Table of Contents

1. Introduction

This document describes the Sakila sample database -- its history, installation, structure and usage.

The Sakila sample database is developed and maintained by Mike Hillyer of the MySQL AB documentation team and is intended to provide a standard schema that can be used for examples in books, tutorials, articles, samples, etc. The Sakila sample database also serves to highlight the latest features of MySQL including Views, Stored Procedures, Triggers, etc.

The Sakila sample database is the result of support and feedback from the MySQL user community and feedback and user input is always appreciated. Please direct all feedback to docs@mysql.com

Additional information on the Sakila sample database and its usage can be found through the MySQL forums [http://forums.mysql.com/list.php?121].

Feedback, bug reports, and requests can be directed to the MySQL AB documentation team at `<docs@mysql.com>`.

2. History

The Sakila sample database was designed as a replacement to the `World` [http://downloads.mysql.com/docs/#examples] sample database, also provided by MySQL AB.

The World sample database provides a set of tables containing information on the countries and cities of the world and is useful for basic queries, but lacked structures for testing MySQL-specific functionality and new features found in MySQL 5.

Development of the Sakila sample database began in early 2005 . Early designs were based on the database used in the Dell Whitepaper Three Approaches to MySQL Applications on Dell PowerEdge Servers [http://www.dell.com/downloads/global/solutions/mysql_apps.pdf].

Where Dell's sample database was designed to represent an online DVD store, the Sakila sample database is designed to represent a DVD rental store. The Sakila sample database still borrows film

and actor names from the Dell sample database.

Development was accomplished using MySQL Query Browser for schema design with the tables being populated by a combination of MySQL Query Browser and custom scripts, in addition to contributor efforts (see Section 6, “Acknowledgements”).

After the basic schema was completed, various views, stored routines, and triggers were added to the schema, after which the sample data was populated. After a series of review versions, the first official version of the Sakila sample database was released in March 2006.

3. Installation

The Sakila sample database is divided into two installation files: `sakila-schema.sql` and `sakila-data.sql`.

The `sakila-schema.sql` file contains all the `CREATE` statements required to create the structure of the Sakila database including tables, views, stored procedures and triggers.

The `sakila-data.sql` file contains the `INSERT` statements required to populate the structure created by the `sakila-schema.sql` file, along with definitions for triggers that must be created after the initial data load.

To install the Sakila sample database, follow these steps:

1. Extract the installation archive to a temporary location on the MySQL server machine such as `C:\temp\` or `/tmp/`.

2. Connect to the MySQL server using the `mysql` command-line client using the following command:

   ```
   mysql -u root -p
   ```

   Enter your password when prompted. Non-root accounts can be used as long as those accounts have privileges to create new databases.

3. Execute the `sakila-schema.sql` script to create the database structure with the following command:

   ```
   SOURCE C:/temp/sakila-schema.sql;
   ```

   Replace *`C:/temp/`* with the path to the `sakila-schema.sql` on the server.

4. Execute the `sakila-data.sql` script to populate the database structure with the following command:

   ```
   SOURCE C:/temp/sakila-data.sql;
   ```

   Replace *`C:/temp/`* with the path to the `sakila-data.sql` on the server.

5. Confirm the sample database is installed correctly:

   ```
   mysql> USE sakila;
   Database changed

   mysql> SHOW TABLES;
   +----------------------------+
   | Tables_in_sakila           |
   +----------------------------+
   | actor                      |
   | address                    |
   | category                   |
   | city                       |
   | country                    |
   | customer                   |
   | customer_list              |
   ```

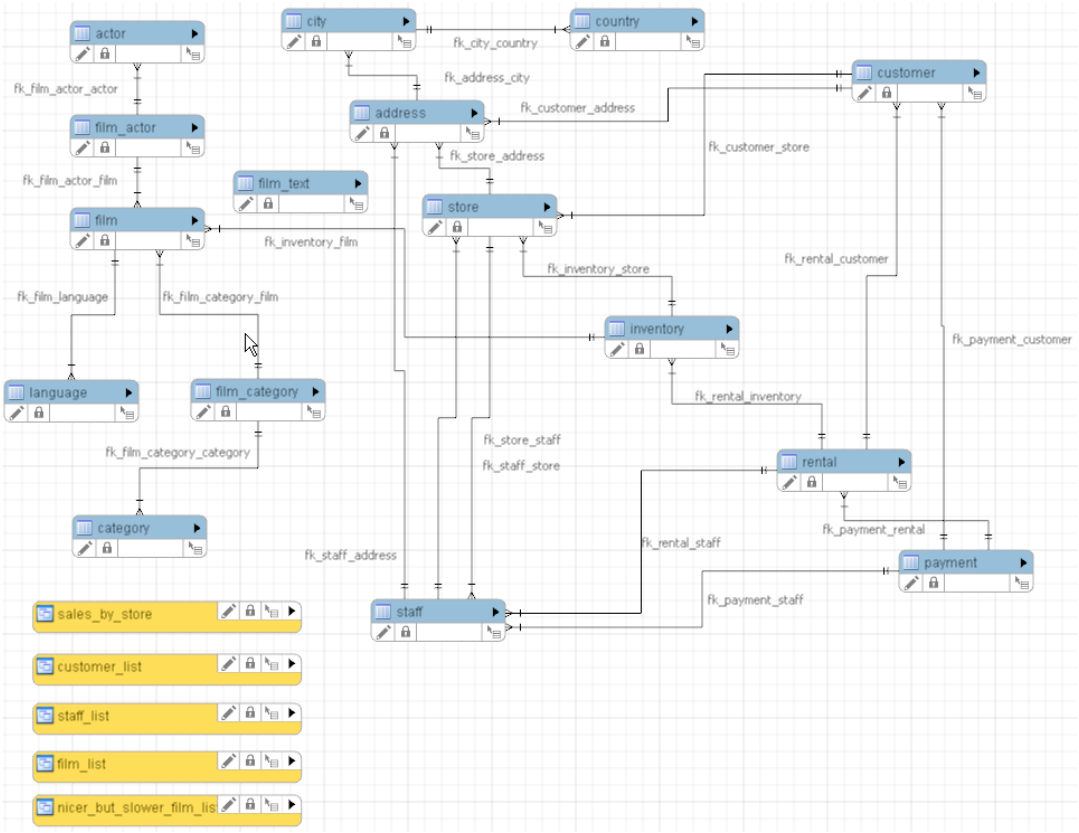
```
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
22 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM film;
+----------+
| COUNT(*) |
+----------+
| 1000     |
+----------+
1 row in set (0.02 sec)

mysql> SELECT COUNT(*) FROM film_text;
+----------+
| COUNT(*) |
+----------+
| 1000     |
+----------+
1 row in set (0.00 sec)
```

4. Structure

This section provides an overview of the structure of the Sakila sample database.

The following diagram provides an overview of the Sakila structure. The source file (for use with MySQL Workbench) is included in the Sakila distribution and is named `sakila.mwb`.

Figure 1. The Sakila schema



4.1. Tables

This section describes the tables that make up the Sakila sample database, in alphabetical order.

> **Note**
>
> All tables, with the exception of the `film_text` table, contain a `last_update` column that serves as a timestamp marking the last time each row was updated. This column will not be listed for each table in the table's column list.

4.1.1. The actor Table

The `actor` table lists information for all actors.

The actor table is joined to the film table by means of the film_actor table.

Columns:

- `actor_id` - Surrogate primary key used to uniquely identify each actor in the table.
- `first_name` - The actor's first name.
- `last_name` - The actor's last name.

4.1.2. The address Table

The `address` table contains address information for customers, staff, and stores.

The address table primary key appears as a foreign key in the customer, staff and store tables.

Columns:

- `address_id` - Surrogate primary key used to uniquely identify each address in the table.
- `address` - First line of an address.
- `address2` - Optional second line of an address.
- `district` - The region of an address, this may be a state, province, prefecture, etc.
- `city_id` - A foreign key pointing to the city table.
- `postal_code` - The postal code or zip code of the address (where applicable).
- `phone` - The telephone number for the address.

4.1.3. The category Table

The `category` table contains a list of categories which the various categories that can be assigned to a film.

The category table is joined to the film table by means of the film_category tables.

Columns:

- `category_id` - Surrogate primary key used to uniquely identify each category in the table.
- `name` - The name of the category.

4.1.4. The city Table

The `City` table contains a list of cities.

The city table is referred to by a foreign key in the address table and refers to the country table using a foreign key.

Columns:

- `city_id` - Surrogate primary key used to uniquely identify each city in the table.
- `city` - The name of the city.
- `country_id` - Foreign key identifying the country that the city belongs to.

4.1.5. The country Table

The `country` table contains a list of countries.

The country table is referred to by a foreign key in the city table.

Columns:

- `country_id` - Surrogate primary key used to uniquely identify each country in the table.
- `country` - The name of the country.

4.1.6. The customer Table

The `customer` table contains a list of all customers.

The customer table is referred to in the payment and rental tables and refers to the address and store tables using foreign keys.

Columns:

- `customer_id` - Surrogate primary key used to uniquely identify each customer in the table.
- `store_id` - Foreign key identifying the customer's 'home store'. Customers are not limited to renting only from this store, but this is the store they generally shop at.
- `first_name` - The customer's first name.
- `last_name` - The customer's last name.
- `email` - The customer's email address.
- `address_id` - Foreign key identifying the customer's address in the address table.
- `active` - Indicates whether the customer is an active customer. Setting this to `FALSE` serves as an alternative to deleting a customer outright. Most queries should have a `WHERE active = TRUE` clause.
- `create_date` - The date the customer was added to the system. This date is automatically set using a trigger during an `INSERT`.

4.1.7. The film Table

The `film` table is a list of all films potentially in stock in the stores. Actually copies of each film and represented in the inventory table.

The film table refers to the language table and is referred to by the film_category, film_actor and inventory tables.

Columns:

- `film_id` - Surrogate primary key used to uniquely identify each film in the table.
- `title` - The title of the film.
- `description` - A short description or plot summary of the film.
- `release_year` - The year in which the movie was released.
- `language_id` - Foreign key pointing at the language table, identified the language of the film.
- `original_language_id` - Foreign key pointing at the language table, identified the original language of the film. Used when a film has been dubbed into a new language.
- `rental_duration` - The period in days that the film is rented for.
- `rental_rate` - The cost to rent the film for the period specified in the `rental_duration` column.
- `length` - The duration of the film, in minutes.
- `replacement_cost` - The amount charged to the customer if the film is not returned or is re-

turned in a damaged state.

- `rating` - The rating assigned to the film. Can be one of: `G`, `PG`, `PG-13`, `R` or `NC-17`.
- `special_features` - Lists which common special features are included on the DVD. Can be zero or more of: `Trailers`, `Commentaries`, `Deleted Scenes`, `Behind the Scenes`.

4.1.8. The film_actor Table

The `film_actor` table is used to support a many-to-many relationship between films and actors. For each actor in a given film, there will be one row in the film_actor table listing the actor and film.

The film_actor table refers to the film and actor tables using foreign keys.

Columns:

- `actor_id` - Foreign key identifying the actor.
- `film_id` - Foreign key identifying the film.

4.1.9. The film_category Table

The `film_category` table is used to support a many-to-many relationship between films and categories. For each category applied to a film, there will be one row in the film_category table listing the category and film.

The film_category table refers to the film and category tables using foreign keys.

Columns:

- `film_id` - Foreign key identifying the film.
- `category_id` - Foreign key identifying the category.

4.1.10. The film_text Table

The `film_text` table is the only table in the Sakila sample database that uses a `MyISAM` storage engine. This table is provided to allow for fulltext searching of the titles and descriptions of the films listed in the film table.

The film_text table contains the `film_id`, `title` and `description` columns of the film table, with the contents of the table kept in sync with the film table by means of triggers on the film table's `INSERT`, `UPDATE` and `DELETE` operations (see Section 4.5, “Triggers”).

Columns:

- `film_id` - Surrogate primary key used to uniquely identify each film in the table.
- `title` - The title of the film.
- `description` - A short description or plot summary of the film.

The contents of the film_text table should never be modified directly, all changes should be made to the film table instead.

4.1.11. The inventory Table

The `inventory` table contains one row for each copy of a given film in a given store.

The inventory table refers to the film and store tables using foreign keys and is referred to by the rental table.

Columns:

- `inventory_id` - Surrogate primary key used to uniquely identify each item in inventory.
- `film_id` - Foreign key pointing to the film this item represents.
- `store_id` - Foreign key pointing to the store stocking this item.

4.1.12. The language Table

The `language` table is a lookup table listing the possible languages a film can be listed as having for its language and original language.

The language table is referred to by the film table.

Columns:

- `language_id` - Surrogate primary key used to uniquely identify each language.
- `name` - The English name of the language.

4.1.13. The payment Table

The `payment` table records each payment made by a customer, with information such as the amount and the rental being paid for (when applicable).

The payment table refers to the customer, rental and staff tables.

Columns:

- `payment_id` - Surrogate primary key used to uniquely identify each payment.
- `customer_id` - The customer whose balance the payment is being applied to. Foreign key reference to the `customer` table.
- `staff_id` - The staff member who processed the payment. Foreign key reference to the `staff` table.
- `rental_id` - The rental that the payment is being applied to. This is optional because some payments are for outstanding fees and may not be directly related to a rental.
- `amount` - The amount of the payment.
- `payment_date` - The date the payment was processed.

4.1.14. The rental Table

The `rental` table contains one row for each rental of each inventory item with information about who rented what item, when it was rented and when it was returned.

The rental table refers to the inventory, customer and staff tables and is referred to by the payment table.

Columns:

- `rental_id` - Surrogate primary key that uniquely identifies the rental.
- `rental_date` - The date/time that the item was rented.
- `inventory_id` - The item being rented.
- `customer_id` - The customer renting the item.
- `return_date` - The date/time the item was returned.
- `staff_id` - The staff member who processed the rental.

4.1.15. The staff Table

The `staff` table lists all staff members, including information on email address, login information, and picture.

The staff table refers to the store and address tables using foreign keys, and is referred to by the rental, payment and store tables.

Columns:

- `staff_id` - Surrogate primary key that uniquely identifies the staff member.
- `first_name` - The first name of the staff member.
- `last_name` - The last name of the staff member.
- `address_id` - Foreign key to the staff member's address in the address table.
- `picture` - A BLOB containing a photograph of the employee.
- `email` - The staff member's email address.
- `store_id` - The staff member's 'home store'. The employee can work at other stores but is generally assigned to the store listed.
- `active` - Whether this is an active employee. If an employee leaves their row is not deleted from this table, instead this column is set to `FALSE`.
- `username` - The username used by the staff member to access the rental system.
- `password` - The password used by the staff member to access the rental system. The password should be stored as a hash using the `SHA1()` function.

4.1.16. The store Table

The `store` table lists all stores in the system. All inventory is assigned to specific stores, and staff and customers are assigned a 'home store'.

The store table refers to the staff and address tables using foreign keys and is referred to by the staff, customer and inventory tables.

Columns:

- `store_id` - Surrogate primary key that uniquely identifies the store.
- `manager_staff_id` - Foreign key identifying the manager of this store.
- `address_id` - Foreign key identifying the address of this store.

4.2. Views

This section describes the views that are included with the Sakila sample database, in alphabetical order.

4.2.1. The actor_info View

The `actor_info` view provides a list of all actors, including the films they have performed in, broken down by category.

The staff_list view incorporates data from the film, actor, category, film_actor and film_category tables.

4.2.2. The customer_list View

The `customer_list` view provides a list of customers, with first name and last name concatenated together and address information combined into a single view.

The customer_list view incorporates data from the customer, address, city and country tables.

4.2.3. The film_list View

The `film_list` view contains a formatted view of the film table, with a comma-separated list of the film's actors.

The film_list view incorporates data from the film, category, film_category, actor and film_actor tables.

4.2.4. The nicer_but_slower_film_list View

The `nicer_but_slower_film_list` view contains a formatted view of the film table, with a comma-separated list of the film's actors.

The nicer_but_slower_film_list view differs from the film_list view in the list of actors. The lettercase of the actor names is adjusted so that the first letter of each name is capitalized rather than have the name in all-caps.

As indicated in its name, the nicer_but_slower_film_list performs additional processing and therefore takes longer to return data than the film_list view.

The nicer_but_slower_film_list view incorporates data from the film, category, film_category, actor and film_actor tables.

4.2.5. The sales_by_film_category View

The `sales_by_film_category` view provides a list of total sales, broken down by individual film category.

Because a film can be listed in multiple categories, it is not advisable to calculate aggregate sales by totalling the rows of this view.

The sales_by_film_category view incorporates data from the category, payment, rental, inventory, film, film_category and category tables.

4.2.6. The sales_by_store View

The `sales_by_store` view provides a list of total sales, broken down by store.

The view returns the store location, manager name, and total sales.

The sales_by_store view incorporates data from the city, country, payment, rental, inventory, store, address and staff tables.

4.2.7. The staff_list View

The `staff_list` view provides a list of all staff members, including address and store information.

The staff_list view incorporates data from the staff and address tables.

4.3. Stored Procedures

This is a list of stored procedures included with the Sakila sample database, in alphabetical order.

All parameters listed are IN parameters unless listed otherwise.

4.3.1. The film_in_stock Stored Procedure

Description

The `film_in_stock` stored procedure is used to determine if there are any copies of a given film in stock at a given store.

Parameters

- `p_film_id` - The id of the film to be checked, from the `film_id` column of the `film` table.
- `p_store_id` - The id of the store to check for, from the `store_id` column of the `store` table.
- `p_film_count` - An OUT parameter that returns a count of the copies of the film in stock.

Return Values

This function returns a table of inventory id numbers for the copies of the film in stock.

Sample Usage

```
mysql> CALL film_in_stock(1,1,@count);
+--------------+
| inventory_id |
+--------------+
| 1            |
| 2            |
| 3            |
| 4            |
+--------------+
4 rows in set (0.06 sec)

Query OK, 0 rows affected (0.06 sec)

mysql> SELECT @count;
+--------+
| @count |
+--------+
| 4      |
+--------+
1 row in set (0.00 sec)
```

```
mysql>
```

4.3.2. The film_not_in_stock Stored Procedure

Description

The `film_not_in_stock` stored procedure is used to determine if 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 the `film_id` column of the `film` table.
- `p_store_id` - The id of the store to check for, from the `store_id` column of the `store` table.
- `p_film_count` - An OUT parameter that returns a count of the copies of the film not in stock.

Return Values

This function returns a table of inventory id numbers for the copies of the film not in stock.

Sample Usage

```
mysql> CALL film_not_in_stock(2,2,@count);
+--------------+
| inventory_id |
+--------------+
| 9            |
+--------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @count;
+--------+
| @count |
+--------+
| 1      |
+--------+
1 row in set (0.00 sec)
```

4.3.3. The rewards_report Stored Procedure

Description

The `rewards_report` stored procedure generates a customizable list of the top customers for the previous month.

Parameters

- `min_monthly_purchases` - The minimum number of purchases/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` - An OUT parameter 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 returned is of the same structure as the customer table.

Sample Usage

```
mysql> CALL rewards_report(7,20.00,@count);

...
| 598         | 1        | WADE        | DELVALLE     | WADE.DELVALLE@sakilacustomer.org         |
| 599         | 2        | AUSTIN      | CINTRON      | AUSTIN.CINTRON@sakilacustomer.org        |
...

42 rows in set (0.11 sec)

Query OK, 0 rows affected (0.67 sec)

mysql> SELECT @count;
+--------+
| @count |
+--------+
| 42     |
+--------+
1 row in set (0.00 sec)
```

4.4. Stored Functions

This section lists the stored functions included with the Sakila sample database.

4.4.1. The get_customer_balance Function

The `get_customer_balance` function returns the current amount owing on a specified customer's account.

Parameters

- `p_customer_id` - The id of the customer to check, from the `customer_id` column of the `customer` table.
- `p_effective_date` - The cutoff date for items that will be applied to the balance. Any rentals/payments/etc after this date are not counted.

Return Values

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

Sample Usage

```
mysql> SELECT get_customer_balance(298,NOW());
+---------------------------------+
| get_customer_balance(298,NOW()) |
+---------------------------------+
|  22.00                          |
+---------------------------------+
1 row in set (0.00 sec)
```

4.4.2. The inventory_held_by_customer Function

The `inventory_held_by_customer` function returns the customer_id of 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 functions returns the customer_id of the customer who is currently renting the item, or NULL if the item is in stock.

Sample Usage

```
mysql> SELECT inventory_held_by_customer(8);
+------------------------------+
| inventory_held_by_customer(8) |
+------------------------------+
| NULL                         |
+------------------------------+
1 row in set (0.14 sec)

mysql> SELECT inventory_held_by_customer(9);
+------------------------------+
| inventory_held_by_customer(9) |
+------------------------------+
| 366                          |
+------------------------------+
1 row in set (0.00 sec)

mysql>
```

4.4.3. The inventory_in_stock Function

The `inventory_function` function returns 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 returns TRUE or FALSE depending on whether the item specified is in stock.

Sample Usage

```
mysql> SELECT inventory_in_stock(9);
+-----------------------+
| inventory_in_stock(9) |
+-----------------------+
| 0                     |
+-----------------------+
1 row in set (0.03 sec)

mysql> SELECT inventory_in_stock(8);
+-----------------------+
| inventory_in_stock(8) |
+-----------------------+
| 1                     |
+-----------------------+
1 row in set (0.00 sec)
```

4.5. Triggers

This section lists the triggers in the Sakila sample database.

4.5.1. The customer_create_date Trigger

The `customer_create_date` trigger sets the create_date column of the customer table to the current time and date as rows are inserted.

4.5.2. The payment_date Trigger

The `payment_date` trigger sets the payment_date column of the payment table to the current time and date as rows are inserted.

4.5.3. The rental_date Trigger

The `rental_date` trigger sets the rental_date column of the rental table to the current time and date as rows are inserted.

4.5.4. The ins_film Trigger

The `ins_film` trigger duplicates all INSERT operations on the film table to the film_text table.

4.5.5. The upd_film Trigger

The `upd_film` trigger duplicates all UPDATE operations on the film table to the film_text table.

4.5.6. The del_film Trigger

The `del_film` trigger duplicates all DELETE operations on the film table to the film_text table.

5. Usage Examples

These are a few usage examples of how to perform common operations using the Sakila sample database. While these operations are good candidates for stored procedures and views, such implementation is intentionally left as an exercise to the user.

Rent a DVD

To rent a DVD, first confirm that the given inventory item is in stock, then insert a row into the rental table. After the rental is created, insert a row into the payment table. Depending on business rules, you may also need to check if the customer has an outstanding balance before processing the rental.

```
mysql> SELECT INVENTORY_IN_STOCK(10);
+-------------------------+
| INVENTORY_IN_STOCK(10) |
+-------------------------+
| 1                       |
+-------------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO rental(rental_date, inventory_id, customer_id, staff_id)
    -> VALUES(NOW(), 10, 3, 1);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @balance := get_customer_balance(3, NOW());
+-----------------------------------------+
| @balance := get_customer_balance(3, NOW()) |
+-----------------------------------------+
| 4.99                                    |
+-----------------------------------------+
1 row in set (0.01 sec)

mysql> INSERT INTO payment (customer_id, staff_id, rental_id, amount,  payment_date)
    -> VALUES(3,1,LAST_INSERT_ID(), @balance, NOW());
Query OK, 1 row affected (0.00 sec)
```

Return a DVD

To return a DVD, we update the rental table and set the return date. To do this, we first need to identify the rental_id to update based on the inventory_id of the item being returned. Depending on the situation we may then need to check the customer balance and perhaps process a payment for overdue fees by inserting a row into the payment table.

```
mysql> SELECT rental_id
    -> FROM rental
    -> WHERE inventory_id = 10
    -> AND customer_id = 3
```

```
    -> AND return_date IS NULL;
+-----------+
| rental_id |
+-----------+
| 16050     |
+-----------+
1 row in set (0.00 sec)

mysql> UPDATE rental
    -> SET return_date = NOW()
    -> WHERE rental_id = @rentID;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT get_customer_balance(3, NOW());
+--------------------------------+
| get_customer_balance(3, NOW()) |
+--------------------------------+
| 0.00                           |
+--------------------------------+
1 row in set (0.09 sec)
```

Find Overdue DVDs

Many DVD stores produce a daily list of overdue rentals so that customers can be contacted and asked to return their overdue DVDs.

To create such a list, we search the rental table for films with a return date that is NULL and where the rental date is further in the past than the rental duration specified in the film table. If so, the film is overdue and we should return the name of the film along with the customer name and phone number.

```
mysql> SELECT CONCAT(customer.last_name, ', ', customer.first_name) AS customer,

    -> address.phone, film.title
    -> FROM rental INNER JOIN customer ON rental.customer_id = customer.customer
_id
    -> INNER JOIN address ON customer.address_id = address.address_id
    -> INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
    -> INNER JOIN film ON inventory.film_id = film.film_id
    -> WHERE rental.return_date IS NULL
    -> AND rental_date + INTERVAL film.rental_duration DAY < CURRENT_DATE()
    -> LIMIT 5;
+------------------+--------------+------------------+
| customer         | phone        | title            |
+------------------+--------------+------------------+
| OLVERA, DWAYNE   | 62127829280  | ACADEMY DINOSAUR |
| HUEY, BRANDON    | 99883471275  | ACE GOLDFINGER   |
| BROWN, ELIZABETH | 10655648674  | AFFAIR PREJUDICE |
| OWENS, CARMEN    | 272234298332 | AFFAIR PREJUDICE |
| HANNON, SETH     | 864392582257 | AFRICAN EGG      |
+------------------+--------------+------------------+
5 rows in set (0.02 sec)
```

6. Acknowledgements

The following individuals and organizations have contributed to the development of the Sakila sample database.

- Roland Bouman, Author [http://rpbouman.blogspot.com/] - Provided valuable feedback throughout the development process, contributed sample views and stored procedures.
- Ronald Bradford, Developer [http://blog.arabx.com.au/] - Developed first sample application [http://sakila.arabx.com.au/index.htm] for use with the Sakila sample database.
- Dave Jaffe, Dell [http://www.dell.com/mysql] - Provided schema used in Dell whitepaper and secured permission to use parts thereof in the Sakila sample database.
- Giuseppe Maxia, CTO of Stardata [http://www.stardata.it//index_en.html] - Provided valuable feedback throughout the development process, populated some of the sample data, provided some of the sample views and triggers.

- Jay Pipes, MySQL Community Advocate [http://www.jpipes.com/] - Provided some of the sample stored procedures.
- Zak Greant, Community Advocate and Author [http://zak.greant.com] - Provided advice and feedback on licensing.

In addition to the individuals mentioned above, there are various individuals in MySQL AB and the community that have provided feedback during the course of development.

7. License for the Sakila Sample Database

The contents of the sakila-schema.sql and sakila-data.sql files are licensed under the New BSD license.

Information on the New BSD license can be found at http://www.opensource.org/licenses/bsd-license.php and http://en.wikipedia.org/wiki/BSD_License.

The additional materials included in the Sakila distribution, including this documentation, are not licensed under an open license. Use of this documentation is subject to the following terms:

- Conversion to other formats is allowed, but the actual content may not be altered or edited in any way.
- You may create a printed copy for your own personal use.
- For all other uses, such as selling printed copies or using (parts of) the manual in another publication, prior written agreement from MySQL AB is required.

Please email <docs@mysql.com> for more information or if you are interested in doing a translation.

8. Note for Authors

When using the Sakila sample database for articles and books, it is strongly recommended that you explicitly list the version of the Sakila sample database that is used in your examples. This way readers will download the same version for their use and not encounter any differences in their results that may occur from upgrades to the data or schema.

9. Sakila Change History

9.1. Version 0.8

- Added actor_info view.
- Changed error handler for inventory_held_by_customer function. Function now has an exit handler for NOT FOUND instead of the more cryptic 1329.
- Added template for new BSD license to schema and data files.
- Added READS SQL DATA to the procedures and functions where appropriate to support loading on MySQL 5.1.
- Fixed date range issue in rewards_report procedure (thanks Goplat).

9.2. Version 0.7

- Fixed bug in sales_by_store view that caused the same manager to be listed for every store.
- Fixed bug in inventory_held_by_customer function that caused function to return multiple rows.
- Moved rental_date trigger to sakila-data.sql file to prevent it from interfering with data loading.

9.3. Version 0.6

- Added `film_in_stock` stored procedure.
- Added `film_not_in_stock` stored procedure.
- Added `inventory_help_by_customer` stored function.
- Added `inventory__in_stock` stored function.
- Optimized data file for loading (multi-row INSERT statements, transactions). (Thanks Giuseppe)
- Fixed error in payment table loading script that cause infinitely increasing payment amounts.

9.4. Version 0.5

- Added views `sales_by_store` and `sales_by_film_category` submitted by Jay Pipes.
- Added stored procedure `rewards_report` submitted by Jay Pipes.
- Added stored procedure `get_customer_balance`.
- Added `sakila-data.sql` file to load data into sample database.

9.5. Version 0.4

- Added password column to staff table (VARCHAR(40) BINARY DEFAULT NULL)

9.6. Version 0.3

- Changed address.district to VARCHAR(20)
- Changed customer.first_name to VARCHAR(45)
- Changed customer.last_name to VARCHAR(45)
- Changed customer.email to VARCHAR(50)
- payment.rental_id added - NULLable INT column
- Foreign key added for payment.rental_id to rental.rental_id
- rental.rental_id added, INT Auto_Increment, made into surrogate primary key, old primary key changed to UNIQUE key.

9.7. Version 0.2

- All tables have a `last_update` TIMESTAMP with traditional behavior (DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)
- `actor_id` is now a SMALLINT
- `address_id` is now a SMALLINT
- `category_id` is now a TINYINT
- `city_id` is now a SMALLINT
- `country_id` is now a SMALLINT
- `customer_id` is now a SMALLINT
- `first_name`, `last_name` for customer table are now CHAR instead of VARCHAR
- `customer` table now has email CHAR(50)
- `create_date` on customer table is now DATETIME (to accommodate last_update TIMESTAMP)
- `customer` table has a new ON INSERT trigger that enforces `create_date` column being set to NOW()
- `film_id` is now SMALLINT
- `film.description` now has DEFAULT NULL
- `film.release_year` added with type YEAR
- `film.language_id` and `film.original_language_id` added along with language table. For foreign films that may have been subbed. `original_language_id` can be NULL, `language_id` is NOT NULL
- `film.length` is now SMALLINT
- `film.category_id` column removed
- New table: `film_category` - allows for multiple categories per film
- `film_text.category_id` column removed
- `inventory_id` is now MEDIUMINT
- `payment_id` is now SMALLINT
- `payment.payment_date` is now DATETIME
- Trigger added to `payment` table to enforce that payment_date is set to NOW() upon INSERT
- `rental.rent_date` is now rental.rental_date and is now DATETIME
- Trigger added to `rental` table to enforce that rental_date is set to NOW() upon INSERT
- `staff_id` is now TINYINT
- `staff.email` added (VARCHAR(50))
- `staff.username` added (VARCHAR(16))
- `store_id` is now TINYINT
- VIEW `film_list` updated to handle new film_category table

- VIEW `nicer_but_slower_film_list` updated to handle new film_category table