# **The Sakila Database**

**Sakila** Database was originally created by MySQL and has been open sourced under the terms of the BSD License.

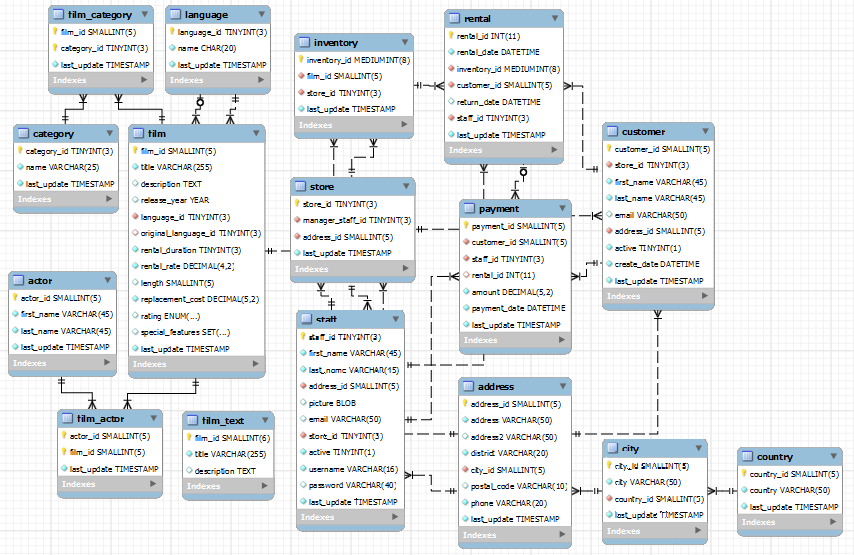
The Sakila database is a nicely normalised schema modelling a DVD rental store, featuring things like films, actors, film-actor relationships, and a central inventory table that connects films, stores, and rentals.

# **Tables**

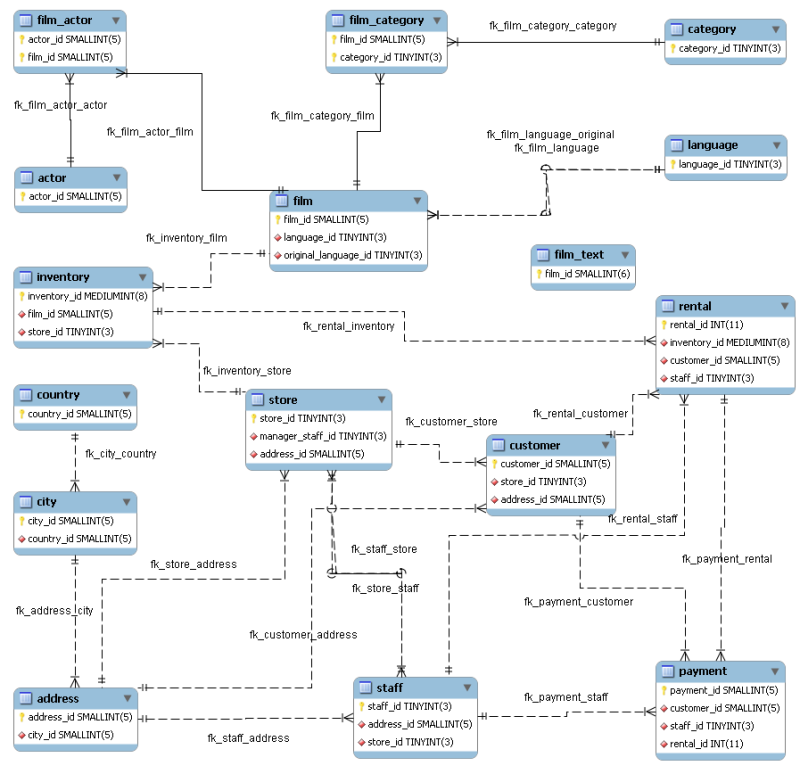
The following sections describe the tables that make up the Sakila sample database, in alphabetic order.

* **actor:** The actor table lists information for all actors.The actor table is joined to the film table by means of the film\_actor table.
* **address:** 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.
* **category:** 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.
* **city:** 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.
* **country:** The country table contains a list of countries. The country table is referred to by a foreign key in the city table.
* **customer:** 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.
* **film:** 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.
* **film\_actor:** The film\_actor table is used to support a many-to-many relationship between films and actors. For eachactor 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.
* **film\_category:** 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.
* **film\_text:** 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”).
* **inventory:** 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.
* **language:** 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.
* **payment:** 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.
* **rental:** 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.
* **Staff:** 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.
* **store:** The store table lists all stores in the system. All inventory is assigned to specific stores, and staff andcustomers are assigned a “home store”.The store table refers to the staff and address tables using foreign keys and is referred to by thestaff, customer, and inventory tables.

# **Schema**



# **EER Diagram**



# **Test Scenarios For Schema testing**

* Check table presence in Database Schema
* Check Table name conventions
* Check number of columns in a table
* Check column names in a table
* Check Data types of columns in table
* Check size of the columns in table
* Check column keys in a table
* Check whether the references for foreign keys are valid.
* Check whether the data type of the primary key and the corresponding foreign keys are the same in the two tables.

# **Test Scenarios for Stored Procedure**

* Check Stored Procedure exist in database
* Check stored proedure with valid input data
* Check stored procedure handle exceptions when you pass invalid input data.
* Check stored procedure display results as expected.
* Check stored procedure inserting data in proper tables.
* Check stored procedure updating data in proper tables.
* Check stored procedure deleting data from proper tables.
* Check calling stored procedure from another stored procedure.

## **Stored Procedures**

**Stored Procedure 1:**

|  |  |
| --- | --- |
| **Stored procedure Name** | SelectAllActor |
| **Description** | Select all records from the “actor” table |
| **Input Parameters** | NA |
| **Output Parameters** | NA |

**Stored Procedure 2:**

|  |  |
| --- | --- |
| **Stored procedure Name** | AllCustomerbyAddress |
| **Description** | Select customers from the “customer” table by a particular address id |
| **Input Parameters** | Address\_id |
| **Output Parameters** | NA |

**Stored Procedure 3:**

|  |  |
| --- | --- |
| **Stored procedure Name** | AllCustomerby\_fname\_address |
| **Description** | Select customers from the “customer” table by customers’s firstName and address id |
| **Input Parameters** | First\_Name, Address\_id |
| **Output Parameters** | NA |

**Stored Procedure 4:**

|  |  |
| --- | --- |
| **Stored procedure Name** | Film\_in\_stock |
| **Description** | Find the films that are in stock from “inventory” table using particular film id and stock id |
| **Input Parameters** | p\_film\_id, p\_store\_id |
| **Output Parameters** | NA |

**Stored Procedure 5:**

|  |  |
| --- | --- |
| **Stored procedure Name** | Payment\_history\_by\_customer\_id |
| **Description** | Find payment history of an individuals by his/her customer id and returns the output as low, medium and high |
| **Input Parameters** | Customer\_id |
| **Output Parameters** | Low, Medium, High |

# **Test Scenarios for Stored Function**

* Check Stored Function exist in database
* Check Stored Function with valid input data
* Check Stored Function handle exceptions when you pass invalid input data.
* Check Stored Function returns results as expected.
* Check Stored Function not used insert/update/delete operations.
* Check Stored Function used only select statements
* Check calling Stored Function from stored procedure.

## **Stored Functions**

**Stored Functions 1:**

|  |  |
| --- | --- |
| **Stored function Name** | get\_customer\_balance |
| **Description** | Returns the amount owing on the customer's account. |
| **Input Parameters** | p\_customer\_id, p\_effective\_date |
| **Output Parameters** | Customer balance |

**Stored Functions 2:**

|  |  |
| --- | --- |
| **Stored function Name** | inventory\_held\_by\_customer |
| **Description** | Returns the customer\_id of the customer who is currently renting the item, or NULL if the item is in stock |
| **Input Parameters** | p\_inventory\_id |
| **Output Parameters** | Inventory held by customer |

**Stored Functions 3:**

|  |  |
| --- | --- |
| **Stored function Name** | inventory\_in\_stock |
| **Description** | Returns TRUE or FALSE to indicate whether the item specified is in stock. |
| **Input Parameters** | p\_inventory\_id |
| **Output Parameters** | Inventory in stock |

# **Triggers Testing**

**Types of triggers**

There are Six types of actions or events in the form of triggers:

* Before Insert: It is activated before the insertion of data into the table.
* After Insert: It is activated after the insertion of data into the table.
* Before Update: It is activated before the update of data in the table.
* After Update: It is activated after the update of the data in the table.
* Before Delete: It is activated before the data is removed from the table.
* After Delete: It is activated after the deletion of data from the table.

**Trigger-1:**

|  |  |
| --- | --- |
| **Trigger Name** | customer\_create\_date |
| **Trigger type** | After Insert |
| **Description** | The customer\_create\_date trigger sets the create\_date column of the customer table to the current time and date as rows are inserted. |
| **Tables** | Customer |

**Trigger-2:**

|  |  |
| --- | --- |
| **Trigger Name** | ins\_film |
| **Trigger type** | Insert |
| **Description** | The ins\_film trigger duplicates all INSERT operations on the film table to the film\_text table |
| **Tables** | film |

**Trigger-3:**

|  |  |
| --- | --- |
| **Trigger Name** | upd\_film |
| **Trigger type** | Update |
| **Description** | The upd\_film trigger duplicates all UPDATE operations on the film table to the film\_text table. |
| **Tables** | film |

**Trigger-4:**

|  |  |
| --- | --- |
| **Trigger Name** | del\_film |
| **Trigger type** | Delete |
| **Description** | The del\_film trigger duplicates all DELETE operations on the film table to the film\_text table. |
| **Tables** | film |

**Trigger-5:**

|  |  |
| --- | --- |
| **Trigger Name** | payment\_date |
| **Trigger type** | insert |
| **Description** | The payment\_date trigger sets the payment\_date column of the payment table to the current time and date as rows are inserted. |
| **Tables** | payment |