建库建表sql语句

[1 Introduction 1](#_Toc482045943)

[2 Installation 1](#_Toc482045944)

[2.1 方法一：安装MySQL时选择 FULL 完全安装 1](#_Toc482045945)

[2.2 方法二：下载后导入 2](#_Toc482045946)

[3 Structure 3](#_Toc482045947)

[3.1 表的对应关系 3](#_Toc482045948)

[3.2 Tables 6](#_Toc482045949)

[3.2.1 【01演员表actor】 7](#_Toc482045950)

[3.2.2 【02地址表address】 7](#_Toc482045951)

[3.2.3 【03分类表category】 7](#_Toc482045952)

[3.2.4 【04城市表city】 8](#_Toc482045953)

[3.2.5 【05国家表country】 8](#_Toc482045954)

[3.2.6 【06客户表customer】 8](#_Toc482045955)

[3.2.7 【07电影表film】 8](#_Toc482045956)

[3.2.8 【08 film\_actor表】 9](#_Toc482045957)

[3.2.9 【09 film\_category表】 9](#_Toc482045958)

[3.2.10 【10 film\_text表】 9](#_Toc482045959)

[3.2.11 【11库存表inventory】 10](#_Toc482045960)

[3.2.12 【12语言表language】 10](#_Toc482045961)

[3.2.13 【13付款表payment】 10](#_Toc482045962)

[3.2.14 【14租金表rental】 10](#_Toc482045963)

[3.2.15 【15工作人员表staff】 11](#_Toc482045964)

[3.2.16 【16商店表store】 11](#_Toc482045965)

[3.3 Views 11](#_Toc482045966)

[3.3.1 The actor\_info View 11](#_Toc482045967)

[3.3.2 The customer\_list View 12](#_Toc482045968)

[3.3.3 The film\_list View 13](#_Toc482045969)

[3.3.4 The nicer\_but\_slower\_film\_list View 14](#_Toc482045970)

[3.3.5 The sales\_by\_film\_category View 16](#_Toc482045971)

[3.3.6 The sales\_by\_store View 17](#_Toc482045972)

[3.3.7 3.2.7. The staff\_list View 17](#_Toc482045973)

[3.4 Stored Procedures 18](#_Toc482045974)

[3.4.1 The film\_in\_stock Stored Procedure 18](#_Toc482045975)

[3.4.2 The film\_not\_in\_stock Stored Procedure 19](#_Toc482045976)

[3.4.3 The rewards\_report Stored Procedure 20](#_Toc482045977)

[3.5 Stored Functions 22](#_Toc482045978)

[3.5.1 The get\_customer\_balance Function 22](#_Toc482045979)

[3.5.2 The inventory\_held\_by\_customer Function 24](#_Toc482045980)

[3.5.3 The inventory\_in\_stock Function 24](#_Toc482045981)

[3.6 Triggers 25](#_Toc482045982)

[3.6.1 The customer\_create\_date Trigger 25](#_Toc482045983)

[3.6.2 The payment\_date Trigger 25](#_Toc482045984)

[3.6.3 The rental\_date Trigger 25](#_Toc482045985)

[3.6.4 The ins\_film Trigger 26](#_Toc482045986)

[3.6.5 The upd\_film Trigger 26](#_Toc482045987)

[3.6.6 The del\_film Trigger 26](#_Toc482045988)

# Introduction

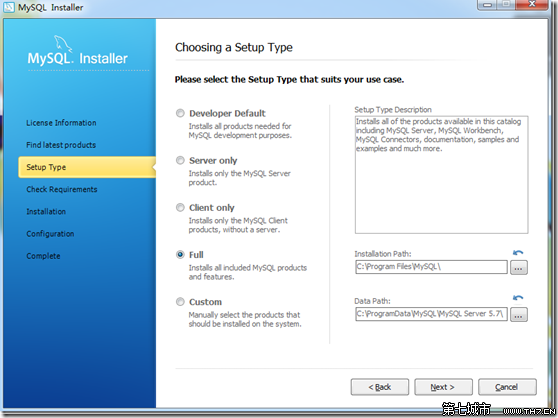
Sakila可以作为数据库设计的参考,也可作为实验数据。我是用作数据仓库和ODI学习的实验数据。

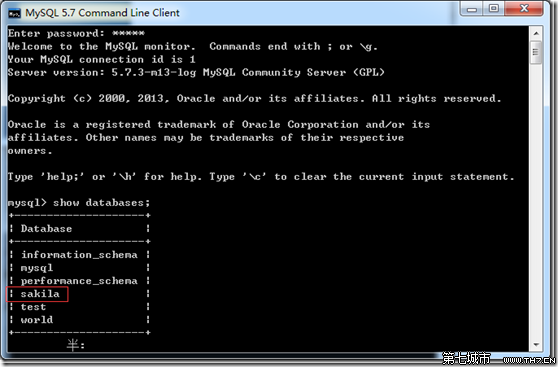
The Sakila sample database was developed by Mike Hillyer, a former member of the MySQL AB documentation team, andis intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so forth.Sakila sample database also serves to highlight the latest features of MySQL such as Views, Stored Procedures, and Triggers.

# Installation

## 方法一：安装MySQL时选择 FULL 完全安装

安装MySQL时选择FULL 完全安装, 默认安装了Sakila sample database





## 方法二：下载后导入

The Sakila sample database is available fromhttp://dev.mysql.com/doc/index-other.html. A downloadable archive is available in compressed file or Zip format. The archive contains three files:sakila-schema.sql,sakila-data.sql, andsakila.mwb.

Thesakila-schema.sqlfile contains all theCREATEstatements required to create the structure of the Sakila database including tables, views, stored procedures, and triggers.

Thesakila-data.sqlfile contains theINSERTstatements required to populate the structure created by thesakila-schema.sqlfile, along with definitions for triggers that must be created after the initial data load.

Thesakila.mwbfile is a MySQL Workbench data model that you can open within MySQL Workbench to examine the database structure. For more information, seeMySQL Workbench.

下载地址：

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

tar格式http://downloads.mysql.com/docs/sakila-db.tar.gz

官方文档http://dev.mysql.com/doc/sakila/en/index.html

解压后得到三个文件：

1. sakila-schema.sql 文件包含创建Sakila数据库的结构：表、视图、存储过程和触发器

2. sakila-data.sql文件包含：使用 INSERT语句填充数据及在初始数据加载后，必须创建的触发器的定义

3. sakila.mwb文件是一个MySQL Workbench数据模型，可以在MySQL的工作台打开查看数据库结构。

导入参考：http://dev.mysql.com/doc/sakila/en/sakila-installation.html

shell>mysql -u root -p

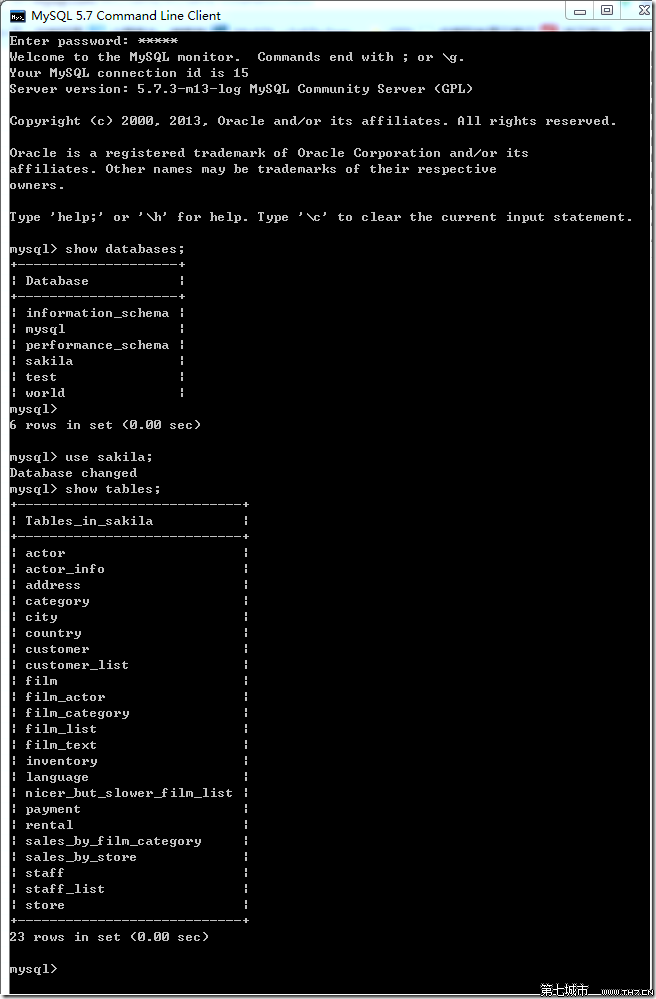
mysql> SOURCE C:/temp/sakila-db/sakila-schema.sql;

mysql> SOURCE C:/temp/sakila-db/sakila-data.sql;

查看导入结果

show databases

use sakila show datatable

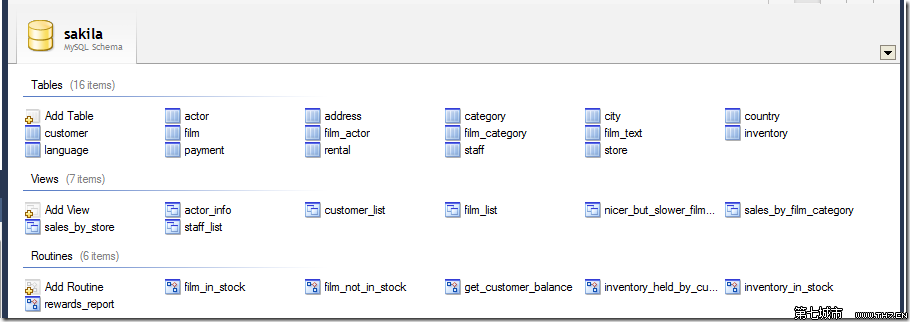


sakila

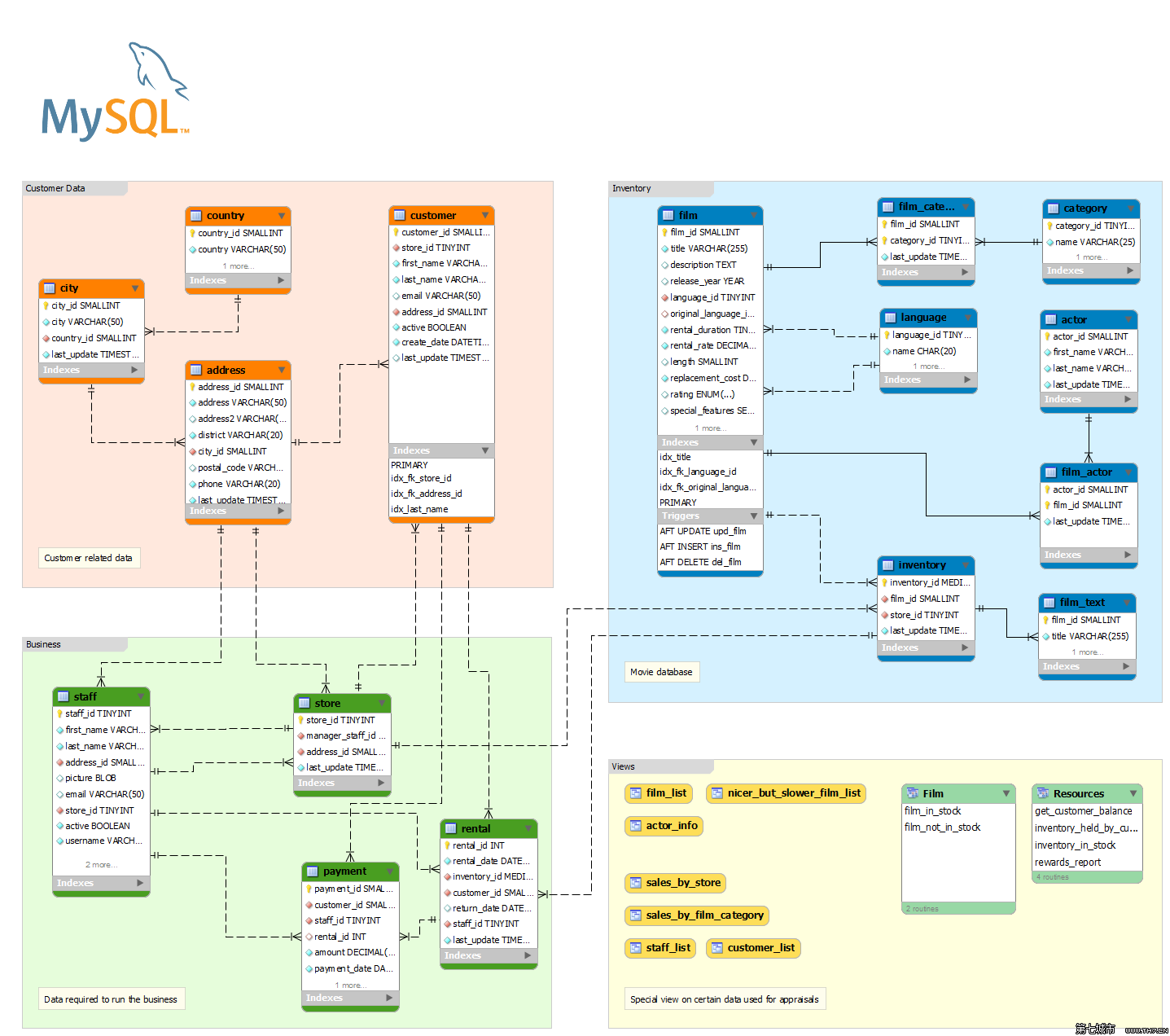
# Structure

## 表的对应关系

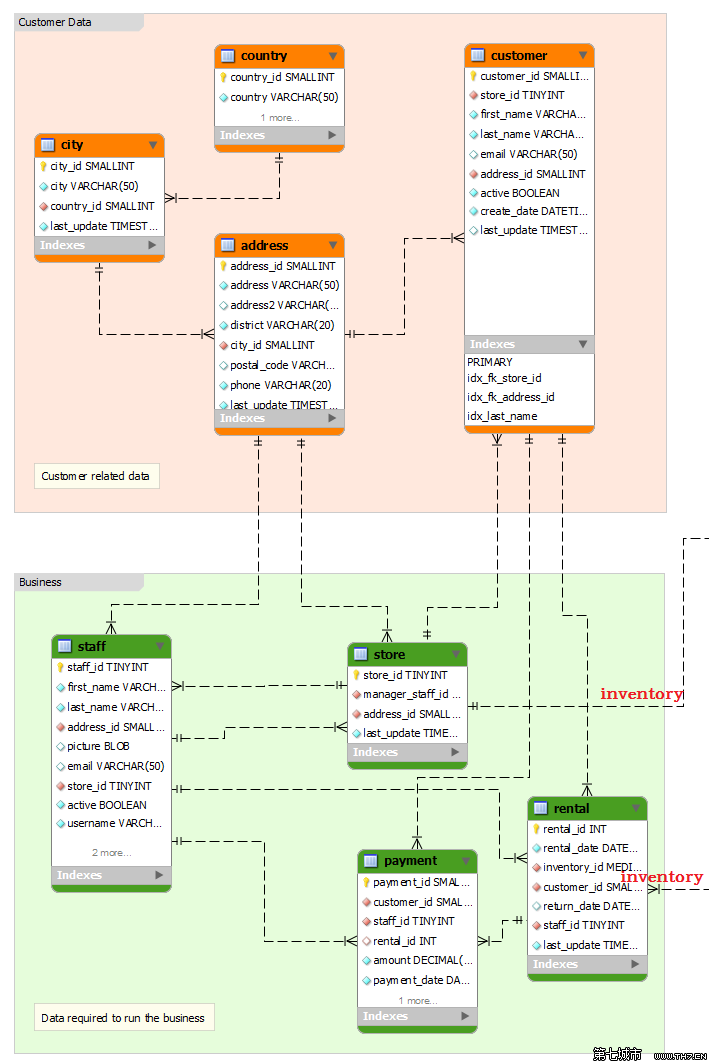
Sakila样例数据库包括16张表格，7个视图，3个Stored Procedures，3个Functions，6个Triggers。英文描述点开相应连接即可查看。



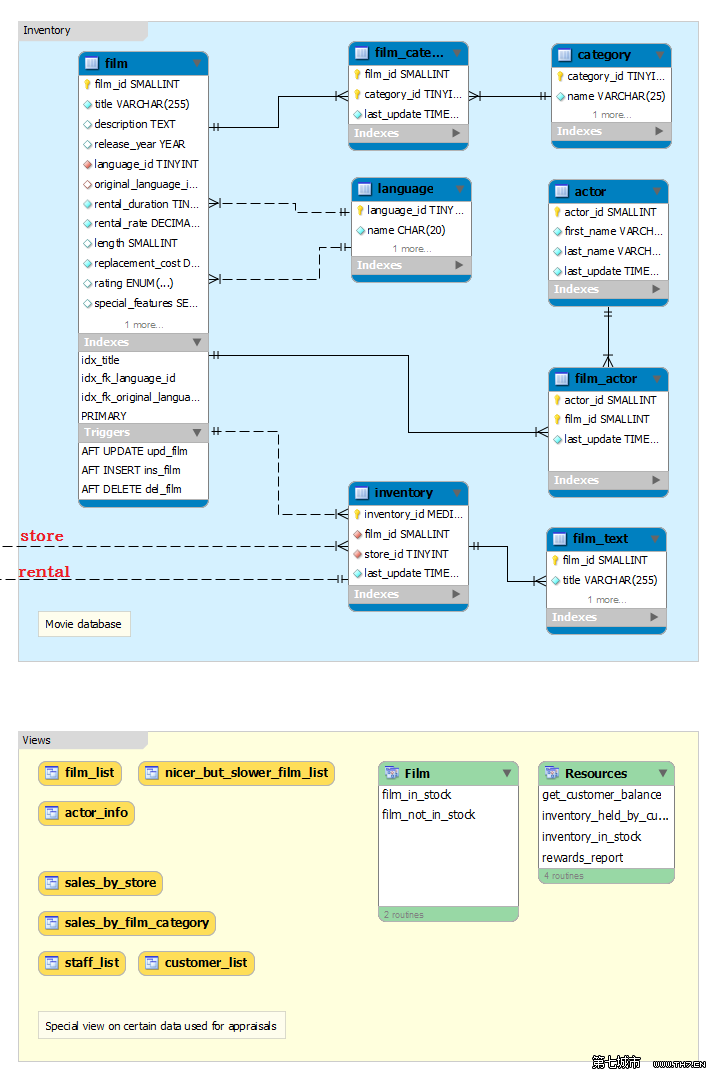
全图：



Customer Data 和 Business Data



Inventory Data



## Tables

(1) The actor Table演员表

(2) The address Table地址表

(3) The category Table电影类别表

(4) The city Table城市表

(5) The country Table国家表

(6) The customer Table客户表

(7) The film Table电影表

(8) The film\_actor Table电影和演员中间对应表

(9) The film\_category Table电影和类别的中间对应表

(10) The film\_text Table只包含title和description两列的电影表子表，用以快速检索

(11) The inventory Table库存表

(12) The language Table语言表

(13) The payment Table付款表

(14) The rental Table租金表

(15) The staff Table工作人员表

(16) The store Table商店表

### 【01演员表actor】

演员表列出了所有演员的信息。演员表和电影表之间是多对多的关系，通过film\_actor表建立关系

1. actor\_id: 代理主键用于唯一标识表中的每个演员
2. first\_name: 演员的名字
3. last\_name: 演员的姓氏
4. last\_update: 该行已创建或最近更新的时间

### 【02地址表address】

地址表包含客户、员工和商店的地址信息。地址表的主键出现在顾客、 员工、和存储表的外键 。

1. address\_id: 代理主键用于唯一标识表中的每个地址
2. address: 地址的第一行
3. address2: 一个可选的第二行地址
4. district: 该地区的所属地区，这可以是国家，省，县等
5. city\_id: 指向城市表的外键
6. postal\_code: 邮政编码
7. phone: 地址的电话号码
8. last\_update: 该行已创建或最近更新的时间

### 【03分类表category】

类别表列出了可以分配到一个电影类别。分类和电影是多对多的关系，通过表film\_category建立关系

1. category\_id: 代理主键用于唯一标识表中的每个类别
2. name: 类别名称
3. last\_update: 该行已创建或最近更新的时间

### 【04城市表city】

城市表包含的城市名单。城市表使用外键来标示国家；在地址表中被作为外键来使用。

1. city\_id: 代理主键用于唯一标识表中的每个城市
2. city: 城市的名字
3. country\_id: 外键，用于标示城市所属的国家
4. last\_update: 该行已创建或最近更新的时间

### 【05国家表country】

国家表中包含的国家名单。国家表是指在城市表的外键 。

1. country\_id: 代理主键用于唯一标识表中的每个国家
2. country: 国家的名称
3. last\_update: 该行已创建或最近更新的时间

### 【06客户表customer】

客户表包含了所有客户的列表 。 客户表在支付表和租金表被作为外键使用；客户表使用外键来表示地址和存储。

1. customer\_id: 代理主键用于唯一标识表中的每个客户
2. store\_id: 一个外键，确定客户所属的store。
3. first\_name: 客户的名字
4. last\_name: 客户的姓氏
5. email: 客户的电子邮件地址
6. address\_id: 使用在地址 表的外键来确定客户的地址
7. active: 表示客户是否是活跃的客户
8. create\_date: 顾客被添加到系统中的日期。使用 INSERT 触发器自动设置。
9. last\_update: 该行已创建或最近更新的时间

说明： active: 此设置为“ FALSE“作为替代客户彻底删除。大多数查询应该有一个 WHERE active = TRUE 字句。 store\_id: 此处的客户不仅限于只由这家商店出租，而是包括客户常常去逛的商店

### 【07电影表film】

电影表是一个可能在商店库存的所有影片名单。每部影片的拷贝的实际库存信息保存在库存表。电影表指使用外键来标示语言表；在film\_category、film\_actor和库存表中作为外键使用。

1. film\_id: 代理主键用于唯一标识表中的每个电影
2. title: 影片的标题
3. description: 一个简短的描述或电影的情节摘要
4. release\_year: 电影发行的年份
5. language\_id: 使用外键来标示语言
6. original\_language\_id: 电影的原始语音。使用外键来标示语言
7. rental\_duration: 租赁期限的长短，以天作为单位
8. rental\_rate: 指定的期限内电影的租金
9. length: 影片的长度，以分钟为单位。
10. replacement\_cost: 如果电影未被归还或损坏状态向客户收取的款项
11. rating: 分配给电影评级。可以是 G， PG，PG - 13 ， R 或NC - 17
12. special\_features: 包括DVD上常见的特殊功能的列表
13. last\_update: 该行已创建或最近更新的时间

特殊功能包括零个或多个拖车、评论、删剪片段、幕后。

### 【08 film\_actor表】

film\_actor表是用来支持许多电影和演员之间的多对多关系 。对于每一个给定的电影演员，将有film\_actor表中列出的演员和电影中的一个行 。

film\_actor表指的是使用外键的电影和演员表。

1. actor\_id: 用于识别演员的外键
2. film\_id: 用于识别电影的外键
3. last\_update: 该行已创建或最近更新的时间

### 【09 film\_category表】

film\_category表是用来支持许多电影和类别之间的多对多关系 。应用于电影的每个类别中，将有film\_category表中列出的类别和电影中的一个行。

film\_category表是指使用外键 的 电影 和类别表 。

1. film\_id: 用于识别电影的外键
2. category\_id: 用于识别类别的外键
3. last\_update: 该行已创建或最近更新的时间

### 【10 film\_text表】

film\_text表是Sakila样例数据库唯一使用MyISAM存储引擎的表。MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快。此表提供允许全文搜索电影表中列出的影片的标题和描述。film\_text表包含的film\_id，标题和描述的列电影表，保存的内容与电影表上的内容同步（指电影表的插入、更新和删除操作）

1. film\_id: 代理主键用于唯一标识表中的每个电影
2. title: 影片的标题
3. description: 一个简短的描述或电影的情节摘要

注意：film\_text表的内容不应该直接修改。所有的变更来自于电影表 。

### 【11库存表inventory】

库存表的一行为存放在一个给定的商店里的一个给定的电影的copy副本。库存表是使用外键来识别电影和存储；在出租表中使用外键来识别库存。

1. inventory\_id: 理主键用于唯一标识每个项目在库存
2. film\_id: 使用外键来识别电影
3. store\_id: 使用外键来识别物品所在的商店
4. last\_update: 该行已创建或最近更新的时间

### 【12语言表language】

语言表是一个查找表，列出可能使用的语言，电影可以有自己的语言和原始语言值 。

语言表在电压表中被作为外键来使用。

1. language\_id: 代理主键用于唯一标识每一种语言
2. name: 语言的英文名称
3. last\_update: 该行已创建或最近更新的时间

### 【13付款表payment】

付款表记录每个客户的付款，如支付的金额和租金的资料。

付款表使用外键来表示客户、出租、和工作人员。

1. payment\_id: 代理主键用于唯一标识每个付款
2. customer\_id: 使用外键来标识付款的客户
3. staff\_id: 工作人员，负责处理支付 。使用外键来标识
4. rental\_id: 租借ID, 外键，参照rental表
5. amount: 付款金额
6. payment\_date: 处理付款的日期
7. last\_update: 该行已创建或最近更新的时间

### 【14租金表rental】

租借表的一行表示每个inventory的租借客户、租借时间、归还时间

租借表是使用外键来标识库存 ，顾客 和工作人员；在支付表中使用了外键来标识租金 。

1. rental\_id: 代理主键唯一标识的租金
2. rental\_date: 该项目租用的日期和时间
3. inventory\_id: 该项目被租用
4. customer\_id: 租用该项目的客户
5. return\_date: 归还日期
6. staff\_id: 处理该项业务的工作人员
7. last\_update: 该行已创建或最近更新的时间

### 【15工作人员表staff】

工作人员表列出了所有的工作人员，包括电子邮件地址，登录信息和图片信息 。

工作人员表是指使用外键来标识存储和地址表；在出租、支付和商店表中作为外键。

1. staff\_id: 代理主键唯一标识的工作人员
2. first\_name: 工作人员的名字
3. last\_name: 工作人员的姓氏
4. address\_id: 工作人员的地址在地址表的外键
5. picture: 工作人员的照片，使用了 BLOB属性
6. email: 工作人员的电子邮件地址
7. store\_id: 工作人员所在的商店，用外键标识
8. active: 是否是活跃的工作人员。
9. username: 用户名，由工作人员用来访问租赁系统
10. password: 工作人员访问租赁系统所使用的密码。使用了 SHA1 函数
11. last\_update: 该行已创建或最近更新的时间
12. active: 是否有效，删除时设置为False

### 【16商店表store】

store表列出了系统中的所有商店 。

store使用外键来标识工作人员和地址；在员工、客户、库存表被作为外键使用。

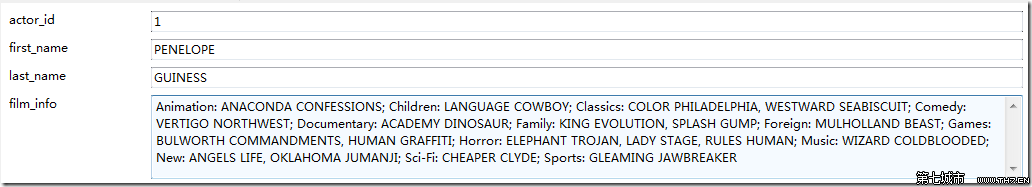
1. store\_id: 代理主键唯一标识的商店
2. manager\_staff\_id: 使用外键来标识这家商店的经理
3. address\_id: 使用外键来确定这家店的地址
4. last\_update: 该行已创建或最近更新的时间

## Views

### The actor\_info View

actor\_info视图提供了所有演员的列表及所演的电影, 电影按category分组.

actor →film\_actor →film→film\_category →category



SELECT

a.actor\_id,

a.first\_name,

a.last\_name,

GROUP\_CONCAT(DISTINCT CONCAT(c.name, ': ',

(SELECT GROUP\_CONCAT(f.title ORDER BY f.title SEPARATOR ', ')

FROM sakila.film f

INNER JOIN sakila.film\_category fc

ON f.film\_id = fc.film\_id

INNER JOIN sakila.film\_actor fa

ON f.film\_id = fa.film\_id

WHERE fc.category\_id = c.category\_id

AND fa.actor\_id = a.actor\_id

)

)

ORDER BY c.name SEPARATOR '; ')

AS film\_info

FROM sakila.actor a

LEFT JOIN sakila.film\_actor fa

ON a.actor\_id = fa.actor\_id

LEFT JOIN sakila.film\_category fc

ON fa.film\_id = fc.film\_id

LEFT JOIN sakila.category c

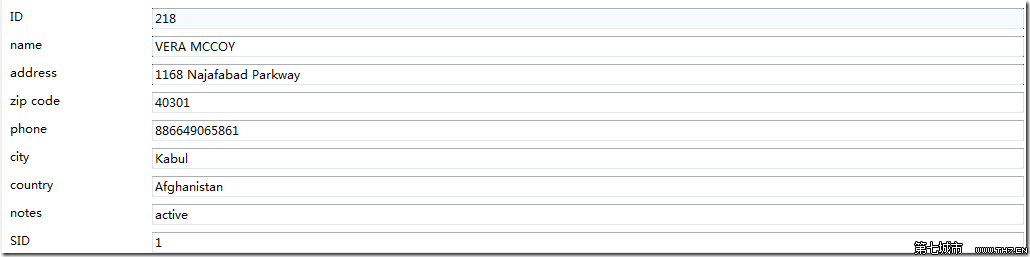
ON fc.category\_id = c.category\_id

GROUP BY a.actor\_id, a.first\_name, a.last\_name

### The customer\_list View

客户列表，firstname和lastname连接成fullname，将address,city,country集成在一个视图里

customer→address→city →country



SELECT

cu.customer\_id AS ID,

CONCAT(

cu.first\_name,

\_utf8 ' ',

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,

IF (

cu.active,

\_utf8 'active',

\_utf8 ''

) AS notes,

cu.store\_id AS SID

FROM

customer AS cu

JOIN address AS 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

### The film\_list View

电影列表视图，包含了每一部电影的信息及电影所对应的演员。电影对应的演员以逗号作为分隔符。连接了film,film\_category,category,film\_actorandactor表的数据

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,

GROUP\_CONCAT(

CONCAT(

actor.first\_name,

\_utf8 ' ',

actor.last\_name

) SEPARATOR ', '

) AS actors

FROM

category

LEFT JOIN film\_category ON category.category\_id = film\_category.category\_id

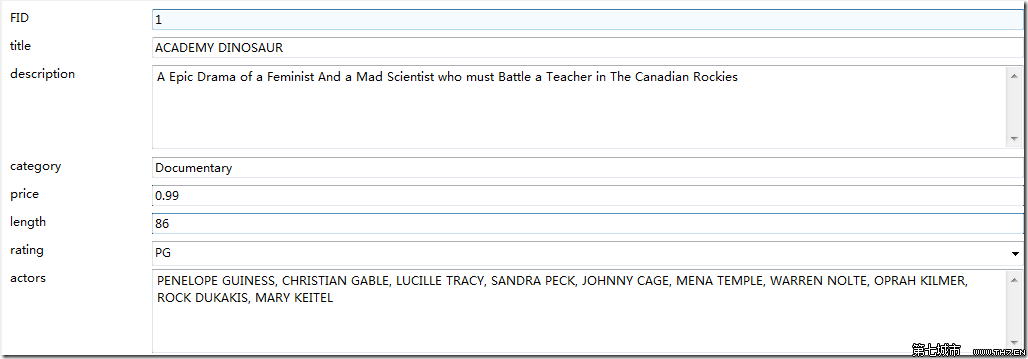
LEFT JOIN film ON film\_category.film\_id = film.film\_id

JOIN film\_actor ON film.film\_id = film\_actor.film\_id

JOIN actor ON film\_actor.actor\_id = actor.actor\_id

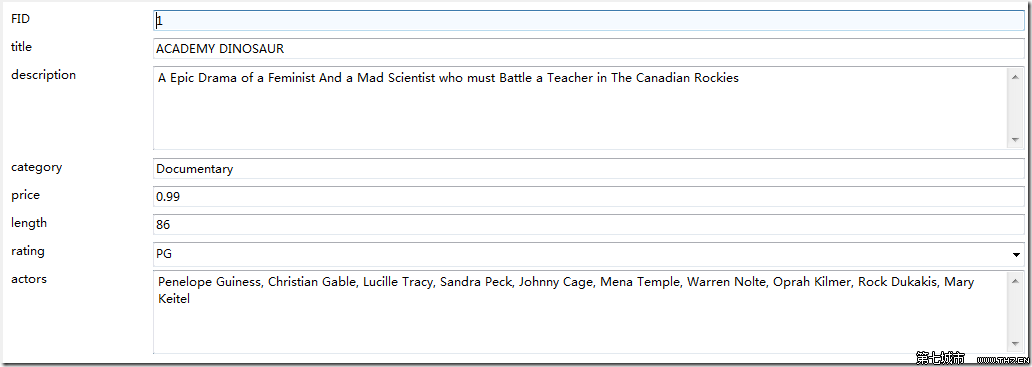
GROUP BY

film.film\_id



### The nicer\_but\_slower\_film\_list View

电影列表视图，包含了每一部电影的信息及电影所对应的演员。电影对应的演员以逗号作为分隔符。连接了film,film\_category,category,film\_actorandactor表的数据。和The film\_list View不同，演员名字只有单词首字母大写了。



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,

GROUP\_CONCAT(

CONCAT(

CONCAT(

UCASE(

SUBSTR(actor.first\_name, 1, 1)

),

LCASE(

SUBSTR(

actor.first\_name,

2,

LENGTH(actor.first\_name)

)

),

\_utf8 ' ',

CONCAT(

UCASE(

SUBSTR(actor.last\_name, 1, 1)

),

LCASE(

SUBSTR(

actor.last\_name,

2,

LENGTH(actor.last\_name)

)

)

)

)

) SEPARATOR ', '

) 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

### The sales\_by\_film\_category View

每个电影种类的销售额 ,payment →rental→inventory→film→film\_category→category



image

SELECT

c.name AS category

, SUM(p.amount) AS total\_sales

FROM payment AS p

INNER JOIN rental AS r ON p.rental\_id = r.rental\_id

INNER JOIN inventory AS i ON r.inventory\_id = i.inventory\_id

INNER JOIN film AS f ON i.film\_id = f.film\_id

INNER JOIN film\_category AS fc ON f.film\_id = fc.film\_id

INNER JOIN category AS c ON fc.category\_id = c.category\_id

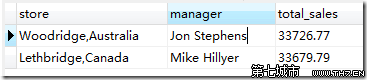
GROUP BY c.name

ORDER BY total\_sales DESC

### The sales\_by\_store View

每个商店的manager及销售额。

payment →rental→inventory→store→staff



SELECT

CONCAT(c.city, \_utf8',', cy.country) AS store

, CONCAT(m.first\_name, \_utf8' ', m.last\_name) AS manager

, SUM(p.amount) AS total\_sales

FROM payment AS p

INNER JOIN rental AS r ON p.rental\_id = r.rental\_id

INNER JOIN inventory AS i ON r.inventory\_id = i.inventory\_id

INNER JOIN store AS s ON i.store\_id = s.store\_id

INNER JOIN address AS a ON s.address\_id = a.address\_id

INNER JOIN city AS c ON a.city\_id = c.city\_id

INNER JOIN country AS cy ON c.country\_id = cy.country\_id

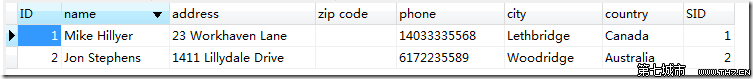
INNER JOIN staff AS m ON s.manager\_staff\_id = m.staff\_id

GROUP BY s.store\_id

ORDER BY cy.country, c.city

### 3.2.7. The staff\_list View

工作人员的列表



SELECT

s.staff\_id AS ID,

CONCAT(

s.first\_name,

\_utf8 ' ',

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 AS s

JOIN address AS 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

## Stored Procedures

### The film\_in\_stock Stored Procedure

Description

Thefilm\_in\_stockstored procedure is used to determine whether any copies of a given film are in stock at a given store.

判断在一个给定的商店里是否有一个给定电影的copy.

Parameters

p\_film\_id: The ID of the film to be checked, from thefilm\_idcolumn of thefilmtable.

p\_store\_id: The ID of the store to check for, from thestore\_idcolumn of thestoretable.

p\_film\_count: AnOUTparameter that returns a count of the copies of the film in stock.

Return Values

This procedure produces a table of inventory ID numbers for the copies of the film in stock, and returns (in thep\_film\_countparameter) a count that indicates the number of rows in that table.

BEGIN

SELECT inventory\_id

FROM inventory

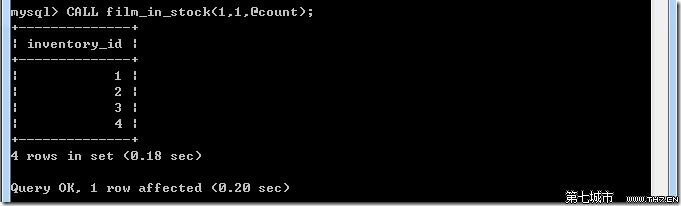
WHERE film\_id = p\_film\_id

AND store\_id = p\_store\_id

AND inventory\_in\_stock(inventory\_id);

SELECT FOUND\_ROWS() INTO p\_film\_count;

END



### The film\_not\_in\_stock Stored Procedure

和The film\_in\_stock Stored Procedure相反，判断给定的电影，是否有copy，不在一个给定的商店

Description

Thefilm\_not\_in\_stockstored procedure is used to determine whether there are any copies of a given film not in stock (rented out) at a given store.

Parameters

p\_film\_id: The ID of the film to be checked, from thefilm\_idcolumn of thefilmtable.

p\_store\_id: The ID of the store to check for, from thestore\_idcolumn of thestoretable.

p\_film\_count: AnOUTparameter that returns a count of the copies of the film not in stock.

Return Values

This procedure produces a table of inventory ID numbers for the copies of the film not in stock, and returns (in thep\_film\_countparameter) a count that indicates the number of rows in that table.

BEGIN

SELECT inventory\_id

FROM inventory

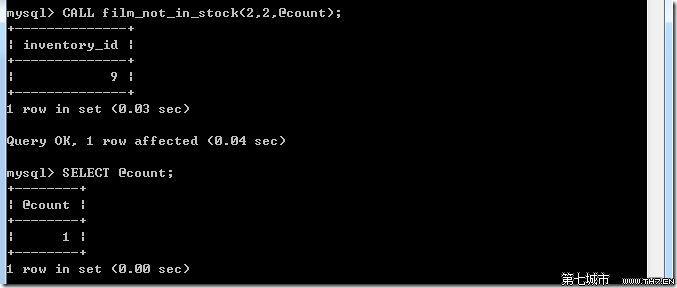
WHERE film\_id = p\_film\_id

AND store\_id = p\_store\_id

AND NOT inventory\_in\_stock(inventory\_id);

SELECT FOUND\_ROWS() INTO p\_film\_count;

END



### The rewards\_report Stored Procedure

过去一个月，数量大于输入参数min\_monthly\_purchases ，消费金额大于输入参数min\_dollar\_amount\_purchased的客户

Description

Therewards\_reportstored procedure generates a customizable list of the top customers for the previous month.

Parameters

min\_monthly\_purchases: The minimum number of purchases or rentals a customer needed to make in the last month to qualify.

min\_dollar\_amount\_purchased: The minimum dollar amount a customer needed to spend in the last month to qualify.

count\_rewardees: AnOUTparameter that returns a count of the customers who met the qualifications specified.

Return Values

This function returns a table of customers who met the qualifications specified. The table has the same structure as thecustomertable. The procedure also returns (in thecount\_rewardeesparameter) a count that indicates the number of rows in that table.

Stop Service net stop uxsms

Start Service net start uxsms

Disable Service sc config uxsms start= disabled

Enable Service sc config uxsms start= auto

proc: BEGIN

DECLARE last\_month\_start DATE;

DECLARE last\_month\_end DATE;

/\* Some sanity checks... \*/

IF min\_monthly\_purchases = 0 THEN

SELECT 'Minimum monthly purchases parameter must be > 0';

LEAVE proc;

END IF;

IF min\_dollar\_amount\_purchased = 0.00 THEN

SELECT 'Minimum monthly dollar amount purchased parameter must be > $0.00';

LEAVE proc;

END IF;

/\* Determine start and end time periods \*/

SET last\_month\_start = DATE\_SUB(CURRENT\_DATE(), INTERVAL 1 MONTH);

SET last\_month\_start = STR\_TO\_DATE(CONCAT(YEAR(last\_month\_start),'-',MONTH(last\_month\_start),'-01'),'%Y-%m-%d');

SET last\_month\_end = LAST\_DAY(last\_month\_start);

/\*

Create a temporary storage area for

Customer IDs.

\*/

CREATE TEMPORARY TABLE tmpCustomer (customer\_id SMALLINT UNSIGNED NOT NULL PRIMARY KEY);

/\*

Find all customers meeting the

monthly purchase requirements

\*/

INSERT INTO tmpCustomer (customer\_id)

SELECT p.customer\_id

FROM payment AS p

WHERE DATE(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;

/\* Populate OUT parameter with count of found customers \*/

SELECT COUNT(\*) FROM tmpCustomer INTO count\_rewardees;

/\*

Output ALL customer information of matching rewardees.

Customize output as needed.

\*/

SELECT c.\*

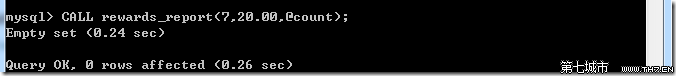
FROM tmpCustomer AS t

INNER JOIN customer AS c ON t.customer\_id = c.customer\_id;

/\* Clean up \*/

DROP TABLE tmpCustomer;

END



因为不满足系统时间，所以结果为0

## Stored Functions

### The get\_customer\_balance Function

返回指定客户到某一个日期欠的钱。 所有的rental\_rate+超期费用-所有的payment

Description

Theget\_customer\_balancefunction returns the current amount owing on a specified customer's account.

Parameters

p\_customer\_id: The ID of the customer to check, from thecustomer\_idcolumn of thecustomertable.

p\_effective\_date: The cutoff date for items that will be applied to the balance. Any rentals, payments, and so forth after this date are not counted.

Return Values

This function returns the amount owing on the customer's account.

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

SELECT IFNULL(SUM(film.rental\_rate),0) INTO v\_rentfees

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;

SELECT IFNULL(SUM(IF((TO\_DAYS(rental.return\_date) - TO\_DAYS(rental.rental\_date)) > film.rental\_duration,

((TO\_DAYS(rental.return\_date) - TO\_DAYS(rental.rental\_date)) - film.rental\_duration),0)),0) INTO v\_overfees

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;SELECT IFNULL(SUM(payment.amount),0) INTO v\_payments

FROM payment

WHERE payment.payment\_date <= p\_effective\_date

AND payment.customer\_id = p\_customer\_id;

RETURN v\_rentfees + v\_overfees - v\_payments;

END

### The inventory\_held\_by\_customer Function

返回指定的inventory现在在那个客户的customer\_id

Theinventory\_held\_by\_customerfunction returns thecustomer\_idof the customer who has rented out the specified inventory item.

Parameters

p\_inventory\_id: The ID of the inventory item to be checked.

Return Values

This function returns thecustomer\_idof the customer who is currently renting the item, orNULLif the item is in stock.

BEGIN

DECLARE v\_customer\_id INT;

DECLARE EXIT HANDLER FOR NOT FOUND RETURN NULL;

SELECT customer\_id INTO v\_customer\_id

FROM rental

WHERE return\_date IS NULL

AND inventory\_id = p\_inventory\_id;

RETURN v\_customer\_id;

END

### The inventory\_in\_stock Function

返回指定的inventory是否在商店里

Theinventory\_in stock functionreturns a boolean value indicating whether the inventory item specified is in stock.

Parameters

p\_inventory\_id: The ID of the inventory item to be checked.

Return Values

This function returnsTRUEorFALSEto indicate whether the item specified is in stock.

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

SELECT COUNT(\*) INTO v\_rentals

FROM rental

WHERE inventory\_id = p\_inventory\_id;

IF v\_rentals = 0 THEN

RETURN TRUE;

END IF;

SELECT COUNT(rental\_id) INTO v\_out

FROM inventory LEFT JOIN rental USING(inventory\_id)

WHERE inventory.inventory\_id = p\_inventory\_id

AND rental.return\_date IS NULL;

IF v\_out > 0 THEN

RETURN FALSE;

ELSE

RETURN TRUE;

END IF;

END

## Triggers

### The customer\_create\_date Trigger

插入新客户时，设置create\_date为当前日期

-- Trigger to enforce create dates on INSERT

CREATE TRIGGER customer\_create\_date BEFORE INSERT ON customer

FOR EACH ROW SET NEW.create\_date = NOW();

### The payment\_date Trigger

payment新增记录时，payment\_date设置为当前日期

-- Trigger to enforce payment\_date during INSERT

CREATE TRIGGER payment\_date BEFORE INSERT ON payment

FOR EACH ROW SET NEW.payment\_date = NOW();

### The rental\_date Trigger

rental表新增记录时，rental\_date设置为当前日期

-- Trigger to enforce rental\_date on INSERT

CREATE TRIGGER rental\_date BEFORE INSERT ON rental

FOR EACH ROW SET NEW.rental\_date = NOW();

### The ins\_film Trigger

确保 film\_text 和 film同步插入

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;;

### The upd\_film Trigger

确保 film\_text 和 film同步更新

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;;

### The del\_film Trigger

确保 film\_text 和 film同步删除

CREATE TRIGGER `del\_film` AFTER DELETE ON `film` FOR EACH ROW BEGIN

DELETE FROM film\_text WHERE film\_id = old.film\_id;

END;;