2017

Database Design for ONLINE COLLEGE BOOK STORE

CS 6360.003 FINAL PROJECT

GEETIKA JAIN, PRABHAT KUMAR JHA, NEELABH PANDEY

Table of Contents

Requirements

Services offered by Online Book store

This is a book purchasing online store where customers can go online, make their user profiles and make multiple purchases of the books. The services offered by the store are as follows.

- Services include –
    - Creating Customer profiles to the Online stores
    - Book Browsing with Image
    - Adding of books to customer specific Cart
    - Keeping a track of Book availability
    - Placing Orders
    - Reflecting the count of books in Available Books
    - Deleting the Customer Profile
    - Storing Order History of every customer
    -

Customer Structure

Each Customer has a unique Customer ID which will be generated when they create their profiles. While creating their profiles as customers they will be asked for their Email IDs and will be prompted to choose a password. Email ID of the customer should be unique to each customer. Other information required to create a Customer will be their Phone numbers, First names, last names and their addresses.

Books

Each customer will browse for books of their choice. Books will be uniquely identified by their Book IDs. But there may be many books of the same ID. Thus, we keep a count of the books of same IDs. Each book will be associated with a Category to which it belongs. A book can be browsed by its Title. All the information regarding a book are available like its price, Publisher, Published date, version. If a customer purchases a book, it will reflect in the count of the book.

Author

A book can be written by one author or many authors. One author may have written multiple books and vice versa. An author is identified uniquely by an Author ID. Also, information about author like Name, Email ID, and address of the author is available.

Category

Each book is allotted to a different category. For example: Physics, Science, Math etc. A category is distinguished by a Category ID and every book is allotted to one category. A category has characteristic like Name and popularity. Also, we keep a count of all the books which belong to a category.

Cart

A cart is associated with every customer directly. Each customer will have exactly one cart and can place purchase orders via it. Multiple books can be added to a cart by the customer and it will also show if the books in the cart are available at the time of adding the book to the cart. The book and price of the book will get added. Also, the time of adding the book to the cart is noted.

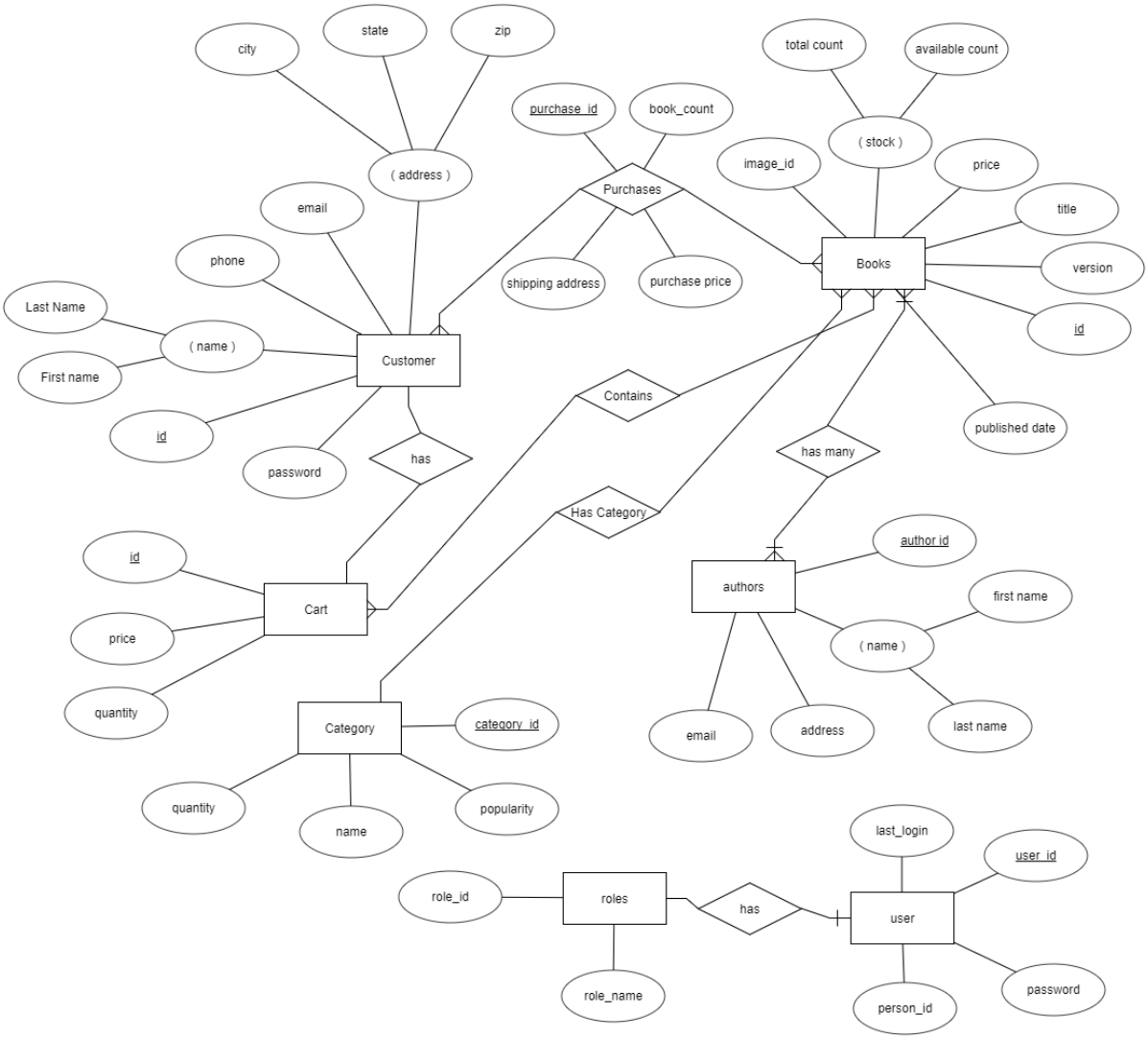
Purchase

A Customer can place the order with the items they have in their cart currently. The books in the order will be checked for availability. If the Books are available, the total price will be calculated. Each purchase is unique by a Purchase ID and the Customer buying the items. Time of purchase is recorded. A customer can choose any shipping address for the delivery of the books they ordered.

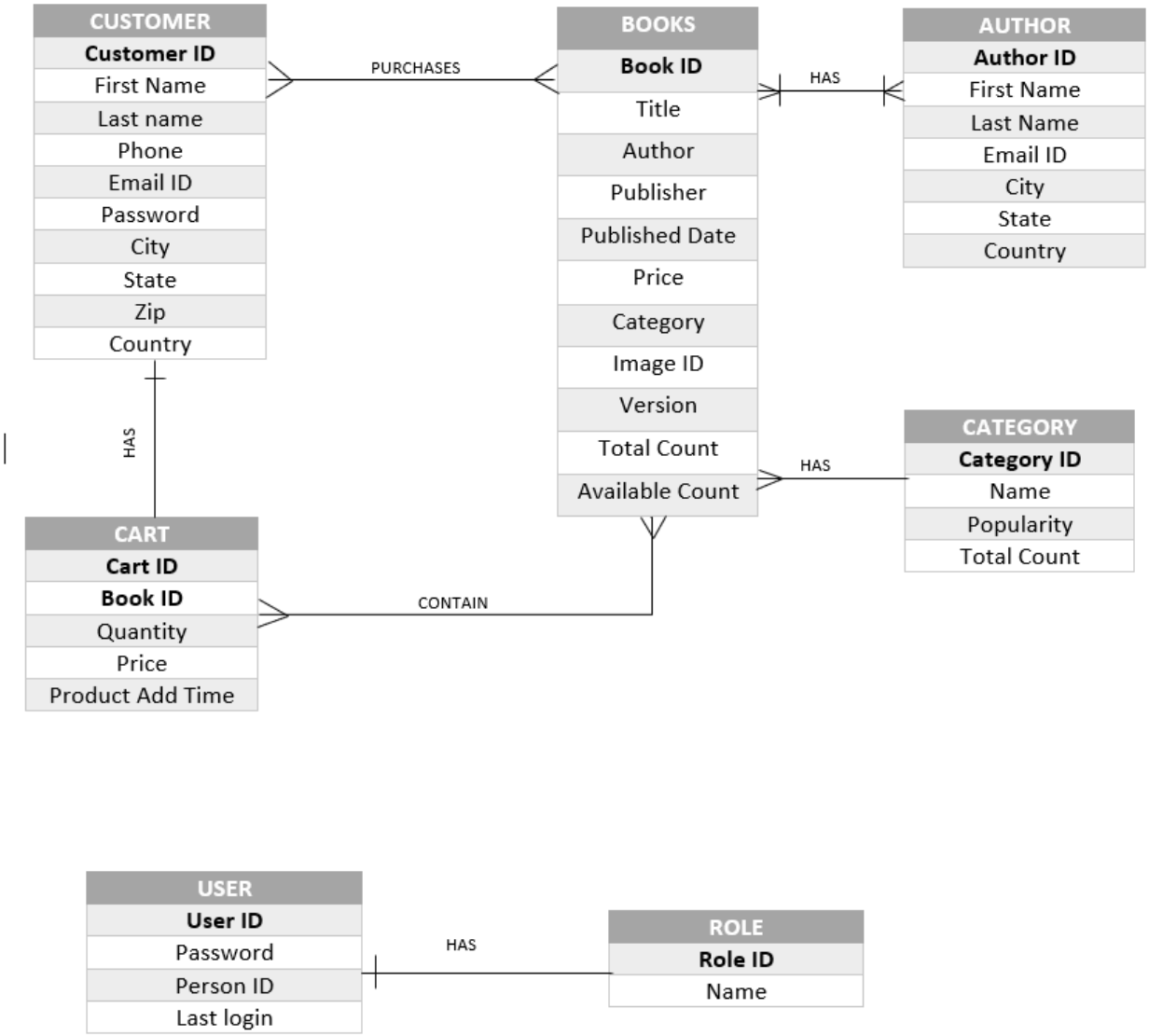
Users

The Users are different roles a profile may have in the system. Each role has different responsibility or roles. A user may be a Customer, or a seller, or Admin. All the Created accounts for a customer is by default assigned to a particular User type.

ER Diagram:


city
state
zip
(address)
email
phone
Last Name
(name)
First name
Customer
id
password
has
Cart
id
price
quantity
Purchases
purchase_id
book_count
shipping address
purchase price
total count
available count
(stock)
image_id
price
title
version
id
Books
Contains
Has Category
has many
published date
authors
author id
first name
(name)
last name
email
address
Category
category_id
quantity
name
popularity
role_id
roles
has
role_name
last_login
user_id
user
password
person_id

Modeling of Requirements as ER-Diagram:



The requirements can be summarized/ derived from ERD as –

1. A Customer can have one cart and a cart can belong to one customer (1:1).
2. A Book can have many authors and an author can write many books (M:N).
3. A book belongs to one category, But a Category can have many books listed under it. (1:M)
4. A book can be added in many Carts and a cart can have many books (M:N)
5. A Customer can purchase Many books and a book can be purchased by many customers(M:N).
6. A User has to have at least one role but a role might not have users (0:1)

Mapping of ERD in Relational Schema

1. CUSTOMER

Customer ID	First name	Last Name	Phone	Email	Password	City	State	Country	Zip

- Primary Key : Customer ID
- Foreign Keys : None

2. BOOKS

Book ID	Title	Version	Publisher	Published	Price	Total count	Available count	Image ID	Category	Discontinued

- Primary Key : BOOK_ID
- Foreign Keys : FOREIGN KEY (Category) REFERENCES CATEGORY (CATEGORY_ID)

3. AUTHOR

Author ID	First name	Last name	Email	City	State	Country

- Primary Key : AUTHOR_ID
- Foreign Keys : None

4. CART

Cart ID	Book ID	Quantity	Price	Product add time

- Primary Key : CART ID, BOOK ID
- Foreign Keys : FOREIGN KEY (BOOK_ID) REFERENCES BOOKS (BOOK_ID)

  FOREIGN KEY (CART_ID) REFERENCES CUSTOMER (CUSTOMER_ID)

5. CATEGORY

Category ID	Name	Popularity	Total count

- Primary Key : CATEGORY ID
- Foreign Keys : None

6. ROLE

Role ID	Name

- Primary Key : ROLE ID
- Foreign Keys : None

7. USER

User ID	Password	Person ID	Last Login

- Primary Key : USER ID
- Foreign Keys : FOREIGN KEY (DNO) REFERENCES FIREDEPARTMENT(DNUMBER)

8. BOOKS_AUTHORS

Book ID	Author ID

- Primary Key : BOOK ID, AUTHOR ID
- Foreign Keys : FOREIGN KEY (BOOK ID) REFERENCES BOOKS (BOOK ID), FOREIGN KEY (AUTHOR ID) REFERENCES AUTHOR (AUTHOR ID)

9. PURCHASE

Purchase ID	Customer ID	Total Price	Total items	Shipping Address	Purchase Date

- Primary Key : PURCHASE ID, CUSTOMER ID
- Foreign Keys : FOREIGN KEY (CUSTOMER ID) REFERENCES CUSTOMER (CUSTOMER ID)

10. PURCHASE_DETAILS

Purchase ID	Book ID	Price	Quantity

- Primary Key : PURCHASE ID, BOOK ID
- Foreign Keys : FOREIGN KEY (PURCHASE ID) REFERENCES PURCHASE (PURCHASE ID), FOREIGN KEY (BOOK ID) REFERENCES BOOKS (BOOK ID)

SQL Statements to create Relations in DB and Add Constraints

```
CREATE TABLE `remedscx_`.`bs_books` (
	`book_id` INT NOT NULL,
	`title` VARCHAR(45) NULL,
	`price` FLOAT NULL,
	`version` VARCHAR(45) NULL,
	`publisher` VARCHAR(45) NULL,
	`published_date` VARCHAR(45) NULL,
	`image_id` VARCHAR(45) NULL,
	`total_count` VARCHAR(45) NULL,
	`available_count` VARCHAR(45) NULL,
	`category` INT DEFAULT NULL,
	`discontinued` INT DEFAULT 0,
	PRIMARY KEY (`book_id`)
);

ALTER TABLE `remedscx_`.`bs_books`
	ADD CONSTRAINT `bs_books_fk1`
	FOREIGN KEY (`category`)
	REFERENCES `remedscx_`.`bs_category` (`category_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;

CREATE TABLE `remedscx_`.`bs_customer` (
	`customer_id` INT NOT NULL,
	`email` VARCHAR(45) NULL,
	`phone` VARCHAR(45) NULL,
	`fname` VARCHAR(45) NULL,
	`lname` VARCHAR(45) NULL,
	`city` VARCHAR(45) NULL,
	`state` VARCHAR(45) NULL,
	`zip` VARCHAR(45) NULL,
	`country` VARCHAR(45) NULL,
	`cart_id` INT DEFAULT NULL,
	PRIMARY KEY (`customer_id`)
);

CREATE TABLE `remedscx_`.`bs_authors` (
	`author_id` INT NOT NULL,
	`fname` VARCHAR(45) NULL,
	`lname` VARCHAR(45) NULL,
	`email` VARCHAR(45) NULL,
	`city` VARCHAR(45) NULL,
	`state` VARCHAR(45) NULL,
	`country` VARCHAR(45) NULL,
	PRIMARY KEY (`author_id`)
);
```

```
CREATE TABLE `remedscx_`.`bs_book_authors` (
	`book_id` INT NOT NULL,
	`author_id` INT NOT NULL,
	PRIMARY KEY (`book_id`, `author_id`)
);


ALTER TABLE `remedscx_`.`bs_book_authors`
	ADD CONSTRAINT `bs_book_author_fk1`
	FOREIGN KEY (`author_id`)
	REFERENCES `remedscx_`.`bs_authors`(`author_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;


ALTER TABLE `remedscx_`.`bs_book_authors`
	ADD CONSTRAINT `bs_book_author_fk2`
	FOREIGN KEY (`book_id`)
	REFERENCES `remedscx_`.`bs_books`(`book_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;


CREATE TABLE `bs_category` (
	`category_id` int(11) NOT NULL,
	`category_name` varchar(45) DEFAULT NULL,
	`category_popularity` varchar(45) DEFAULT NULL,
	`quantity` int(11) DEFAULT NULL,
	PRIMARY KEY (`category_id`)
) ;


CREATE TABLE `remedscx_`.`bs_cart` (
	`cart_id` INT NOT NULL,
	`book_id` INT NOT NULL,
	`quantity` INT NULL,
	`price` VARCHAR(45) NULL,
	`product_add_time` TIMESTAMP NULL,
	PRIMARY KEY (`cart_id`, `book_id`)
);


ALTER TABLE `remedscx_`.`bs_cart`
	ADD CONSTRAINT `bs_cart_fk1`
	FOREIGN KEY (`book_id`)
	REFERENCES `remedscx_`.`bs_books`(`book_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;
```

```
CREATE TABLE `remedscx_`.`bs_purchase` (
	`purchase_id` INT NOT NULL,
	`customer_id` INT NULL,
	`shipping_address` VARCHAR(45) NULL,
	`total_cost` FLOAT NULL,
	`total_items` INT NULL,
	`purchase_date` TIMESTAMP NULL,
	PRIMARY KEY (`purchase_id`)
);


ALTER TABLE `remedscx_`.`bs_purchase`
	ADD CONSTRAINT `bs_purchase_fk1`
	FOREIGN KEY (`customer_id`)
	REFERENCES `remedscx_`.`bs_customer`(`customer_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;


CREATE TABLE `remedscx_`.`bs_purchase_details` (
	`purchase_id` INT NOT NULL,
	`book_id` INT NOT NULL,
	`quantity` INT NULL,
	`price` FLOAT NULL,
	PRIMARY KEY (`purchase_id`, `book_id`)
);


ALTER TABLE `remedscx_`.`bs_purchase_details`
	ADD CONSTRAINT `bs_purchase_details_fk1`
	FOREIGN KEY (`purchase_id`)
	REFERENCES `remedscx_`.`bs_purchase`(`purchase_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;


ALTER TABLE `remedscx_`.`bs_purchase_details`
	ADD CONSTRAINT `bs_purchase_details_fk2`
	FOREIGN KEY (`book_id`)
	REFERENCES `remedscx_`.`bs_books`(`book_id`)
	ON DELETE RESTRICT
	ON UPDATE RESTRICT;


CREATE TABLE `bs_user_roles` (
	`role_id` int(11) NOT NULL,
	`role_name` varchar(45) DEFAULT NULL,
	PRIMARY KEY (`role_id`)
);
```

```
CREATE TABLE `remedscx_`.`bs_users` (
	`user_id` VARCHAR(10) NOT NULL,
	`password` VARCHAR(45) NULL,
	`role_id` INT NULL,
	`person_id` INT NULL,
	`last_login` TIMESTAMP NULL,
	`deleted` INT DEFAULT 0,
	PRIMARY KEY (`user_id`)
);
```

Normalization of Relational Schema

Since a Book can have multiple authors, thus, the Books Relation was split up into Books and Author and is a third relation called Book_Authors acts as a bridge between these two relations.

Finally, the following Functional Dependencies exist in the Relational Schema –

1. CUSTOMER { CustomerID -> FirstName, LastName, Phone, Email, Password, City, State, Country, Zip }
2. BOOKS { BookID -> Title, Version, Publisher, Published, Price, TotalCount, AvailableCount, ImageID, Category }
3. AUTHOR { AuthorID -> FirstName, LastName, Email, City, State, Country }
4. CART { CartID, BookID -> Quantity, Price, ProductAddTime }
5. CATEGORY { CategoryID -> Name, Popularity, TotalCount }
6. ROLE { RoleID -> Name }
7. USER { UserID -> Password, PersonID, LastLogin }
8. PURCHASE { PurchaseID, CustomerID -> TotalPrice, TotalOtems, ShippingAddress, PurchaseDate }
9. PURCHASE_DETAILS { PurchaseID, BookID -> Price, Quantity }

The above Functional Dependencies cause the Schema to be in Third Normal Form.

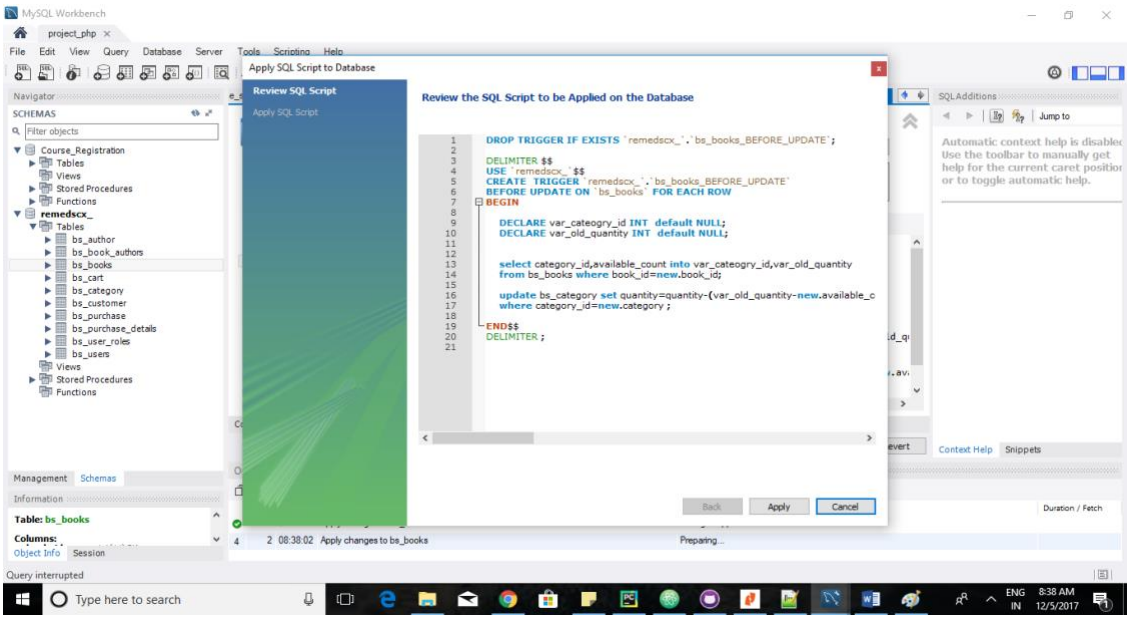
MySQL –> Triggers

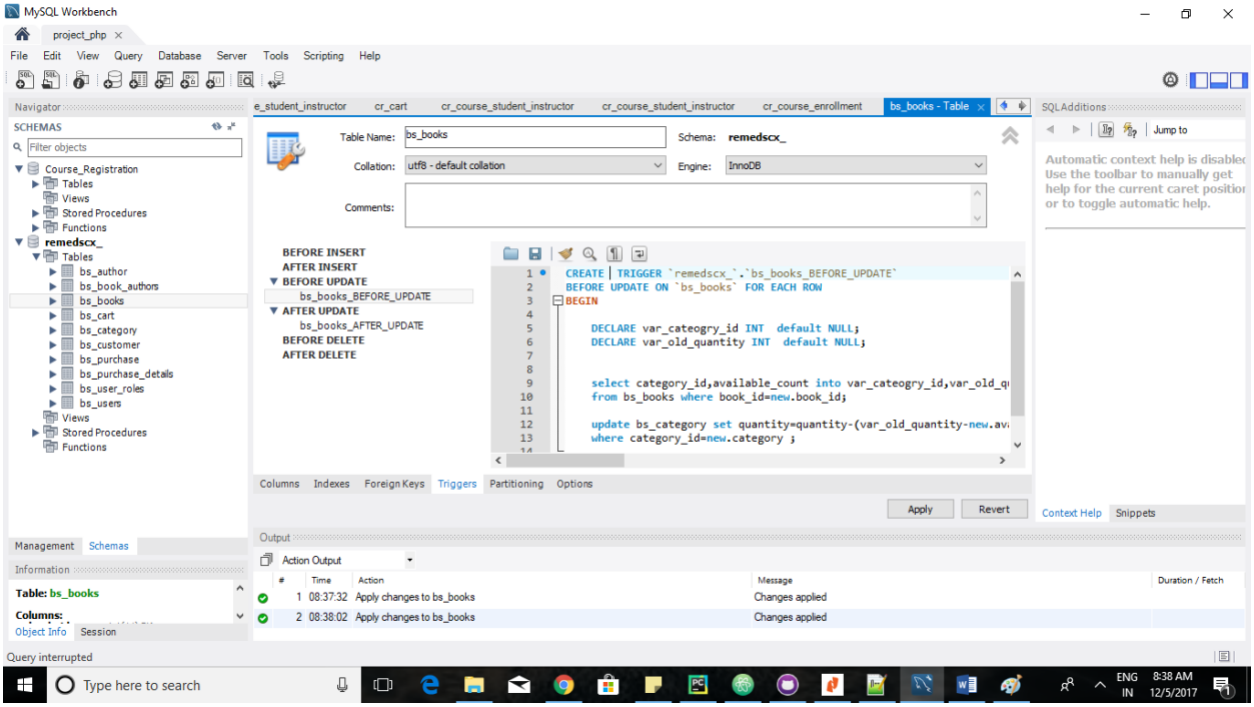
1. Manage Category wise count of Books:

We are keeping category wise count for every book. Every book has one category and in bs_category we are maintain total book count for one category. When ever any purchase has been made for any book, we retrieve the category from bs_books and substract the total count from bs_category.

CODE:

```
CREATE  TRIGGER `remedscx_`.`bs_books_BEFORE_UPDATE`
BEFORE UPDATE ON `bs_books` FOR EACH ROW
BEGIN
        DECLARE var_cateogry_id INT  default NULL;
        DECLARE var_old_quantity INT  default NULL;

        select category,available_count into var_cateogry_id,var_old_quantity
        from bs_books where book_id=new.book_id;

        update bs_category set quantity=quantity-(var_old_quantity-new.available_count)
        where category_id=new.category;
END
```



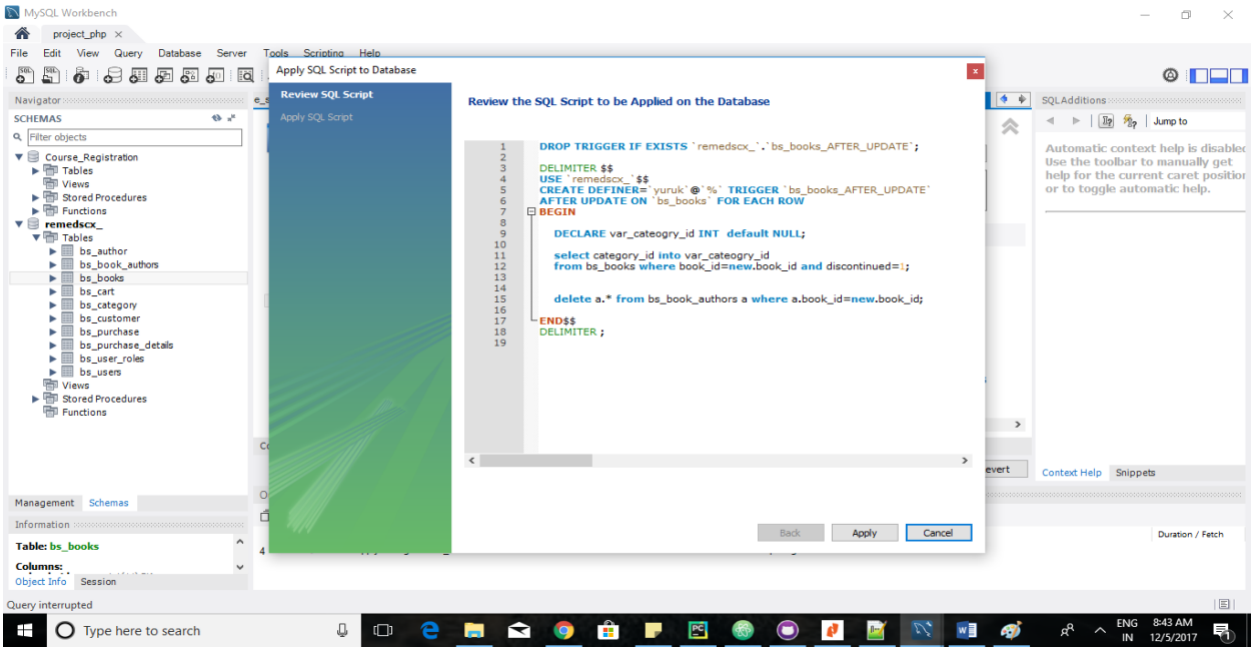


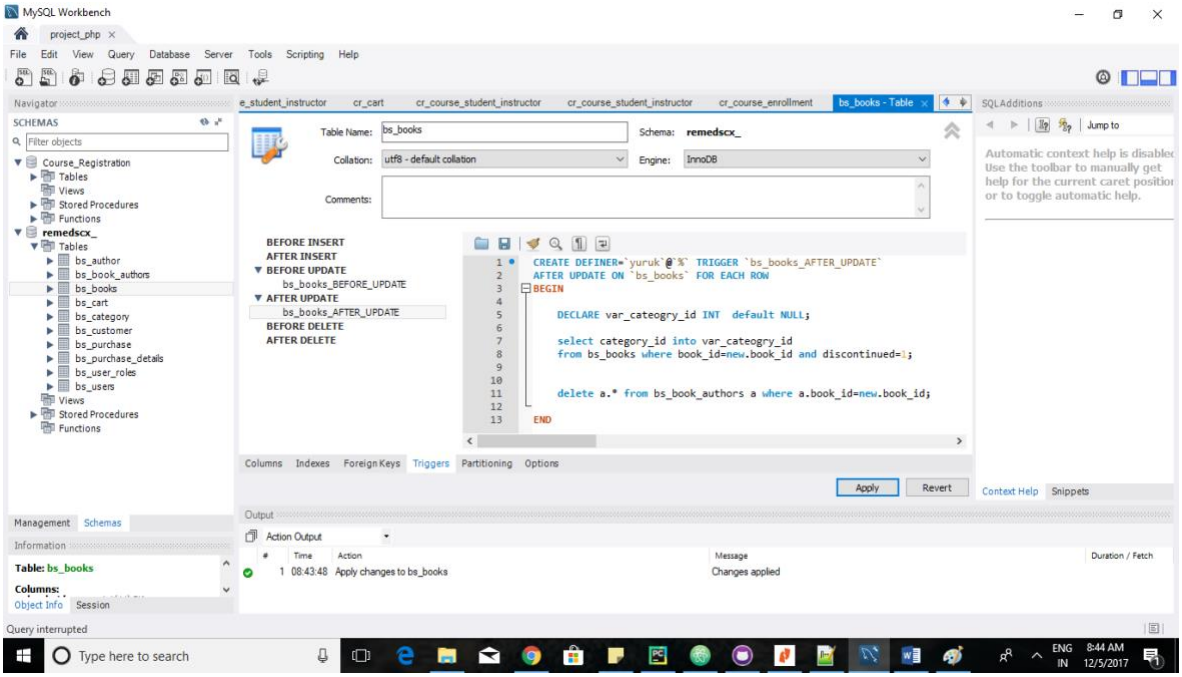
2. On Deletion of a Book:

Suppose admin has deleted one book from bs_books by updating flag discontinued=1. In this scenario, we will delete all mappings from bs_book_authors for that book. Because we are no longer providing this book. Since, authors can have many books thus, we are not deleting any authors.

CODE:

```
CREATE TRIGGER `bs_books_AFTER_UPDATE`
AFTER UPDATE ON `bs_books` FOR EACH ROW
BEGIN
        DECLARE var_cateogry_id INT  default NULL;

        select category_id into var_cateogry_id
        from bs_books where book_id=new.book_id and discontinued=1;

        delete a.* from bs_book_authors a where a.book_id=new.book_id;
END
```


MySQL Workbench
project_php
File Edit View Query Database Server Tools Scripting Help
Apply SQL Script to Database
Review SQL Script
Apply SQL Script
Review the SQL Script to be Applied on the Database
DROP TRIGGER IF EXISTS `remedscx_`.`bs_books_AFTER_UPDATE`;
DELIMITER $$
USE `remedscx_`$$
CREATE DEFINER=`yuruk`@`%` TRIGGER `bs_books_AFTER_UPDATE`
AFTER UPDATE ON `bs_books` FOR EACH ROW
BEGIN
DECLARE var_cateogry_id INT default NULL;
select category_id into var_cateogry_id
from bs_books where book_id=new.book_id and discontinued=1;
delete a.* from bs_book_authors a where a.book_id=new.book_id;
END$$
DELIMITER ;
Back
Apply
Cancel
SCHEMAS
Course_Registration
remedscx_
Tables
bs_author
bs_book_authors
bs_books
bs_cart
bs_category
bs_customer
bs_purchase
bs_purchase_details
bs_user_roles
bs_users
Views
Stored Procedures
Functions
Management
Schemas
Information
Table: bs_books
Columns:
Object Info
Session
SQLAdditions
Jump to
Context Help
Snippets
Duration / Fetch
Query interrupted
Type here to search
ENG
IN
8:43 AM
12/5/2017


MySQL Workbench
project_php
File Edit View Query Database Server Tools Scripting Help
Table Name: bs_books
Schema: remedscx_
Collation: utf8 - default collation
Engine: InnoDB
Comments:
BEFORE INSERT
AFTER INSERT
BEFORE UPDATE
bs_books_BEFORE_UPDATE
AFTER UPDATE
bs_books_AFTER_UPDATE
BEFORE DELETE
AFTER DELETE
CREATE DEFINER=`yuruk`@`%` TRIGGER `bs_books_AFTER_UPDATE`
AFTER UPDATE ON `bs_books` FOR EACH ROW
BEGIN
DECLARE var_cateogry_id INT default NULL;
select category_id into var_cateogry_id
from bs_books where book_id=new.book_id and discontinued=1;
delete a.* from bs_book_authors a where a.book_id=new.book_id;
END
Columns Indexes Foreign Keys Triggers Partitioning Options
Apply
Revert
Context Help
Snippets
Output
Action Output
Time
Action
Message
Duration / Fetch
1 08:43:48 Apply changes to bs_books
Changes applied
Query interrupted
Type here to search
ENG
IN
8:44 AM
12/5/2017

MySQL –> Procedures

1. Register a Customer

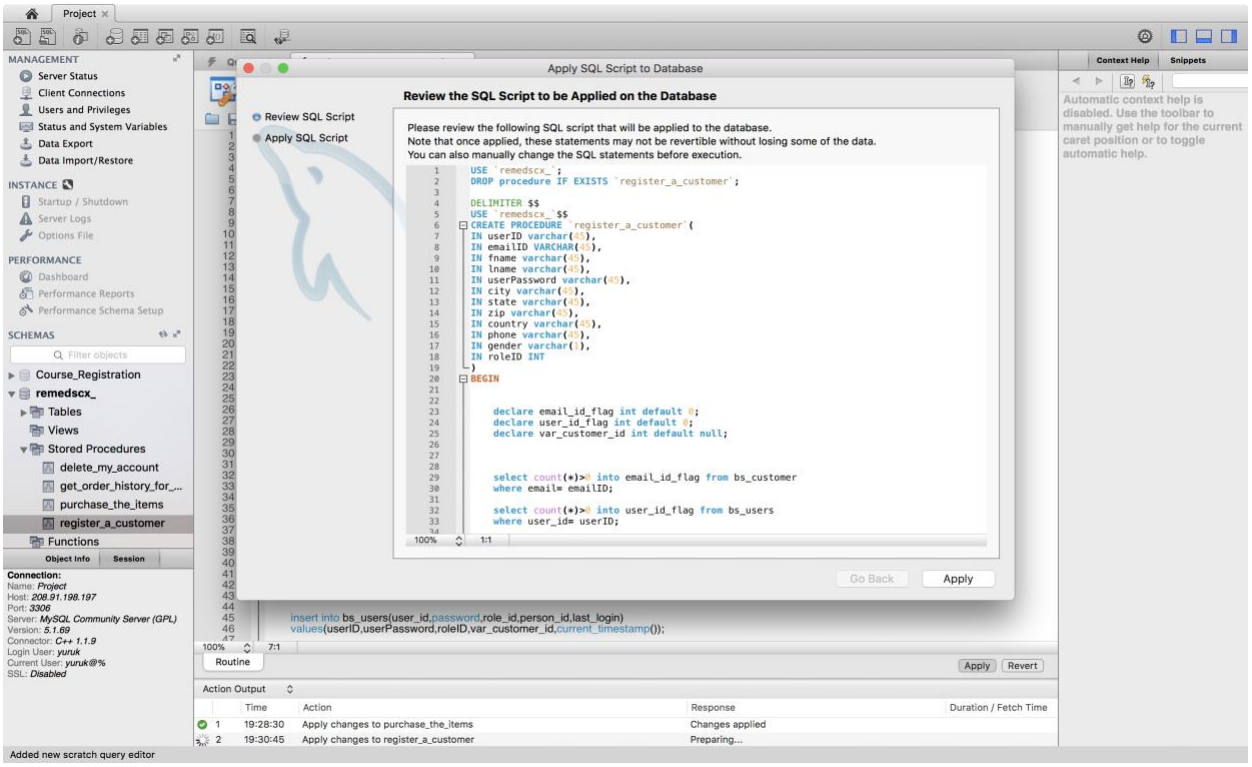
This procedure creates a customer profile by accepting the information required for creating a customer and generates a Customer ID for the Customer Profile. The arguments are mentioned below.

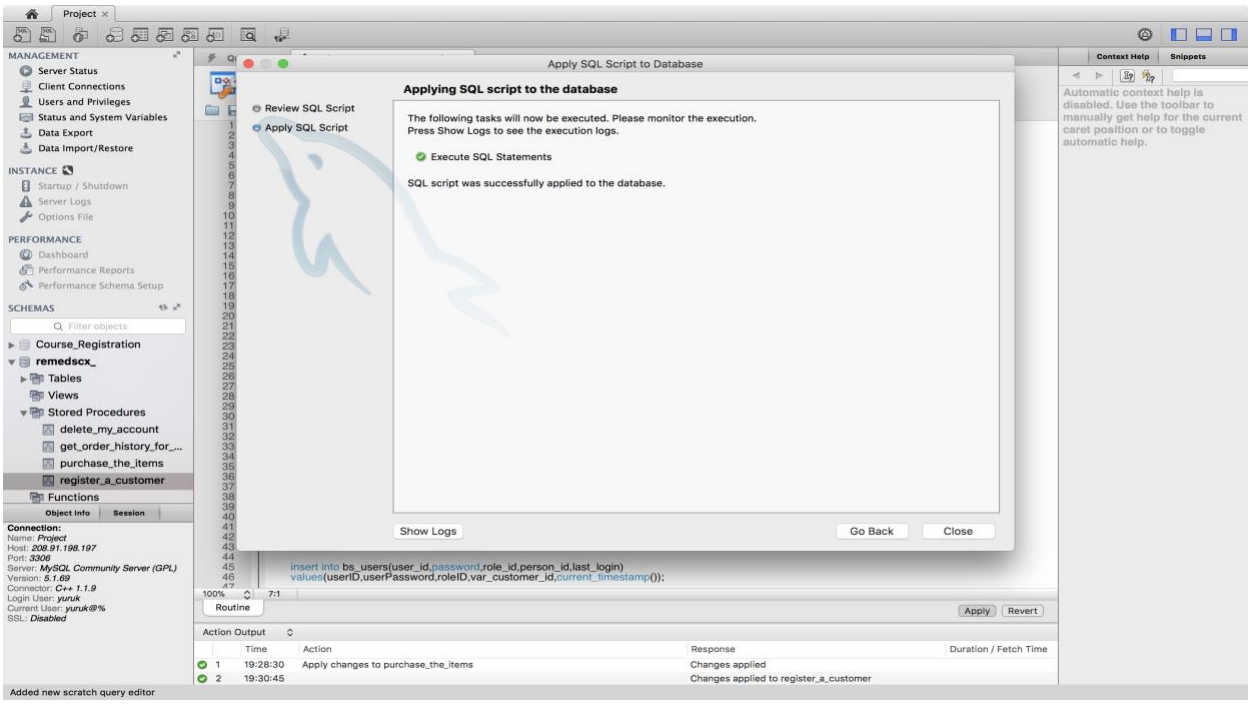
Args: (userID IN, emailID IN, fname IN, lname IN, userPassword IN, city IN, state IN, zip IN, country IN, phone IN, gender IN, roleID IN)

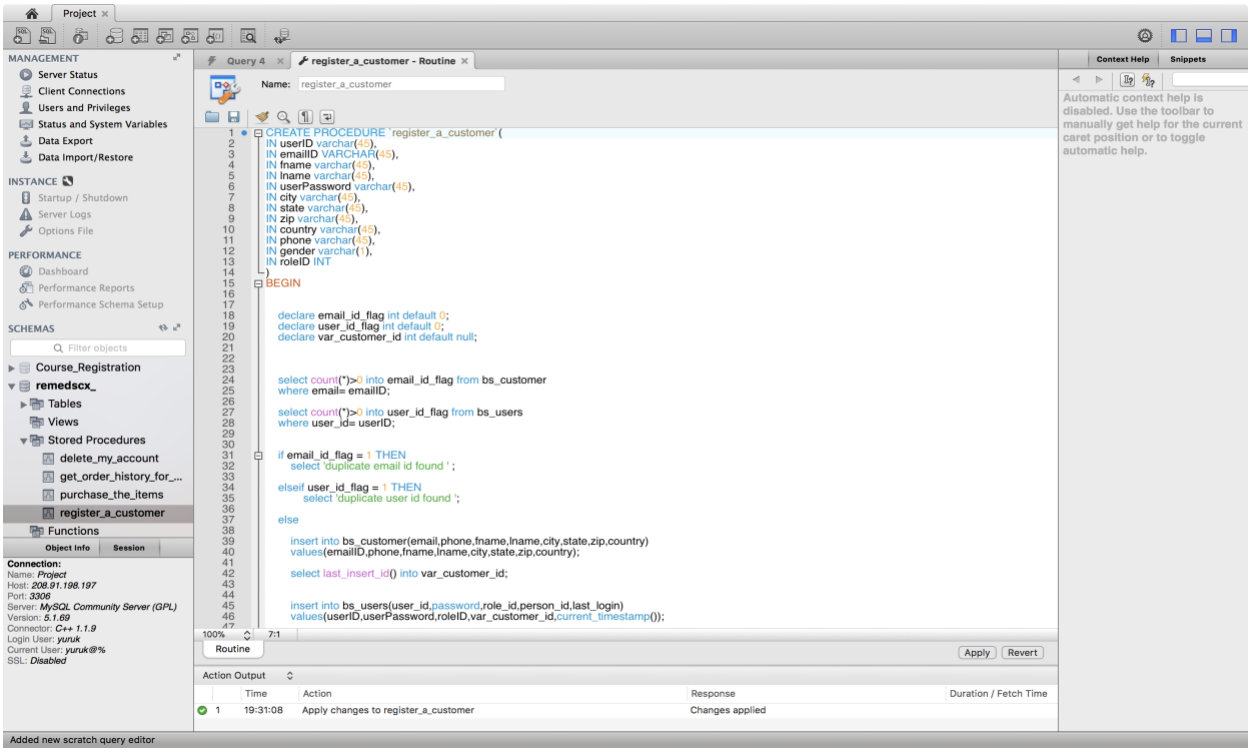
CODE:

```
CREATE PROCEDURE `register_a_customer`(
        IN userID varchar(45),
        IN emailID VARCHAR(45),
        IN fname varchar(45),
        IN lname varchar(45),
        IN userPassword varchar(45),
        IN city varchar(45),
        IN state varchar(45),
        IN zip varchar(45),
        IN country varchar(45),
        IN phone varchar(45),
        IN gender varchar(1),
        IN roleID INT
)
BEGIN
        declare email_id_flag int default 0;
        declare user_id_flag int default 0;
        declare var_customer_id int default null;

        select count(*)>0 into email_id_flag from bs_customer where email= emailID;
        select count(*)>0 into user_id_flag from bs_users where user_id= userID;
        if email_id_flag = 1 THEN
                select 'duplicate email id found ' ;
```

```
elseif user_id_flag = 1 THEN
        select 'duplicate user id found ';
else
        insert into bs_customer(email,phone,fname,lname,city,state,zip,country)
values(emailID,phone,fname,lname,city,state,zip,country);

        select last_insert_id() into var_customer_id;
        insert into bs_users(user_id,password,role_id,person_id,last_login)
        values(userID,userPassword,roleID,var_customer_id,current_timestamp());
        select 'registration complete' as result;
end if;
END
```

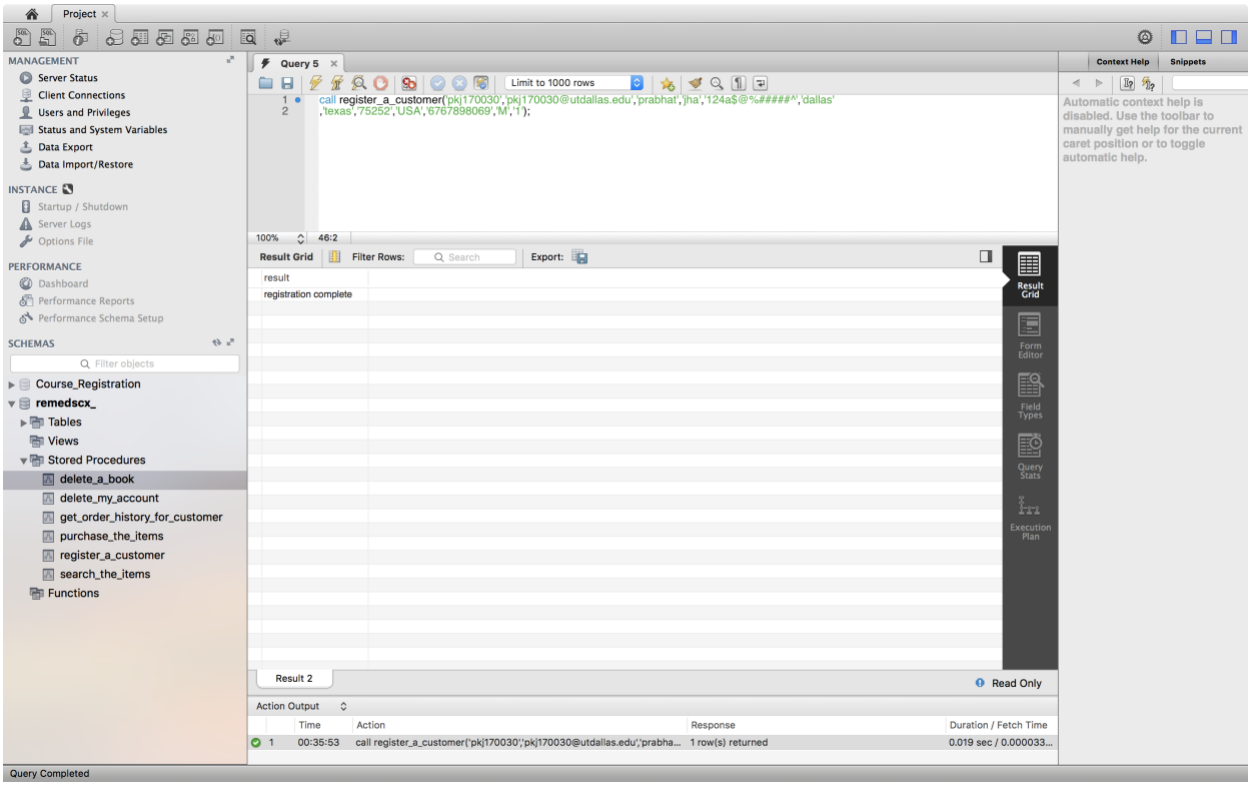




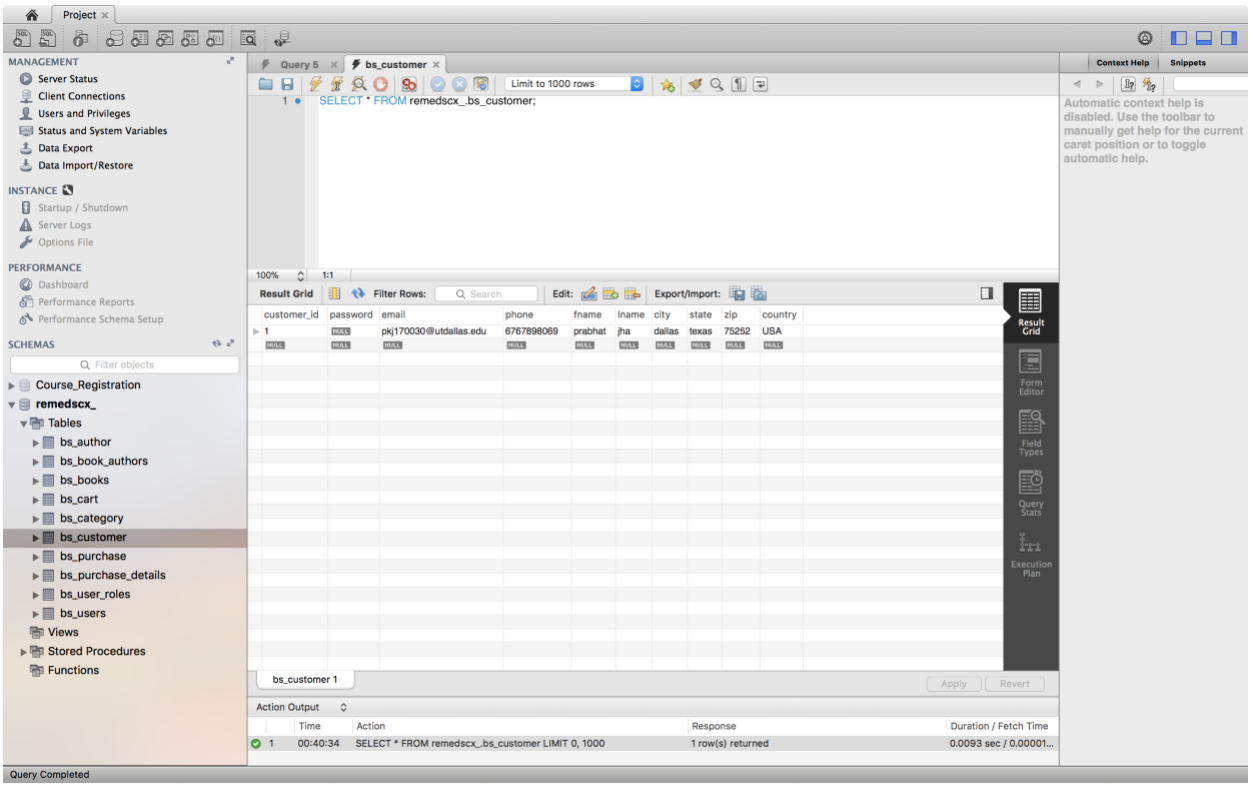


Test Case with Output:

call
register_a_customer('pkj170030','pkj170030@utdallas.edu','prabhat','jha','124a$@%#####^','dallas',
'texas','75252','USA','6767898069','M','1');



The bs_customer Table after the Procedure is called,



2. Search Items

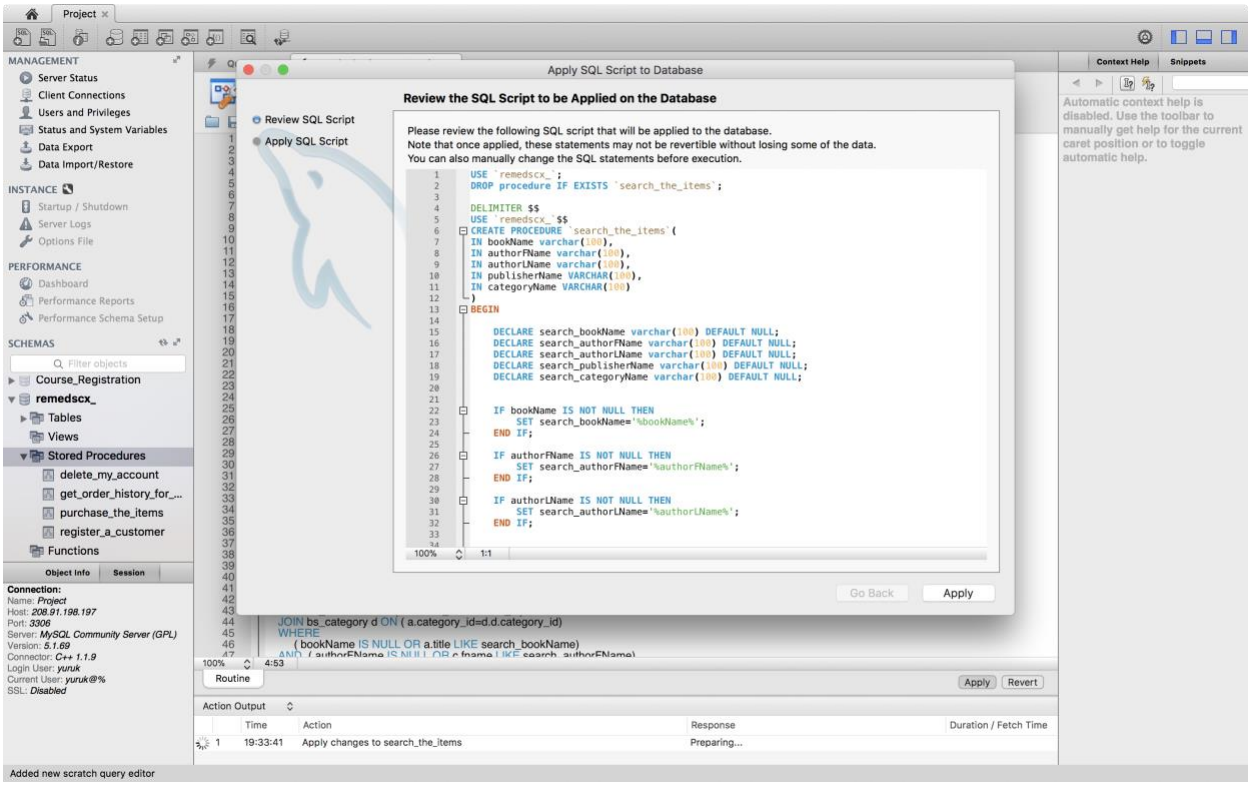
This procedure enables the user to take a deep dive into the database and search for books by the book name, author name, publisher name as well as the category. We can obtain detailed information about the book using this.

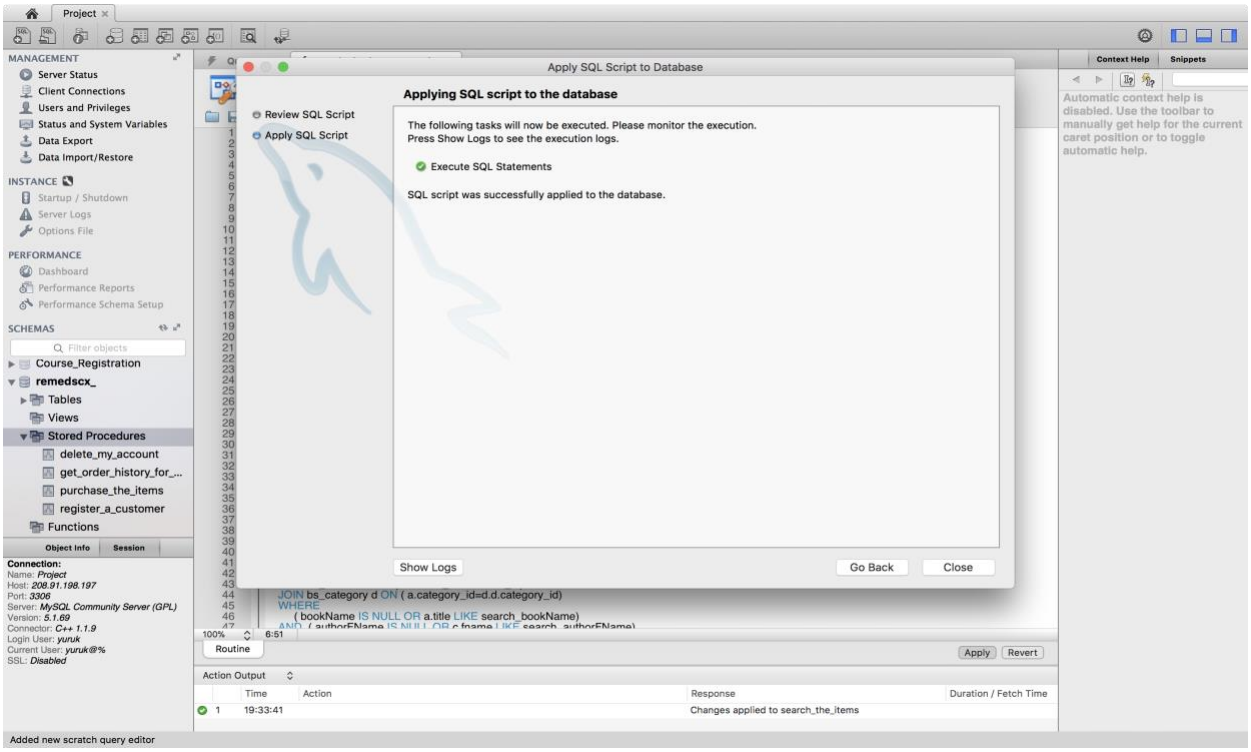
Args: (bookName IN, authorFName IN, authorLName IN, publisherName IN, categoryName IN)

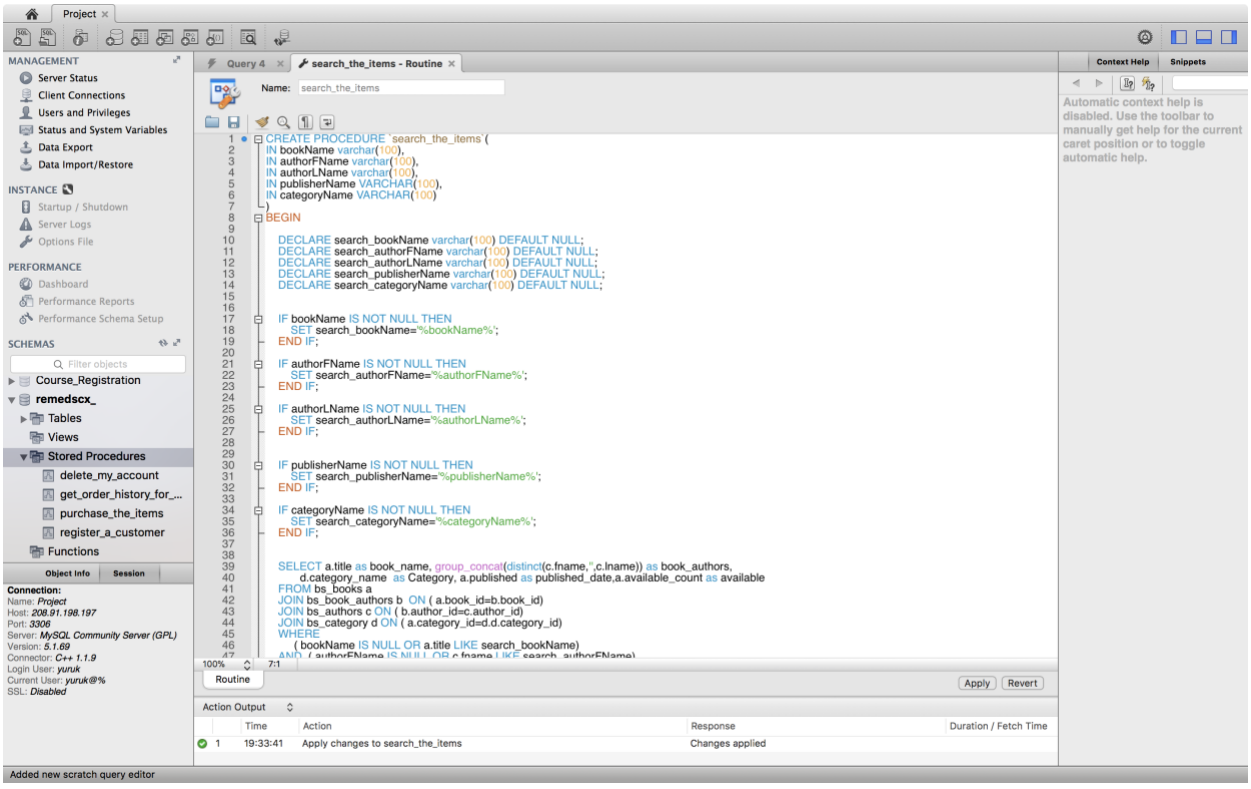
CODE:

```
CREATE PROCEDURE `search_the_items`(
    IN bookName varchar(100),
    IN authorFName varchar(100),
    IN authorLName varchar(100),
    IN publisherName VARCHAR(100),
    IN categoryName VARCHAR(100)
)
BEGIN
    DECLARE search_bookName varchar(100) DEFAULT NULL;
    DECLARE search_authorFName varchar(100) DEFAULT NULL;
    DECLARE search_authorLName varchar(100) DEFAULT NULL;
    DECLARE search_publisherName varchar(100) DEFAULT NULL;
    DECLARE search_categoryName varchar(100) DEFAULT NULL;
    IF bookName IS NOT NULL THEN
        SET search_bookName='%bookName%';
    END IF;
    IF authorFName IS NOT NULL THEN
        SET search_authorFName='%authorFName%';
    END IF;
    IF authorLName IS NOT NULL THEN
        SET search_authorLName='%authorLName%';
    END IF;
    IF publisherName IS NOT NULL THEN
        SET search_publisherName='%publisherName%';
    END IF;
```

```
IF categoryName IS NOT NULL THEN
        SET search_categoryName='%categoryName%';
END IF;
SELECT a.title as book_name, group_concat(distinct(c.fname,'',c.lname)) as book_authors,
d.category_name  as Category, a.published as published_date,a.available_count as available
FROM bs_books a
JOIN bs_book_authors b  ON ( a.book_id=b.book_id)
JOIN bs_authors c ON ( b.author_id=c.author_id)
JOIN bs_category d ON ( a.category_id=d.d.category_id)
WHERE
        ( bookName IS NULL OR a.title LIKE search_bookName)
        AND  ( authorFName IS NULL OR c.fname LIKE search_authorFName)
        AND  ( authorLName IS NULL OR c.lname LIKE search_authorLName)
        AND  ( publisherName IS NULL OR a.publisher LIKE search_publisherName)
        AND  ( categoryName IS NULL OR d.category_name LIKE search_categoryName)
GROUP BY a.book_id;
END
```




Project
Apply SQL Script to Database
Applying SQL script to the database
Review SQL Script
Apply SQL Script
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Execute SQL Statements
SQL script was successfully applied to the database.
Show Logs
Go Back
Close
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1 19:33:41 Changes applied to search_the_items
Added new scratch query editor


Project
Query 4
search_the_items - Routine
Name: search_the_items
CREATE PROCEDURE `search_the_items`(
IN bookName varchar(100),
IN authorFName varchar(100),
IN authorLName varchar(100),
IN publisherName VARCHAR(100),
IN categoryName VARCHAR(100)
)
BEGIN
DECLARE search_bookName varchar(100) DEFAULT NULL;
DECLARE search_authorFName varchar(100) DEFAULT NULL;
DECLARE search_authorLName varchar(100) DEFAULT NULL;
DECLARE search_publisherName varchar(100) DEFAULT NULL;
DECLARE search_categoryName varchar(100) DEFAULT NULL;
IF bookName IS NOT NULL THEN
SET search_bookName='%bookName%';
END IF;
IF authorFName IS NOT NULL THEN
SET search_authorFName='%authorFName%';
END IF;
IF authorLName IS NOT NULL THEN
SET search_authorLName='%authorLName%';
END IF;
IF publisherName IS NOT NULL THEN
SET search_publisherName='%publisherName%';
END IF;
IF categoryName IS NOT NULL THEN
SET search_categoryName='%categoryName%';
END IF;
SELECT a.title as book_name, group_concat(distinct(c.fname,",c.lname)) as book_authors,
d.category_name as Category, a.published as published_date,a.available_count as available
FROM bs_books a
JOIN bs_book_authors b ON (a.book_id=b.book_id)
JOIN bs_authors c ON (b.author_id=c.author_id)
JOIN bs_category d ON (a.category_id=d.d.category_id)
WHERE
(bookName IS NULL OR a.title LIKE search_bookName)
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1 19:33:41 Apply changes to search_the_items Changes applied
Added new scratch query editor

3. Complete the Purchase of Item in Cart

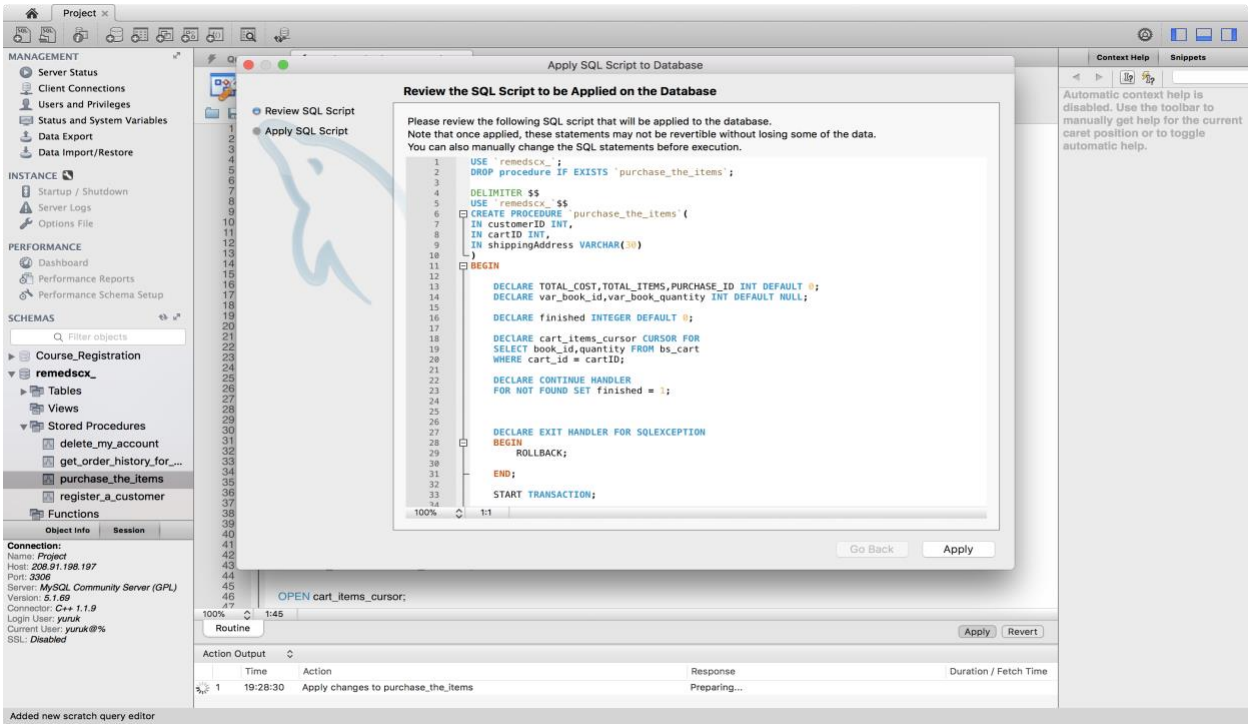
This procedure checks out and completes the purchase of items that are in cart. As such, users can add a plethora of books in their cart and when they require to check out and complete the purchase, this procedure needs to be used. This will, add up all the costs of the purchases as well as the quantities. It will also loop through the cart items, decreasing all the corresponding counts and make correct updates in other relations.

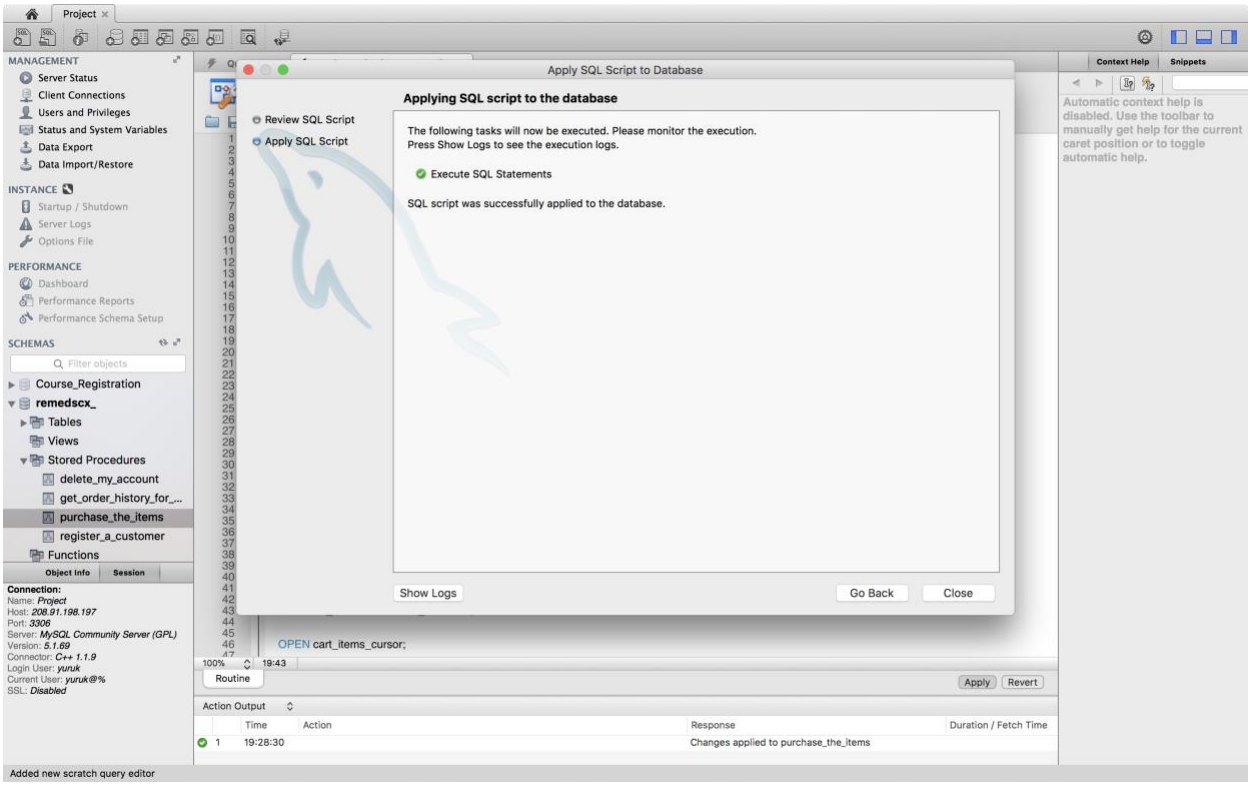
Args: (customerID IN, cartID IN, shippingAddress IN)

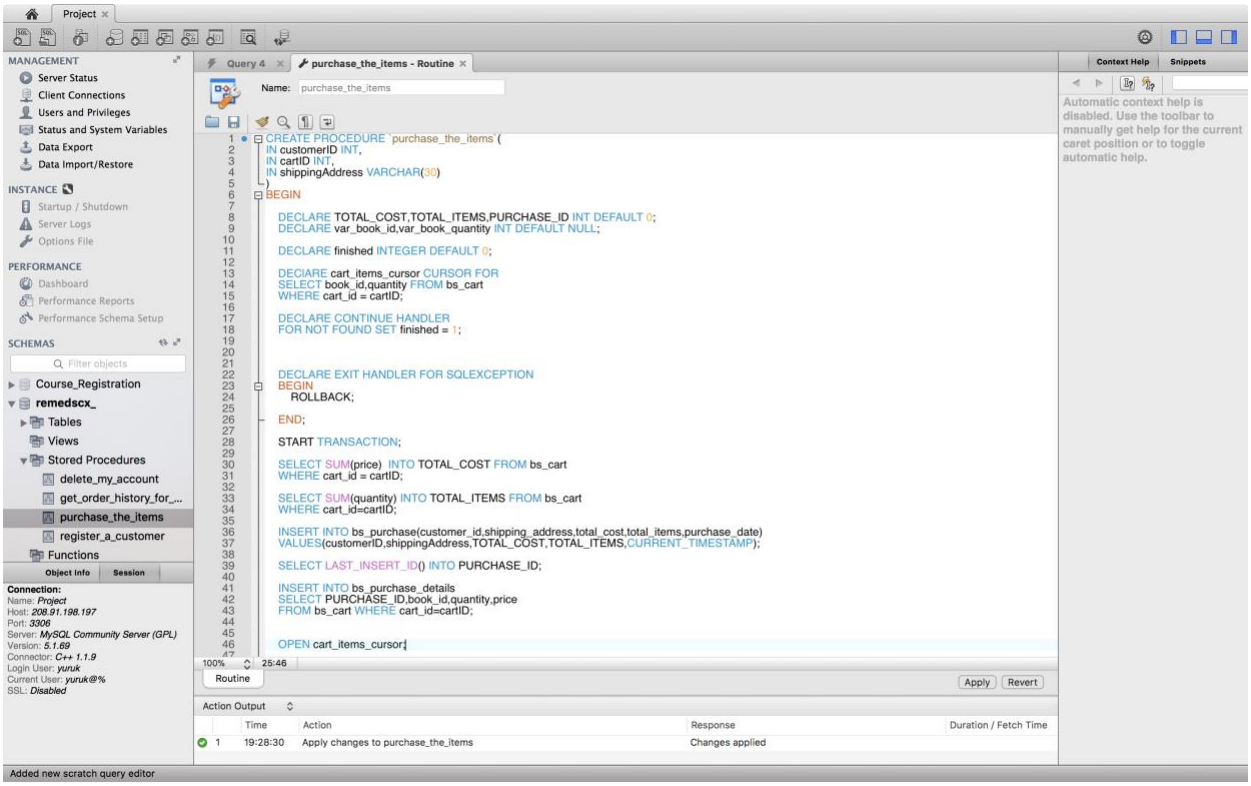
CODE:

```
CREATE PROCEDURE `purchase_the_items`(
        IN customerID INT,
        IN cartID INT,
        IN shippingAddress VARCHAR(30)
)
BEGIN
        DECLARE TOTAL_COST,TOTAL_ITEMS,PURCHASE_ID INT DEFAULT 0;
        DECLARE var_book_id,var_book_quantity INT DEFAULT NULL;
        DECLARE finished INTEGER DEFAULT 0;

        DEClARE cart_items_cursor CURSOR FOR
        SELECT book_id,quantity FROM bs_cart
        WHERE cart_id = cartID;

        DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET finished = 1;

        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                ROLLBACK;
        END;

        START TRANSACTION;
```

```
        SELECT SUM(price)  INTO TOTAL_COST FROM bs_cart WHERE cart_id = cartID;

        SELECT SUM(quantity) INTO TOTAL_ITEMS FROM bs_cart WHERE cart_id=cartID;

        INSERT INTO
bs_purchase(customer_id,shipping_address,total_cost,total_items,purchase_date)
VALUES(customerID,shippingAddress,TOTAL_COST,TOTAL_ITEMS,CURRENT_TIMESTAMP);

    SELECT LAST_INSERT_ID() INTO PURCHASE_ID;

    INSERT INTO bs_purchase_details
    SELECT PURCHASE_ID,book_id,quantity,price
    FROM bs_cart WHERE cart_id=cartID;

    OPEN cart_items_cursor;
        cart_items: LOOP
            FETCH cart_items_cursor INTO var_book_id,var_book_quantity;
            IF finished = 1 THEN
                LEAVE cart_items;
            END IF;
            UPDATE bs_books a
            SET a.total_count=a.total_count-var_book_quantity
            WHERE a.book_id=var_book_id;
        END LOOP cart_items;
    CLOSE cart_items_cursor;
    DELETE a.* FROM bs_cart a WHERE cart_id=cartID;
END
```


Project
Apply SQL Script to Database
Review the SQL Script to be Applied on the Database
Review SQL Script
Apply SQL Script
Please review the following SQL script that will be applied to the database.
Note that once applied, these statements may not be revertible without losing some of the data.
You can also manually change the SQL statements before execution.
USE `remedscx_` ;
DROP procedure IF EXISTS `purchase_the_items`;
DELIMITER $$
USE `remedscx_`$$
CREATE PROCEDURE `purchase_the_items`(
IN customerID INT,
IN cartID INT,
IN shippingAddress VARCHAR(30)
)
BEGIN
DECLARE TOTAL_COST,TOTAL_ITEMS,PURCHASE_ID INT DEFAULT 0;
DECLARE var_book_id,var_book_quantity INT DEFAULT NULL;
DECLARE finished INTEGER DEFAULT 0;
DECLARE cart_items_cursor CURSOR FOR
SELECT book_id,quantity FROM bs_cart
WHERE cart_id = cartID;
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
ROLLBACK;
END;
START TRANSACTION;
Go Back
Apply
OPEN cart_items_cursor;
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:28:30
Apply changes to purchase_the_items
Preparing...
Added new scratch query editor


Project
Apply SQL Script to Database
Applying SQL script to the database
Review SQL Script
Apply SQL Script
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Execute SQL Statements
SQL script was successfully applied to the database.
Show Logs
Go Back
Close
OPEN cart_items_cursor;
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:28:30
Changes applied to purchase_the_items
Added new scratch query editor


Project
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
Course_Registration
remedscx_
Tables
Views
Stored Procedures
delete_my_account
get_order_history_for_...
purchase_the_items
register_a_customer
Functions
Object Info
Session
Connection:
Name: Project
Host: 208.91.198.197
Port: 3306
Server: MySQL Community Server (GPL)
Version: 5.1.69
Connector: C++ 1.1.9
Login User: yuruk
Current User: yuruk@%
SSL: Disabled
Query 4
purchase_the_items - Routine
Name: purchase_the_items
CREATE PROCEDURE `purchase_the_items`(
IN customerID INT,
IN cartID INT,
IN shippingAddress VARCHAR(30)
)
BEGIN
DECLARE TOTAL_COST,TOTAL_ITEMS,PURCHASE_ID INT DEFAULT 0;
DECLARE var_book_id,var_book_quantity INT DEFAULT NULL;
DECLARE finished INTEGER DEFAULT 0;
DECIARE cart_items_cursor CURSOR FOR
SELECT book_id,quantity FROM bs_cart
WHERE cart_id = cartID;
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
ROLLBACK;
END;
START TRANSACTION;
SELECT SUM(price) INTO TOTAL_COST FROM bs_cart
WHERE cart_id = cartID;
SELECT SUM(quantity) INTO TOTAL_ITEMS FROM bs_cart
WHERE cart_id=cartID;
INSERT INTO bs_purchase(customer_id,shipping_address,total_cost,total_items,purchase_date)
VALUES(customerID,shippingAddress,TOTAL_COST,TOTAL_ITEMS,CURRENT_TIMESTAMP);
SELECT LAST_INSERT_ID() INTO PURCHASE_ID;
INSERT INTO bs_purchase_details
SELECT PURCHASE_ID,book_id,quantity,price
FROM bs_cart WHERE cart_id=cartID;
OPEN cart_items_cursor;
100%
25:46
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:28:30
Apply changes to purchase_the_items
Changes applied
Context Help
Snippets
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Added new scratch query editor

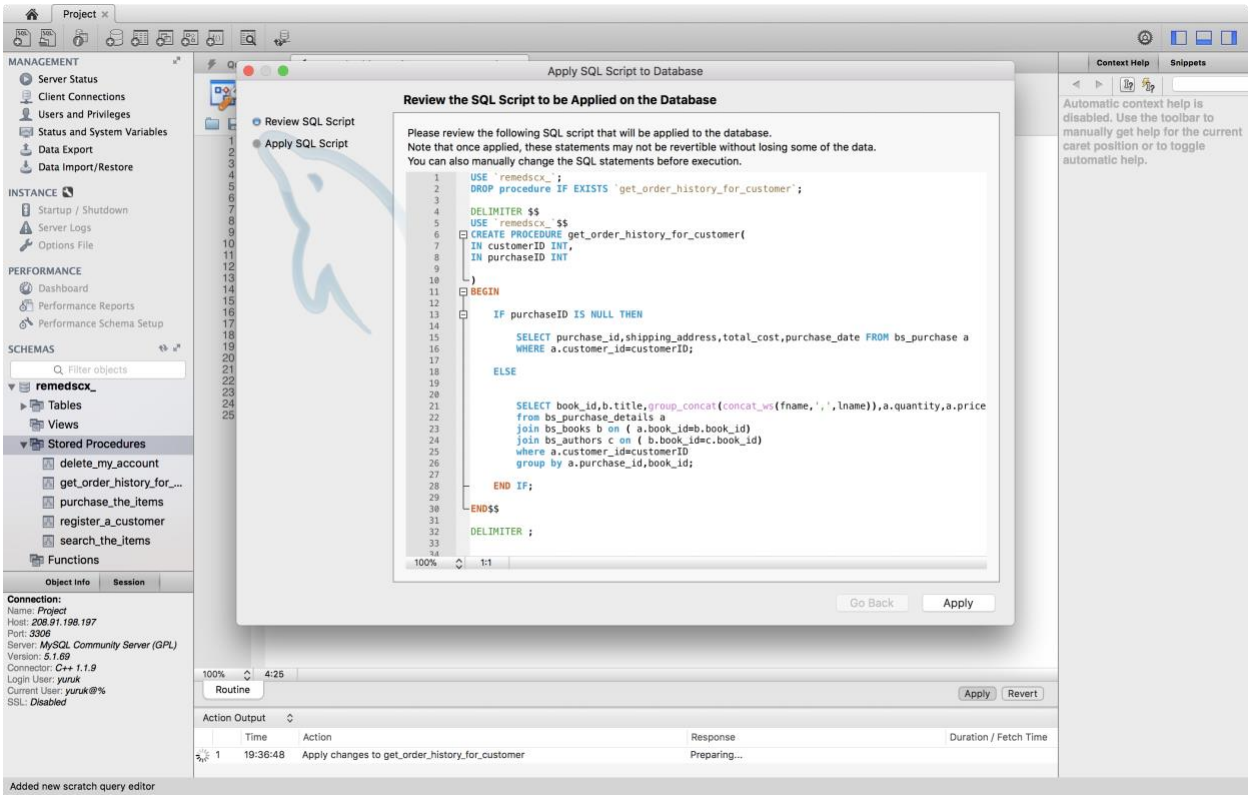
4. Get Order History

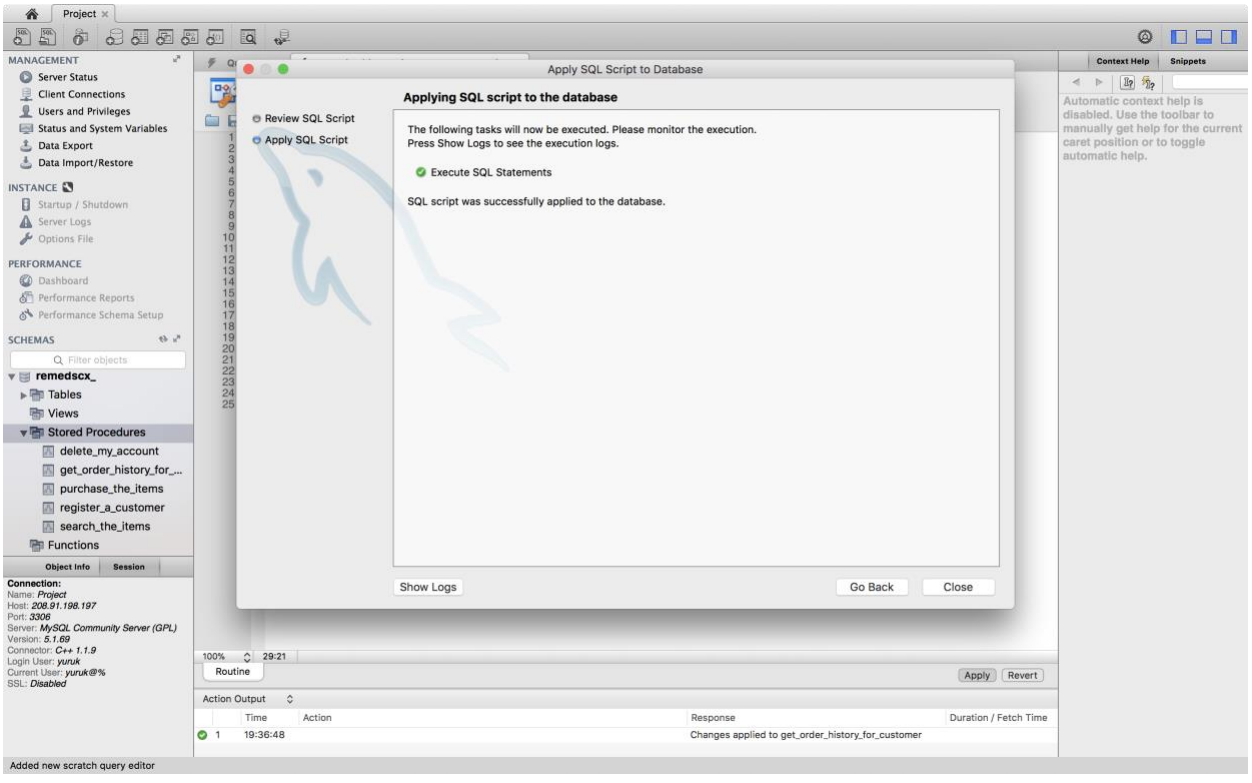
The user can use this procedure in two ways. Either he can get his entire order history OR he can get the details about a particular purchase by providing the purchaseID.

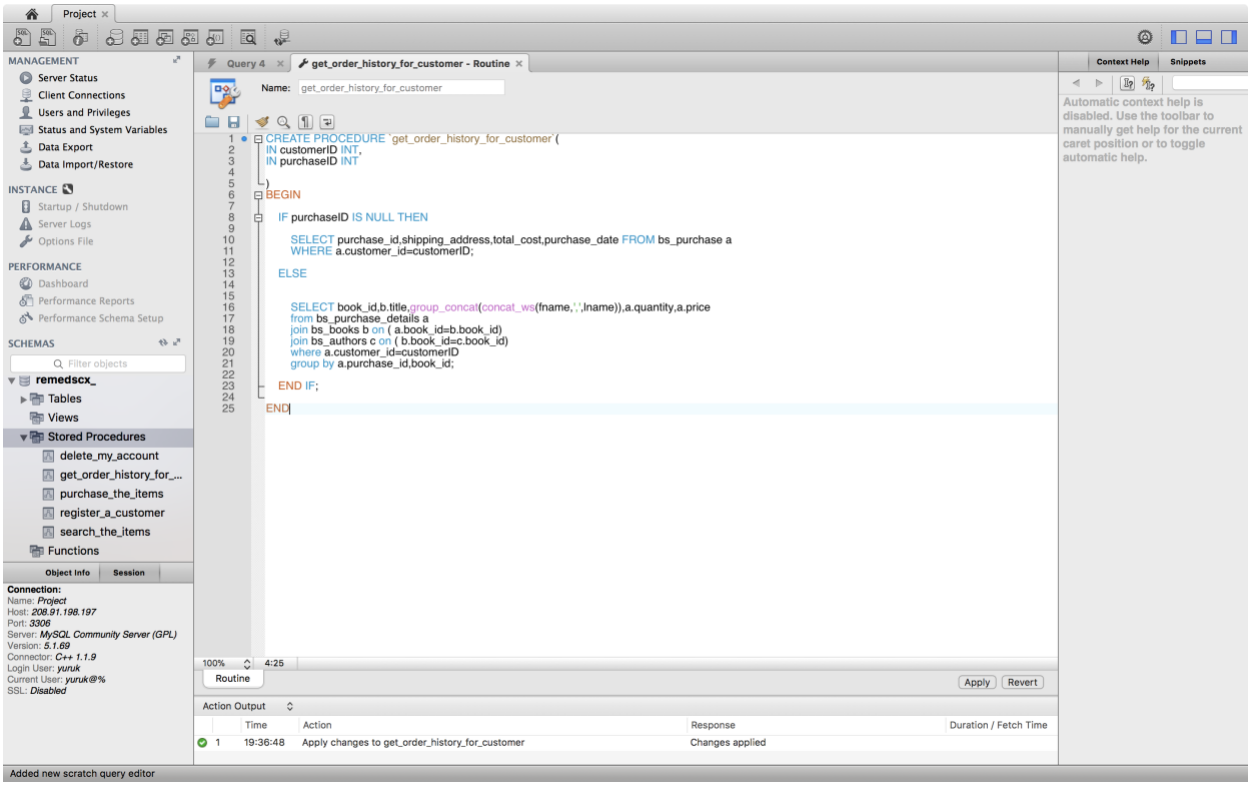
Args: (customerID IN, purchaseID IN)

CODE:

```
CREATE PROCEDURE get_order_history_for_customer(
        IN customerID INT,
        IN purchaseID INT
)
BEGIN
        IF purchaseID IS NULL THEN
                SELECT purchase_id,shipping_address,total_cost,purchase_date FROM bs_purchase a
                WHERE a.customer_id=customerID;
        ELSE
                SELECT book_id,b.title,group_concat(concat_ws(fname,',',lname)),a.quantity,a.price
                from bs_purchase_details a
                join bs_books b on ( a.book_id=b.book_id)
                join bs_authors c on ( b.book_id=c.book_id)
                where a.customer_id=customerID
                group by a.purchase_id,book_id;
        END IF;
END
```


Project
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
remedscx_
Tables
Views
Stored Procedures
delete_my_account
get_order_history_for_...
purchase_the_items
register_a_customer
search_the_items
Functions
Object Info
Session
Connection:
Name: Project
Host: 208.91.198.197
Port: 3306
Server: MySQL Community Server (GPL)
Version: 5.1.69
Connector: C++ 1.1.9
Login User: yuruk
Current User: yuruk@%
SSL: Disabled
Apply SQL Script to Database
Review the SQL Script to be Applied on the Database
Review SQL Script
Apply SQL Script
Please review the following SQL script that will be applied to the database.
Note that once applied, these statements may not be revertible without losing some of the data.
You can also manually change the SQL statements before execution.
USE `remedscx_`;
DROP procedure IF EXISTS `get_order_history_for_customer`;
DELIMITER $$
USE `remedscx_`$$
CREATE PROCEDURE get_order_history_for_customer(
IN customerID INT,
IN purchaseID INT
)
BEGIN
IF purchaseID IS NULL THEN
SELECT purchase_id,shipping_address,total_cost,purchase_date FROM bs_purchase a
WHERE a.customer_id=customerID;
ELSE
SELECT book_id,b.title,group_concat(concat_ws(fname,',',lname)),a.quantity,a.price
from bs_purchase_details a
join bs_books b on (a.book_id=b.book_id)
join bs_authors c on (b.book_id=c.book_id)
where a.customer_id=customerID
group by a.purchase_id,book_id;
END IF;
END$$
DELIMITER ;
100%
1:1
Go Back
Apply
Routine
4:25
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:36:48
Apply changes to get_order_history_for_customer
Preparing...
Context Help
Snippets
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Added new scratch query editor


Project
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
remedscx_
Tables
Views
Stored Procedures
delete_my_account
get_order_history_for_...
purchase_the_items
register_a_customer
search_the_items
Functions
Object Info
Session
Connection:
Name: Project
Host: 208.91.198.197
Port: 3306
Server: MySQL Community Server (GPL)
Version: 5.1.69
Connector: C++ 1.1.9
Login User: yuruk
Current User: yuruk@%
SSL: Disabled
Apply SQL Script to Database
Applying SQL script to the database
Review SQL Script
Apply SQL Script
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Execute SQL Statements
SQL script was successfully applied to the database.
Show Logs
Go Back
Close
100%
29:21
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:36:48
Changes applied to get_order_history_for_customer
Context Help
Snippets
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Added new scratch query editor


Project
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
remedscx_
Tables
Views
Stored Procedures
delete_my_account
get_order_history_for_...
purchase_the_items
register_a_customer
search_the_items
Functions
Object Info
Session
Connection:
Name: Project
Host: 208.91.198.197
Port: 3306
Server: MySQL Community Server (GPL)
Version: 5.1.69
Connector: C++ 1.1.9
Login User: yuruk
Current User: yuruk@%
SSL: Disabled
Query 4
get_order_history_for_customer - Routine
Name: get_order_history_for_customer
CREATE PROCEDURE `get_order_history_for_customer`(
IN customerID INT,
IN purchaseID INT
)
BEGIN
IF purchaseID IS NULL THEN
SELECT purchase_id,shipping_address,total_cost,purchase_date FROM bs_purchase a
WHERE a.customer_id=customerID;
ELSE
SELECT book_id,b.title,group_concat(concat_ws(fname,' ',lname)),a.quantity,a.price
from bs_purchase_details a
join bs_books b on (a.book_id=b.book_id)
join bs_authors c on (b.book_id=c.book_id)
where a.customer_id=customerID
group by a.purchase_id,book_id;
END IF;
END
100%
4:25
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:36:48
Apply changes to get_order_history_for_customer
Changes applied
Context Help
Snippets
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Added new scratch query editor

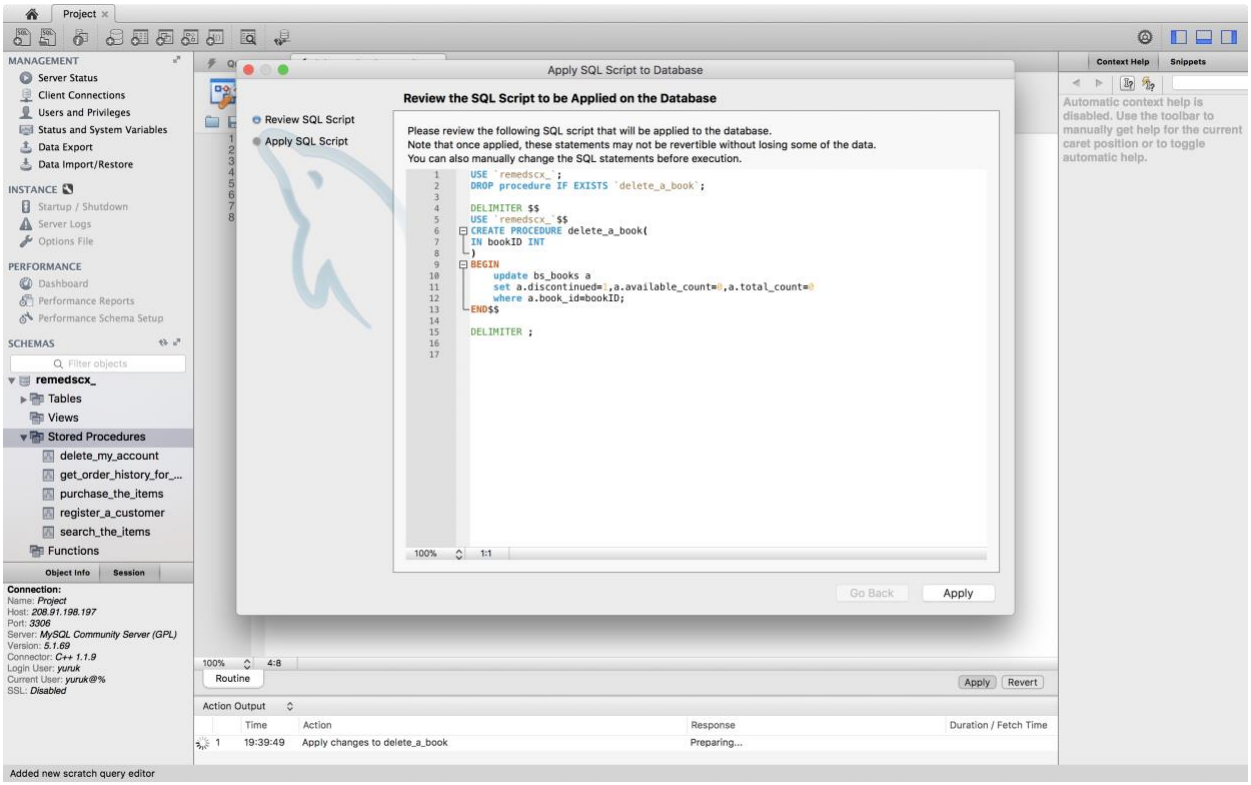
5. Delete a Book

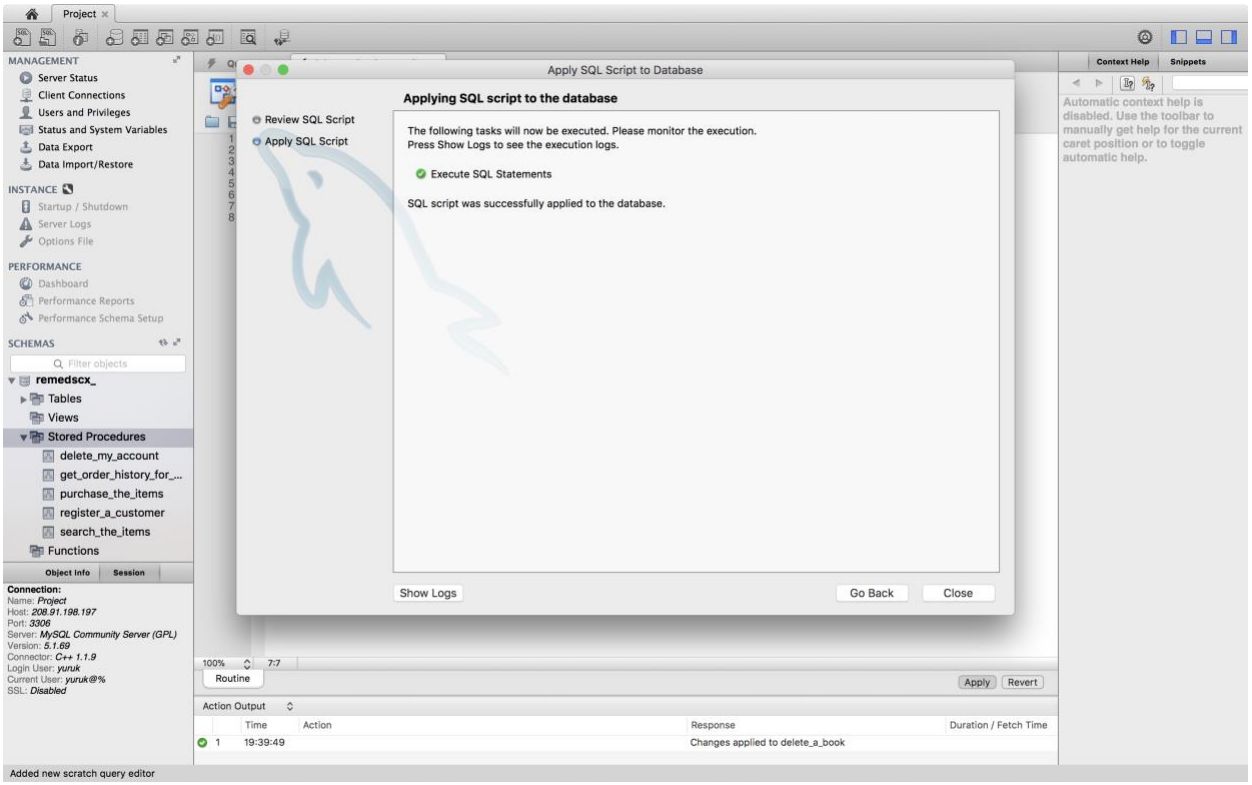
When a book needs to be removed from the Book Store, we call upon this procedure. This procedure updates the relevant availability counts and total counts for that particular book.

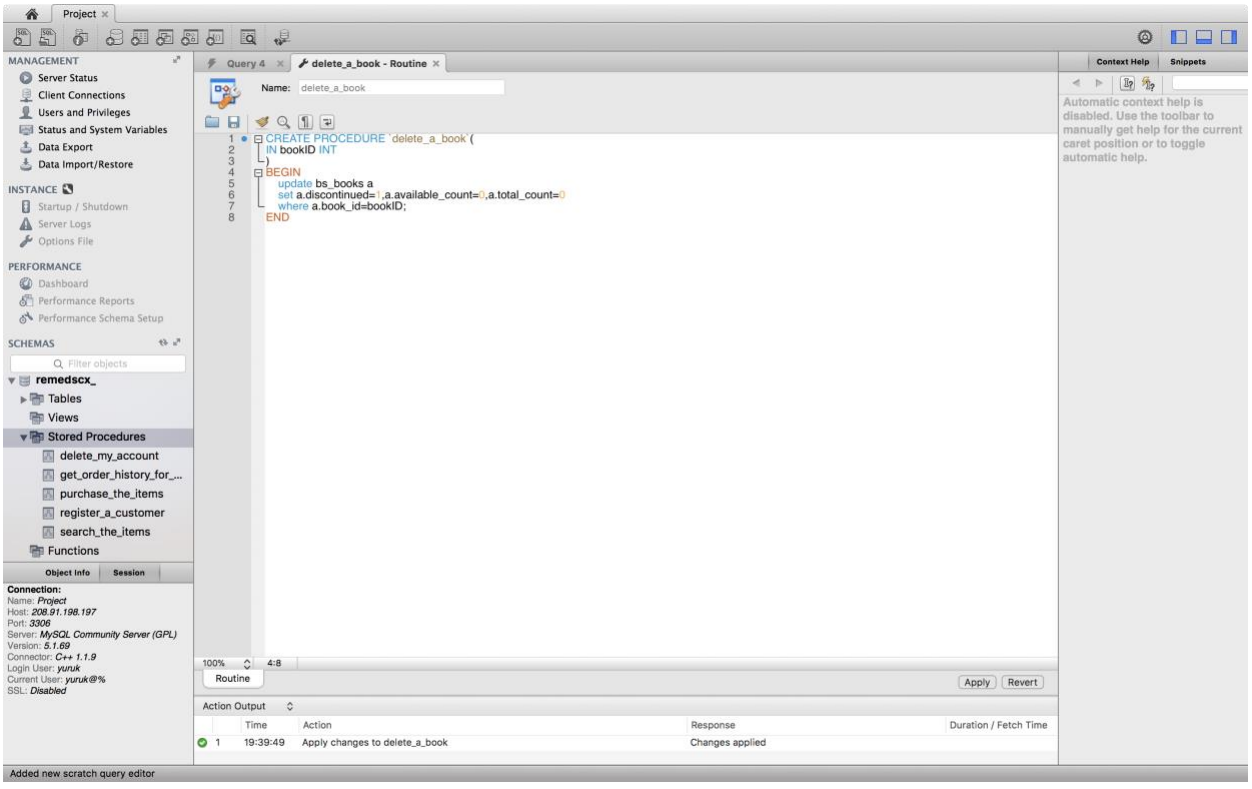
Args: (bookID IN)

CODE:

```
CREATE PROCEDURE delete_a_book(
        IN bookID INT
)
BEGIN
        update bs_books a
        set a.discontinued=1,a.available_count=0,a.total_count=0
        where a.book_id=bookID;
END
```




Project
Apply SQL Script to Database
Applying SQL script to the database
Review SQL Script
Apply SQL Script
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Execute SQL Statements
SQL script was successfully applied to the database.
Show Logs
Go Back
Close
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1 19:39:49 Changes applied to delete_a_book
Added new scratch query editor


Project
Query 4
delete_a_book - Routine
Name: delete_a_book
CREATE PROCEDURE `delete_a_book`(
IN bookID INT
)
BEGIN
update bs_books a
set a.discontinued='',a.available_count='0',a.total_count='0'
where a.book_id=bookID;
END
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1 19:39:49 Apply changes to delete_a_book Changes applied
Added new scratch query editor

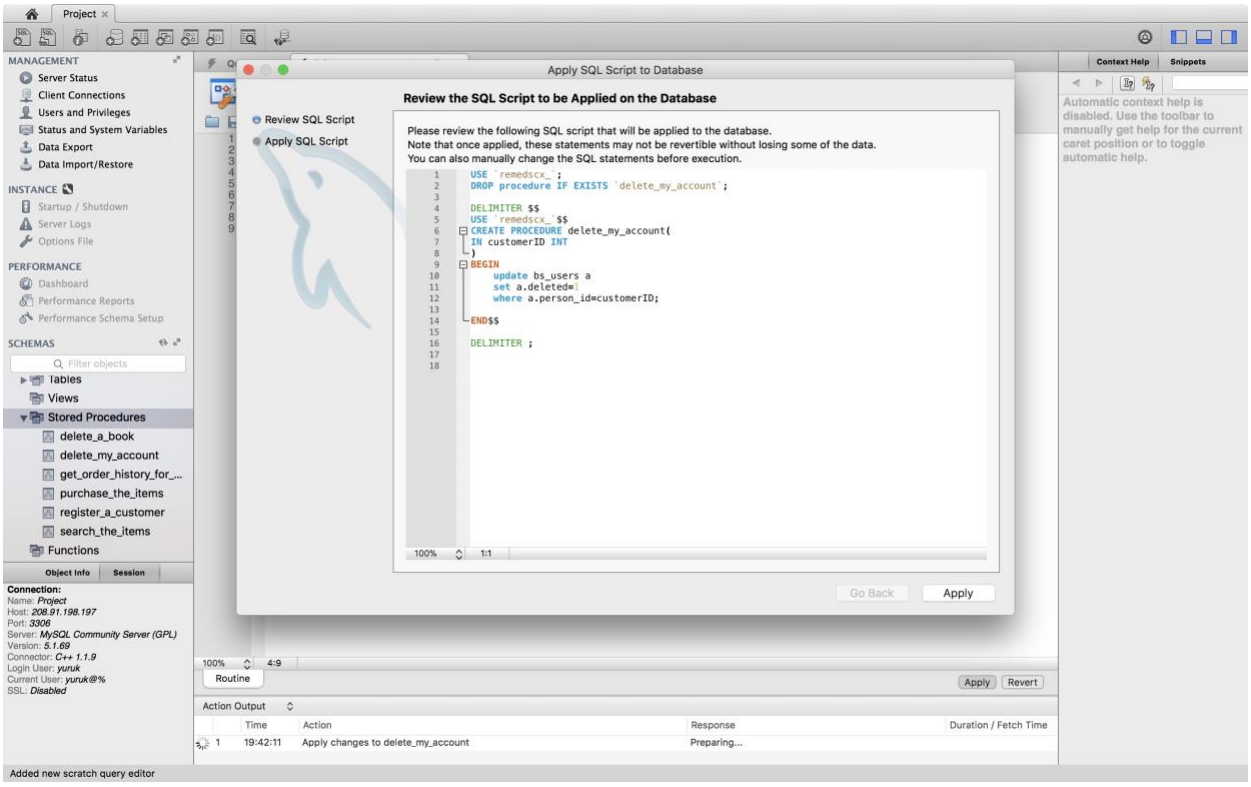
6. Deleting a Customer Profile

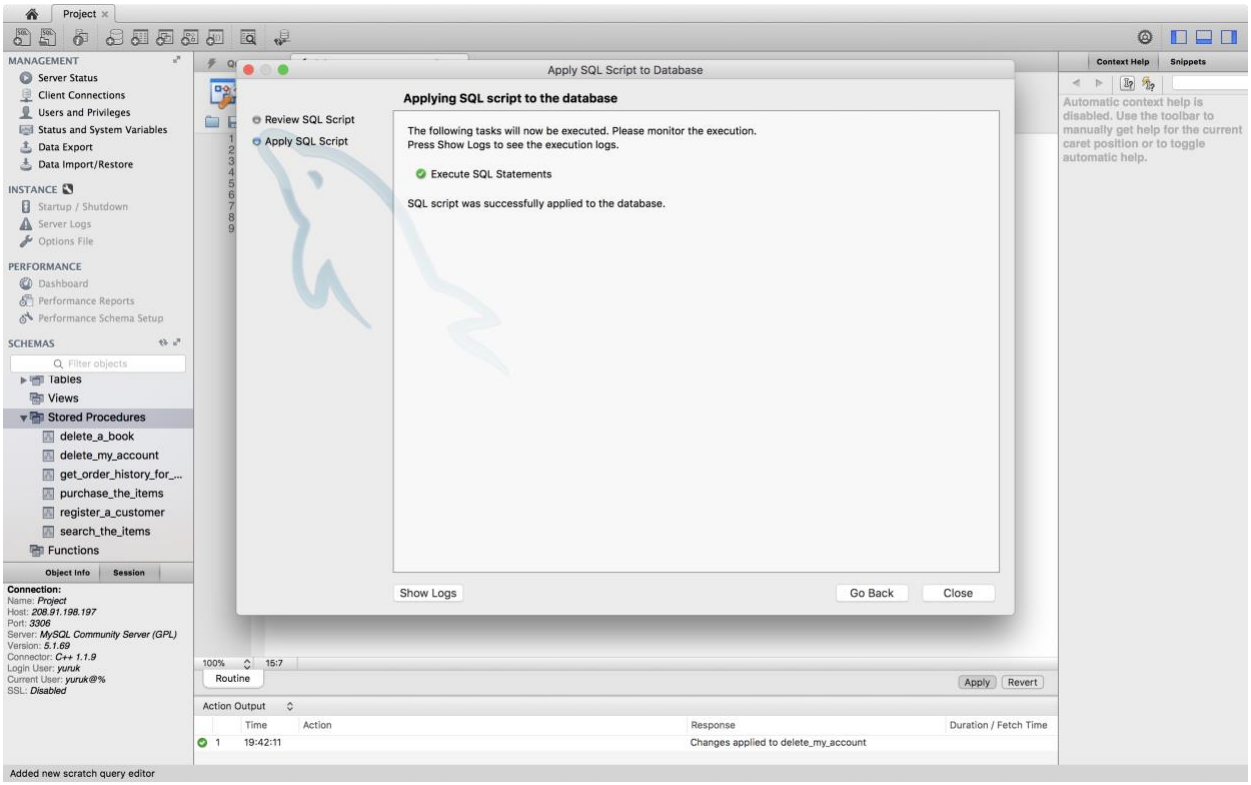
When a customer wants to quit the profile on the Online Book store, this procedure is called and the associated customer profile is removed.

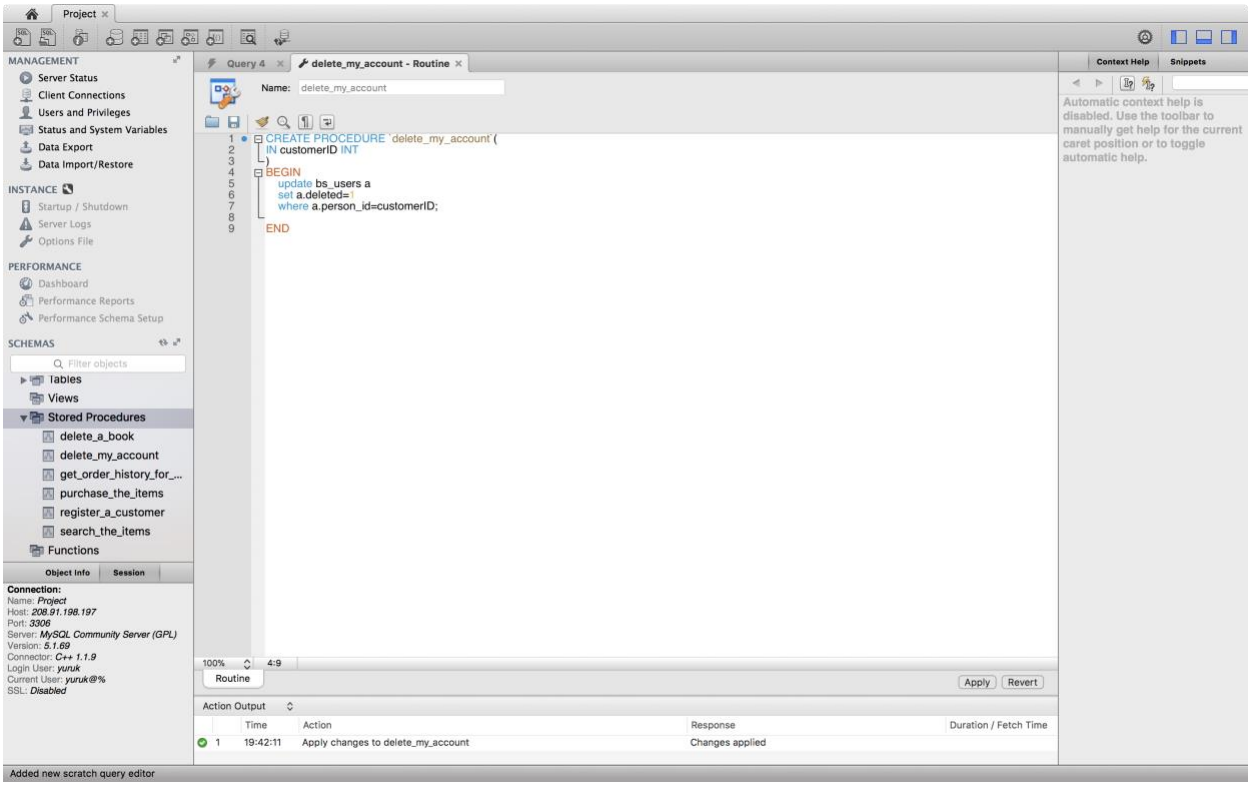
Args: (customerID IN)

CODE:

```
CREATE PROCEDURE delete_my_account (
        IN customerID INT
)
BEGIN
        update bs_users a
        set a.deleted=1
        where a.person_id=customerID;
END
```




Project
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
Tables
Views
Stored Procedures
delete_a_book
delete_my_account
get_order_history_for_...
purchase_the_items
register_a_customer
search_the_items
Functions
Object Info
Session
Connection:
Name: Project
Host: 208.91.198.197
Port: 3306
Server: MySQL Community Server (GPL)
Version: 5.1.69
Connector: C++ 1.1.9
Login User: yuruk
Current User: yuruk@%
SSL: Disabled
Apply SQL Script to Database
Applying SQL script to the database
Review SQL Script
Apply SQL Script
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Execute SQL Statements
SQL script was successfully applied to the database.
Show Logs
Go Back
Close
Context Help
Snippets
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:42:11
Changes applied to delete_my_account
Added new scratch query editor


Project
Query 4
delete_my_account - Routine
Name: delete_my_account
CREATE PROCEDURE `delete_my_account`(
IN customerID INT
)
BEGIN
update bs_users a
set a.deleted=1
where a.person_id=customerID;
END
Routine
Apply
Revert
Action Output
Time
Action
Response
Duration / Fetch Time
1
19:42:11
Apply changes to delete_my_account
Changes applied
Added new scratch query editor