT,CITY_ID,
POSTAL_CODE,PHONE,LOCATION,LAST_UPDATE) values ( ?, ?, ?, ?, ?, ?, ?, ?, ?)
Source table column cannot convert to destination column
---------------------------------------------------------------------------
Source column detail information :
        column name : location
        column type : -4
        column size : -1
        C data type : -2
Destination column detail information :
        column type : 1
        column size : 255
---------------------------------------------------------------------------*/

Error (5328) : [DBMaker] the driver not capable of doing conversion
insert into "SYSADM"."FILM" (FILM_ID,TITLE,DESCRIPTION,RELEASE_YEAR,
                             LANGUAGE_ID,ORIGINAL_LANGUAGE_ID,
                             RENTAL_DURATION,RENTAL_RATE,"LENGTH",
                             REPLACEMENT_COST,RATING,SPECIAL_FEATURES,
                             LAST_UPDATE)
values ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Source table column cannot convert to destination column
---------------------------------------------------------------------------
Source column detail information :
        column name : release_year
        column type : 5
        column size : 4
        C data type : -15
Destination column detail information :
        column type : 1
        column size : 4
---------------------------------------------------------------------------*/
```

3.1.3 FIXING THE MIGRATION WIZARD

How to migrate data from the Address table?

The Address table of the MySQL sakila database contains spatial data types geometry,dbmaker are automatically converted to BLOB by default, but the data is incorrect.

Therefore, you need to manually convert the geometry type to char (255)

See Chapter 7.2.1 SCRIPTS FOR CREATING TABLES

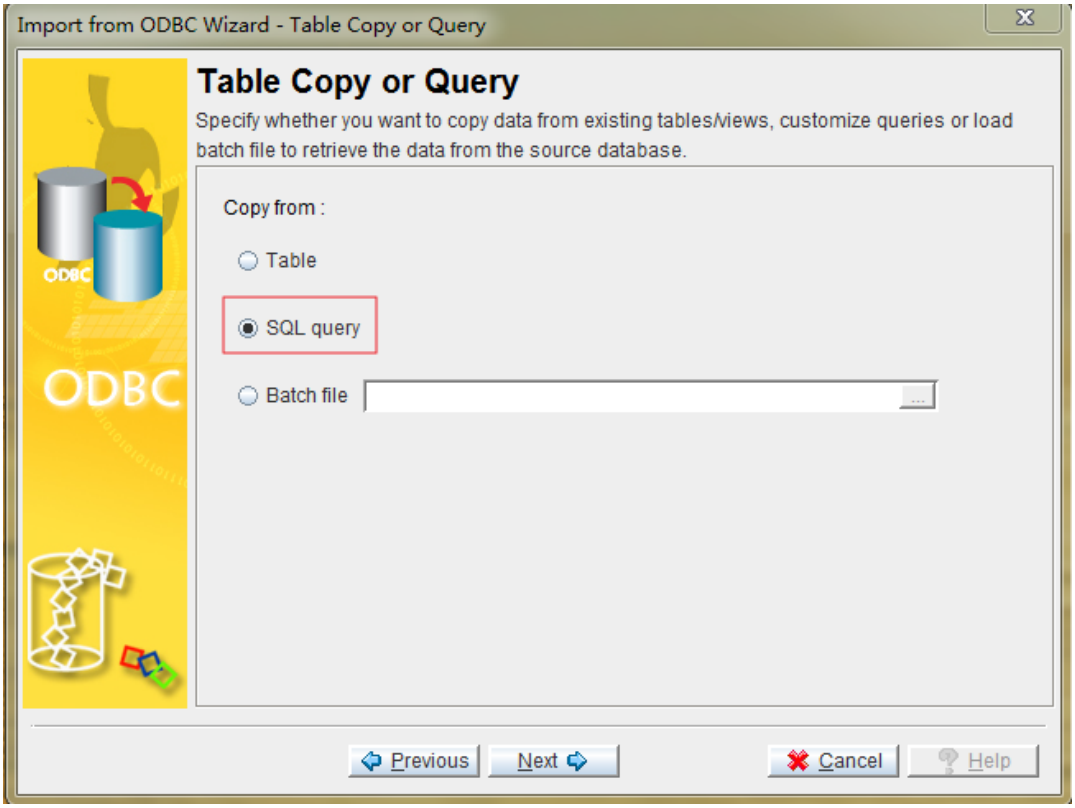
MySQL Table：Address	*DBMaker Table：Address*
`location` geometry NOT NULL	*"location" char(255) NOT NULL*

When importing data, using SQL query to set up St_ASText (' address'. ' Location'), after importing the data, the data format is shown in the following table.

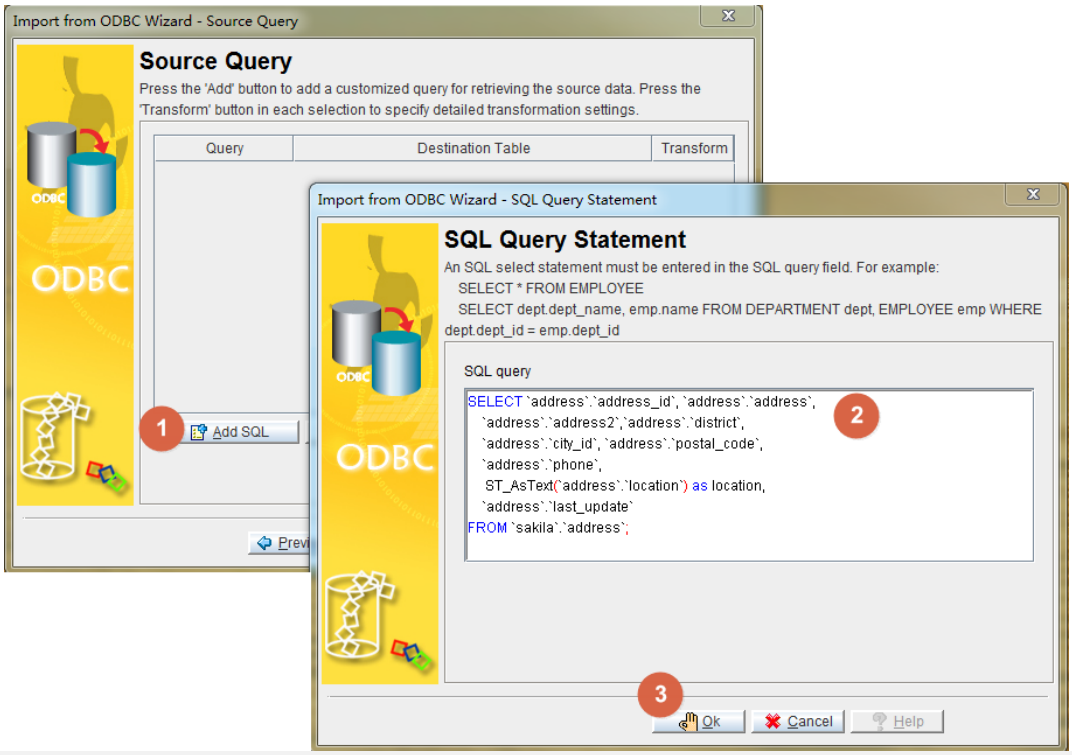
ADDR...	ADDRESS	ADDR...	DISTRICT	CITY_ID	POSTAL_...	PHONE	LOCATION	...	LAST_UPDATE
1	47 MySakila Drive	(NULL)	Alberta	300			POINT(-112.8185647 49.6999986)	...	2014-09-25 22:30:27
2	28 MySQL Boulevard	(NULL)	QLD	576			POINT(153.1408538 -27.6333361)	...	2014-09-25 22:30:09
3	23 Workhaven Lane	(NULL)	Alberta	300		14033335568	POINT(-112.8185673 49.6999951)	...	2014-09-25 22:30:27
4	1411 Lillydale Drive	(NULL)	QLD	576		6172235589	POINT(153.1913094 -27.6333373)	...	2014-09-25 22:30:09
5	1913 Hanoi Way		Nagasaki	463	35200	28303384290	POINT(129.7227851 33.1591726)	...	2014-09-25 22:31:53

You are now back at the Migrate Database Wizard, repeat the steps above 1~4.

- Step 5: Select the "SQL query" radiobutton and click **Next**.

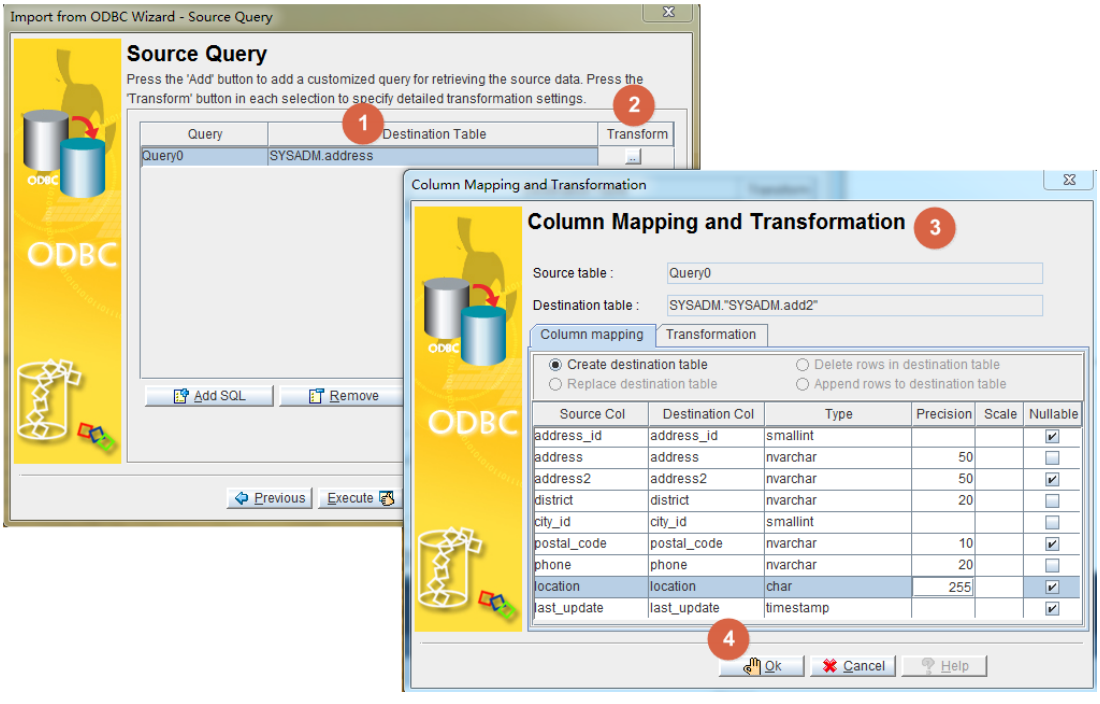


- Step 6: Enter the following SQL Query Statement to define the Source Query

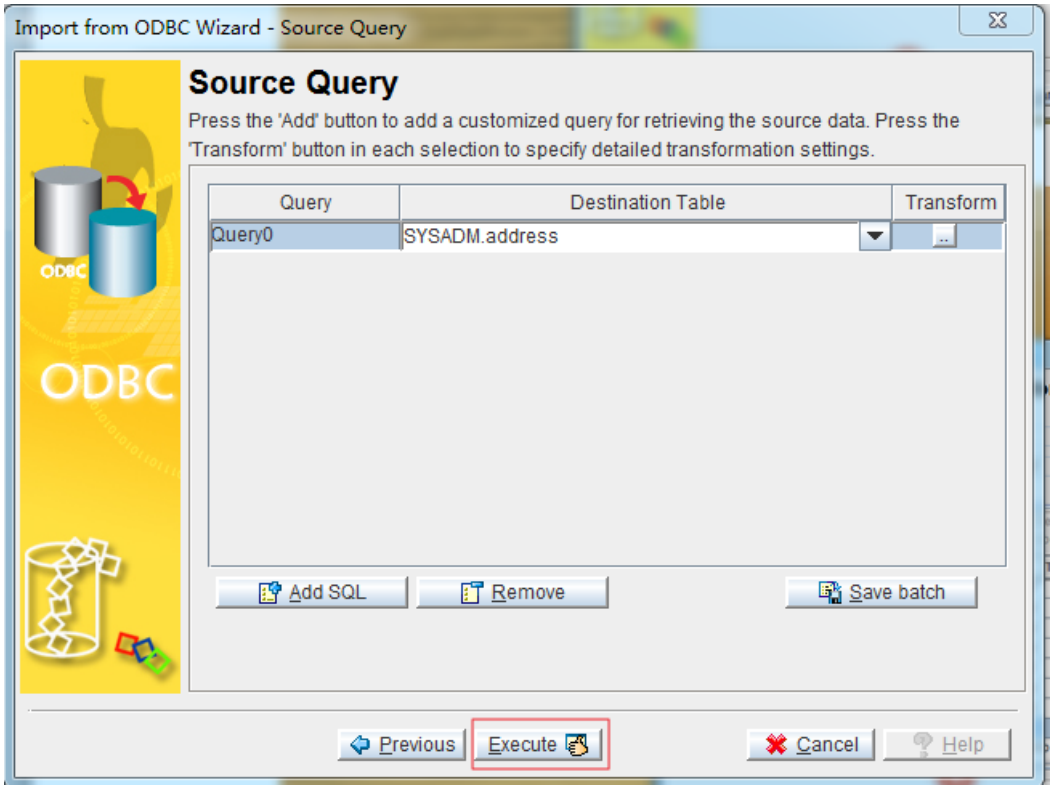



```
SELECT `address`.`address_id`, `address`.`address`,
    `address`.`address2`,`address`.`district`,
    `address`.`city_id`, `address`.`postal_code`,
    `address`.`phone`,
     ST_AsText(`address`.`location`) as location,
    `address`.`last_update`
FROM `sakila`.`address`;
```

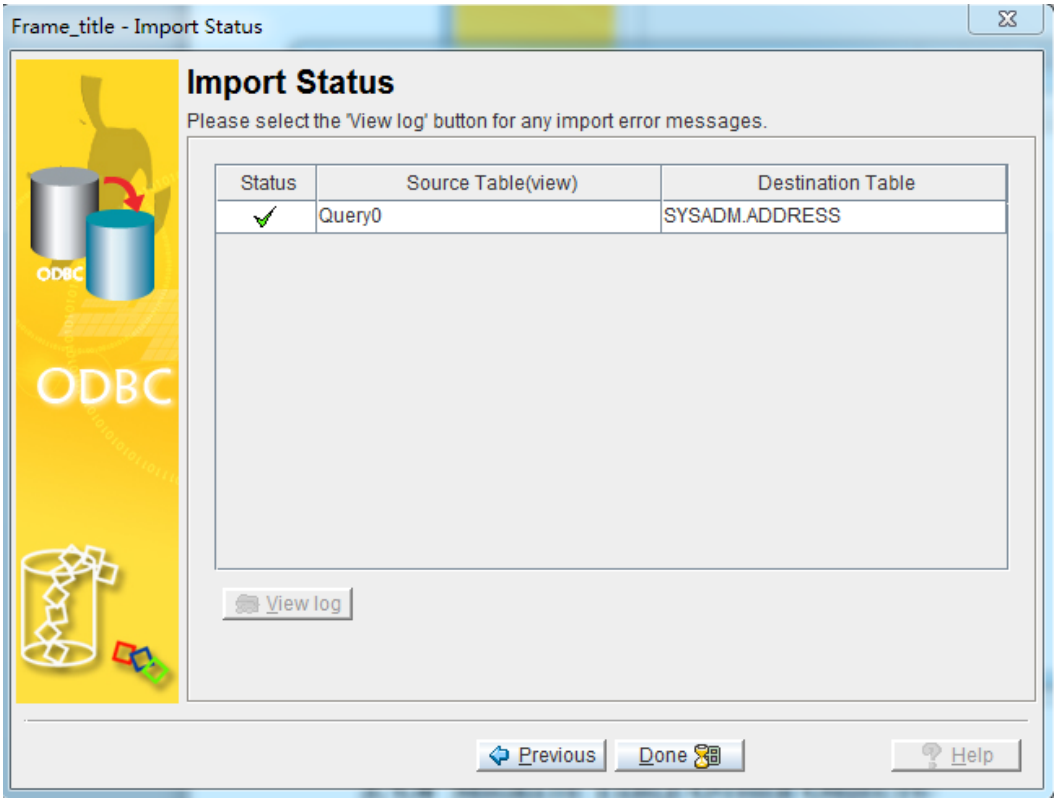
- Step 7: Check the correctness of the steps according to the number tags 1 through 4



- Step 8: Click **Execute** to begin the migration. DBMaker wizard will issue an error if there are problems with the SELECT statement.



- Step 9: Complete the wizard by clicking **Done**.



How to migrate data from the Film table?

The Film table of the MySQL sakila database contains **year** data types,dbmaker are automatically converted to int by default, but the data format is unreasonable.

Therefore, you need to manually convert the **year** type to char (4)

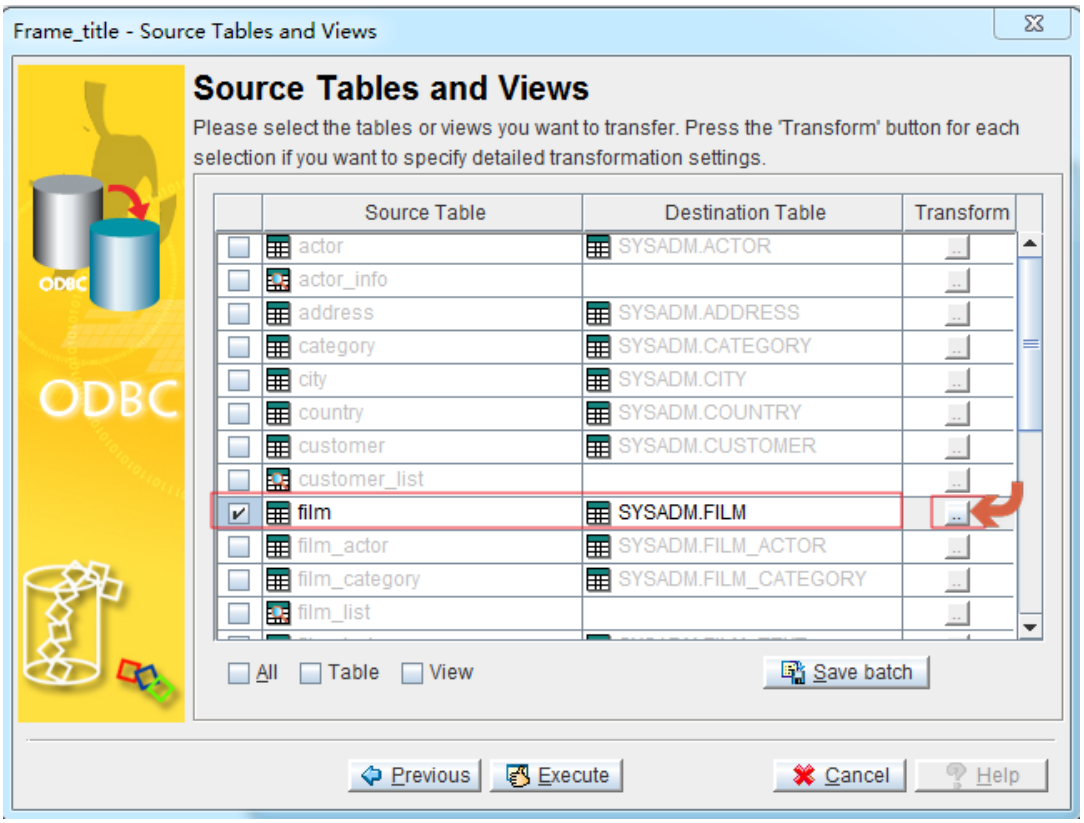
Edit the **jsyscom_dbmaker.xml** file, and add the following bold content

```
<?xml version="1.0" encoding="UTF-8"?>
<syscom language_choice="0" login_db_name="SAKILA" login_id_name="SYSADM">
<datatransfer>
<frame import_odbc_option="3" xpos="717" ypos="348"/>
<datatype dmType="char" precision="4" type="year" udf=""/>
</datatransfer>
</syscom>
```

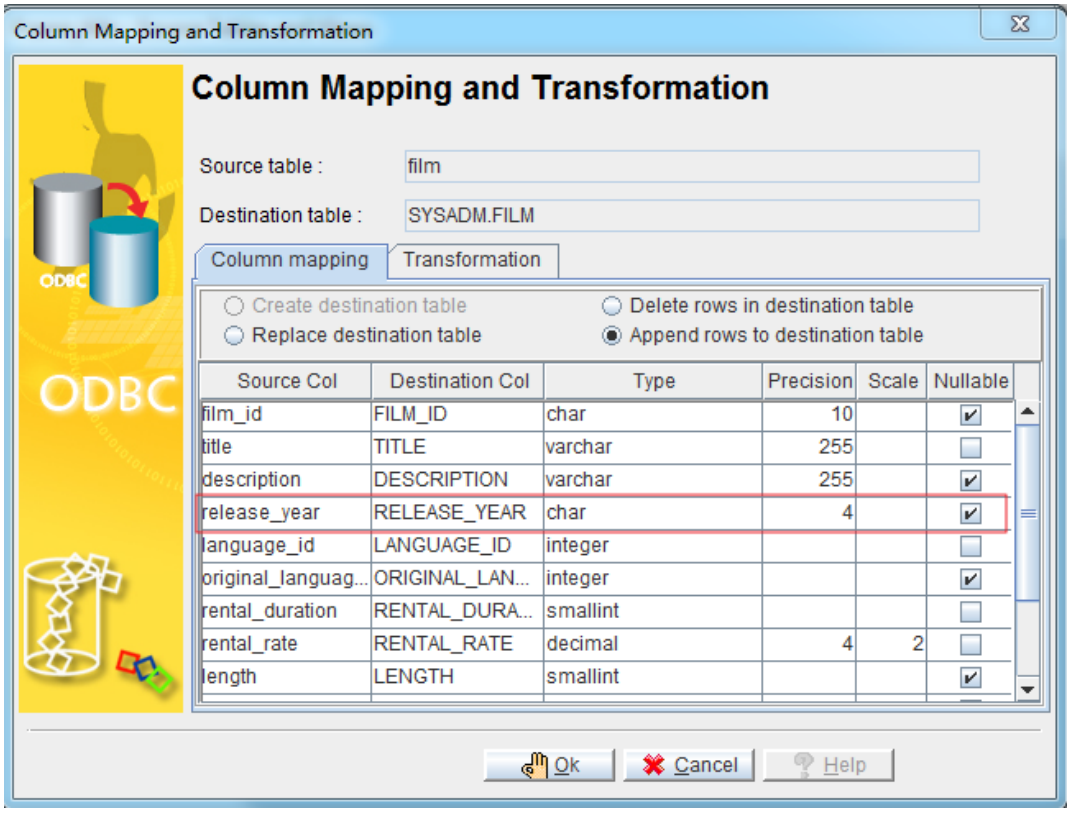
Note: jsyscom_dbmaker.xml is the dbmaker jtools configuration file, located in the DBMaker installation root directory, which automatically generates.

You need to reopen JData Transfer Tool, and repeat the steps above 1~5.

- Step 6: Select the Source Table "film" and click **Transfrorm** button.

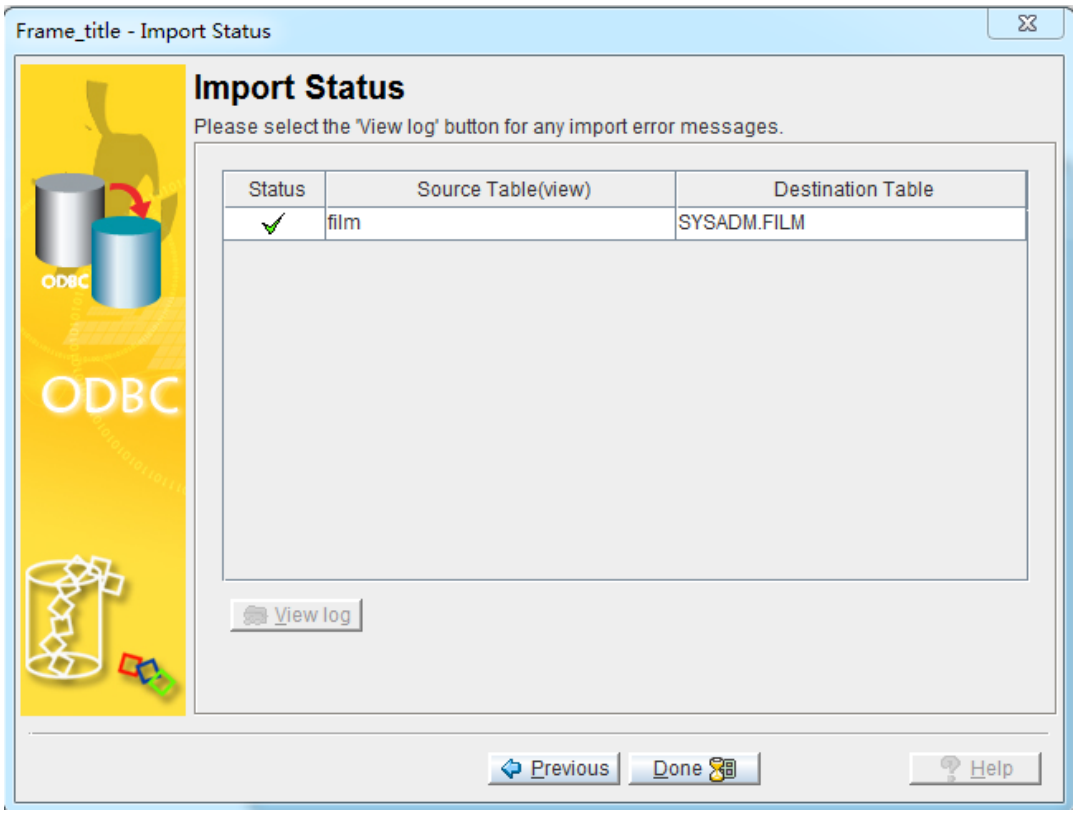


Check if the release_year column is mapped correctly or not



- Step 7: Click **Execute** to begin the migration

- Step 8: Click **Done** to close the wizard and end the migration process.



3.1.4 MIGRATE CONSTRAINTS OBJECTS

Primary keys. Syntax and behavior are very similar.

MySQL

Syntax: PRIMARY KEY (`actor_id`)

DBMaker

ALTER TABLE "SYSADM"."ACTOR" PRIMARY KEY (ACTOR_ID);

Foreign keys. Syntax and behavior are not the same.

MySQL

Syntax: CONSTRAINT "fk_address_city"

DBMaker

alter table ADDRESS foreign key "fk_address_city" (CITY_ID) references SYSADM.CITY("CITY_ID") on update CASCADE on delete CASCADE;

Indexes. Syntax and behavior are not the same.

MySQL

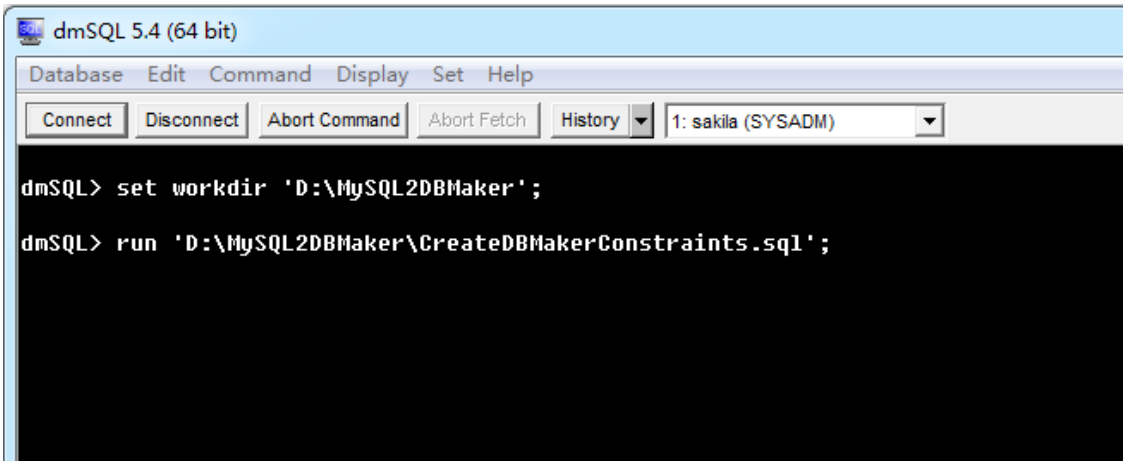
Syntax: KEY "idx_actor_last_name" ("last_name")

DBMaker

create index idx_actor_last_name on SYSADM.ACTOR (LAST_NAME);

This script takes care of creating all indexes and key relationships to match the schema of the MySQL Sakila database. To run the script the database must be started.

See the Chapter 7.2.3 for CreateDBMaker Constraints.sql

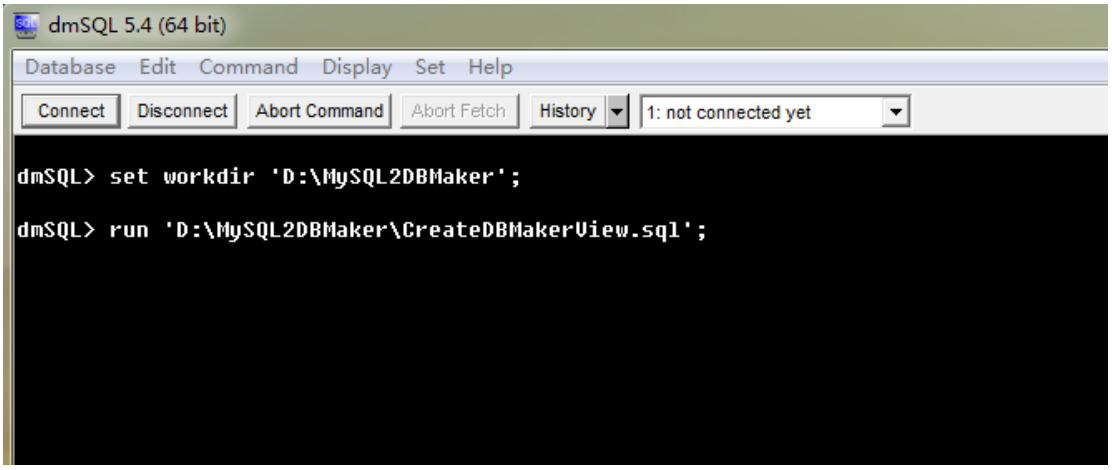


3.1.5 MIGRATE VIEW OBJECTS

There are seven views defined in the MySQL Sakila database. The queries used to generate the views in MySQL are very similar to the ones in DBMaker, requiring some modifications. But actor_info and sales_by_store could not be converted successfully due to some limitations.

This script is launched with the following command in dmsql tool. To run the script the database must be started.

See Chapter 7.2.2 for CreateDBMakerView.sql

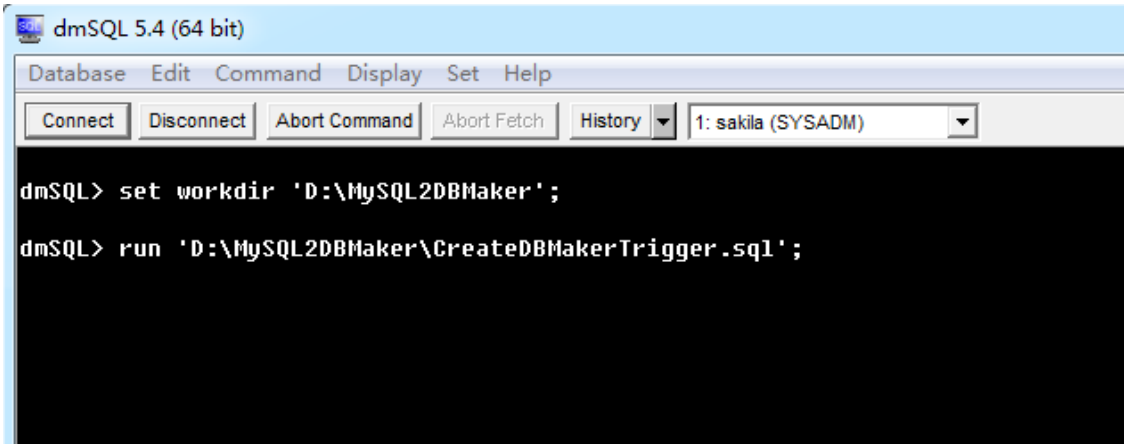


3.1.6 MIGRATE TRIGGER OBJECTS

There are six triggers defined in the MySQL Sakila database, the syntax and behavior of trigger are very similar, only requiring a few small modifications.

This script is launched with the following command in dmsql tool. To run the script the database must be started.

See Chapter 7.2.4 for CreateDBMakerTrigger.sql

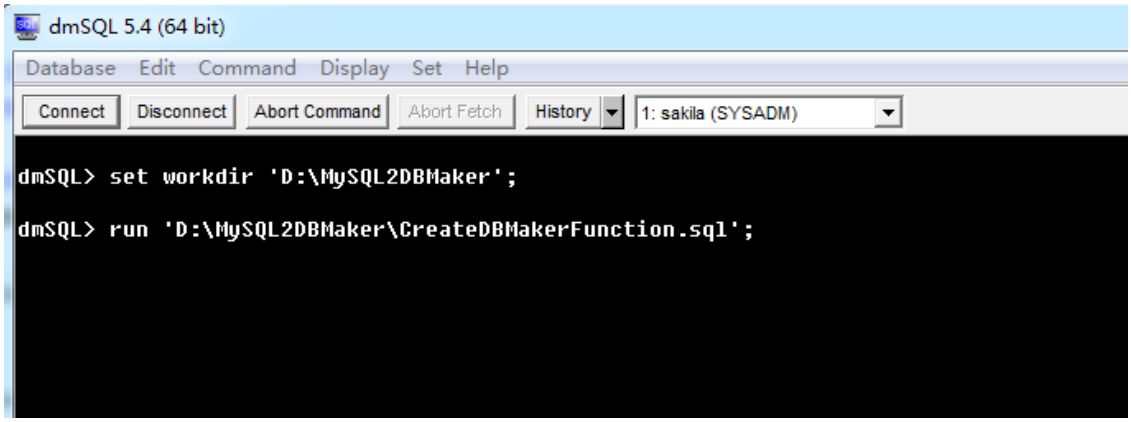


3.1.7 MIGRATE FUNCTION OBJECTS

There are three functions defined in the MySQL Sakila database. The syntax and behavior of MySQL Function are very different from DBMaker. DBMaker cannot convert them, but you can convert Function to a stored procedure with the same behavior.

This script is launched with the following command in dmsql tool. To run the script the database must be started.

See Chapter 7.2.5 for CreateDBMakerFunction.sql

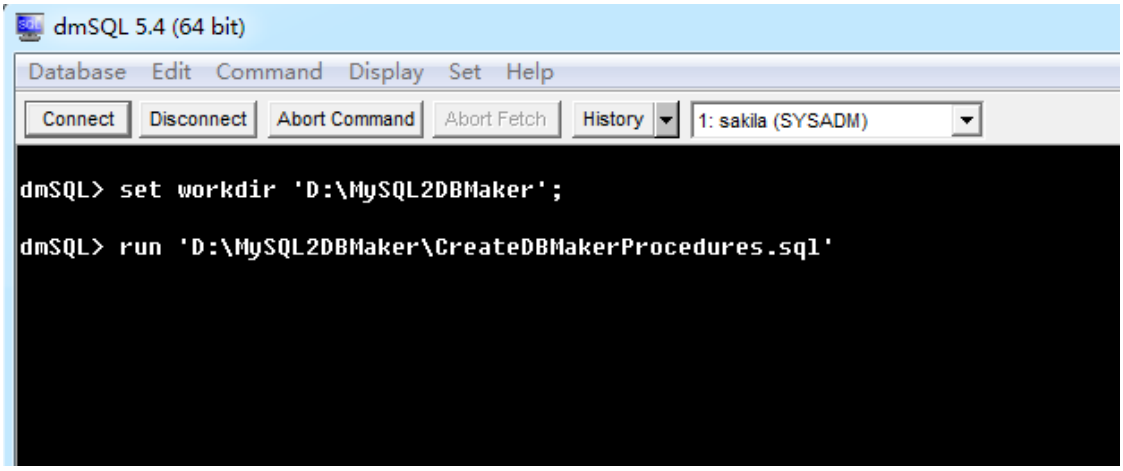


3.1.8 MIGRATE STORED PROCEDURE OBJECTS

There are three stored procedures defined in the MySQL Sakila database. Migrating these to DBMaker database requires some modifications.

This script is launched with the following command in dmsql tool. To run the script the database must be started.

See Chapter 7.2.6 for CreateDBMakerProcedures.sql



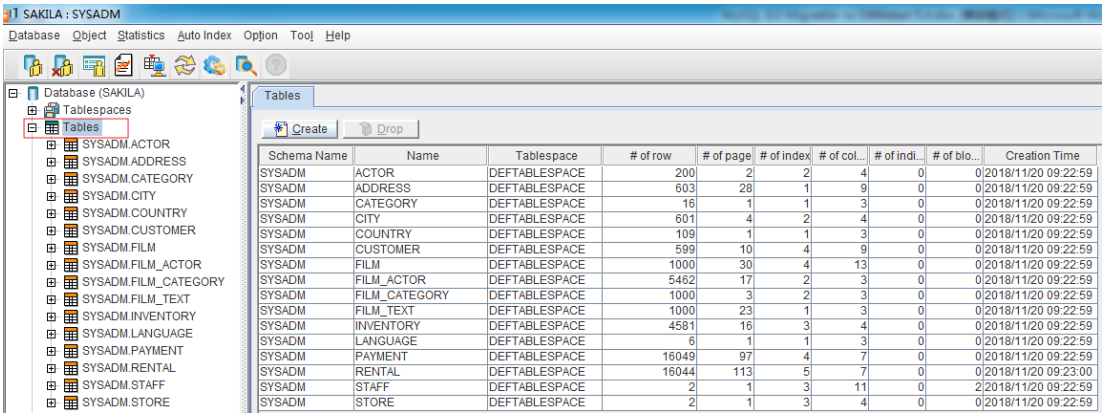
3.2 Migration Validating

First, it is important to find out which objects must be validated in the destination database after migration is complete:

- Tables
- Views
- Data
- Indexes
- Foreign keys
- Triggers
- Functions
- Stored Procedures

3.2.1 TABLES

In JDBATool highlight the Tables in the left pane and go to 'Tables' tab



Schema Name	Name	Tablespace	# of row	# of page	# of index	# of col...	# of indi...	# of blo...	Creation Time
SYSADM	ACTOR	DEFTABLESPACE	200	2	2	4	0	0	2018/11/20 09:22:59
SYSADM	ADDRESS	DEFTABLESPACE	603	28	1	9	0	0	2018/11/20 09:22:59
SYSADM	CATEGORY	DEFTABLESPACE	16	1	1	3	0	0	2018/11/20 09:22:59
SYSADM	CITY	DEFTABLESPACE	601	4	2	4	0	0	2018/11/20 09:22:59
SYSADM	COUNTRY	DEFTABLESPACE	109	1	1	3	0	0	2018/11/20 09:22:59
SYSADM	CUSTOMER	DEFTABLESPACE	599	10	4	9	0	0	2018/11/20 09:22:59
SYSADM	FILM	DEFTABLESPACE	1000	30	4	13	0	0	2018/11/20 09:22:59
SYSADM	FILM_ACTOR	DEFTABLESPACE	5462	17	2	3	0	0	2018/11/20 09:22:59
SYSADM	FILM_CATEGORY	DEFTABLESPACE	1000	3	2	3	0	0	2018/11/20 09:22:59
SYSADM	FILM_TEXT	DEFTABLESPACE	1000	23	1	3	0	0	2018/11/20 09:22:59
SYSADM	INVENTORY	DEFTABLESPACE	4581	16	3	4	0	0	2018/11/20 09:22:59
SYSADM	LANGUAGE	DEFTABLESPACE	6	1	1	3	0	0	2018/11/20 09:22:59
SYSADM	PAYMENT	DEFTABLESPACE	16049	97	4	7	0	0	2018/11/20 09:22:59
SYSADM	RENTAL	DEFTABLESPACE	16044	113	5	7	0	0	2018/11/20 09:23:00
SYSADM	STAFF	DEFTABLESPACE	2	1	3	11	0	2	2018/11/20 09:22:59
SYSADM	STORE	DEFTABLESPACE	2	1	3	4	0	0	2018/11/20 09:22:59

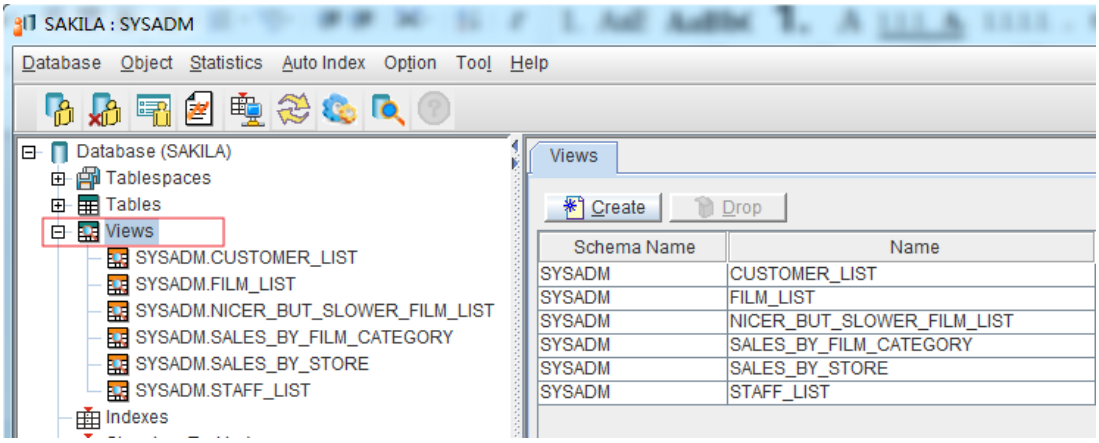
Or

Using the following query:

```
SELECT TABLE_OWNER, TABLE_NAME FROM SYSTEM.SYSTABLE WHERE
TABLE_OWNER = 'SYSADM' AND TABLE_TYPE='TABLE';
```

3.2.2 VIEWS

In JDBATool highlight the Views in the left pane and go to 'Views' tab



Or

Using the following query:

```
SELECT TABLE_NAME FROM SYSTEM.SYSTABLE WHERE TABLE_TYPE='VIEW';
```

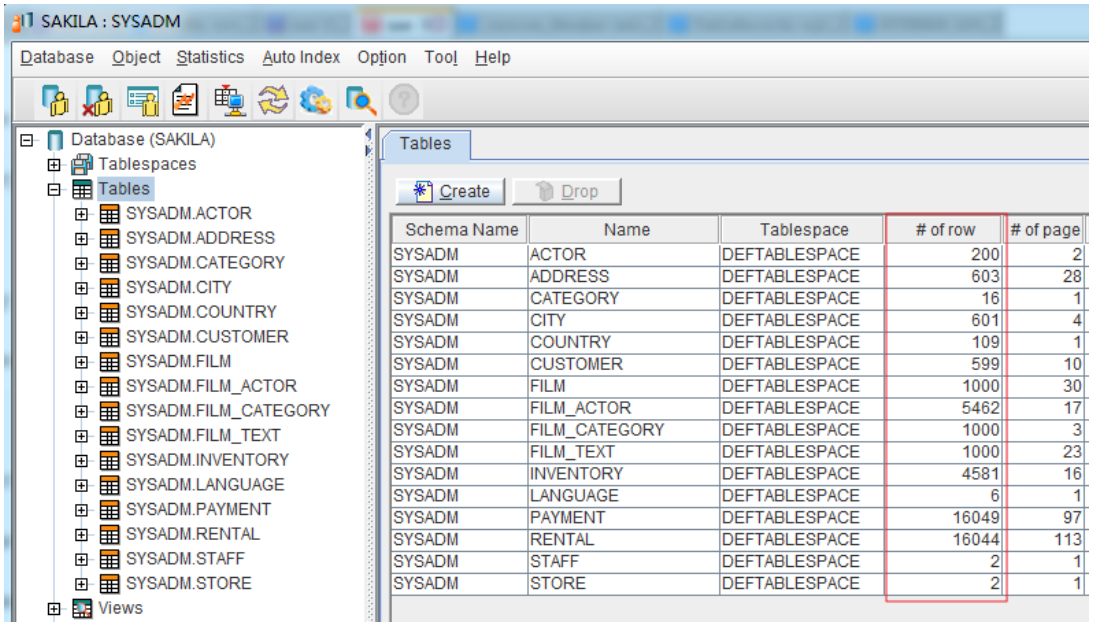
3.2.3 DATA

Also, it is necessary to verify that MySQL and DBMaker tables have the same count of rows. Both DBMS allows getting number of rows in a table using the following query:

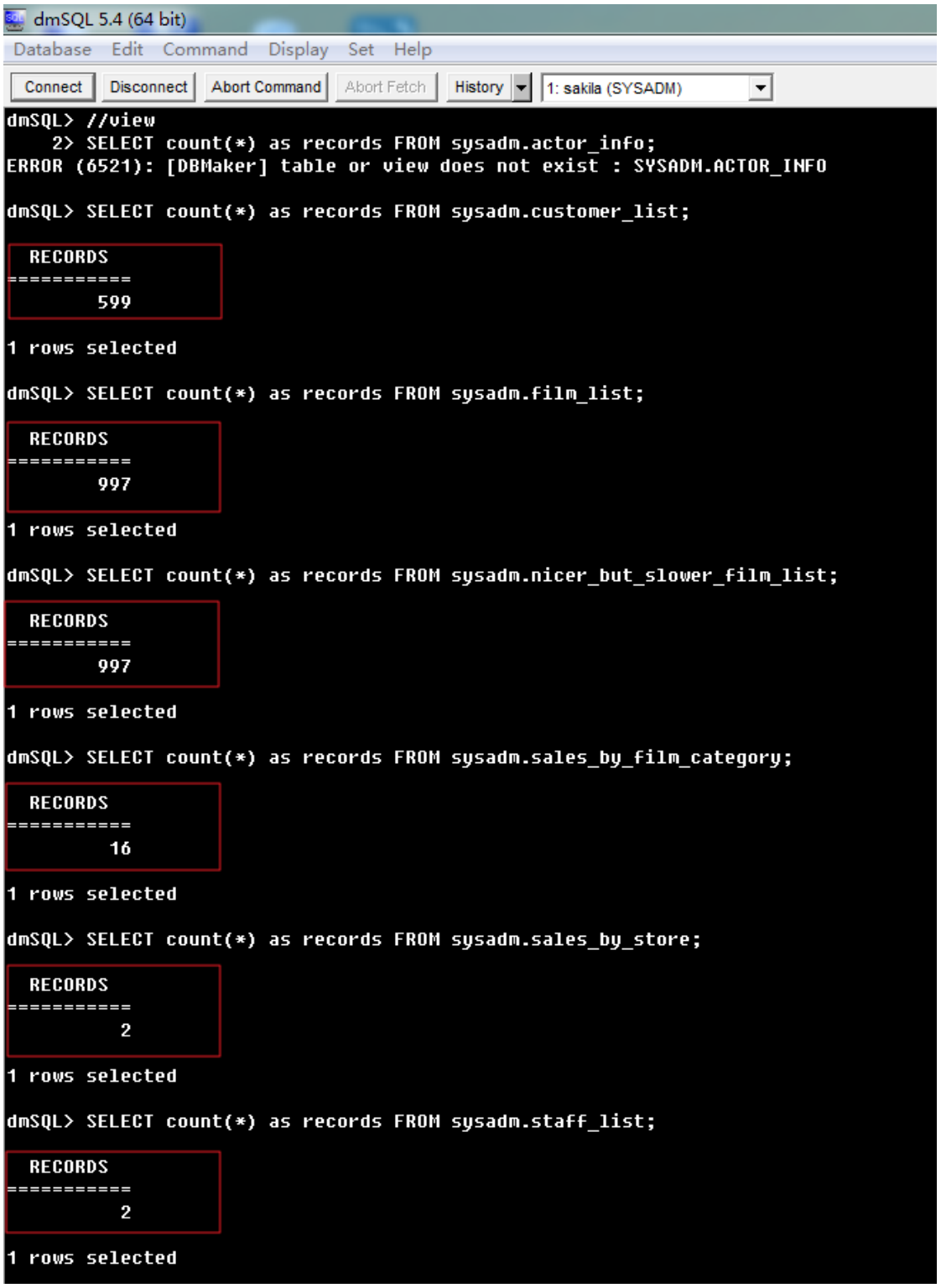
```
SELECT COUNT(*) FROM table_name/view_name;
```

Or

In JDBATool highlight the Tables in the left pane and go to 'Tables' tab

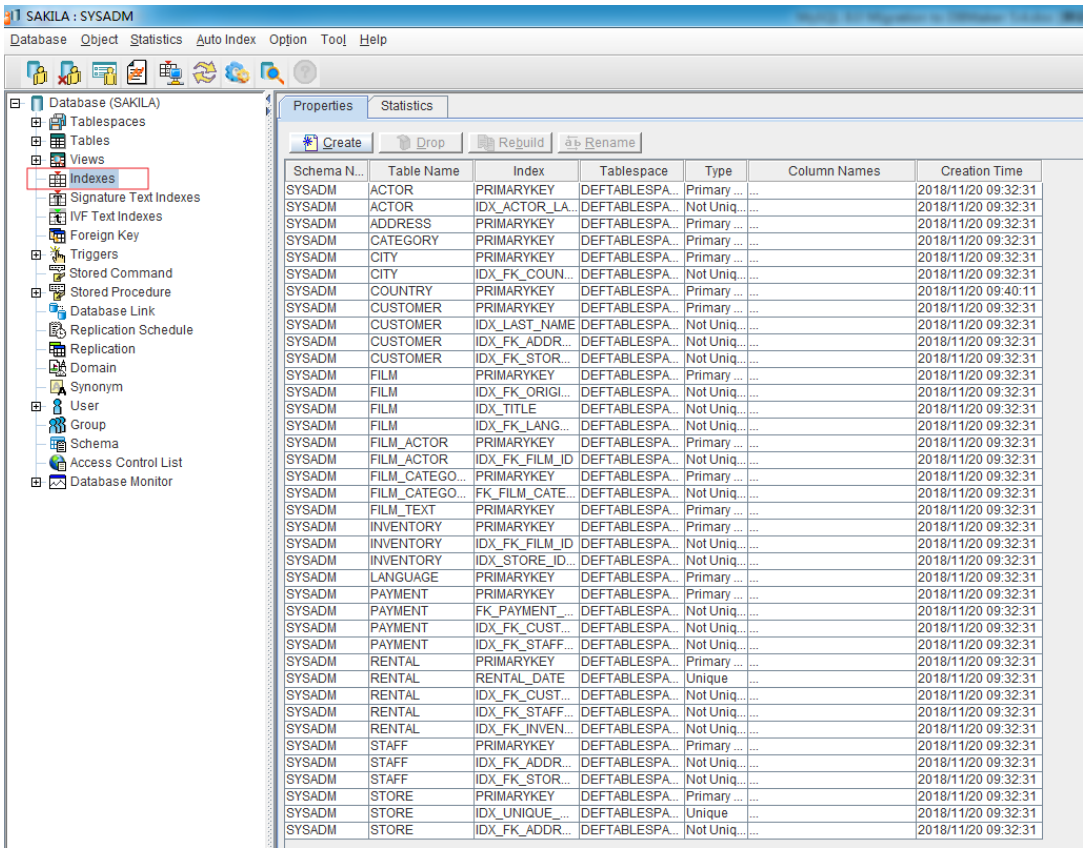


You can also execute SQL queries to ensure that the result set is correct.



3.2.4 INDEXES

In JDBATool highlight the Index in the left pane and go to 'Properties' tab



Schema N...	Table Name	Index	Tablespace	Type	Column Names	Creation Time
SYSADM	ACTOR	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	ACTOR	IDX_ACTOR_LA...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	ADDRESS	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	CATEGORY	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	CITY	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	CITY	IDX_FK_COUN...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	COUNTRY	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:40:11
SYSADM	CUSTOMER	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	CUSTOMER	IDX_LAST_NAME	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	CUSTOMER	IDX_FK_ADDR...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	CUSTOMER	IDX_FK_STOR...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	FILM	IDX_FK_ORIGI...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM	IDX_TITLE	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM	IDX_FK_LANG...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM_ACTOR	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	FILM_ACTOR	IDX_FK_FILM_ID	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM_CATEGO...	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	FILM_CATEGO...	FK_FILM_CATE...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	FILM_TEXT	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	INVENTORY	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	INVENTORY	IDX_FK_FILM_ID	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	INVENTORY	IDX_STORE_ID...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	LANGUAGE	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	PAYMENT	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	PAYMENT	FK_PAYMENT_...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	PAYMENT	IDX_FK_CUST...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	PAYMENT	IDX_FK_STAFF...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	RENTAL	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	RENTAL	RENTAL_DATE	DEFTABLESPA...	Unique	...	2018/11/20 09:32:31
SYSADM	RENTAL	IDX_FK_CUST...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	RENTAL	IDX_FK_STAFF...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	RENTAL	IDX_FK_INVEN...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	STAFF	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	STAFF	IDX_FK_ADDR...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	STAFF	IDX_FK_STOR...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31
SYSADM	STORE	PRIMARYKEY	DEFTABLESPA...	Primary ...	...	2018/11/20 09:32:31
SYSADM	STORE	IDX_UNIQUE_...	DEFTABLESPA...	Unique	...	2018/11/20 09:32:31
SYSADM	STORE	IDX_FK_ADDR...	DEFTABLESPA...	Not Uniq...	...	2018/11/20 09:32:31

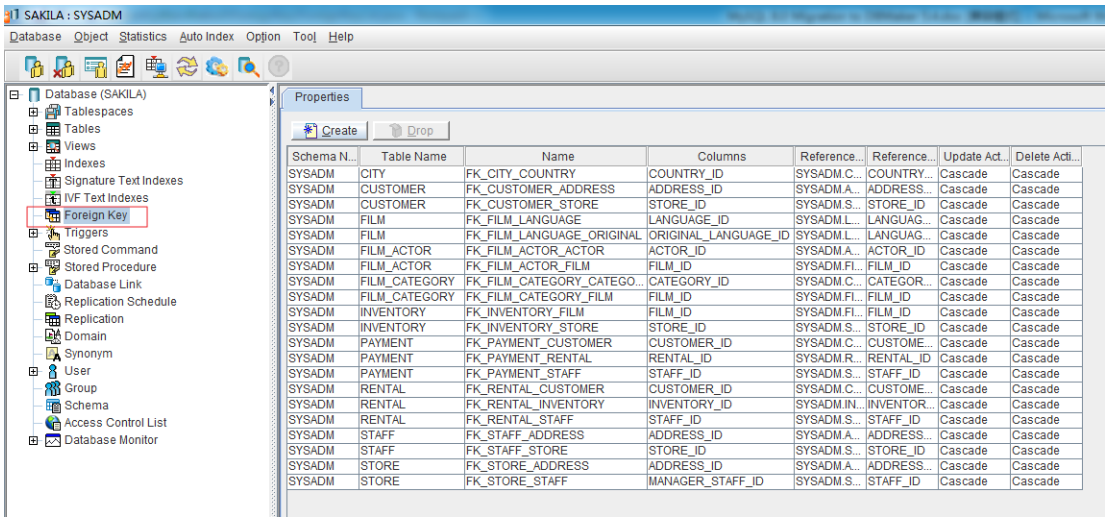
Or

Using the following query:

```
SELECT INDEX_NAME FROM SYSTEM.SYSINDEX WHERE TABLE_OWNER='SYSADM';
```

3.2.5 FOREIGN KEYS

In JDBATool highlight the Foreign key in the left pane and go to 'Properties' tab



Schema N...	Table Name	Name	Columns	Reference...	Reference...	Update Act...	Delete Acti...
SYSADM	CITY	FK_CITY_COUNTRY	COUNTRY_ID	SYSADM.C...	COUNTRY...	Cascade	Cascade
SYSADM	CUSTOMER	FK_CUSTOMER_ADDRESS	ADDRESS_ID	SYSADM.A...	ADDRESS...	Cascade	Cascade
SYSADM	CUSTOMER	FK_CUSTOMER_STORE	STORE_ID	SYSADM.S...	STORE_ID	Cascade	Cascade
SYSADM	FILM	FK_FILM_LANGUAGE	LANGUAGE_ID	SYSADM.L...	LANGUAG...	Cascade	Cascade
SYSADM	FILM	FK_FILM_LANGUAGE_ORIGINAL	ORIGINAL_LANGUAGE_ID	SYSADM.L...	LANGUAG...	Cascade	Cascade
SYSADM	FILM_ACTOR	FK_FILM_ACTOR_ACTOR	ACTOR_ID	SYSADM.A...	ACTOR_ID	Cascade	Cascade
SYSADM	FILM_ACTOR	FK_FILM_ACTOR_FILM	FILM_ID	SYSADM.FI...	FILM_ID	Cascade	Cascade
SYSADM	FILM_CATEGORY	FK_FILM_CATEGORY_CATEGO...	CATEGORY_ID	SYSADM.C...	CATEGOR...	Cascade	Cascade
SYSADM	FILM_CATEGORY	FK_FILM_CATEGORY_FILM	FILM_ID	SYSADM.FI...	FILM_ID	Cascade	Cascade
SYSADM	INVENTORY	FK_INVENTORY_FILM	FILM_ID	SYSADM.FI...	FILM_ID	Cascade	Cascade
SYSADM	INVENTORY	FK_INVENTORY_STORE	STORE_ID	SYSADM.S...	STORE_ID	Cascade	Cascade
SYSADM	PAYMENT	FK_PAYMENT_CUSTOMER	CUSTOMER_ID	SYSADM.C...	CUSTOME...	Cascade	Cascade
SYSADM	PAYMENT	FK_PAYMENT_RENTAL	RENTAL_ID	SYSADM.R...	RENTAL_ID	Cascade	Cascade
SYSADM	PAYMENT	FK_PAYMENT_STAFF	STAFF_ID	SYSADM.S...	STAFF_ID	Cascade	Cascade
SYSADM	RENTAL	FK_RENTAL_CUSTOMER	CUSTOMER_ID	SYSADM.C...	CUSTOME...	Cascade	Cascade
SYSADM	RENTAL	FK_RENTAL_INVENTORY	INVENTORY_ID	SYSADM.IN...	INVENTOR...	Cascade	Cascade
SYSADM	RENTAL	FK_RENTAL_STAFF	STAFF_ID	SYSADM.S...	STAFF_ID	Cascade	Cascade
SYSADM	STAFF	FK_STAFF_ADDRESS	ADDRESS_ID	SYSADM.A...	ADDRESS...	Cascade	Cascade
SYSADM	STAFF	FK_STAFF_STORE	STORE_ID	SYSADM.S...	STORE_ID	Cascade	Cascade
SYSADM	STORE	FK_STORE_ADDRESS	ADDRESS_ID	SYSADM.A...	ADDRESS...	Cascade	Cascade
SYSADM	STORE	FK_STORE_STAFF	MANAGER_STAFF_ID	SYSADM.S...	STAFF_ID	Cascade	Cascade

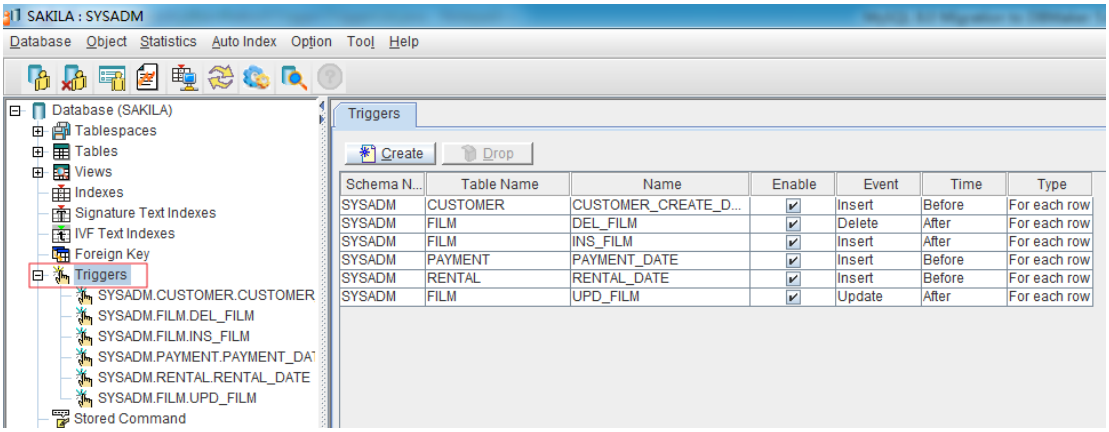
Or

Using the following query:

```
SELECT FK_NAME, FK_TBL_OWNER, FK_TBL_NAME FROM
SYSTEM.SYSFOREIGNKEY;
```

3.2.6 TRIGGERS

In JDBATool highlight the Triggers in the left pane and go to 'Triggers' tab



Or

Using the following query:

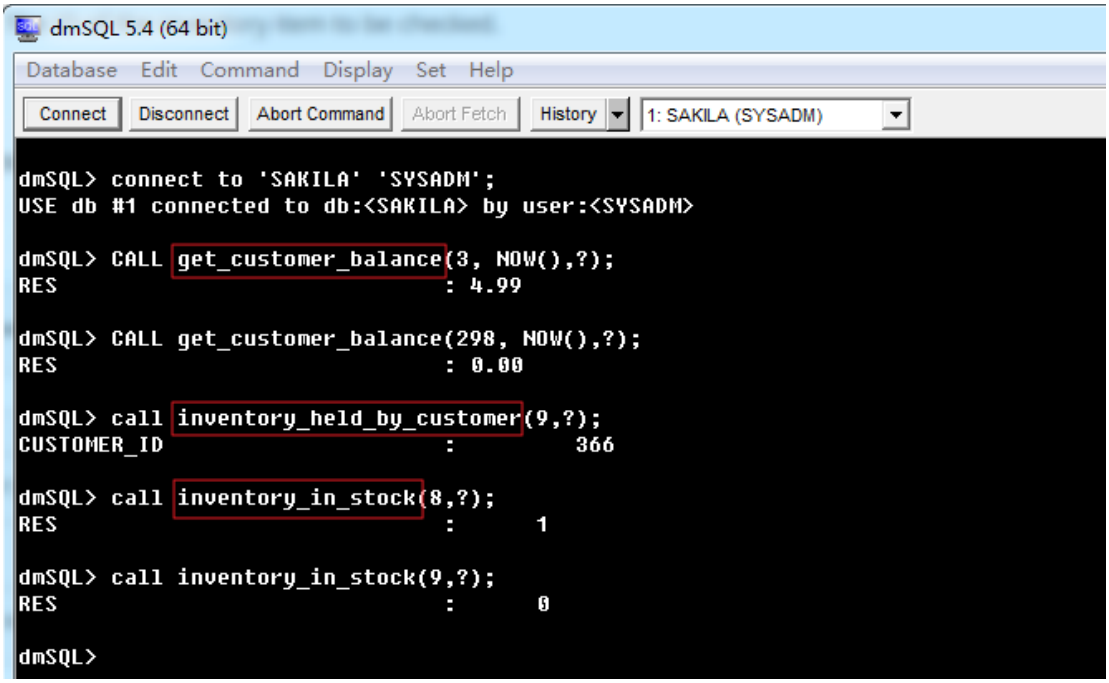
```
SELECT TBOWNER, TBNAME, TRIGNAME from SYSTEM.SYSTRIGGER where
RESERVE1 != 1;
```

3.2.7 FUNCTIONS

Three functions are converted to stored procedures, and using call XXX () to verify correct Behaviors.

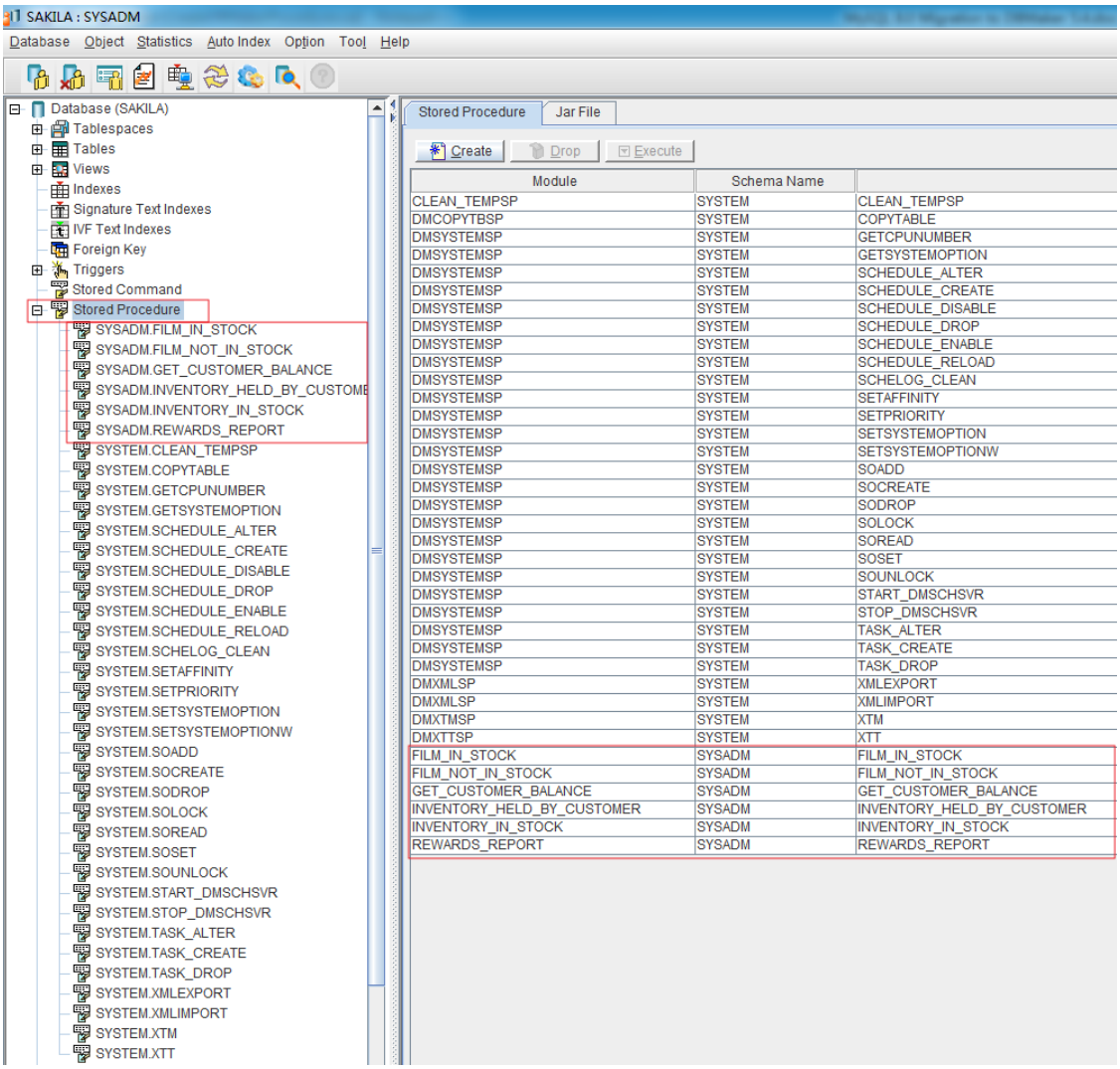
See Chapters 3.1.7 Migrate Function Objects

At a minimum, you should execute a few SQL queries to ensure the result sets are correct.



3.2.8 STORED PROCEDURES

In JDBATool highlight the Stored Procedure in the left pane and go to 'Stored Procedure' tab

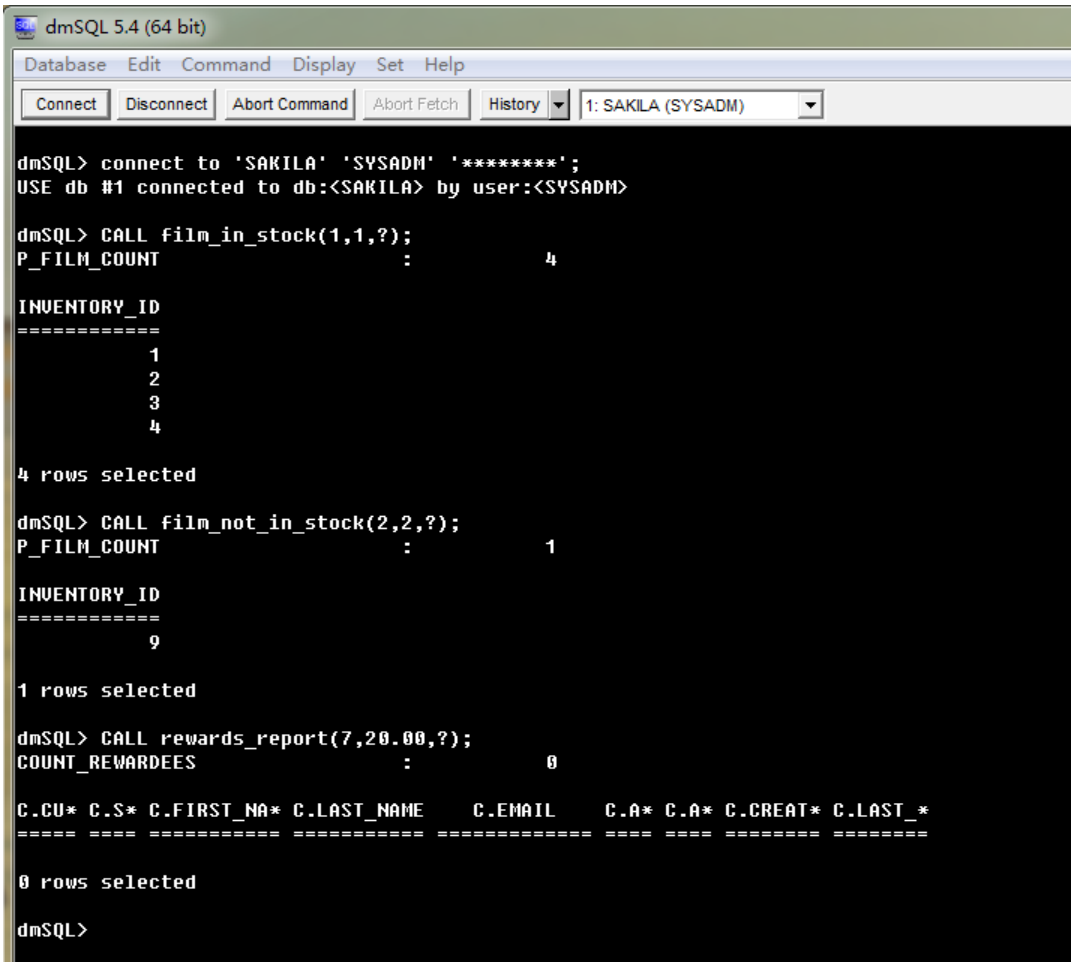


Or

Using the following query:

```
SELECT PROC_NAME FROM SYSTEM.SYSPROCINFO WHERE
PROC_OWNER='SYSADM';
```

At a minimum, you should execute a few SQL queries to ensure the result sets are correct.


dmSQL 5.4 (64 bit)
Database Edit Command Display Set Help
Connect Disconnect Abort Command Abort Fetch History 1: SAKILA (SYSADM)
dmSQL> connect to 'SAKILA' 'SYSADM' '********';
USE db #1 connected to db:<SAKILA> by user:<SYSADM>
dmSQL> CALL film_in_stock(1,1,?);
P_FILM_COUNT : 4
INVENTORY_ID
1
2
3
4
4 rows selected
dmSQL> CALL film_not_in_stock(2,2,?);
P_FILM_COUNT : 1
INVENTORY_ID
9
1 rows selected
dmSQL> CALL rewards_report(7,20.00,?);
COUNT_REWARDEES : 0
C.CU* C.S* C.FIRST_NA* C.LAST_NAME C.EMAIL C.A* C.A* C.CREAT* C.LAST_*
0 rows selected
dmSQL>

4. Migration Limitations

Although the migration sakila database is complete, it has the following limitations:

- **View are not migrated**

View: actor_info	*Not supported.* *Workaround:* *None*
View: sales_by_store	*ERROR (6145): [DBMaker] non-aggregate column must be in the GROUP BY list or ORDER BY's column must be in the projection list* *Workaround:* *Remove* *ORDER BY "cy"."country" , "c"."city"*

- **Datatypes limitations**

ENUM datatype *"rating" enum('G','PG','PG-13','R','NC-17') DEFAULT 'G'*	*Not supported.* *Workaround:* *Rating char(X)*
Set datatype *"special_features" set('Trailers','Commentaries','Deleted Scenes','Behind the Scenes')*	*Not supported.* *Workaround:* *"special_features" char(X)*
geometry datatypes *Note: MySQL server 5.7.5 and above*	*Not supported.* *Workaround:* *See the Chapter 3.1.3*

- **Index limitations**

SPATIAL KEY `idx_location` (`location`) *Note: MySQL server 5.7.5 and above*	*Not supported*

- **Function limitations**

User define Function: *get_customer_balance* *inventory_held_by_customer* *inventory_in_stock*	*Not supported.* *Workaround:* *can only be converted to stored procedure with the same behavior*

- **Other limitations**

[ON DELETE reference_option] *[ON UPDATE reference_option]* *reference_option: CASCADE* *| SET NULL* *| SET DEFAULT* *| NO ACTION* *| **RESTRICT***	*There are four available options in DBMaker.* *delete-rule ::= ON DELETE CASCADE* *| SET NULL* *| SET DEFAULT* *| NO ACTION*

5. Summary

DBMaker provides a Database Migration Wizard and data type mapping to quickly enable migration of data from different databases, such as MySQL. Migrating table objects and data using the wizards is fairly the easiest way.

Another, migrate views and database logic, a manual approach is more appropriate due to the differences in the SQL syntax between DBMaker and other databases. Fortunately, there are many similarities in the SQL script which makes the migration process easier.

We will continue to improve the automated migration tool.

6. References

1. *MySQL Documentation*

   https://dev.mysql.com/doc/refman/8.0/en/

2. *MySQL Sakila Sample Database Documentation*

   https://dev.mysql.com/doc/sakila/en/sakila-usage.html

3. *Database reference - sakila*

   http://elsasoft.com/samples/mysql_sakila/MySQL.localhost.sakila/default.htm

4. *DBMaker Documentation*

   DBMaker Database Administrator's Guide

   dmSQL User's Guide

   DBMaker SQL Stored Procedure User's Guide

   DBMaker SQL Command and Function Reference manual

   JDBA Tool User's Guide

7. Appendix for DBMaker Client

7.1 Mapping Data Types

DBMaker data types are described in the Internal Data Types section of DBMaker SQL.pdf.

MySQL data types are fully described in MySQL documentation.

MySQL and DBMaker have similar data types. Some of them are equivalent while others are not. When planning MySQL to DBMaker migration it is important to remember the following table of the correct types mapping.

Table-1 Mapping MySQL Data Types to DBMaker Data Types

Data Type	MySQL 8.0	DBMaker 5.4	Description
Numeric Types	TINYINT	SMALLINT	
	SMALLINT	SMALLINT	
	MEDIUMINT	INTEGER	
	INT, INTEGER	INT, INTEGER	
	BIGINT	BIGINT	
	DECIMAL(p,s) , DEC(p,s)	DECIMAL(p,s)	
	FLOAT	FLOAT	
	DOUBLE	DOUBLE	
	BIT	BINARY(n)	
String Types	CHAR	CHAR(n)	
	VARCHAR(n)	VARCHAR(n)	
	BINARY	BINARY(n)	
	VARBINARY	BINARY(n)	
	TINYBLOB	BLOB	
	BLOB	BLOB	
	MEDIUMBLOB	BLOB	
	LONGBLOB	BLOB	
	TINYTEXT	CLOB	

	TEXT	CLOB	
	MEDIUMTEXT	CLOB	
	LONGTEXT	CLOB	
	ENUM	CHAR(n)	
	SET	CHAR(n)	Rarely used
Date and Time Types	DATE	DATE	DBMaker does not allow to store '0000-00-00' into date columns.
	TIME	TIME	
	DATETIME	TIMESTAMP	
	TIMESTAMP	TIMESTAMP	
	YEAR	CHAR(4)	
Spatial Data Types	GEOMETRY	Not supported	A spatial value of any type
	POINT	Not supported	A point (a pair of X-Y coordinates)
	LINESTRING	Not supported	A curve (one or more POINT values)
	POLYGON	Not supported	A polygon
	GEOMETRYCOLLECTION	Not supported	A collection of GEOMETRYvalues
	MULTILINESTRING	Not supported	A collection of LINESTRINGvalues
	MULTIPOINT	Not supported	A collection of POINTvalues
	MULTIPOLYGON	Not supported	A collection of POLYGONvalues
JSON data type	JSON	Jsoncols	JSONCOLS Type is a column set of dynamic columns
AUTO_INCREMENT	BIGINT AUTO_INCREMENT	BIGSERIAL	DBMaker uses SERIAL type and its modifications for the same purpose.
	INTEGER AUTO_INCREMENT	SERIAL	
	SMALLINT AUTO_INCREMENT	SERIAL	
	TINYINT AUTO_INCREMENT	SERIAL	

UNSIGNED attribute	BIGINT UNSIGNED	BIGINT	DBMaker do not support the UNSIGNED attribute.
	INT UNSIGNED	BIGINT	
	MEDIUMINT UNSIGNED	INTEGER	
	SMALLINT UNSIGNED	INTEGER	
	TINYINT UNSIGNED	INTEGER	

7.2 Import DBMaker script files

In some instances, the same SQL statements will work properly in DBMaker database, but typically you will need to make a few modifications to ensure the business logic executes as expected.

7.2.1 SCRIPTS FOR CREATING TABLES

Save the following script as CreateDBMakerTable.sql

```
//Table structure for table "actor"
DROP TABLE IF EXISTS "actor";
CREATE TABLE "actor" (
  "actor_id" serial,
  "first_name" varchar(45) NOT NULL,
  "last_name" varchar(45) NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "category"
DROP TABLE IF EXISTS "category";
CREATE TABLE "category" (
  "category_id" serial,
  "name" varchar(25) NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "country"
DROP TABLE IF EXISTS "country";
CREATE TABLE "country" (
  "country_id" serial,
  "country" varchar(50) NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "city"
DROP TABLE IF EXISTS "city";
CREATE TABLE "city" (
  "city_id" serial,
  "city" varchar(50) NOT NULL,
  "country_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
);

//Table structure for table "language"
DROP TABLE IF EXISTS "language";
CREATE TABLE "language" (
  "language_id" serial,
  "name" char(20) NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;
```

```
//Table structure for table "address"
DROP TABLE IF EXISTS "address";
CREATE TABLE "address" (
  "address_id" serial,
  "address" varchar(50) NOT NULL,
  "address2" varchar(50) DEFAULT NULL,
  "district" varchar(20) NOT NULL,
  "city_id" int NOT NULL,
  "postal_code" varchar(10) DEFAULT NULL,
  "phone" varchar(20) NOT NULL,
  "location" char(255) NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "film"
DROP TABLE IF EXISTS "film";
CREATE TABLE "film" (
  "film_id" serial,
  "title" varchar(255) NOT NULL,
  "description" varchar(255),
  "release_year" char(4) DEFAULT NULL,
  "language_id" INT NOT NULL,
  "original_language_id" INT DEFAULT NULL,
  "rental_duration" smallint NOT NULL DEFAULT 3,
  "rental_rate" decimal(4,2) NOT NULL DEFAULT 4.99,
  "length" smallint DEFAULT NULL,
  "replacement_cost" decimal(5,2) NOT NULL DEFAULT 19.99,
  "rating" varchar(20) DEFAULT 'G',
  "special_features" varchar(255) DEFAULT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "film_actor"
DROP TABLE IF EXISTS "film_actor";
CREATE TABLE "film_actor" (
  "actor_id" int NOT NULL,
  "film_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "film_category"
DROP TABLE IF EXISTS "film_category";
CREATE TABLE "film_category" (
  "film_id" int NOT NULL,
  "category_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
);

//Table structure for table "film_text"
DROP TABLE IF EXISTS "film_text";
CREATE TABLE "film_text" (
  "film_id" smallint NOT NULL,
  "title" varchar(255) NOT NULL,
  "description" clob
) ;

//Table structure for table "staff"
DROP TABLE IF EXISTS "staff";
CREATE TABLE "staff" (
  "staff_id" serial,
  "first_name" varchar(45) NOT NULL,
```

```
  "last_name" varchar(45) NOT NULL,
  "address_id" int NOT NULL,
  "picture" blob,
  "email" varchar(50) DEFAULT NULL,
  "store_id" int  NOT NULL,
  "active" smallint NOT NULL DEFAULT 1,
  "username" varchar(16) NOT NULL,
  "password" varchar(40) DEFAULT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "store"
DROP TABLE IF EXISTS "store";
CREATE TABLE "store" (
  "store_id" serial,
  "manager_staff_id" int NOT NULL,
  "address_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "inventory"
DROP TABLE IF EXISTS "inventory";
CREATE TABLE "inventory" (
  "inventory_id" serial,
  "film_id" int NOT NULL,
  "store_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "payment"
DROP TABLE IF EXISTS "payment";
CREATE TABLE "payment" (
  "payment_id" serial,
  "customer_id" int NOT NULL,
  "staff_id" int NOT NULL,
  "rental_id" int DEFAULT NULL,
  "amount" decimal(5,2) NOT NULL,
  "payment_date" timestamp NOT NULL,
  "last_update" timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
);

//Table structure for table "customer"
DROP TABLE IF EXISTS "customer";
CREATE TABLE "customer" (
  "customer_id" serial,
  "store_id" int NOT NULL,
  "first_name" varchar(45) NOT NULL,
  "last_name" varchar(45) NOT NULL,
  "email" varchar(50) DEFAULT NULL,
  "address_id" int NOT NULL,
  "active" smallint NOT NULL DEFAULT 1,
  "create_date" timestamp NOT NULL,
  "last_update" timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;

//Table structure for table "rental"
DROP TABLE IF EXISTS "rental";
CREATE TABLE "rental" (
  "rental_id" serial,
  "rental_date" timestamp NOT NULL,
  "inventory_id" int NOT NULL,
  "customer_id" int NOT NULL,
```

```
  "return_date" timestamp DEFAULT NULL,
  "staff_id" int NOT NULL,
  "last_update" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
) ;
```

7.2.2 SCRIPTS FOR CREATING VIEWS

Save the following script as CreateDBMakerView.sql.

```
//view name : actor_info

//view name: customer_list
CREATE VIEW "customer_list" AS
        select
                "cu"."customer_id" AS "ID",
                concat("cu"."first_name","cu"."last_name") AS "name",
                "a"."address" AS "address",
                "a"."postal_code" AS "zip code",
                "a"."phone" AS "phone",
                "city"."city" AS "city",
                "country"."country" AS "country",
                CASE WHEN "cu"."active"=1 THEN 'active' WHEN "cu"."active"=0 THEN '' END  AS
"notes",
                "cu"."store_id" AS "SID"
        from
                ((("customer" "cu"
                join "address" "a" on(("cu"."address_id" = "a"."address_id")))
                join "city" on(("a"."city_id" = "city"."city_id")))
                join "country" on(("city"."country_id" = "country"."country_id")));

//view name : film_list
CREATE VIEW "film_list" AS
   SELECT
      "film"."film_id" AS "FID",
      "film"."title" AS "title",
      "film"."description" AS "description",
      "category"."name" AS "category",
      "film"."rental_rate" AS "price",
      "film"."length" AS "length",
      "film"."rating" AS "rating"  ,
       CAST(xmlagg( CONCAT(CONCAT(CONCAT("actor"."first_name",' '),"actor"."last_name"),',')) as
varchar(255)) AS "actors"
   FROM
      ((("category"
      LEFT JOIN "film_category" ON (("category"."category_id" = "film_category"."category_id")))
      LEFT JOIN "film" ON (("film_category"."film_id" = "film"."film_id")))
      JOIN "film_actor" ON (("film"."film_id" = "film_actor"."film_id")))
      JOIN "actor" ON (("film_actor"."actor_id" = "actor"."actor_id")))
   GROUP BY "film"."film_id" , "category"."name",price,rating,length,title,description;

//view:   nicer_but_slower_film_list
CREATE VIEW "nicer_but_slower_film_list" AS
   SELECT
      "film"."film_id" AS "FID",
      "film"."title" AS "title",
      "film"."description" AS "description",
      "category"."name" AS "category",
      "film"."rental_rate" AS "price",
      "film"."length" AS "length",
      "film"."rating" AS "rating",
      cast(
                xmlagg(
```

```
                        CONCAT(CONCAT(concat(CONCAT(UCASE(SUBSTRING(first_name,
1,1)),LCASE(SUBSTRING(first_name, 2, LENGTH(first_name)))),'
') ,CONCAT(UCASE(SUBSTRING(last_name, 1,1)),LCASE(SUBSTRING(last_name, 2,
LENGTH(last_name))))),',')
                        ) as varchar(255)) AS "actors"
   FROM
      ((("category"
      LEFT JOIN "film_category" ON (("category"."category_id" = "film_category"."category_id")))
      LEFT JOIN "film" ON (("film_category"."film_id" = "film"."film_id")))
      JOIN "film_actor" ON (("film"."film_id" = "film_actor"."film_id")))
      JOIN "actor" ON (("film_actor"."actor_id" = "actor"."actor_id")))
   GROUP BY "film"."film_id" , "category"."name",price,rating,length,title,description;

//view:film_list
CREATE VIEW "film_list" AS
   SELECT
      "film"."film_id" AS "FID",
      "film"."title" AS "title",
      "film"."description" AS "description",
      "category"."name" AS "category",
      "film"."rental_rate" AS "price",
      "film"."length" AS "length",
      "film"."rating" AS "rating",
      CAST(xmlagg( CONCAT(CONCAT(CONCAT("actor"."first_name",' '),"actor"."last_name"),',')) as
varchar(255))
                  AS "actors"
   FROM
      ((("category"
      LEFT JOIN "film_category" ON (("category"."category_id" = "film_category"."category_id")))
      LEFT JOIN "film" ON (("film_category"."film_id" = "film"."film_id")))
      JOIN "film_actor" ON (("film"."film_id" = "film_actor"."film_id")))
      JOIN "actor" ON (("film_actor"."actor_id" = "actor"."actor_id")))
   GROUP BY "film"."film_id" , "category"."name",price,rating,length,title,description;

//view name:sales_by_film_category
CREATE VIEW "sales_by_film_category" AS
   SELECT
      "c"."name" AS "category", SUM("p"."amount") AS "total_sales"
   FROM
      (((((("payment" "p"
      JOIN "rental" "r" ON (("p"."rental_id" = "r"."rental_id")))
      JOIN "inventory" "i" ON (("r"."inventory_id" = "i"."inventory_id")))
      JOIN "film" "f" ON (("i"."film_id" = "f"."film_id")))
      JOIN "film_category" "fc" ON (("f"."film_id" = "fc"."film_id")))
      JOIN "category" "c" ON (("fc"."category_id" = "c"."category_id")))
   GROUP BY "c"."name"
   ORDER BY "total_sales" DESC;

//view name:sales_by_store
CREATE VIEW "sales_by_store" AS
   SELECT
      CONCAT("c"."city", "cy"."country") AS "store",
      CONCAT("m"."first_name",
            "m"."last_name") AS "manager",
      SUM("p"."amount") AS "total_sales"
   FROM
      (((((((("payment" "p"
      JOIN "rental" "r" ON (("p"."rental_id" = "r"."rental_id")))
      JOIN "inventory" "i" ON (("r"."inventory_id" = "i"."inventory_id")))
      JOIN "store" "s" ON (("i"."store_id" = "s"."store_id")))
      JOIN "address" "a" ON (("s"."address_id" = "a"."address_id")))
      JOIN "city" "c" ON (("a"."city_id" = "c"."city_id")))
```

```
    JOIN "country" "cy" ON (("c"."country_id" = "cy"."country_id")))
    JOIN "staff" "m" ON (("s"."manager_staff_id" = "m"."staff_id")))
 GROUP BY "s"."store_id", "store", "manager";
// ORDER BY "cy"."country" , "c"."city";

//view name:staff_list
CREATE VIEW "staff_list" AS
  SELECT
    "s"."staff_id" AS "ID",
    CONCAT("s"."first_name",
          "s"."last_name") AS "name",
    "a"."address" AS "address",
    "a"."postal_code" AS "zip code",
    "a"."phone" AS "phone",
    "city"."city" AS "city",
    "country"."country" AS "country",
    "s"."store_id" AS "SID"
  FROM
    ((("staff" "s"
    JOIN "address" "a" ON (("s"."address_id" = "a"."address_id")))
    JOIN "city" ON (("a"."city_id" = "city"."city_id")))
    JOIN "country" ON (("city"."country_id" = "country"."country_id")));
```

7.2.3 SCRIPTS FOR CREATING CONSTRAINTS

Save the following script as CreateDBMaker Constraints.sql.

```
//Primary key
ALTER TABLE "SYSADM"."ACTOR" PRIMARY KEY (ACTOR_ID);
ALTER TABLE "SYSADM"."category" PRIMARY KEY (category_id);
ALTER TABLE "SYSADM"."city" PRIMARY KEY (city_id);
ALTER TABLE "SYSADM"."address" PRIMARY KEY (address_id);
ALTER TABLE "SYSADM"."language" PRIMARY KEY (language_id);
ALTER TABLE "SYSADM"."film" PRIMARY KEY (film_id);
ALTER TABLE "SYSADM"."film_actor" PRIMARY KEY ("actor_id","film_id");
ALTER TABLE "SYSADM"."film_category" PRIMARY KEY ("film_id","category_id");
ALTER TABLE "SYSADM"."film_text" PRIMARY KEY (film_id);
ALTER TABLE "SYSADM"."staff" PRIMARY KEY (staff_id);
ALTER TABLE "SYSADM"."store" PRIMARY KEY (store_id);
ALTER TABLE "SYSADM"."inventory" PRIMARY KEY (inventory_id);
ALTER TABLE "SYSADM"."payment" PRIMARY KEY (payment_id);
ALTER TABLE "SYSADM"."customer" PRIMARY KEY (customer_id);
ALTER TABLE "SYSADM"."rental" PRIMARY KEY (rental_id);
ALTER TABLE "SYSADM"."country" PRIMARY KEY (country_id);

//index
create index idx_actor_last_name on SYSADM.ACTOR (LAST_NAME) IN DEFTABLESPACE fillfactor 100;

create index idx_fk_country_id on SYSADM.city (country_id) IN DEFTABLESPACE fillfactor 100;

create index idx_title on SYSADM.film (title) IN DEFTABLESPACE fillfactor 100;
create index idx_fk_language_id on SYSADM.film (language_id) IN DEFTABLESPACE fillfactor 100;
create index idx_fk_original_language_id on SYSADM.film (original_language_id) IN DEFTABLESPACE
fillfactor 100;

create index idx_fk_film_id on SYSADM.film_actor (film_id) IN DEFTABLESPACE fillfactor 100;

create index fk_film_category_category on SYSADM.film_category (category_id) IN DEFTABLESPACE
fillfactor 100;

create SIGNATURE TEXT INDEX idx_title_description on SYSADM.film_text ("title","description") ;
```

```
create index idx_fk_inventory_id on SYSADM.rental ("inventory_id") IN DEFTABLESPACE fillfactor 100;
create index idx_fk_customer_id on SYSADM.rental (customer_id) IN DEFTABLESPACE fillfactor 100;
create index idx_fk_staff_id on SYSADM.rental (staff_id) IN DEFTABLESPACE fillfactor 100;
create UNIQUE index rental_date on SYSADM.rental ("rental_date","inventory_id","customer_id") IN
DEFTABLESPACE fillfactor 100;

create index idx_fk_store_id on SYSADM.staff ("store_id") IN DEFTABLESPACE fillfactor 100;
create index idx_fk_address_id on SYSADM.staff (address_id) IN DEFTABLESPACE fillfactor 100;

create index idx_fk_address_id on SYSADM.store ("address_id") IN DEFTABLESPACE fillfactor 100;
create UNIQUE index idx_unique_manager on SYSADM.store ("manager_staff_id") IN DEFTABLESPACE
fillfactor 100;

create index idx_fk_film_id on SYSADM.inventory (film_id) IN DEFTABLESPACE fillfactor 100;
create index idx_store_id_film_id on SYSADM.inventory ("store_id","film_id") IN DEFTABLESPACE
fillfactor 100;

create index idx_fk_staff_id on SYSADM.payment (staff_id) IN DEFTABLESPACE fillfactor 100;
create index idx_fk_customer_id on SYSADM.payment (customer_id) IN DEFTABLESPACE fillfactor 100;
create index fk_payment_rental on SYSADM.payment (rental_id) IN DEFTABLESPACE fillfactor 100;

create index idx_fk_store_id on SYSADM.customer (store_id) IN DEFTABLESPACE fillfactor 100;
create index idx_fk_address_id on SYSADM.customer (address_id) IN DEFTABLESPACE fillfactor 100;
create index idx_last_name on SYSADM.customer (last_name) IN DEFTABLESPACE fillfactor 100;

//foreign key
alter table staff  FOREIGN KEY  "fk_staff_address" ("address_id") REFERENCES "address" ("address_id")
ON DELETE CASCADE ON UPDATE CASCADE;
alter table staff  FOREIGN KEY  "fk_staff_store" ("store_id") REFERENCES "store" ("store_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table city FOREIGN KEY "fk_city_country" ("country_id") REFERENCES "country" ("country_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table film FOREIGN KEY "fk_film_language" ("language_id") REFERENCES "language"
("language_id") ON DELETE CASCADE ON UPDATE CASCADE;
alter table film FOREIGN KEY "fk_film_language_original" ("original_language_id") REFERENCES
"language" ("language_id") ON DELETE CASCADE ON UPDATE CASCADE;

alter table film_actor FOREIGN KEY  "fk_film_actor_actor" ("actor_id") REFERENCES "actor" ("actor_id")
ON DELETE CASCADE ON UPDATE CASCADE;
alter table film_actor FOREIGN KEY  "fk_film_actor_film" ("film_id") REFERENCES "film" ("film_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table film_category FOREIGN KEY  "fk_film_category_category" ("category_id") REFERENCES
"category" ("category_id") ON DELETE CASCADE ON UPDATE CASCADE;
alter table film_category FOREIGN KEY  "fk_film_category_film" ("film_id") REFERENCES "film" ("film_id")
ON DELETE CASCADE ON UPDATE CASCADE;

alter table store FOREIGN KEY  "fk_store_address"("address_id") REFERENCES "address" ("address_id")
ON DELETE CASCADE ON UPDATE CASCADE;
alter table store FOREIGN KEY  "fk_store_staff"("manager_staff_id") REFERENCES "staff" ("staff_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table inventory FOREIGN KEY  "fk_inventory_film" ("film_id") REFERENCES "film" ("film_id") ON
DELETE CASCADE ON UPDATE CASCADE;
alter table inventory FOREIGN KEY  "fk_inventory_store" ("store_id") REFERENCES "store" ("store_id")
ON DELETE CASCADE ON UPDATE CASCADE;

alter table payment FOREIGN KEY  "fk_payment_customer" ("customer_id") REFERENCES "customer"
("customer_id") ON DELETE CASCADE ON UPDATE CASCADE;
```

```
alter table payment FOREIGN KEY  "fk_payment_rental" ("rental_id") REFERENCES "rental" ("rental_id")
ON DELETE CASCADE ON UPDATE CASCADE;
alter table payment FOREIGN KEY  "fk_payment_staff" ("staff_id") REFERENCES "staff" ("staff_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table customer FOREIGN KEY "fk_customer_address" ("address_id") REFERENCES "address"
("address_id") ON DELETE CASCADE ON UPDATE CASCADE;
alter table customer FOREIGN KEY "fk_customer_store" ("store_id") REFERENCES "store" ("store_id")
ON DELETE CASCADE ON UPDATE CASCADE;

alter table rental FOREIGN KEY  "fk_rental_customer" ("customer_id") REFERENCES "customer"
("customer_id") ON DELETE CASCADE ON UPDATE CASCADE;
alter table rental FOREIGN KEY  "fk_rental_inventory" ("inventory_id") REFERENCES "inventory"
("inventory_id") ON DELETE CASCADE ON UPDATE CASCADE;
alter table rental FOREIGN KEY  "fk_rental_staff" ("staff_id") REFERENCES "staff" ("staff_id") ON
DELETE CASCADE ON UPDATE CASCADE;

alter table ADDRESS foreign key "fk_address_city" (CITY_ID) references SYSADM.CITY( "CITY_ID" ) on
update CASCADE on delete CASCADE;

update statistics;
```

7.2.4 SCRIPTS FOR CREATING TRIGGERS

Save the following script as CreateDBMakerTrigger.sql.

```
//trigger
set block delimiter @@;
@@
CREATE TRIGGER "ins_film" AFTER INSERT ON "film" FOR EACH ROW BEGIN
    INSERT INTO film_text (film_id, title, description)
        VALUES (new.film_id, new.title, new.description);
  END ;
@@

@@
CREATE TRIGGER "upd_film" AFTER UPDATE ON "film" FOR EACH ROW BEGIN
    IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
    THEN
        UPDATE film_text
            SET title=new.title,
                description=new.description,
                film_id=new.film_id
        WHERE film_id=old.film_id;
    END IF;
  END ;
@@

@@
CREATE TRIGGER "del_film" AFTER DELETE ON "film" FOR EACH ROW BEGIN
    DELETE FROM film_text WHERE film_id = old.film_id;
END;
@@

@@
CREATE TRIGGER "payment_date" BEFORE INSERT ON "payment" FOR EACH ROW begin SET
NEW.payment_date = NOW(); end;
@@

@@
CREATE TRIGGER "rental_date" BEFORE INSERT ON "rental" FOR EACH ROW begin SET
```

```
NEW.rental_date = NOW(); end;
@@

@@
CREATE TRIGGER "customer_create_date" BEFORE INSERT ON "customer" FOR EACH ROW begin
SET NEW.create_date = NOW(); end;
@@
```

7.2.5 SCRIPTS FOR CREATING FUNCTIONS

Save the following script as CreateDBMakerFunction.sql.

```
set block delimiter @@;
@@
//function: inventory_in_stock
CREATE OR REPLACE PROCEDURE SYSADM."inventory_in_stock"(p_inventory_id INT,OUT RES
SMALLINT)
LANGUAGE SQL
BEGIN
   DECLARE v_rentals INT;
   DECLARE v_out     INT;

   #AN ITEM IS IN-STOCK IF THERE ARE EITHER NO ROWS IN THE rental TABLE
   #FOR THE ITEM OR ALL ROWS HAVE return_date POPULATED

         DECLARE cur1 CURSOR FOR
   SELECT COUNT(*) AS RENS
   FROM rental
   WHERE inventory_id = p_inventory_id;

         OPEN cur1;
   FETCH cur1 INTO v_rentals;

   IF v_rentals = 0 THEN
     SET RES = 1;
         ELSE
           SET RES = 0;
   END IF;
         CLOSE cur1;

   if v_rentals > 0 THEN
                  DECLARE cur2 CURSOR FOR
                  SELECT COUNT(rental_id)
                  FROM inventory LEFT JOIN rental USING(inventory_id)
                  WHERE inventory.inventory_id = p_inventory_id
                  AND rental.return_date IS NULL;

                  OPEN cur2;
                  FETCH cur2 INTO v_out;
                  IF v_out > 0 THEN
                    SET RES = 0;
                  ELSE
                    SET RES = 1;
                  END IF;
                  close cur2;
         END IF;
END;
@@

@@
//Function:get_customer_balance
CREATE OR REPLACE PROCEDURE SYSADM."get_customer_balance"(in p_customer_id INT, in
```

```
p_effective_date TIMESTAMP ,out res decimal(5,2))
LANGUAGE SQL
BEGIN

      #OK, WE NEED TO CALCULATE THE CURRENT BALANCE GIVEN A CUSTOMER_ID AND A
DATE
      #THAT WE WANT THE BALANCE TO BE EFFECTIVE FOR. THE BALANCE IS:
      #   1) RENTAL FEES FOR ALL PREVIOUS RENTALS
      #   2) ONE DOLLAR FOR EVERY DAY THE PREVIOUS RENTALS ARE OVERDUE
      #   3) IF A FILM IS MORE THAN RENTAL_DURATION * 2 OVERDUE, CHARGE THE
REPLACEMENT_COST
      #   4) SUBTRACT ALL PAYMENTS MADE BEFORE THE DATE SPECIFIED

  DECLARE v_rentfees DECIMAL(5,2); #FEES PAID TO RENT THE VIDEOS INITIALLY
  DECLARE v_overfees INTEGER;      #LATE FEES FOR PRIOR RENTALS
  DECLARE v_payments DECIMAL(5,2); #SUM OF PAYMENTS MADE PREVIOUSLY

DECLARE cur1 CURSOR FOR
  SELECT IFNULL(SUM(film.rental_rate),0)
    FROM film, inventory, rental
    WHERE film.film_id = inventory.film_id
      AND inventory.inventory_id = rental.inventory_id
      AND rental.rental_date <= p_effective_date
      AND rental.customer_id = p_customer_id;
   open cur1;
   FETCH cur1 INTO v_rentfees;
   close cur1;

   DECLARE cur2 CURSOR FOR
  select ifnull(sum(case when(DAYS_BETWEEN('0001-01-01'd,DATEPART(rental.return_date))-
DAYS_BETWEEN('0001-01-01'd,DATEPART(rental.rental_date)))> film.rental_duration
       then
                        (DAYS_BETWEEN('0001-01-01'd,DATEPART(rental.return_date))-
DAYS_BETWEEN('0001-01-01'd,DATEPART(rental.rental_date)))- film.rental_duration
                    else
                            0
                    end),0)
 FROM
    rental,
    inventory,
    film
WHERE
    film.film_id = inventory.film_id
        AND inventory.inventory_id = rental.inventory_id
      AND rental.rental_date <= p_effective_date
      AND rental.customer_id = p_customer_id;
 open cur2;
   FETCH cur2 INTO v_overfees;
   close cur2;

 DECLARE cur3 CURSOR FOR
  SELECT IFNULL(SUM(payment.amount),0)
    FROM payment
    WHERE payment.payment_date <= p_effective_date
    AND payment.customer_id = p_customer_id;
  open cur3;
   FETCH cur3 INTO v_payments;
   close cur3;

  set res = v_rentfees + v_overfees - v_payments;
END;
@@
```

```
@@
//function:inventory_held_by_customer
CREATE OR REPLACE PROCEDURE SYSADM."inventory_held_by_customer"(in p_inventory_id INT,out
customer_id int)
LANGUAGE SQL
BEGIN
  DECLARE v_customer_id INT;
  DECLARE EXIT HANDLER FOR NOT FOUND set customer_id = NULL;

  DECLARE cur1 CURSOR FOR
  SELECT customer_id
  FROM rental
  WHERE return_date IS NULL
  AND inventory_id = p_inventory_id;

  open cur1;
  FETCH cur1 INTO v_customer_id;
  close cur1;

  set customer_id = v_customer_id;
END;
@@
```

7.2.6 SCRIPTS FOR CREATING STORED PROCEDURES

Save the following script as CreateDBMakerProcedures.sql.

```
set block delimiter @@;

@@
CREATE OR REPLACE PROCEDURE SYSADM."rewards_report"(
    IN min_monthly_purchases int ,
          IN min_dollar_amount_purchased DECIMAL(10,2) ,
          OUT count_rewardees INT)
language sql
BEGIN

    DECLARE last_month_start date;
    DECLARE last_month_end varchar(50);

    /* Determine start and end time periods */
          SET last_month_start = TIMESTAMPADD('M',-1,CURRENT_timestamp()) ;
          SET last_month_end = LAST_DAY(last_month_start);

    /*
       Create a temporary storage area for
       Customer IDs.
    */
          drop table if exists tmpCustomer;
    CREATE TEMPORARY TABLE tmpCustomer (customer_id SMALLINT NOT NULL );

    /*
       Find all customers meeting the
       monthly purchase requirements
    */
    SELECT p.customer_id
    FROM payment AS p
    WHERE DATEPART(p.payment_date) BETWEEN last_month_start AND last_month_end
    GROUP BY customer_id
    HAVING SUM(p.amount) > min_dollar_amount_purchased
    AND COUNT(customer_id) > min_monthly_purchases
```

```
        INTO tmpCustomer (customer_id);

    /* Populate OUT parameter with count of found customers */
        DECLARE cur1 CURSOR FOR
                SELECT count(*) as p_count
                FROM tmpCustomer;

                OPEN cur1;
                FETCH cur1 INTO count_rewardees;
        close cur1;
    /*
       Output ALL customer information of matching rewardees.
       Customize output as needed.
    */
        DECLARE cur2 CURSOR with return FOR
    SELECT c.*
    FROM tmpCustomer AS t
    INNER JOIN customer AS c ON t.customer_id = c.customer_id;

        open cur2;

    /* Clean up */
    DROP TABLE tmpCustomer;
END;
@@

@@
CREATE OR REPLACE PROCEDURE SYSADM.film_not_in_stock(IN p_film_id INT, IN p_store_id INT,
OUT p_film_count INT)
language sql
BEGIN
     declare tmp_inventory_id int;
     declare tmp_func_Id int;
          DECLARE foundrows int;

          /*
       Create a temporary storage inventory_id for inventory.
    */
        DROP TABLE IF EXISTS tmpInventory;
    CREATE TEMP TABLE tmpInventory (inventory_id SMALLINT, inventory_in_stock SMALLINT);

          DECLARE cur1 CURSOR FOR
                SELECT inventory_id
                FROM inventory
                WHERE film_id = p_film_id
                AND store_id = p_store_id ;
        OPEN cur1;
        loop
                FETCH cur1 INTO tmp_inventory_id;
                IF tmp_inventory_id != NULL THEN
                   call inventory_in_stock(tmp_inventory_id,tmp_func_Id);
                   Insert into tmpInventory values(tmp_inventory_id,tmp_func_Id);

                   IF tmp_func_Id = 0 THEN
                     set foundrows = foundrows+1;
                   END IF;
                ELSE
                  LEAVE;
                END IF;

    end loop;
        close cur1;
```

```
        set p_film_count = foundrows;

        DECLARE cur2 CURSOR with return FOR
                SELECT inventory_id
                FROM tmpInventory
                WHERE inventory_in_stock = 0;
        OPEN cur2;

END;
@@

@@
CREATE OR REPLACE PROCEDURE SYSADM.film_in_stock(IN p_film_id INT, IN p_store_id INT, OUT
p_film_count INT)
language sql
BEGIN
    declare tmp_inventory_id int;
    declare tmp_func_Id int;
         DECLARE foundrows int;

         /*
     Create a temporary storage inventory_id for inventory.
  */
        DROP TABLE IF EXISTS tmpInventory;
  CREATE TEMP TABLE tmpInventory (inventory_id SMALLINT, inventory_in_stock SMALLINT);

         DECLARE cur1 CURSOR FOR
                SELECT inventory_id
                FROM inventory
                WHERE film_id = p_film_id
                AND store_id = p_store_id ;
        OPEN cur1;
        loop
                FETCH cur1 INTO tmp_inventory_id;
                IF tmp_inventory_id != NULL THEN
                   call inventory_in_stock(tmp_inventory_id,tmp_func_Id);
                   Insert into tmpInventory values(tmp_inventory_id,tmp_func_Id);

                   IF tmp_func_Id = 1 THEN
                     set foundrows = foundrows+1;
                   END IF;
                ELSE
                  LEAVE;
                END IF;

  end loop;
        close cur1;

        set p_film_count = foundrows;

        DECLARE cur2 CURSOR with return FOR
                SELECT inventory_id
                FROM tmpInventory
                WHERE inventory_in_stock = 1;
        OPEN cur2;

END;
@@
```