Sakila Sample Database

Table of Contents

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

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2025-08-04 (revision: 83188)

1 Preface and Legal Notices

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

Legal Notices

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

2 Introduction

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

Additional information on the Sakila sample database and its usage can be found through the MySQL forums.

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 using the http://www.mysql.com/company/contact/. For bug reports, use MySQL Bugs.

3 History

The Sakila sample database was designed as a replacement to the `world` sample database, also provided by Oracle.

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 lacks structures for testing MySQL-specific functionality and features found in MySQL 5 and higher.

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.

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 8, “Acknowledgments”).

After the basic schema was completed, various views, stored routines, and triggers were added to the schema; then 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.

4 Installation

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

> **Note**
>
> Sakila contains MySQL version specific comments, in that the sakila schema and data depends on the version of your MySQL server. For example, MySQL server 5.7.5 added support for spatial data indexing to `InnoDB`, so the **address** table will include a spatial-aware **location** column for MySQL 5.7.5 and higher.

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.

The `sakila.mwb` file is a MySQL Workbench data model that you can open within MySQL Workbench to examine the database structure. For more information, see MySQL Workbench.

To install the Sakila sample database, follow these steps:

1. Extract the installation archive to a temporary location such as `C:\temp\` or `/tmp/`. When you unpack the archive, it creates a directory named `sakila-db` that contains the `sakila-schema.sql` and `sakila-data.sql` files.

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

   ```
   $> mysql -u root -p
   ```

   Enter your password when prompted. A non-`root` account can be used, provided that the account has privileges to create new databases.

3. Execute the `sakila-schema.sql` script to create the database structure, and execute the `sakila-data.sql` script to populate the database structure, by using the following commands:

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

   Replace the paths to the `sakila-schema.sql` and `sakila-data.sql` files with the actual paths on your system.

   Note

   On Windows, use slashes rather than backslashes when executing the `SOURCE` command.

4. Confirm that the sample database is installed correctly. Execute the following statements. You should see output similar to that shown here.

   ```
   mysql> USE sakila;
   Database changed

   mysql> SHOW FULL TABLES;
   +----------------------------+------------+
   | Tables_in_sakila           | Table_type |
   +----------------------------+------------+
   | actor                      | BASE TABLE |
   | actor_info                 | VIEW       |
   | address                    | BASE TABLE |
   | category                   | BASE TABLE |
   | city                       | BASE TABLE |
   | country                    | BASE TABLE |
   | customer                   | BASE TABLE |
   | customer_list              | VIEW       |
   | film                       | BASE TABLE |
   | film_actor                 | BASE TABLE |
   | film_category              | BASE TABLE |
   | film_list                  | VIEW       |
   | film_text                  | BASE TABLE |
   | inventory                  | BASE TABLE |
   | language                   | BASE TABLE |
   | nicer_but_slower_film_list | VIEW       |
   | payment                    | BASE TABLE |
   | rental                     | BASE TABLE |
   | sales_by_film_category     | VIEW       |
   | sales_by_store             | VIEW       |
   | staff                      | BASE TABLE |
   | staff_list                 | VIEW       |
   | store                      | BASE TABLE |
   +----------------------------+------------+
   23 rows in set (0.01 sec)

   mysql> SELECT COUNT(*) FROM film;
   +----------+
   ```

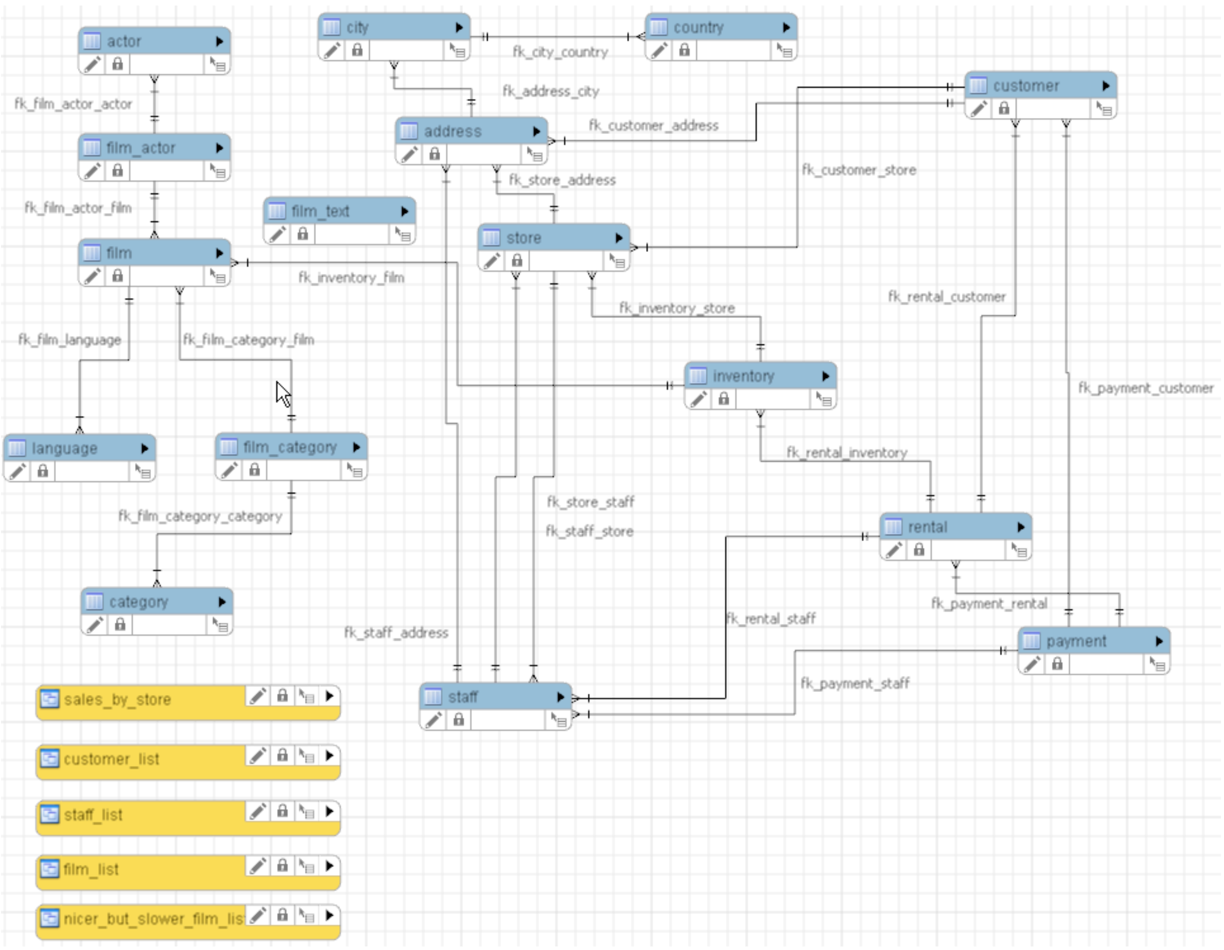
```
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

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

5 Structure

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

Figure 1 The Sakila Schema



5.1 Tables

The following sections describe the tables that make up the Sakila sample database, in alphabetic order.

5.1.1 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`: A surrogate primary key used to uniquely identify each actor in the table.
- `first_name`: The actor first name.
- `last_name`: The actor last name.
- `last_update`: When the row was created or most recently updated.

5.1.2 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`: A surrogate primary key used to uniquely identify each address in the table.
- `address`: The first line of an address.
- `address2`: An 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.
- `last_update`: When the row was created or most recently updated.
- `location`: A Geometry column with a spatial index on it.

> **Note**
>
> The spatial `location` column is supported as of MySQL 5.7.5. This column is added only when executing the Sakila SQL files against MySQL server 5.7.5 and higher. Additionally, `SPATIAL KEY idx_location` is also added.

5.1.3 The category Table

The `category` table lists the categories that can be assigned to a film.

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

Columns

- `category_id`: A surrogate primary key used to uniquely identify each category in the table.
- `name`: The name of the category.
- `last_update`: When the row was created or most recently updated.

5.1.4 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`: A surrogate primary key used to uniquely identify each city in the table.
- `city`: The name of the city.
- `country_id`: A foreign key identifying the country that the city belongs to.
- `last_update`: When the row was created or most recently updated.

5.1.5 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`: A surrogate primary key used to uniquely identify each country in the table.
- `country`: The name of the country.
- `last_update`: When the row was created or most recently updated.

5.1.6 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`: A surrogate primary key used to uniquely identify each customer in the table.
- `store_id`: A foreign key identifying the customer “home store.” Customers are not limited to renting only from this store, but this is the store at which they generally shop.
- `first_name`: The customer first name.
- `last_name`: The customer last name.
- `email`: The customer email address.
- `address_id`: A foreign key identifying the customer 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`.
- `last_update`: When the row was created or most recently updated.

5.1.7 The film Table

The `film` table is a list of all films potentially in stock in the stores. The actual in-stock copies of each film are 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: A 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: A foreign key pointing at the language table; identifies the language of the film.
- original_language_id: A foreign key pointing at the language table; identifies the original language of the film. Used when a film has been dubbed into a new language.
- rental_duration: The length of the rental period, in days.
- 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 returned 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.
- last_update: When the row was created or most recently updated.

5.1.8 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: A foreign key identifying the actor.
- film_id: A foreign key identifying the film.
- last_update: When the row was created or most recently updated.

5.1.9 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: A foreign key identifying the film.
- category_id: A foreign key identifying the category.

- last_update: When the row was created or most recently updated.

5.1.10 The film_text 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 synchrony with the film table by means of triggers on film table INSERT, UPDATE and DELETE operations (see Section 5.5, “Triggers”).

Before MySQL server 5.6.10, the film_text table was the only table in the Sakila sample database that used the MyISAM storage engine. This is because full-text search is used for titles and descriptions of films listed in the film table. MyISAM was used because full-text search support with InnoDB was not available until MySQL server 5.6.10.

Columns

- film_id: A 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.

5.1.11 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: A surrogate primary key used to uniquely identify each item in inventory.
- film_id: A foreign key pointing to the film this item represents.
- store_id: A foreign key pointing to the store stocking this item.
- last_update: When the row was created or most recently updated.

5.1.12 The language Table

The language table is a lookup table listing the possible languages that films can have for their language and original language values.

The language table is referred to by the film table.

Columns

- language_id: A surrogate primary key used to uniquely identify each language.
- name: The English name of the language.
- last_update: When the row was created or most recently updated.

5.1.13 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`: A surrogate primary key used to uniquely identify each payment.
- `customer_id`: The customer whose balance the payment is being applied to. This is a foreign key reference to the `customer` table.
- `staff_id`: The staff member who processed the payment. This is a 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.
- `last_update`: When the row was created or most recently updated.

5.1.14 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`: A surrogate primary key that uniquely identifies the rental.
- `rental_date`: The date and time that the item was rented.
- `inventory_id`: The item being rented.
- `customer_id`: The customer renting the item.
- `return_date`: The date and time the item was returned.
- `staff_id`: The staff member who processed the rental.
- `last_update`: When the row was created or most recently updated.

5.1.15 The staff Table

The `staff` table lists all staff members, including information for 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`: A 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`: A foreign key to the staff member address in the `address` table.
- `picture`: A `BLOB` containing a photograph of the employee.

- `email`: The staff member email address.
- `store_id`: The staff member “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 employees leave, their rows are not deleted from this table; instead, this column is set to `FALSE`.
- `username`: The user name 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 `SHA2()` function.
- `last_update`: When the row was created or most recently updated.

5.1.16 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`: A surrogate primary key that uniquely identifies the store.
- `manager_staff_id`: A foreign key identifying the manager of this store.
- `address_id`: A foreign key identifying the address of this store.
- `last_update`: When the row was created or most recently updated.

5.2 Views

The following sections describe the views that are included with the Sakila sample database, in alphabetic order.

5.2.1 The actor_info View

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

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

5.2.2 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.

5.2.3 The film_list View

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

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

5.2.4 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 having the name in all-caps.

As indicated in its name, the nicer_but_slower_film_list view 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.

5.2.5 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.

5.2.6 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.

5.2.7 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.

5.3 Stored Procedures

The following sections describe the stored procedures included with the Sakila sample database, in alphabetic order.

All parameters listed are IN parameters unless listed otherwise.

5.3.1 The film_in_stock Stored Procedure

Description

The film_in_stock stored procedure determines whether any copies of a given film are 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 procedure produces a table of inventory ID numbers for the copies of the film in stock, and returns (in the `p_film_count` parameter) a count that indicates the number of rows in that table.

Sample Usage

```
mysql> CALL film_in_stock(1,1,@count);
+--------------+
| inventory_id |
+--------------+
|            1 |
|            2 |
|            3 |
|            4 |
+--------------+
4 rows in set (0.01 sec)

Query OK, 1 row affected (0.01 sec)

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

5.3.2 The film_not_in_stock Stored Procedure

Description

The `film_not_in_stock` stored procedure determines 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 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 procedure produces a table of inventory ID numbers for the copies of the film not in stock, and returns (in the `p_film_count` parameter) a count that indicates the number of rows in that table.

Sample Usage

```
mysql> CALL film_not_in_stock(2,2,@count);
+--------------+
| inventory_id |
+--------------+
|            9 |
+--------------+
1 row in set (0.01 sec)

Query OK, 1 row affected (0.01 sec)

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

5.3.3 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 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`: An `OUT` parameter that returns a count of the customers who met the qualifications specified.

Return Values

This procedure produces a table of customers who met the qualifications specified. The table has the same structure as the `customer` table. The procedure also returns (in the `count_rewardees` parameter) a count that indicates the number of rows in that table.

Sample Usage

```
mysql> CALL rewards_report(7,20.00,@count);
...
| 598         | 1        | WADE        | DELVALLE     | WADE.DELVALLE@sakilacustomer.org         | 604
| 599         | 2        | AUSTIN      | CINTRON      | AUSTIN.CINTRON@sakilacustomer.org        | 605
...

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)
```

5.4 Stored Functions

The following sections describe the stored functions included with the Sakila sample database.

5.4.1 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, and so forth after this date are not counted.

Return Values

This function 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)
```

5.4.2 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 function 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 (0.00 sec)

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

5.4.3 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` to indicate whether the item specified is in stock.

Sample Usage

```
mysql> SELECT inventory_in_stock(9);
+-----------------------+
| inventory_in_stock(9) |
+-----------------------+
|                     0 |
+-----------------------+
1 row in set (0.00 sec)

mysql> SELECT inventory_in_stock(8);
```

```
+-----------------------+
| inventory_in_stock(8) |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

5.5 Triggers

The following sections describe the triggers in the Sakila sample database.

5.5.1 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.

5.5.2 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.

5.5.3 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.

5.5.4 The ins_film Trigger

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

5.5.5 The upd_film Trigger

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

5.5.6 The del_film Trigger

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

6 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
- Return a DVD
- Find Overdue DVDs

Rent a DVD

To rent a DVD, first confirm that the given inventory item is in stock, and then insert a row into the `rental` table. After the `rental` table is created, insert a row into the `payment` table. Depending on business rules, you may also need to check whether 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 (0.01 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> SET @rentID = LAST_INSERT_ID(),
                  @balance = get_customer_balance(3, NOW());
Query OK, 0 rows affected (0.14 sec)

mysql> SELECT @rentID, @balance;
+---------+----------+
| @rentID | @balance |
+---------+----------+
|   16050 |     4.99 |
+---------+----------+
1 row in set (0.00 sec)

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

Return a DVD

To return a DVD, update the `rental` table and set the return date. To do this, first identify the `rental_id` to update based on the `inventory_id` of the item being returned. Depending on the situation, it may be necessary 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
           INTO @rentID;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT @rentID;
+---------+
| @rentID |
+---------+
|   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 (0.13 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, 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 produce 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()
           ORDER BY title
           LIMIT 5;
+----------------+--------------+------------------+
| customer       | phone        | title            |
+----------------+--------------+------------------+
| OLVERA, DWAYNE | 62127829280  | ACADEMY DINOSAUR |
| HUEY, BRANDON  | 99883471275  | ACE GOLDFINGER   |
| OWENS, CARMEN  | 272234298332 | AFFAIR PREJUDICE |
| HANNON, SETH   | 864392582257 | AFRICAN EGG      |
| COLE, TRACY    | 371490777743 | ALI FOREVER      |
+----------------+--------------+------------------+
5 rows in set (0.10 sec)
```

7 Known Issues

The design of the Sakila sample database assumes that a staff member of a given store rents inventory items to customers only from that store, not from other stores. This assumption is manifest in that the `rental`, `inventory`, `staff`, and `store` tables have relationships that form a loop. A customer can have only a single store, but a staff member is not similarly constrained. Were a staff member to rent items from other stores, data in the `rental` table could become inconsistent.

The solution to this issue is left to the reader. Here are some possible approaches:

- Add a `store_id` column to the `rental` table and have foreign keys in the table also reference that column to ensure that not only `customer_id` and `inventory_id` but also `staff_id` in the `inventory` table have the same store.
- Add `INSERT` and `UPDATE` triggers on the `rental` table.

8 Acknowledgments

The following individuals and organizations contributed to the initial development of the Sakila sample database. This historical list is no longer updated.

- `Roland Bouman`: provided valuable feedback throughout the development process, contributed sample views and stored procedures.
- `Ronald Bradford`: developed the first sample application for use with the Sakila sample database.
- `Dave Jaffe`: provided schema used in Dell whitepaper and secured permission to use parts thereof in the Sakila sample database.
- `Giuseppe Maxia`: provided valuable feedback throughout the development process, populated some of the sample data, provided some of the sample views and triggers.

  For v1.0, he combined sakila and sakila-spatial by adding MySQL version specific comments within the SQL files.
- `Jay Pipes`: provided some of the sample stored procedures.
- `Zak Greant`: provided advice and feedback on licensing.

In addition to the individuals mentioned previously, various other individuals at MySQL and in the MySQL community have provided feedback during the course of development.

9 License for the Sakila Sample Database

For more information, please Contact http://www.mysql.com/about/contact/.

10 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.

11 Sakila Change History

This section describes changes made in each version of the Sakila sample database.

Version 1.5

- Fixed MySQL Bug #112552: Accented characters were missing from the `address` fields.

Version 1.4

- Fixed MySQL Bug #112131: Made the film_text.film_id field unsigned to match the other film_id definitions.

- Films without an actor were not returned by the film_list and nicer_but_slower_film_list views.

Version 1.3

- Fixed MySQL Bug #106951: Accented characters were missing from the `city` and `country` fields; their values were updated using the `world` database. In addition, the acute accent character itself was also missing.

- Fixed MySQL Bug #107158: Removed five rows in the payment table that had a null rental_id value.

Version 1.2

- Database objects now use `utf8mb4` rather than `utf8`. This change caused a `Specified key was too long; max key length is 767 bytes` error in MySQL 5.6 for the `film.title` column, which was declared as `VARCHAR(255)`. The actual maximum title length is 27 characters, so the column was redeclared as `VARCHAR(128)` to avoid exceeding the maximum key length.

- `sakila-schema.sql` and `sakila-data.sql` include a `SET NAMES utf8mb4` statement.

- `sakila-data.sql` was converted from DOS (CRLF) line endings to Unix (LF) line endings.

- The `address.location` column is a `GEOMETRY` column that has a `SPATIAL` index. As of MySQL 8.0.3, `SPATIAL` indexes are ignored unless the index spatial column has an `SRID` attribute. The `location` column was changed to include an `SRID 0` attribute for MySQL 8.0.3 and higher.

- The `staff.password` column was declared as `VARCHAR(40) BINARY`. This is use of `BINARY` as shorthand in a character column declaration for specifying a `_bin` collation, which is deprecated as of MySQL 8.0.17. The column was redeclared as what `BINARY` is shorthand for, that is, `VARCHAR(40) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin`.

- In the `rewards_report()` stored procedure, the `min_dollar_amount_purchased` parameter was declared as `DECIMAL(10,2) UNSIGNED`. Use of `UNSIGNED` with `DECIMAL` is deprecated as of MySQL 8.0.17. The parameter was redeclared without `UNSIGNED`.

- The `film_in_stock()` and `film_not_in_stock()` stored procedures used the `FOUND_ROWS()` function, which is deprecated as of MySQL 8.0.17. In each procedure, the `FOUND_ROWS()` query was replaced by a query that uses `COUNT(*)` with the same `FROM` and `WHERE` clauses as its associated query. This is more expensive than using `FOUND_ROWS()` but produces the same result.

- The `film_text` table uses `MyISAM` rather than `InnoDB` prior to MySQL 5.6.10 to avoid table-creation failure in older versions. (However, we still recommend upgrading to MySQL 5.6.10 or higher.)

- The `sakila.mwb` file for MySQL Workbench was updated per the preceding changes.

Version 1.1

- Removed all `MyISAM` references. The `film_text` table, and its `FULLTEXT` definition, now uses `InnoDB`. If you use an older MySQL server version (5.6.10 and lower), we recommend upgrading MySQL. If you cannot upgrade, change the `ENGINE` value for the `film_text` table to `MyISAM` in the `sakila-schema.sql` SQL file.

Version 1.0

- Merged `sakila-schema.sql` and `sakila-spatial-schema.sql` into a single file by using MySQL version-specific comments.

  Spatial data, such as `address.location`, is inserted into the sakila database as of MySQL server 5.7.5 (when spatial indexing support was added to `InnoDB`). Also, `InnoDB` full-text search is used as of MySQL server 5.6.10, when before `MyISAM` was used.

Version 0.9

- Added an additional copy of the Sakila example database that includes spatial data with the geometry data type. This is available as a separate download, and requires MySQL server 5.7.5 or later. To use this database, load the `sakila-spatial-schema.sql` file rather than the `sakila-schema.sql` file.
- Modified `GROUP BY` clause of the `nicer_but_slower_film_list` and `film_list` view definitions to be compatible with `ONLY_FULL_GROUP_BY` SQL mode, which is enabled by default as of MySQL 5.7.5.

Version 0.8

- Corrected `upd_film` trigger definition to include changes to `film_id` values.
- 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 stored procedures and functions where appropriate to support loading on MySQL 5.1.
- Fixed date-range issue in the `rewards_report` procedure (thanks Goplat).

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.

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 (multiple-row `INSERT` statements, transactions). (Thanks Giuseppe)
- Fixed error in `payment` table loading script that caused infinitely increasing payment amounts.

Version 0.5

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

Version 0.4

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

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)`.
- Added `payment.rental_id` column (an `INT NULL` 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.

Version 0.2

- All tables have a `last_update` `TIMESTAMP` column 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 subtitled. `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`.
- `film_list` view updated to handle new `film_category` table.
- `nicer_but_slower_film_list` view updated to handle new `film_category` table.