공학석사 학위논문

DEVELOPEMENT OF DATABASE AND STORED PROCEDURES FOR E-COMMERCE SYSTEM

전자상거래 시스템을 위한 데이터베이스 및 저장프로시저 개발

지도교수 박 휴 찬

2019년 8월

한국해양대학교 대학원

컴퓨터공학과

MOHAMMAD HAZIQ BIN MASLAN

Table of Contents

List of Figures

Development of Database and Stored Procedure for E-Commerce Systems

Mohammad Haziq Bin Maslan

Department of Computer Engineering,

Graduate School, Korea Maritime Ocean University

Advised by:

Prof. Hyu-Chan Park

Abstract

E-commerce has been an important entity in our daily life. Mobile commerce will make up to 45% of total revenue of e-commerce transaction by 2020. Database design is important in creating e-commerce application. Previous work on the database design for e-commerce application is improved.

This thesis further improve previous work on database design for e-commerce application. The method of using stored procedure technique in creating an e-commerce application is also applied. We propose two design that is applied for e-commerce application, which is the database design and the stored procedure technique.

The first proposes an improve design on e-commerce application database. We introduce a simple design that is applied and used in e-commerce application, further improving the previous design. The design focus on a simple approach to a database for e-commerce application.

The second proposes on using the stored procedure technique in creating the database. Stored procedure used as a middleware for the application and the database. The stored procedure is categorised in two, which is the basic and advance stored procedure techniques. We explained on applying stored procedure technique on an e-commerce application.

The proposed stored procedure and database can applied to the creation of an e-commerce application.

Chapter 1 : Introduction

1.1 Background of Research

E-commerce has been an entity that is part of our daily life. More and more business are adapting the e-commerce business model to improve their existing business. Creating a new application will likely create new chances not to just the business but also the developers and the users itself [1]. To show the importance of e-commerce, in 2017 there were 1.66 billion online buyers. A case study shows that mobile commerce will make up to 45% of total e-commerce revenue by 2020. To stay relative in the business world most small and medium business need to start adapting to the e-commerce way of doing business.

E-commerce has the potential to drive the economy of a country to the next level with assured streamlined online business operations, enhanced income and profitability. Even with all the potential, that e-commerce can provide for the owners and customers, there is challenges in using an e-commerce system or application for a business.

Everything starts with building a web site or an application that will fill or help the business. A good website or application needs a good foundation to begin with. The website or application must have a good foundation and a good aim in terms to fulfil the needs of the application itself. To achieve this there is a few important aspects in terms of creating the application, and one of them is database and stored procedure.

Database is an important aspect of any application. The techniques to design database for any given application is becoming more complicated even for the experienced programmer. A guide or way to design a database for e-commerce using the stored procedure technique is needed in order to further improve the needs and demands of e-commerce in the market.

To create an effective e-commerce website or application we must start from the database design, and an approach from the stored procedure side of the database design. Stored procedure can have its advantages on use and an effective stored procedure can help its user and its designer.

1.2 Research Objectives

1) Preparing a base framework for database creation in e-commerce sector

Database is important in any application, and with the growing of E-Commerce sector a framework is needed for creating database in an E-Commerce application. This framework is to help programmers or designers in creating their own database for their own e-commerce program in the future.

2) Stored procedure design for e-commerce application

Stored procedure technique is important as it can improve performance of an e-commerce application. Having a design that can be used as a guide is important to help programmers. This research may achieve that stored procedure will be widely used in the creation of e-commerce application.

3) Testing how effective in using stored procedure for e-commerce system

This research is to test the impact of using stored procedure in an e-commerce application. Will it improve and increase the effectivity of the e-commerce application, or will it affect the design of e-commerce application in a negative way.

Chapter 2 : Literature Review

2.1 E-Commerce

E-commerce means electronic commerce. Any business that is done over the internet is considered E-commerce. E-commerce is usually related to a website in the internet which provide, products or services from the portal.

The act to transform, create and redefine relationship between organization and individual through electronic communication and digital information passing is e-commerce [1].

E-commerce is changing the that way business approaches their customers to sell the products and services. New methodologies are evolving and the relationship between geographic distances in forming a business has been lowered. [2]. With the rapid devolvement of the communication technologies, e-commerce will be the future of shopping. E-commerce will have a very important role in the 21st century for business of both small and large companies.

With the internet, distances between customer and business become closer [3]. Today we can see that more and more business is using e-commerce as a means to provide services and products, and with e-commerce being easily accessible by everyone, e-commerce is growing at a very fast pace.

In 2018 e-commerce has achieved a sales grow of 18%. Worldwide consumers purchase around $2.86 trillion compared to $2.43 trillion of 2017 on the web, as reported by Jessica Young [3]. This shows an amazing growth and importance for the e-commerce market.

E-commerce has become an important aspect of the business, and some business started to adopt multiple digital channels to promote their business. A Forbes case studies shows that business that adopts multiple digital channels outperforms business that does single or dual channels by 300%. With this, we can see the significance of E-commerce in today growing business.

2.2 Database and Stored Procedure

Database is a collection of information that is organized so that it can be easily accessed managed and updated. Database is an important aspect in a creation of any type of system. A good database design will affect the system significantly and how we approach the database itself will change on how we approach the system. Database design plays an important role in e-commerce application.

Database is organized into rows, columns and tables and it is indexed to make sure that retrieving information from a database is faster. New information can be added to a database, and data can be updated and deleted. With the help of Structured Query Languages (SQL), database has become more manageable and easier to control.

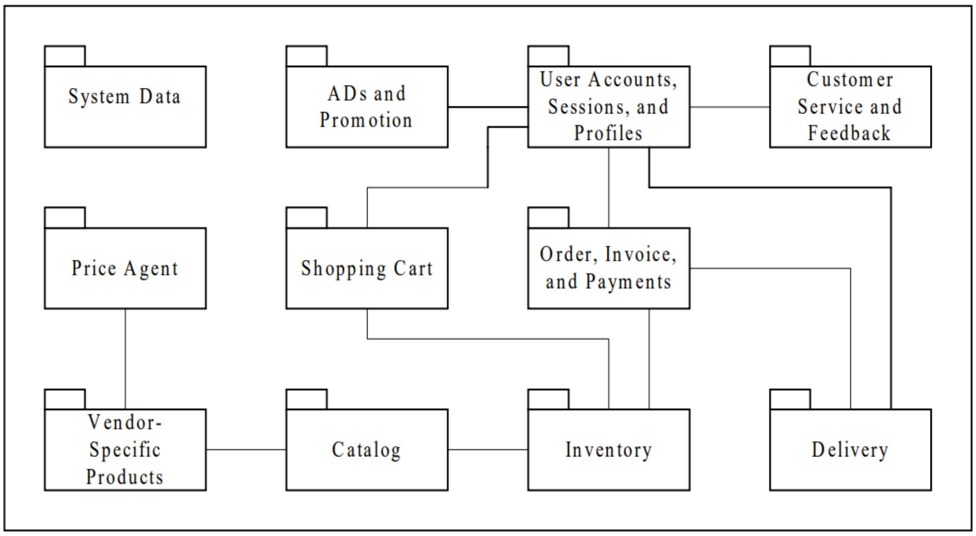


Fig. 2.1 Database Design Example for E-Commerce [5]

Figure 2.1 shows a database design for e-commerce proposed by Il Yeol Seong [5]. The design shows the important component that is needed in an e-commerce application. The database design may change with time, but the basic structure stays the same. The application programmer may change the design by the need of its application. This design shows how the database is linked to one another and shows the important of each database. This design can be further improved by simplifying more on the design itself. Some parts can be combined as a sector or group, for example the inventory and catalogue. There is no need to create two separate databases when we can apply stored procedure to reuse the data inside the databases.

A stored procedure is a program written in procedure and trigger language that is stored as part of the database. Stored procedures can be called by client applications or by other stored procedures or triggers. Triggers are almost identical to stored procedures with one exception, the way they are called. Triggers are called automatically when a change to a row in a database table occurs. This paper examines stored procedures first followed by triggers. As we will see, most of what is said about stored procedures applies to triggers as well. A stored procedure can also access or modify data in a database, but it is not tied to a specific database or object.

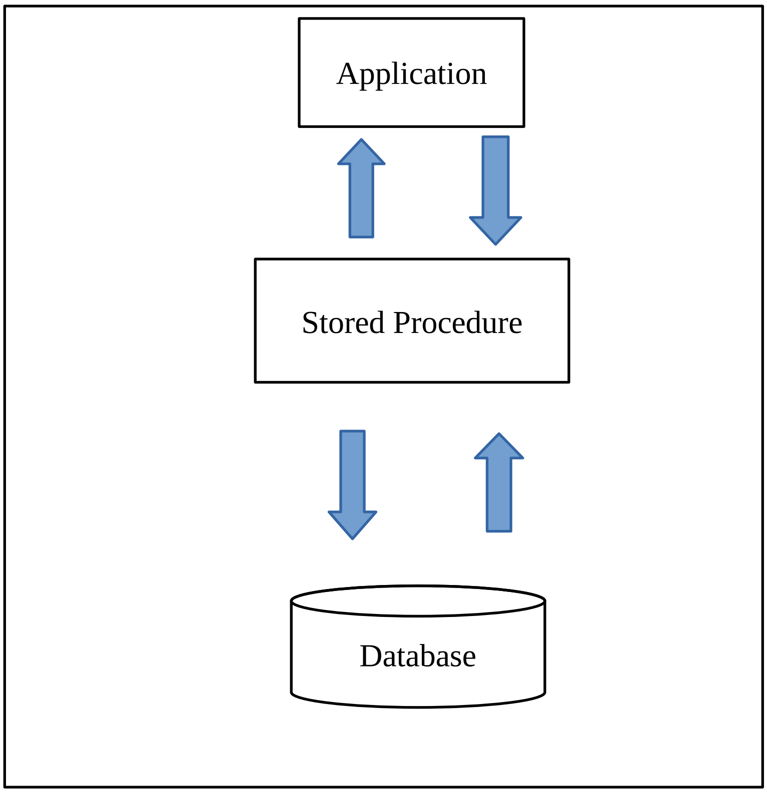


Fig. 2.2 the Workings of Stored Procedure

Figure 2.2 shows how a stored procedure interact with the application and the database. With the use of a stored procedure, any information that is needed by an application from a database will go through the stored procedure before accessing the database. This made the control of an application much easier and much manageable especially in the database side. Stored Procedure will act as a middleware between the application and database itself and stored procedure will control the flow of the data entering and exiting the database.

As told by Guy Harisson and Steven Feuerstein [4], a stored procedure can improve the performance, reliability and maintainability of any My-SQL based application, but it is not a universal solution and should only be used when it is needed and appropriately.

Chapter 3 : Database and Stored Procedure Design

3.1 Database Design

A new database design is proposed to further improve the existing design of [5]. This database design focus on the simplicity for the user and the database will be divided into a few sections

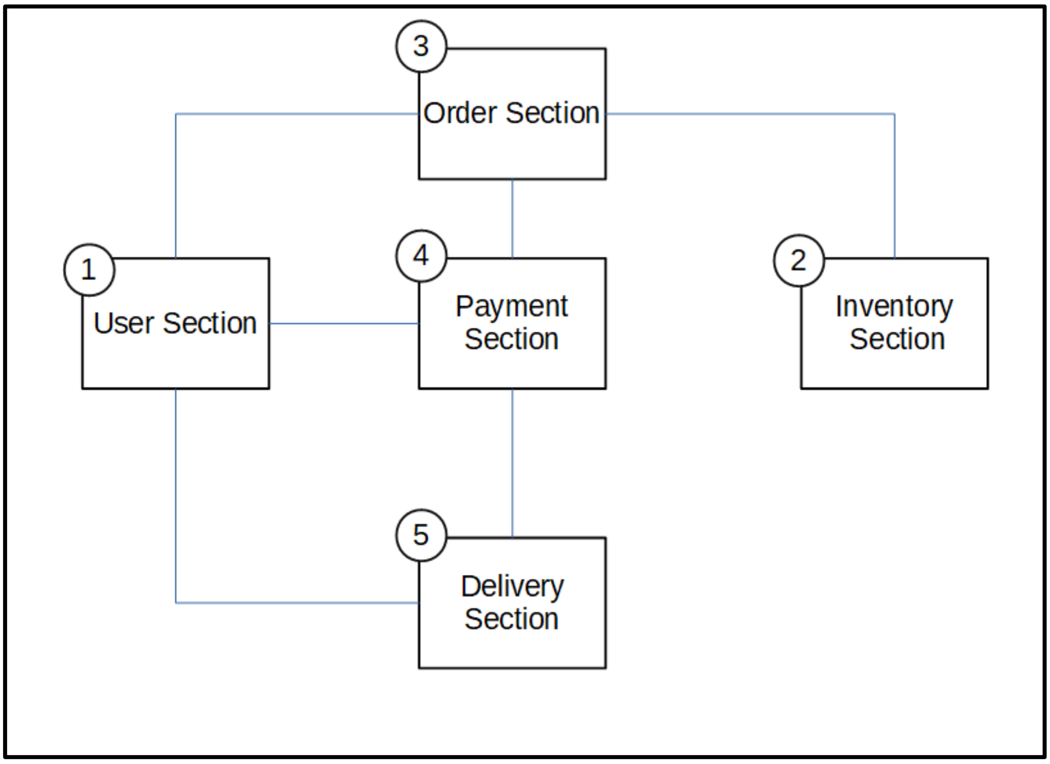


Fig. 3.1 Overall Structure of Database

As shown in figure 3.1, the base database design will consist of five main sections, which are user section, inventory section, order section, payment section and delivery section. Every section has its own database tables and each tables have its own use. The reason that the database is divided into section is to give an ease of design in creating a database table. This also will give a better understanding on the needs of the database

1) **User Section**

As shown in figure 3.2, the user section is designed to be used by both the user and admin of the e-commerce application. It consists of four tables, which are the user_account, user, address and addr_state

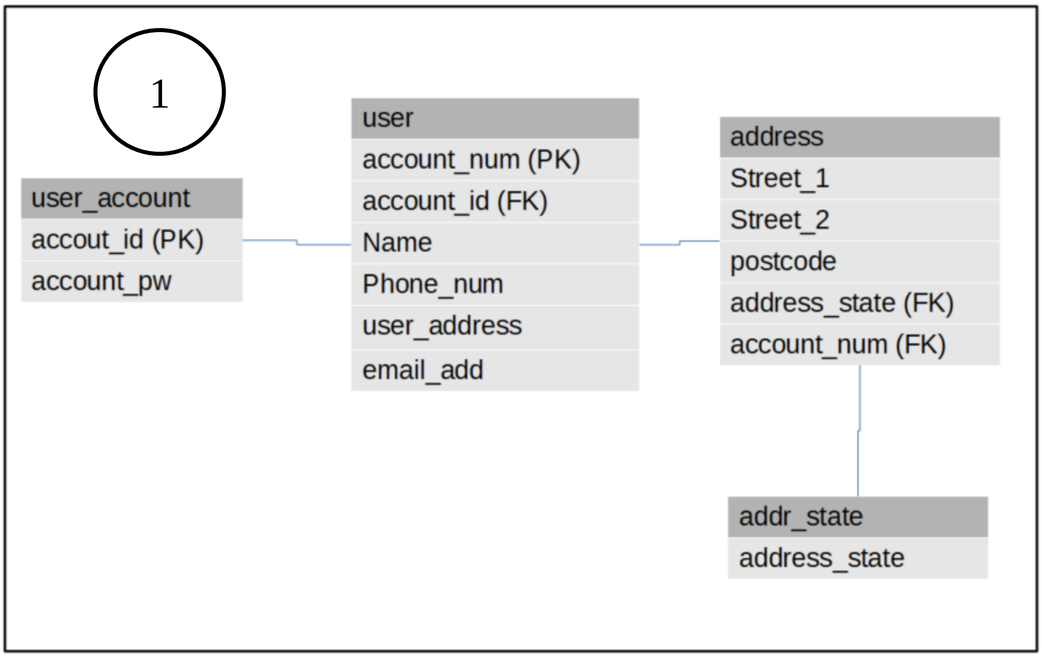


Fig. 3.2 User Section

User account table is the main table used to keep user login data. Any login data from the user will be stored here, and before accessing the user table, all users must go through the user account table. User table will keep the user information and data, which is needed by the e-commerce application. This user table may be changed appropriately based on the needs of specific e-commerce application.

The address table is for storing the address of the users. It is designed in a separate table to give the admin more control and more details on the address itself. Different tables will use the address, so it is better to have more detailed information. The addr_state table is to keep the state of address in the table.

2) Inventory Section

As shown in figure 3.3, the inventory section is one of the important sections in the database design. This inventory section would be one of the mostly used section among all. This section is expected to have a higher usage traffic compared to other sections.

The inventory section consists of shop_cart, inventory, inventory_images, inventory_category and event. The inventory table would be the main table for the e-commerce application.

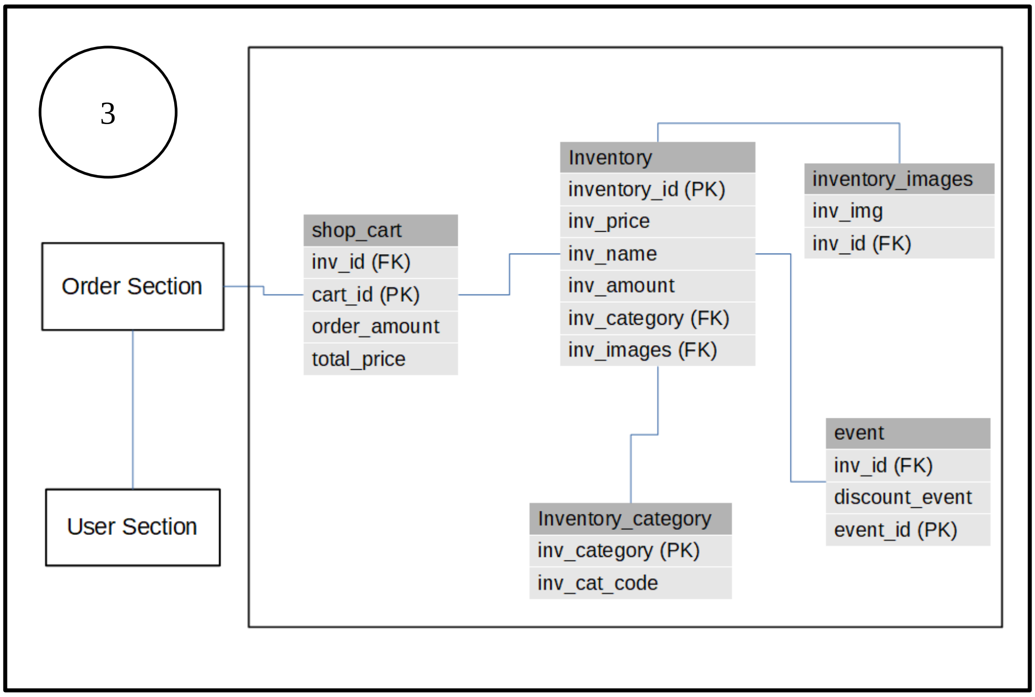


Fig. 3.3 Inventory Section

This is where all the inventory or products information will be updated for the application. The inventory is divided into category, which will be inserted in the inventory_category table. With this, the category for the inventory is more organized. It will be easier for the user in terms of searching a product by inventory and it will help the admin in terms of sorting out inventory according to the category. The images for the inventory would be stored inside the inventory_images table. One inventory might have more than one product image, so this would help further in sorting up the images for the inventory as the foreign key of the table is inv_id.

The event table is for special events regarding a product, the event could be a special offer or discount for the specific product. Event can be removed or added in this table. The shop_cart table is to keep temporary data that is used by the user when they are browsing the inventory. All this data will be linked to the user section and the order section.

3) Order Section

As shown in figure 3.4 the order section consists of three tables which is order_hist, order and order_detail. User will use order section when they are making orders.

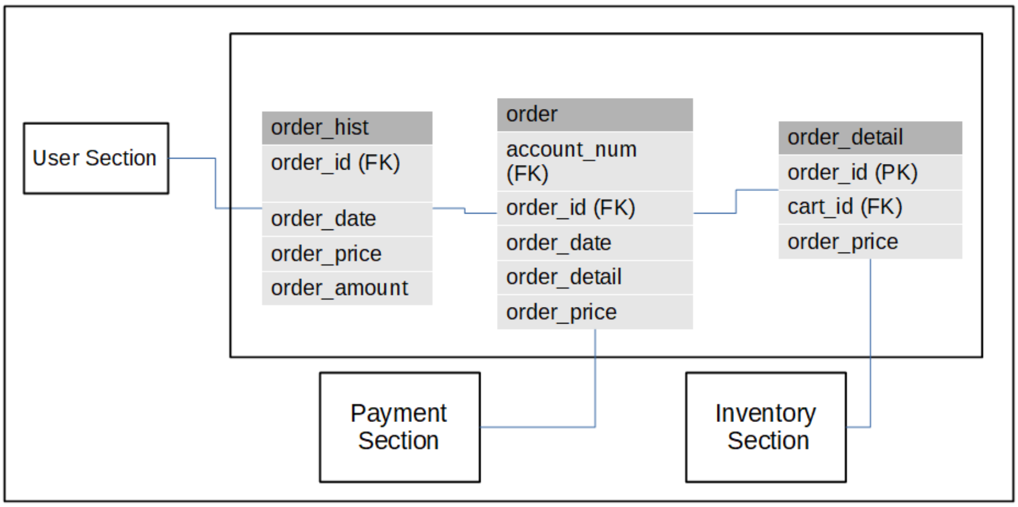


Fig. 3.4 Order Section

The main table would be the order table where it will be linked to the user. Any order made from the inventory will first go to the order table. The order detail is for user to check any order made from the cart in the inventory section.

The order history is for previous order made by the user in the e-commerce application. This will be linked to the user section or block and other parts of the database table.

4) Payment Section

As shown in figure 3.5 the payment section is where a payment is made after passing through the order section. This section consists of three tables, which are credit_card, payment and invoice.

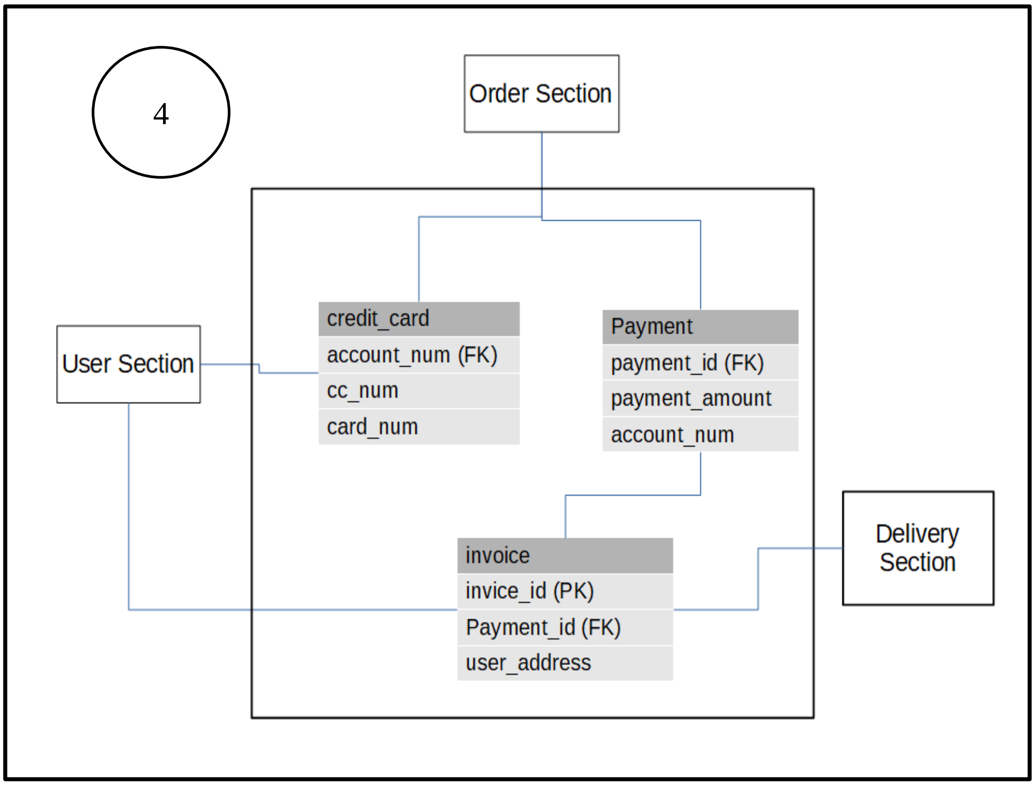


Fig. 3.5 Payment Section

Credit_card table is where the user credit card detail is kept and where the transaction will be done. The payment table is where all payment is recorded for the user and admin and e-commerce application to process. Invoice is done when the payment is made and confirmed, and the invoice will be generated and send to the user with the user_address taken from the user section.

5) Delivery Section

As shown in the figure 3.6, the delivery section consist of two tables, which are delivery and del_status.

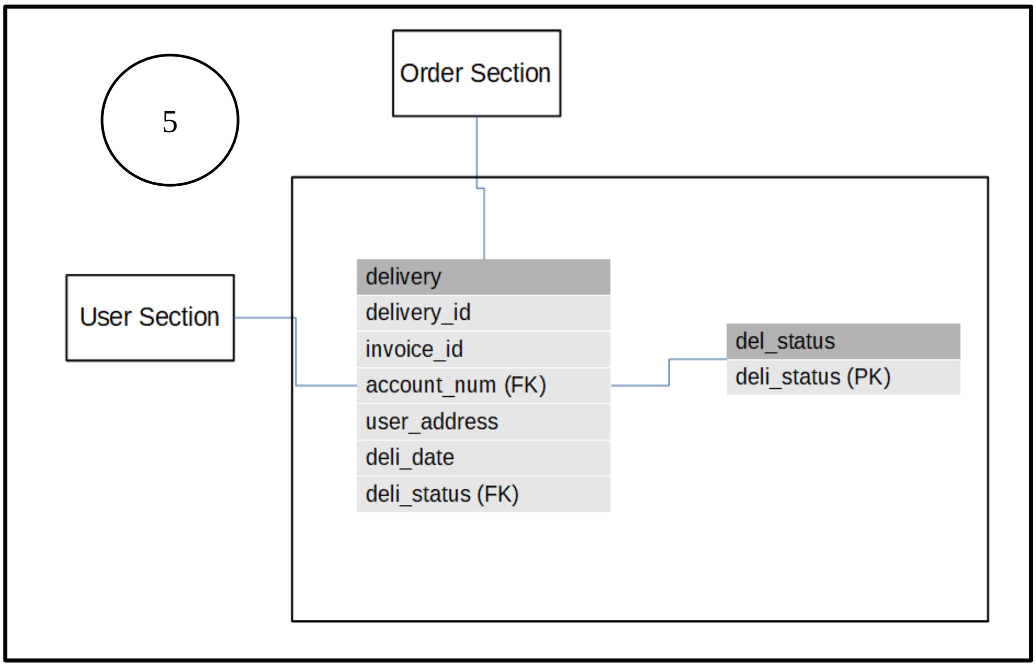


Fig. 3.6 Delivery Section

The delivery table is where any delivery will be recorded and checked before the delivery is done to the user. The del_status is for the admin to set what type of status that the admin wants in the e-commerce application,` for example delivered or in process. The delivery will only be done after payment is made.

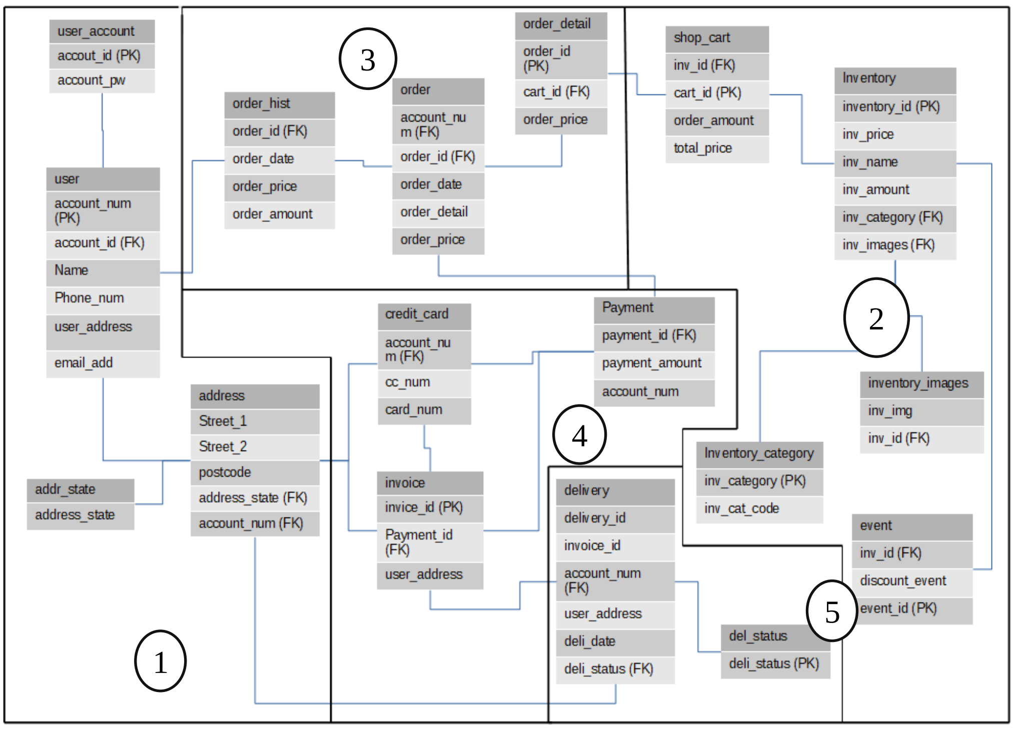


Fig. 3.7 Database Table

As shown in the figure 3.7 the database design is focused around simplicity and connectivity between database tables. The table are divided into a few sections and all the sections have its own tables.

All the tables will be linked or connected to one others by specific stored procedure. The stored procedure will control the data flow from one table to another, and it will decide which data will enter in to a specific table.

3.2 Basic Stored Procedures Design

Stored procedure design can be approach in a simple way. It is a simple way to retrieve the data we need, and these basic designs can be used repeatedly. The design is usually just retrieving from and inserting data into the database.

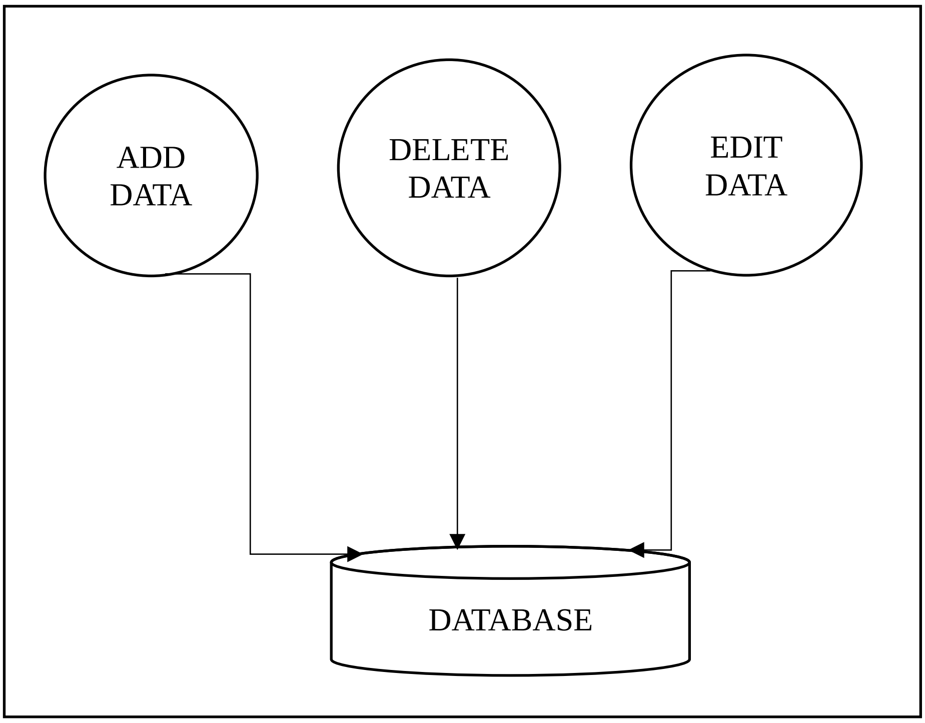


Fig. 3.8 Basic Stored Procedure Design

A basic stored procedure design will normally consist of three parts for each database. Those are add data, edit data and delete data from a database. Stored procedure is designed usually to be very specific to a given action. A basic stored procedure design will usually involve three actions that are required in almost all database.

Some databases or table will have the same stored procedure such as add data, but will be designed specifically for a certain action. Stored procedure is coded in a way that it can be used for any kind of tables depending on the need of the situation. There is no limit for the number of stored procedures that can be stored in a database for a specific table.

```
Create Procedure `AddCategory`(UCName Varchar (50))

BEGIN
Insert into category (CategoryName) Values (UCName);
END
```

As shown in the code above, it is an example of a basic stored procedure design. It is mainly used to update a database where it will insert the data into CategoryName. It can improve the performance in a way that the query can be repeated without typing the code repeatedly.

```
Create Procedure ‘ChangeProductAvailability’ (in CPAID int, in
CPAStatus
varchar (255))
Begin
        Declare AstatusUpdate Varchar (255);
                Set AstatusUpdate = CPAStatus;
                Update Product
                set ProductAvialablity = AstatusUpdate
                Where ProductID = CPAID;
END
```

Another example is a simple stored procedure that can actually be used in terms of updating a data with the given query. Rather than entering a specific data, the stored procedure itself will do the data management and enter the data into the table. It will improve the performance of certain aspect of the DBMS.

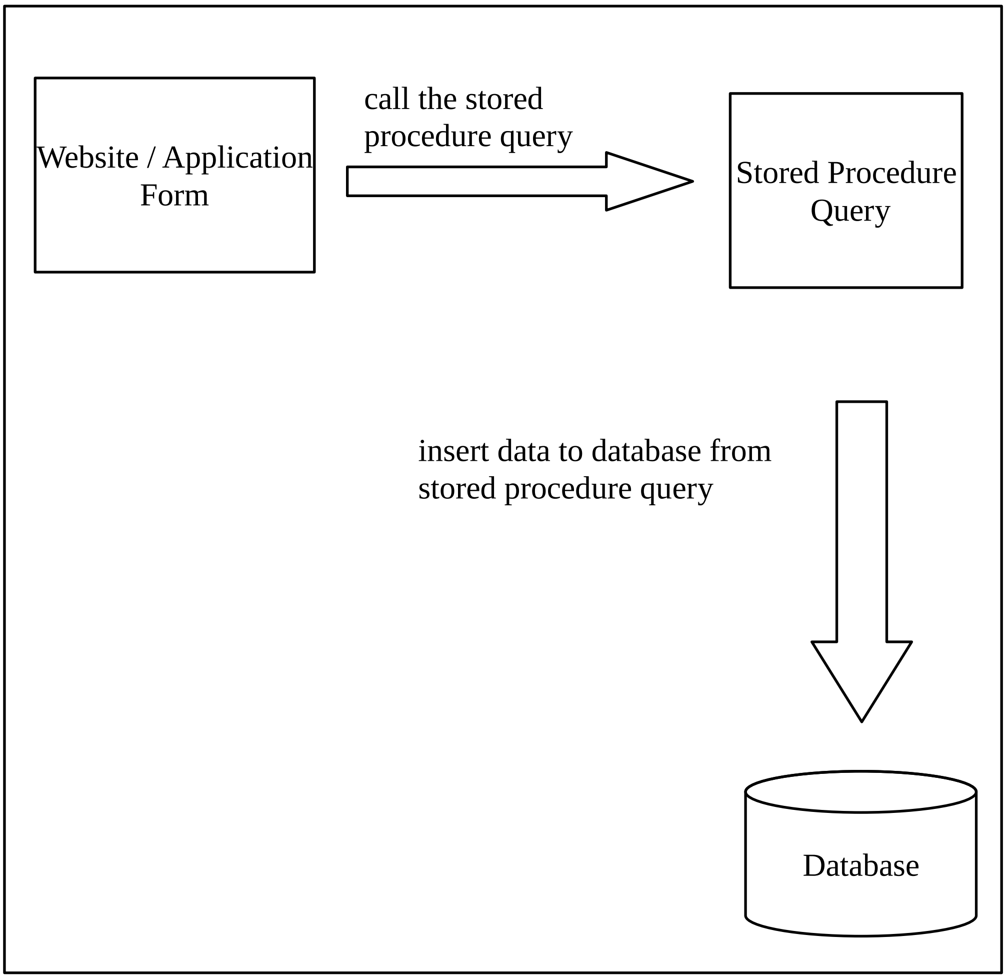

Website / Application Form
call the stored procedure query
Stored Procedure Query
insert data to database from stored procedure query
Database

Fig. 3.9 Stored Procedure

As shown in the figure 3.9 we can see how a simple stored procedure works. The stored procedure query can be used to insert a data into the database. Compared to the normal method where the data is directly inserted into the database the stored procedure will insert the data into the database. The stored procedure will only be activated when needed. This gives an advantage to the programmer where the programmer will only have to apply a specific stored procedure for a specific action. The basic stored procedure will provide more control over the program that is being created by the programmer.

This is the stored procedure for the databases. Most stored procedure would have the same type of add, delete or edit data because, they are basic operations on data. The stored procedures are created for the tables in Fig 3.7 database tables.

1) add_user

The add_user stored procedure is used to register any user to the database. It is a basicstored procedure to add data but it can involve multiple tables which are user_account , user and address.

2) edit_user

The edit_user is to edit any information that is already registered inside the database. This can be used both by the admin and by the user itself.

3) delete_user

The delete_user is to remove any user from the database. This stored procedure can remove specific data that is linked to a specific user.

4) add_addr_state

The add_addr_state is used to add new state into the addr_state table.

5) delete_addr_state

Remove any data from the addr_state table.

6) add_inventory

Add data to the inventory table and add related image. This will mostly be used by the admin of the website.

7) delete_inventory

Delete data from the inventory table.

8) edit_inventory

Edit specific data from the inventory table

9) add_event

Add any event that is related to the inventory. The data will be inserted into the event table.

10) delete_event

Remove data from the event table.

11) add_inv_category

Add data to the inventory_category table.

12) delete_inv_category

Remove data from the inventory_category table

13) add_shop_cart

Add data to shop_cart table

14) delete_shop_cart

Clear all the current data in the shop_cart table.

15) edit_shop_cart

Edit or change the data in the shop_cart table.

16) add_delivery_stats

Add delivery status in the del_status table.

17) delete_delivery_stats

Remove a data from del_status table.

18) delete_delivery

Remove any data from the delivery table.

3.3 Advance Stored Procedure Design

Stored procedure can be used to extensively in a database and the more complicated stored procedure will work better. One stored procedure can be used for multiple databases, and it can act as a bridge between databases. Some of the designed stored procedure are listed below.

1) Updating Order Table Stored Procedure

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateOrderTable`
 (in UOTDate date, in UOTItemID int, in UOTCustID int, in UOTAmount int, in
 UOTOrderStatus varchar (255))
 BEGIN
Declare CustomerAddress varchar (255);
Declare CustomerName Varchar (255);
Declare CustomerPhone int (50);
Declare ItemName varchar (255);
Declare ItemPrice int (50);
Declare ItemAmount int (50);
Set CustomerAddress = (Select CustAddress from Customer Where CustID = UOTCustID);
Set CustomerName = (Select CustName from Customer Where CustID = UOTCustID);
Set CustomerPhone = (Select CustPhone from Customer Where CustID = UOTCustID);
Set ItemName = (Select ProductName from Product where ProductID = UOTItemID);
Set ItemPrice = (Select ProductPrice from Product where ProductID = UOTItemID);
Set ItemAmount = (Select ProductAmount from Product where ProductID = UOTItemID);
 update Product set ProductAmount = ProductAmount - UOTAmount where ProductID =
 UOTItemID;
 Insert into CustOrder
 (OrderDate,OrderAddress,OrderItemName,OrderItemAmount,OrderItemPrice,OrderCustNa
 me,OrderCustPhone,OrderStatus)
values
 (
 UOTDate, CustomerAddress, ItemName,  UOTAmount, ItemPrice * UOTAmount,
 CustomerName, CustomerPhone, UOTOrderStatus
 );
  END
```

Fig. 3.10 UpdateOrderTable Codes

Fig 3.10 shows an example of a created stored procedure. This stored procedure retrieve data from the application and replace it with the existing data in the database.

2) Purchasing_item

This is an advance stored procedure that is implemented when a user purchases an item from the e-commerce application. It will deduct an inventory value from the inventory table, and take the price amount of purchased item from the order table. The item detail will also be taken from the order table and will be recorded inside the invoice table. With this stored procedure, we can see a few different transactions are involved between tables.

3) Delivery

The delivery stored procedure is to transfer data between the invoice, the user address and the status of the delivery. When a payment is made and an invoice is created the delivery, stored procedure will record the data and transfer data from the address table to the delivery table.

4) Updating_order

The updating_order stored procedure is used to mainly update any order. Any order made previously will be updated from the order table to the order history table, and the information will be taken from order detail. Rather than making three different stored procedure for one same query, one stored procedure is made to ensure all the transactions are done at the same time to avoid any mistakes.

Chapter 4 : Implementation and Experiment

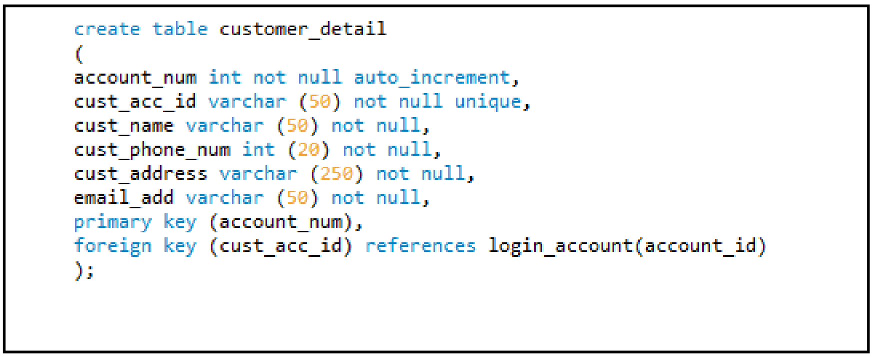
4.1 Database Implementation

The system used for the implementation is Windows 10. The processors is Intel (R) Core (TM) i7-7700HQ CPU @2.80GHz with a RAM of 8GB. The system is running on a 64-bit Operating system.

Two applications are involved for the implementation. For the database implementation, MySQL Workbench 8.0 CE is used. Visual Studio 2017 is used for the application design.

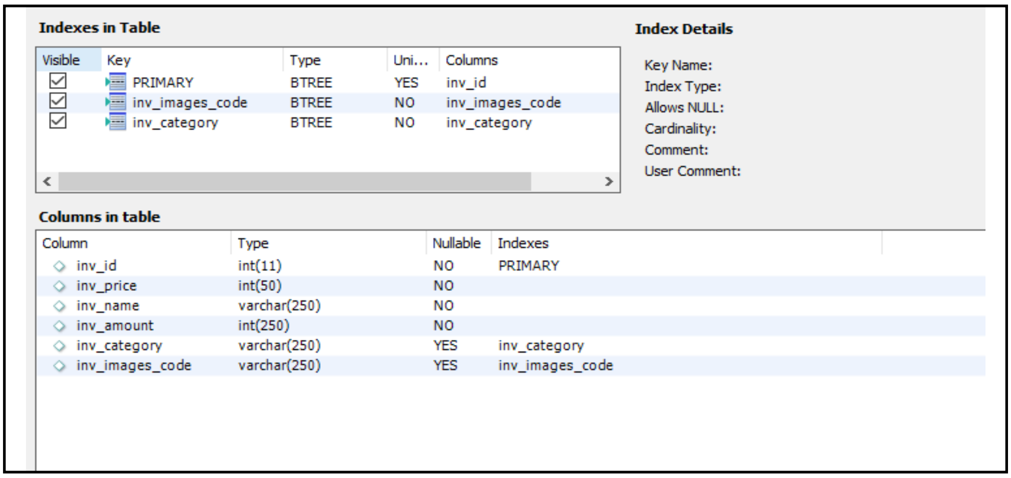
1) Customer Detail Table

As we can see from the figure 4.1, there is a few different SQL involved in declaring the data type. The customer_detail table is connected directly to login_account table where customer will use to login to the application or website.



```
create table customer_detail
(
account_num int not null auto_increment,
cust_acc_id varchar (50) not null unique,
cust_name varchar (50) not null,
cust_phone_num int (20) not null,
cust_address varchar (250) not null,
email_add varchar (50) not null,
primary key (account_num),
foreign key (cust_acc_id) references login_account(account_id)
);
```

Fig. 4.1 SQL for customer_detail



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	inv_id
✓	inv_images_code	BTREE	NO	inv_images_code
✓	inv_category	BTREE	NO	inv_category

Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

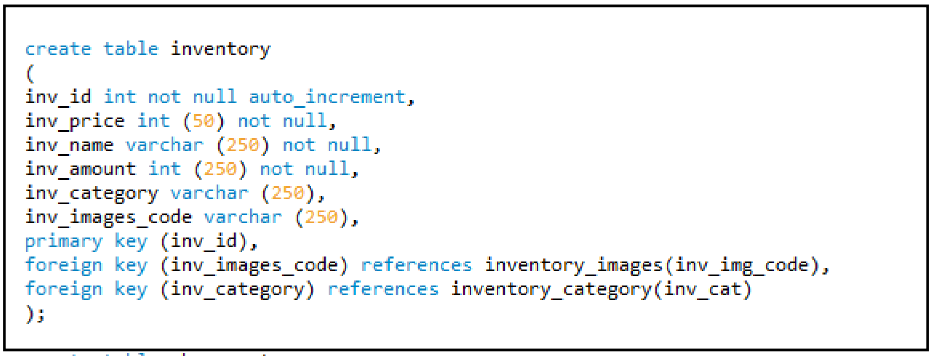
Column	Type	Nullable	Indexes
inv_id	int(11)	NO	PRIMARY
inv_price	int(50)	NO	
inv_name	varchar(250)	NO	
inv_amount	int(250)	NO	
inv_category	varchar(250)	YES	inv_category
inv_images_code	varchar(250)	YES	inv_images_code

Fig. 4.2 Table for customer_detail

Inventory_images table uses a different SQL, it is directly link to the Inventory table but it is a different entity that uses to store the Images for the inventory. For the image, we use the medium blog query, it is used in SQL to keep image data in the database.

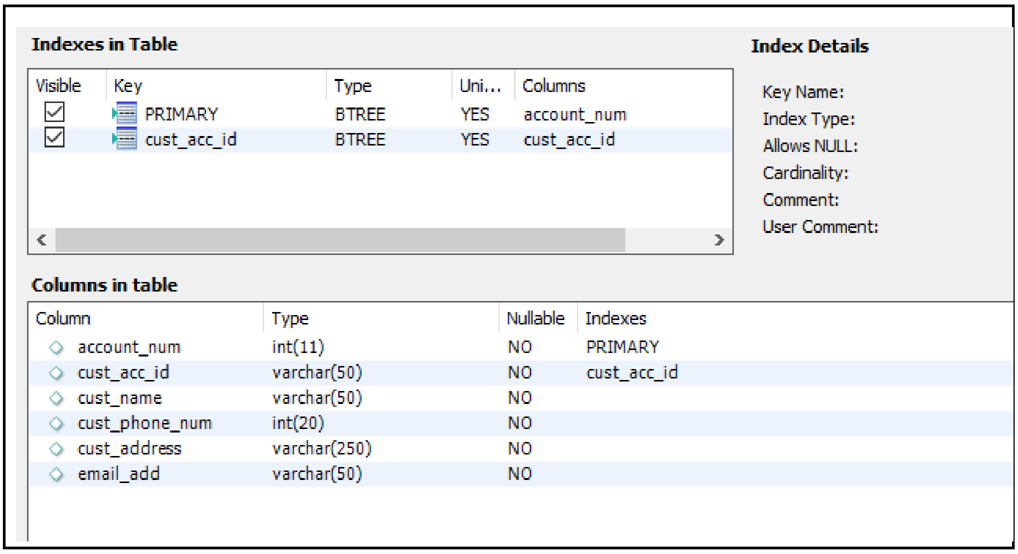
Fig 4.3 and 4.4 shows the SQL and the table involve in the inventory table. Fig 4.5 and 4.6 shows the SQL and table for inventory_images. These two tables are related to one another but is created differently to provide more control for the admin.

2) Inventory Table



```
create table inventory
(
inv_id int not null auto_increment,
inv_price int (50) not null,
inv_name varchar (250) not null,
inv_amount int (250) not null,
inv_category varchar (250),
inv_images_code varchar (250),
primary key (inv_id),
foreign key (inv_images_code) references inventory_images(inv_img_code),
foreign key (inv_category) references inventory_category(inv_cat)
);
```

Fig. 4.3 SQL for inventory



Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	PRIMARY	BTREE	YES	account_num
☑	cust_acc_id	BTREE	YES	cust_acc_id

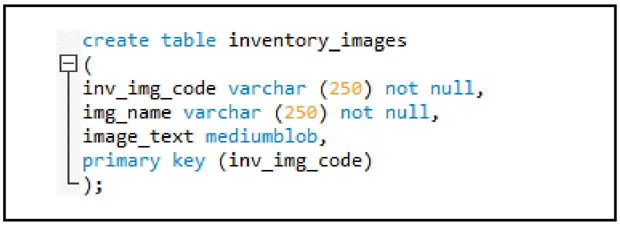
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

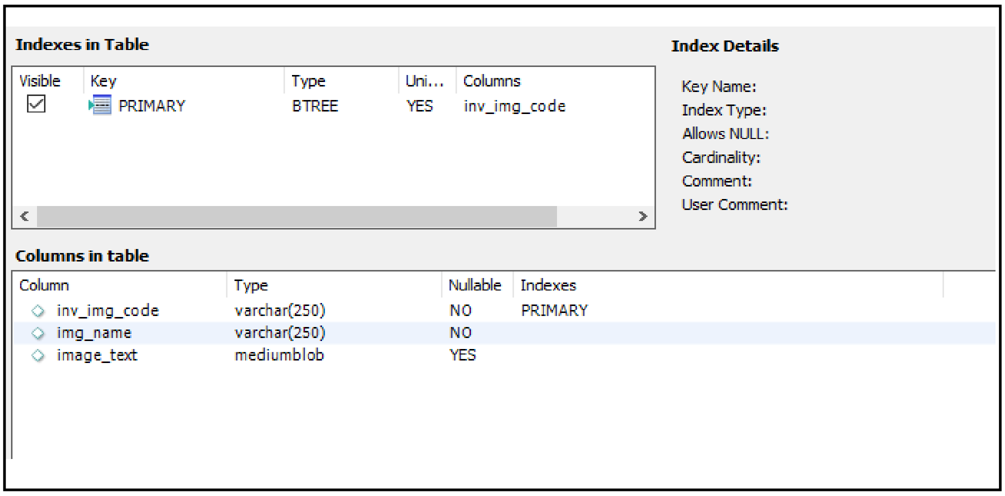
Column	Type	Nullable	Indexes
account_num	int(11)	NO	PRIMARY
cust_acc_id	varchar(50)	NO	cust_acc_id
cust_name	varchar(50)	NO	
cust_phone_num	int(20)	NO	
cust_address	varchar(250)	NO	
email_add	varchar(50)	NO	

Fig. 4.4 Table for inventory



```
create table inventory_images
(
inv_img_code varchar (250) not null,
img_name varchar (250) not null,
image_text mediumblob,
primary key (inv_img_code)
);
```

Fig. 4.5 SQL for inventory_images



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	inv_img_code

Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

Column	Type	Nullable	Indexes
inv_img_code	varchar(250)	NO	PRIMARY
img_name	varchar(250)	NO	
image_text	mediumblob	YES	

Fig 4.6 Table for inventory_images

4.2 Stored Procedure Implementation

Stored procedure are implemented differently. From the basic stored procedures to the more advance stored procedures. We will see the basic stored procedures and then the advance design stored procedure.

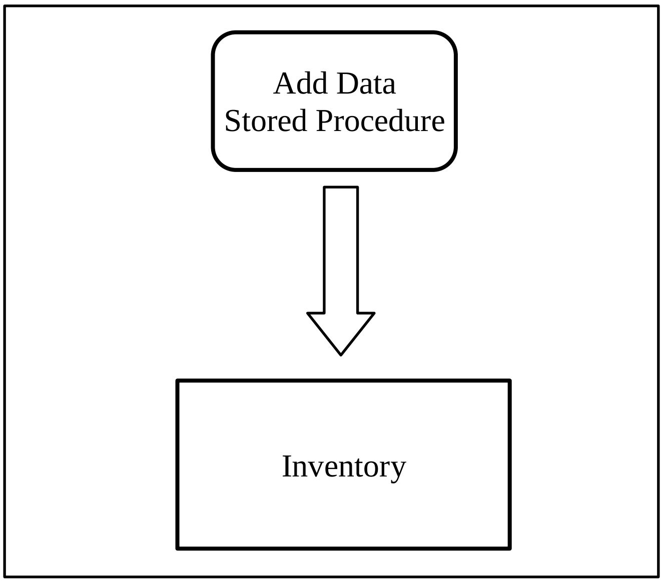
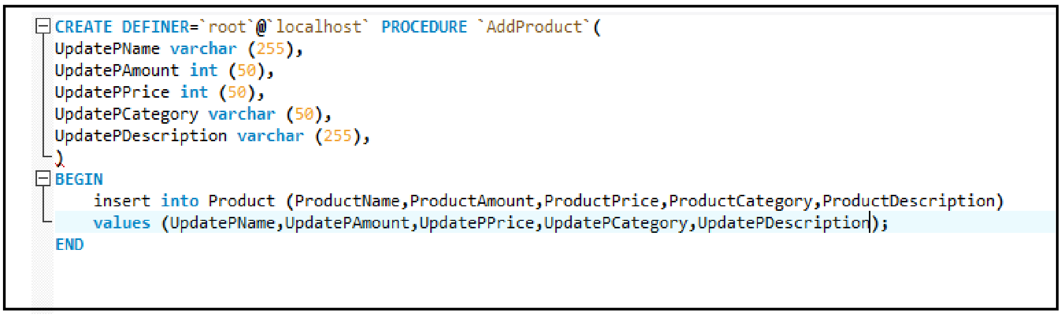


Fig. 4.7 Stored Procedure for Add Data

Basic stored procedure as shown in figure 4.7 uses simple query. This stored procedure is more direct and created specifically for the task on storing data into the inventory table.



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddProduct`(
UpdatePName varchar (255),
UpdatePAmount int (50),
UpdatePPrice int (50),
UpdatePCategory varchar (50),
UpdatePDescription varchar (255),
)
BEGIN
    insert into Product (ProductName,ProductAmount,ProductPrice,ProductCategory,ProductDescription)
    values (UpdatePName,UpdatePAmount,UpdatePPrice,UpdatePCategory,UpdatePDescription);
END
```

Fig. 4.8 add_inventory Stored Procedure

As shown in figure 4.8 the stored procedure will retrieve the data and insert it into the given database in the query, where in this case is the inventory table. This stored procedure is one way and usually only involve one SQL statement and one table.

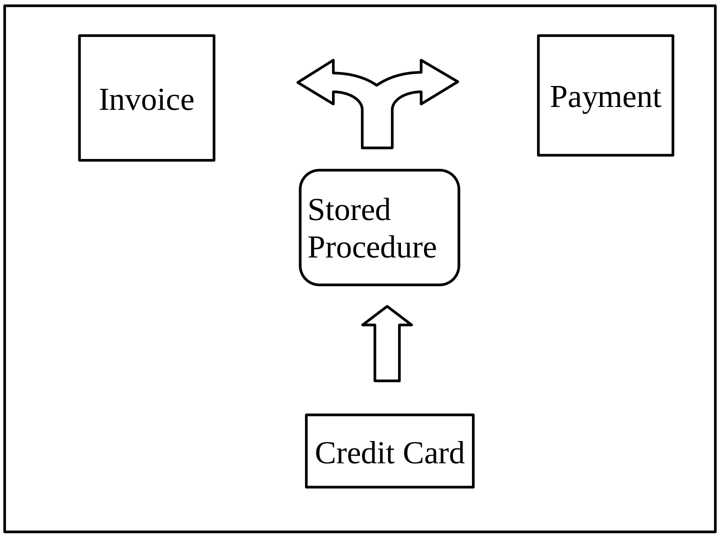
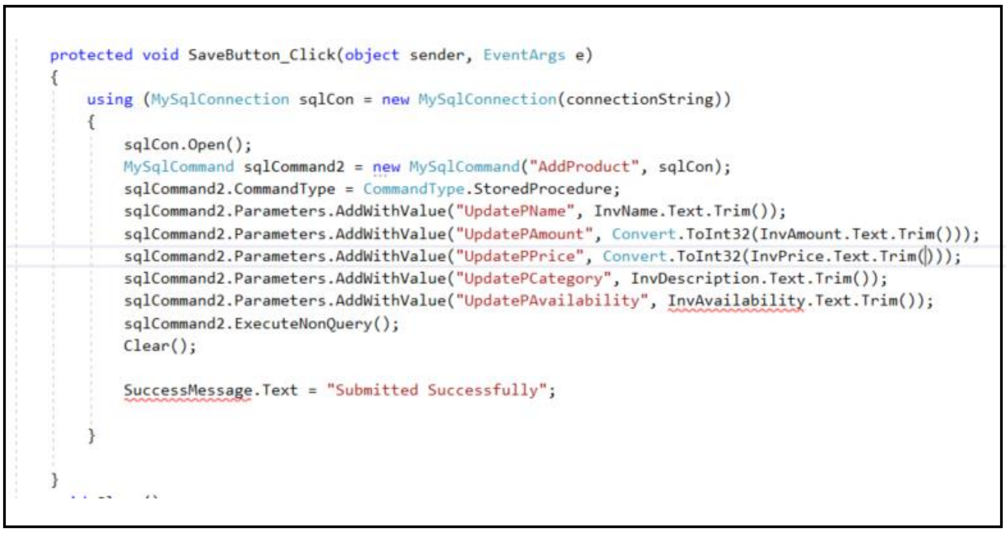


Fig. 4.9 Stored Procedure for Credit Card

As shown in the figure 4.9, this stored procedure is created to connect three different tables with just one transaction. What it will do is, when this is called it will update the table invoice and payment based on the credit card used to pay for the transaction.

Stored procedure may be a trigger from the main application to the database program. As shown in the figure 4.10, a trigger is inserted in the codes for the application. A connection needs to be set up with the database before the stored procedure can be called and triggered. When the connection is set up the stored procedure will be declared, and the command type will be declared. This trigger will be inserted in the respective button to where the admin or programmer wants it to be called.



```
protected void SaveButton_Click(object sender, EventArgs e)
{
    using (MySqlConnection sqlCon = new MySqlConnection(connectionString))
    {
        sqlCon.Open();
        MySqlCommand sqlCommand2 = new MySqlCommand("AddProduct", sqlCon);
        sqlCommand2.CommandType = CommandType.StoredProcedure;
        sqlCommand2.Parameters.AddWithValue("UpdatePName", InvName.Text.Trim());
        sqlCommand2.Parameters.AddWithValue("UpdatePAmount", Convert.ToInt32(InvAmount.Text.Trim()));
        sqlCommand2.Parameters.AddWithValue("UpdatePPrice", Convert.ToInt32(InvPrice.Text.Trim()));
        sqlCommand2.Parameters.AddWithValue("UpdatePCategory", InvDescription.Text.Trim());
        sqlCommand2.Parameters.AddWithValue("UpdatePAvailability", InvAvailability.Text.Trim());
        sqlCommand2.ExecuteNonQuery();
        Clear();

        SuccessMessage.Text = "Submitted Successfully";

    }

}
```

Fig. 4.10 Stored Procedure Trigger

4.3 Website Implementation

The website was implemented in two sites that are the user site and the admin site. The two sites have a few major differences, for example, the user site is more focused on the ease of experience and simplicity. It gives the user a more basic but necessary control over what they want to view and what they want to search.

The admin site focuses on more of detailed list and structures. There are less images involved to save space and to ensure unimportant information is involved. The admin of the site will have control on most information displayed at the website.

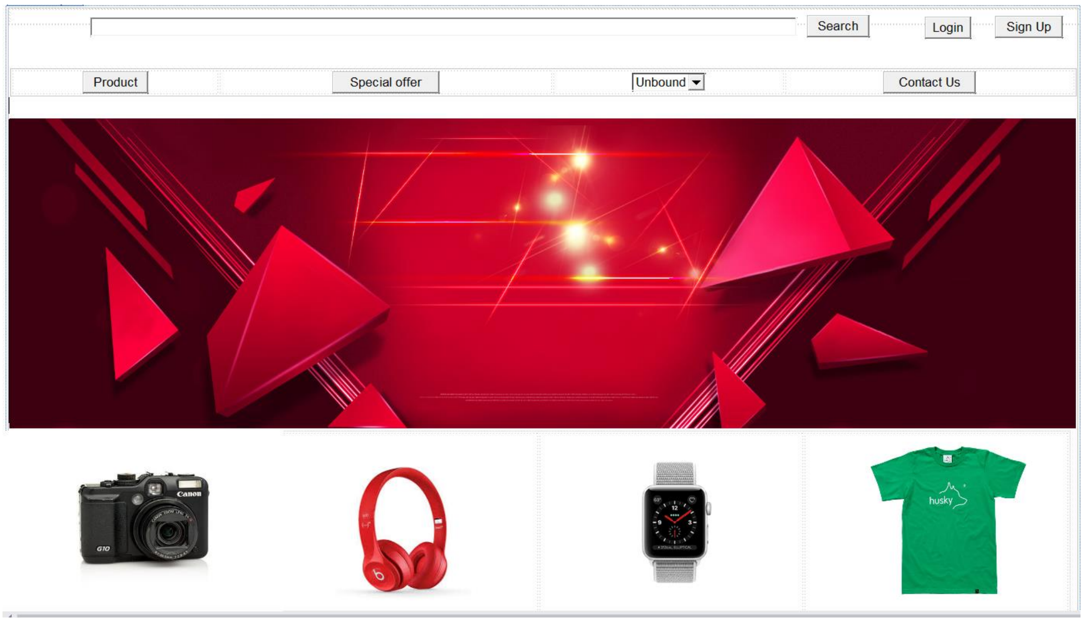
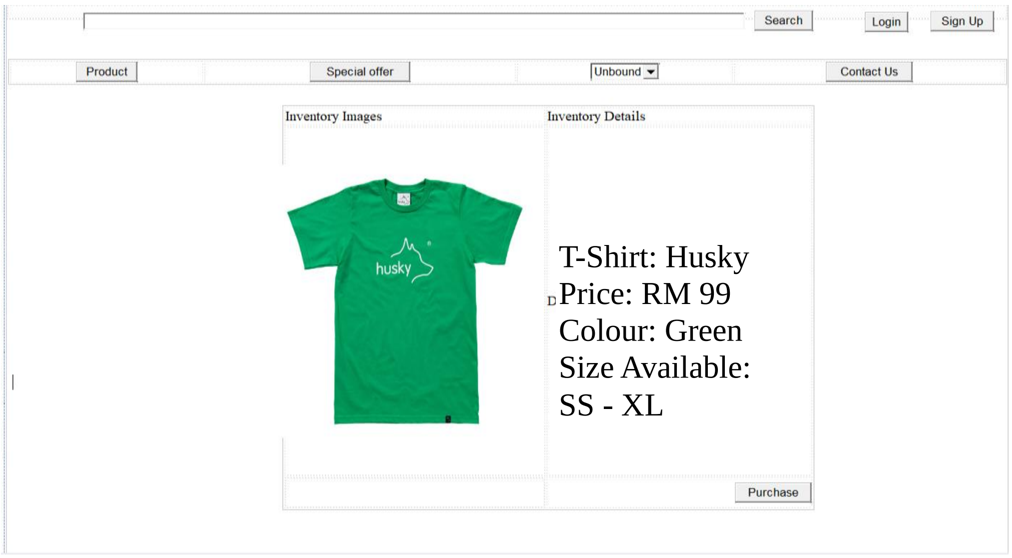


Fig. 4.11 Main Page.

The figure 4.11 shows the website main page, as shown there is a search bar to help the user search on specific item on the site. The site does not require user to login to use it but it is a choice for the user itself.

A few different stored procedures are used in the main page, for example at the search bar a stored procedure is used. The stored procedure retrieve the data from the search bar and implement a search based on the requirement. It will then display the data in a new page with the format that is implemented by the admin. In this instance, at least 1 stored procedure is implemented in each button and the display of banner, and offered items will also be involved with stored procedure.

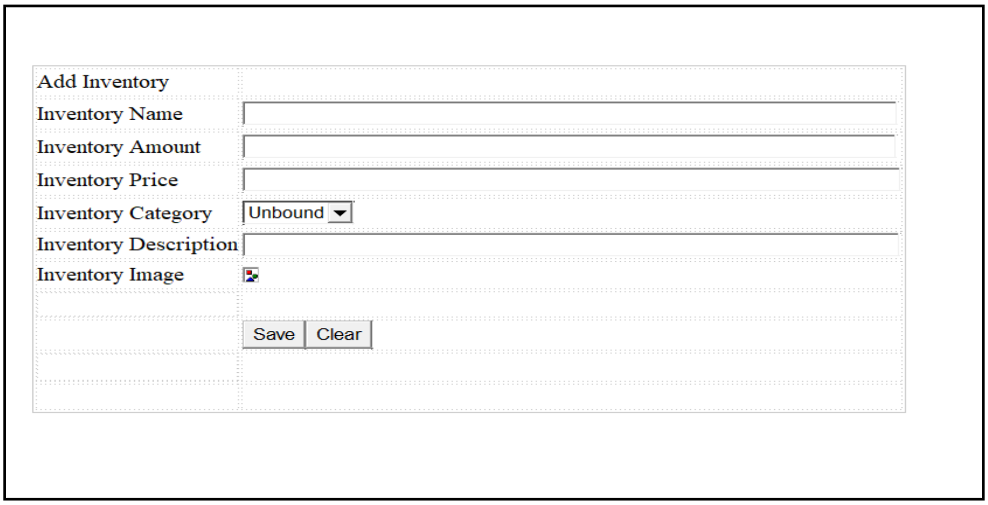


T-Shirt: Husky
Price: RM 99
Colour: Green
Size Available:
SS - XL

Fig. 4.12 Buying a Product

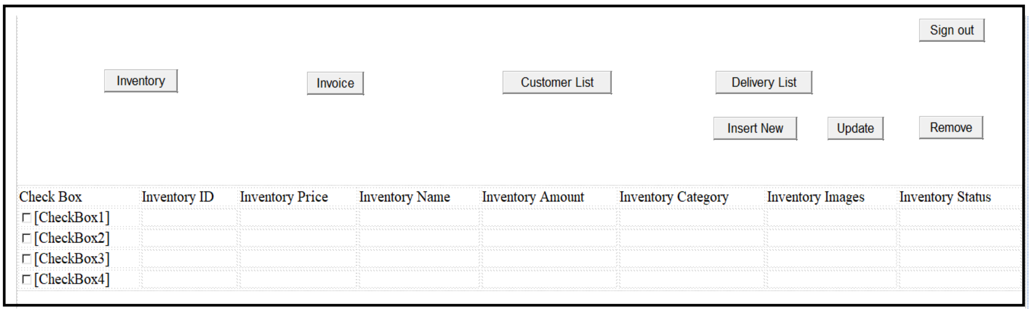
As shown in the figure 4.12 the base design of the website will stay the same. The image will be taken from the inv_image table that is linked to the inventory itself. The content will be changed depending on the needs or the customer choices. After the customer confirmed on a product, it will be inserted in a shopping cart where the details will be hold and collected until the purchased is confirmed.

Figure 4.13 and figure 4.14 above shows how the admin pages are implemented. The admin pages are designed for admin to edit or remove any data from the database with ease. This give admin control on what can be viewed. This implementation .also helps to provide admin a simple way to insert any data for the website.



Add Inventory	
Inventory Name	
Inventory Amount	
Inventory Price	
Inventory Category	Unbound
Inventory Description	
Inventory Image	

Fig. 4.13 Adding Inventory (Admin)



Check Box	Inventory ID	Inventory Price	Inventory Name	Inventory Amount	Inventory Category	Inventory Images	Inventory Status
☐ [CheckBox1]							
☐ [CheckBox2]							
☐ [CheckBox3]							
☐ [CheckBox4]							

Fig. 4.14 Admin Main Page

Chapter 5: Conclusion and Further Works

As a conclusion, a few factors can be recorded in the design of database and stored procedure for e-commerce. The research uses a basic design for e-commerce application that aims to help customer and admin in terms of maintaining or browsing the application.

Stored Procedure can improve a design or flow for any databases. It provides a means of control for the database designer or web programmer in terms of creating the database itself. Stored procedure can be designed as very specific query or broader query depending on the needs of the database. This gives a sense of control over what is needed for the application, and only creating a query that will be used for the application.

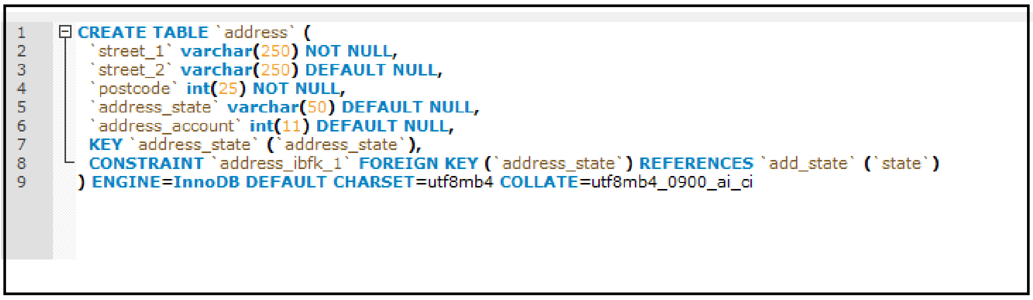
By using stored procedure in any e-commerce application, the query for a specific action can be reused without creating a new query. Some triggers in stored procedure can be used for different situation, and they provide a sense of ease for the design.

Stored procedure query has specific uses, that for each action we need a specific stored procedure. There may be some stored procedures that are usable in multiple actions but it is better to create a stored procedure for each given action.

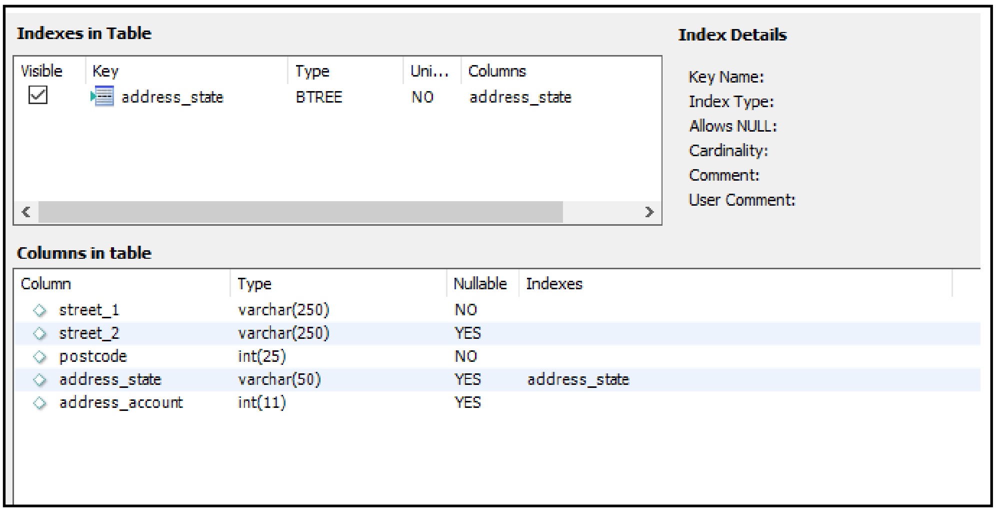
The amount of stored procedure needed for an e-commerce application using stored procedure will increase the more complicated the e-commerce application is. Research can be done to improve the limitation of using stored procedure.

APPENDIX A : Database SQL and Tables

1) address Table



```
CREATE TABLE `address` (
  `street_1` varchar(250) NOT NULL,
  `street_2` varchar(250) DEFAULT NULL,
  `postcode` int(25) NOT NULL,
  `address_state` varchar(50) DEFAULT NULL,
  `address_account` int(11) DEFAULT NULL,
  KEY `address_state` (`address_state`),
  CONSTRAINT `address_ibfk_1` FOREIGN KEY (`address_state`) REFERENCES `add_state` (`state`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	address_state	BTREE	NO	address_state

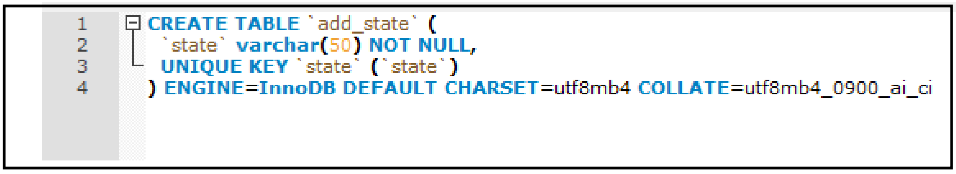
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

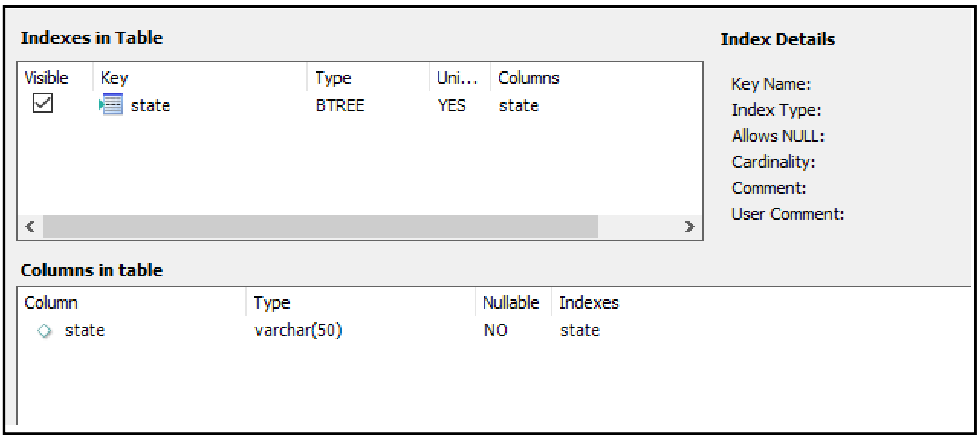
Columns in table

Column	Type	Nullable	Indexes
street_1	varchar(250)	NO	
street_2	varchar(250)	YES	
postcode	int(25)	NO	
address_state	varchar(50)	YES	address_state
address_account	int(11)	YES	

2) addr_state table



```
CREATE TABLE `add_state` (
  `state` varchar(50) NOT NULL,
  UNIQUE KEY `state` (`state`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	state	BTREE	YES	state

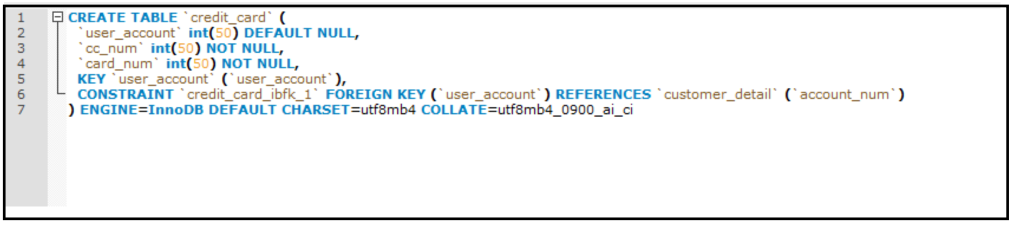
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

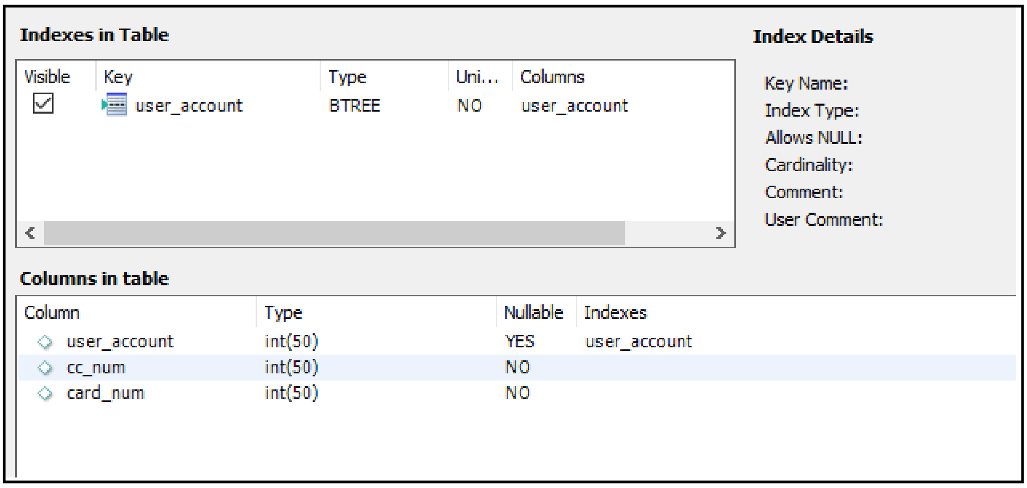
Columns in table

Column	Type	Nullable	Indexes
state	varchar(50)	NO	state

3) credit_card tables



```
CREATE TABLE `credit_card` (
  `user_account` int(50) DEFAULT NULL,
  `cc_num` int(50) NOT NULL,
  `card_num` int(50) NOT NULL,
  KEY `user_account` (`user_account`),
  CONSTRAINT `credit_card_ibfk_1` FOREIGN KEY (`user_account`) REFERENCES `customer_detail` (`account_num`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```


Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	user_account	BTREE	NO	user_account

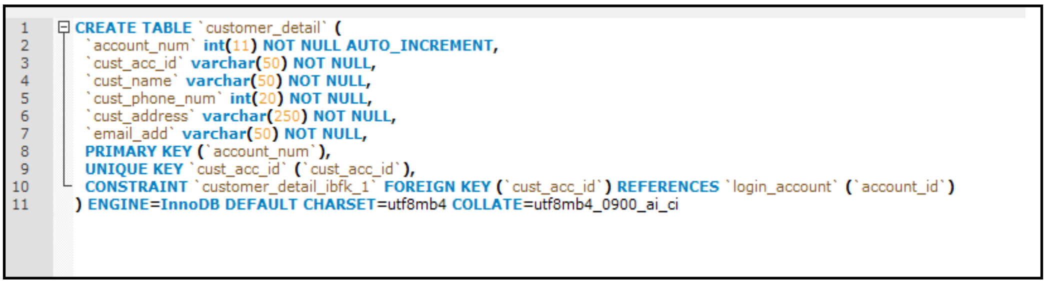
Index Details

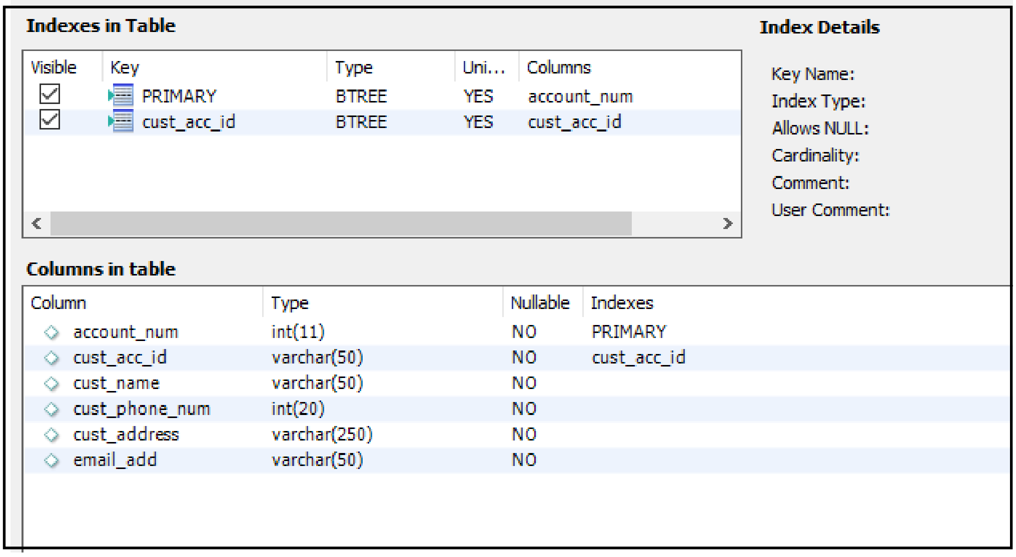
Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

Column	Type	Nullable	Indexes
user_account	int(50)	YES	user_account
cc_num	int(50)	NO	
card_num	int(50)	NO	

4) user tables


```
CREATE TABLE `customer_detail` (
  `account_num` int(11) NOT NULL AUTO_INCREMENT,
  `cust_acc_id` varchar(50) NOT NULL,
  `cust_name` varchar(50) NOT NULL,
  `cust_phone_num` int(20) NOT NULL,
  `cust_address` varchar(250) NOT NULL,
  `email_add` varchar(50) NOT NULL,
  PRIMARY KEY (`account_num`),
  UNIQUE KEY `cust_acc_id` (`cust_acc_id`),
  CONSTRAINT `customer_detail_ibfk_1` FOREIGN KEY (`cust_acc_id`) REFERENCES `login_account` (`account_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```


Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	account_num
✓	cust_acc_id	BTREE	YES	cust_acc_id

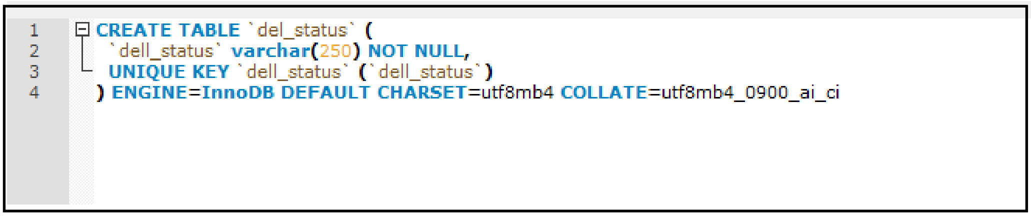
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

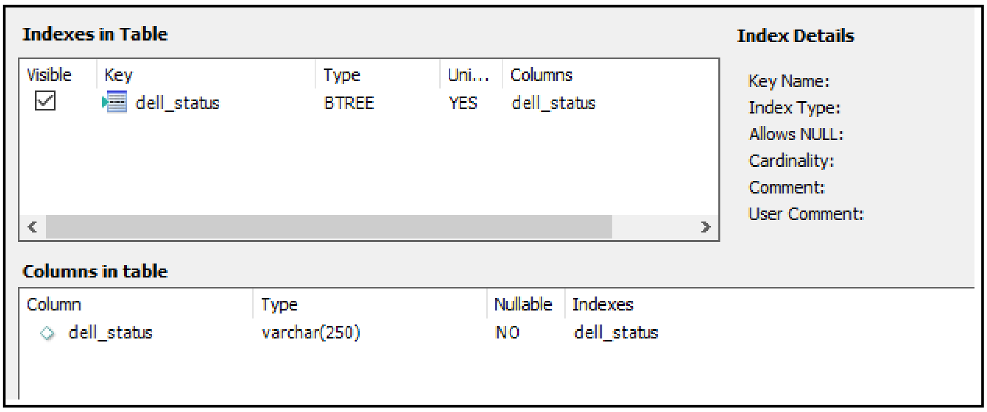
Columns in table

Column	Type	Nullable	Indexes
account_num	int(11)	NO	PRIMARY
cust_acc_id	varchar(50)	NO	cust_acc_id
cust_name	varchar(50)	NO	
cust_phone_num	int(20)	NO	
cust_address	varchar(250)	NO	
email_add	varchar(50)	NO	

5) del_status tables



```
CREATE TABLE `del_status` (
  `dell_status` varchar(250) NOT NULL,
  UNIQUE KEY `dell_status` (`dell_status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	dell_status	BTREE	YES	dell_status

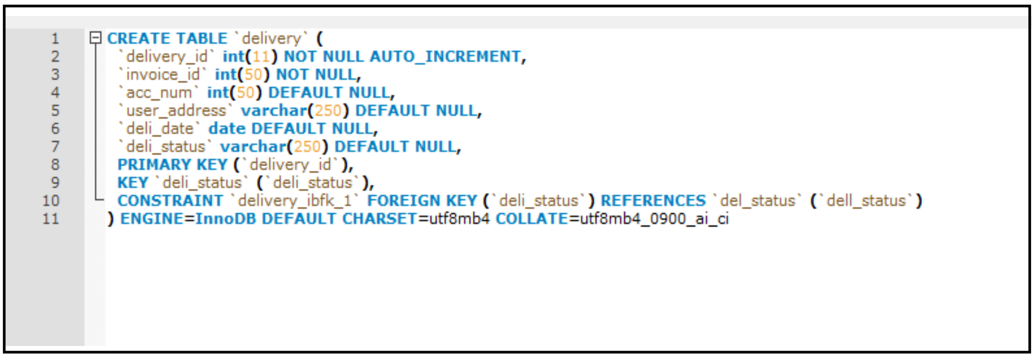
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

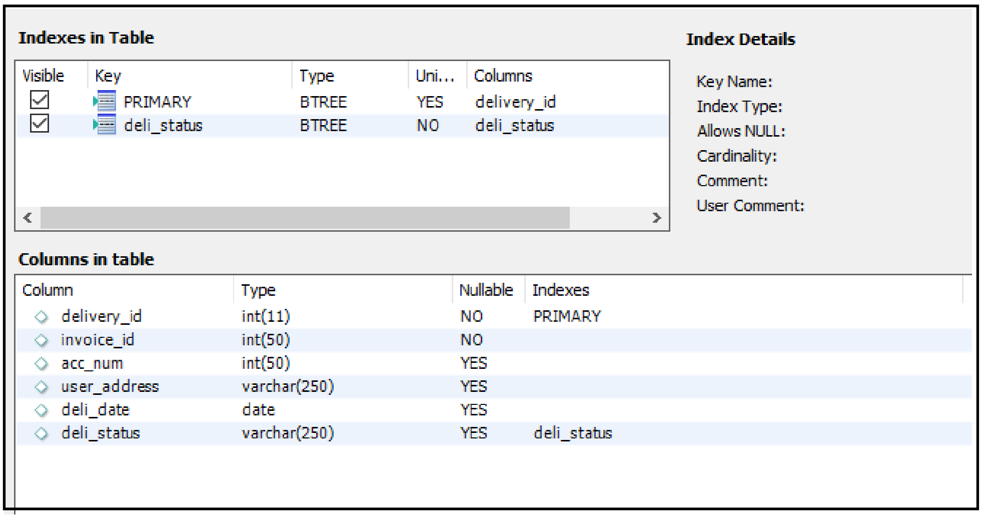
Columns in table

Column	Type	Nullable	Indexes
dell_status	varchar(250)	NO	dell_status

6) delivery tables



```
CREATE TABLE `delivery` (
  `delivery_id` int(11) NOT NULL AUTO_INCREMENT,
  `invoice_id` int(50) NOT NULL,
  `acc_num` int(50) DEFAULT NULL,
  `user_address` varchar(250) DEFAULT NULL,
  `deli_date` date DEFAULT NULL,
  `deli_status` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`delivery_id`),
  KEY `deli_status` (`deli_status`),
  CONSTRAINT `delivery_ibfk_1` FOREIGN KEY (`deli_status`) REFERENCES `del_status` (`dell_status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```


Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	delivery_id
✓	deli_status	BTREE	NO	deli_status

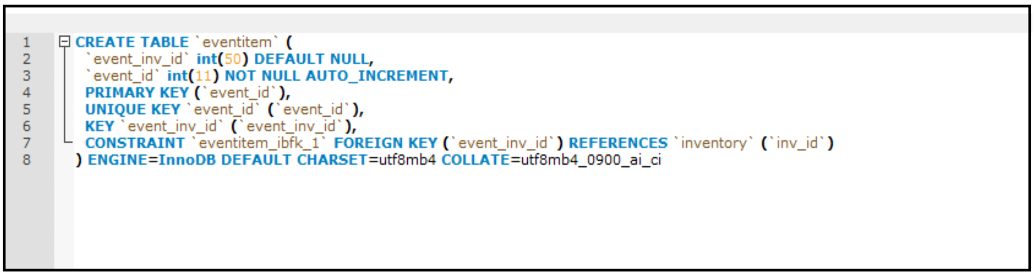
Index Details

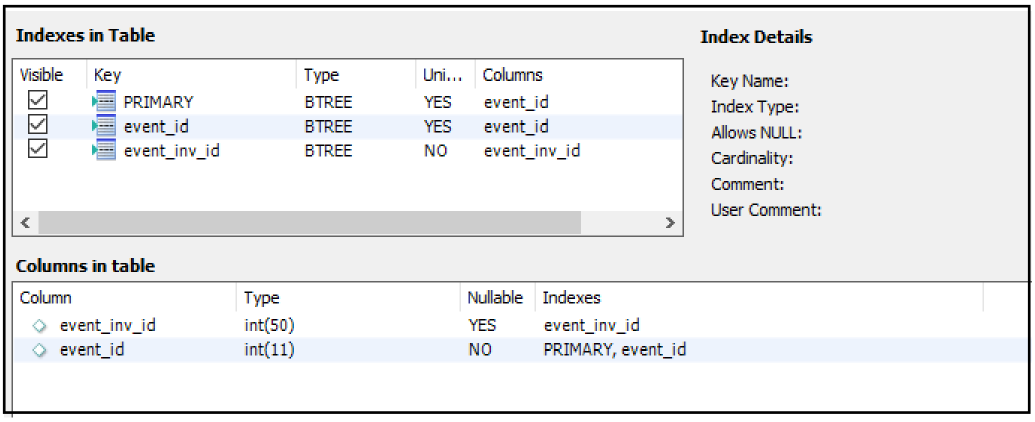
Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

Column	Type	Nullable	Indexes
delivery_id	int(11)	NO	PRIMARY
invoice_id	int(50)	NO	
acc_num	int(50)	YES	
user_address	varchar(250)	YES	
deli_date	date	YES	
deli_status	varchar(250)	YES	deli_status

7) eventitem tables


```
CREATE TABLE `eventitem` (
  `event_inv_id` int(50) DEFAULT NULL,
  `event_id` int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`event_id`),
  UNIQUE KEY `event_id` (`event_id`),
  KEY `event_inv_id` (`event_inv_id`),
  CONSTRAINT `eventitem_ibfk_1` FOREIGN KEY (`event_inv_id`) REFERENCES `inventory` (`inv_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```


Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	event_id
✓	event_id	BTREE	YES	event_id
✓	event_inv_id	BTREE	NO	event_inv_id

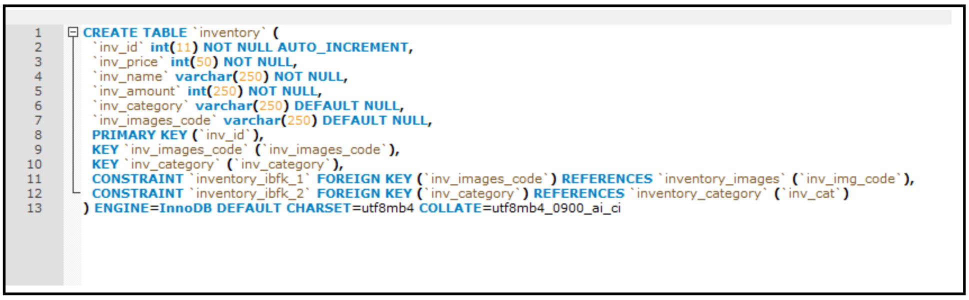
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

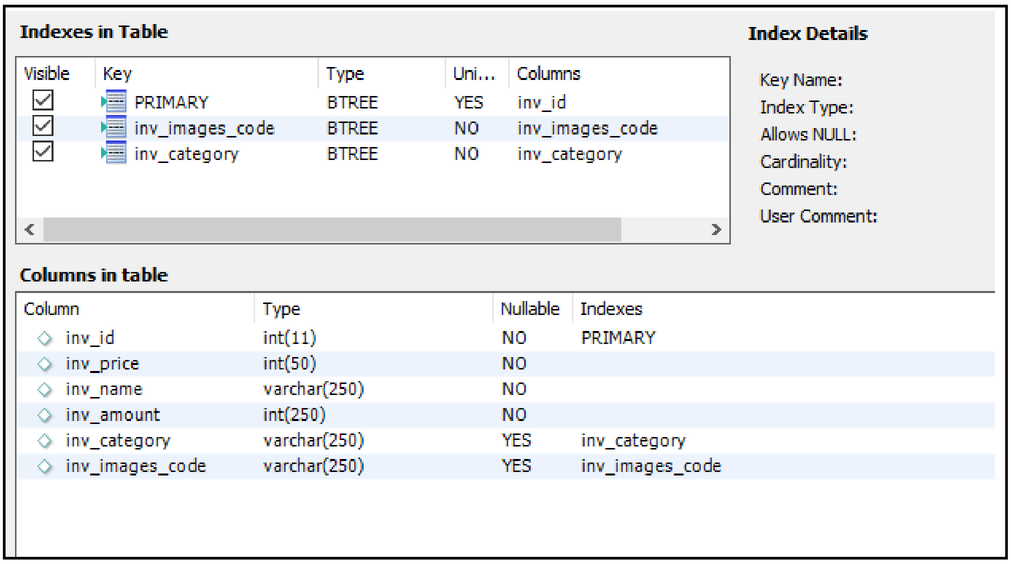
Columns in table

Column	Type	Nullable	Indexes
event_inv_id	int(50)	YES	event_inv_id
event_id	int(11)	NO	PRIMARY, event_id

8) inventory tables



```sql
CREATE TABLE `inventory` (
  `inv_id` int(11) NOT NULL AUTO_INCREMENT,
  `inv_price` int(50) NOT NULL,
  `inv_name` varchar(250) NOT NULL,
  `inv_amount` int(250) NOT NULL,
  `inv_category` varchar(250) DEFAULT NULL,
  `inv_images_code` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`inv_id`),
  KEY `inv_images_code` (`inv_images_code`),
  KEY `inv_category` (`inv_category`),
  CONSTRAINT `inventory_ibfk_1` FOREIGN KEY (`inv_images_code`) REFERENCES `inventory_images` (`inv_img_code`),
  CONSTRAINT `inventory_ibfk_2` FOREIGN KEY (`inv_category`) REFERENCES `inventory_category` (`inv_cat`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	inv_id
✓	inv_images_code	BTREE	NO	inv_images_code
✓	inv_category	BTREE	NO	inv_category

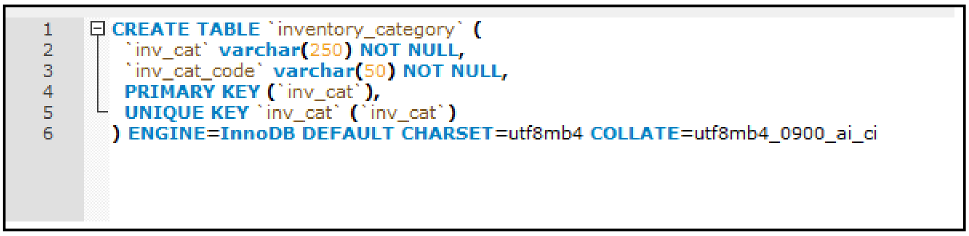
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

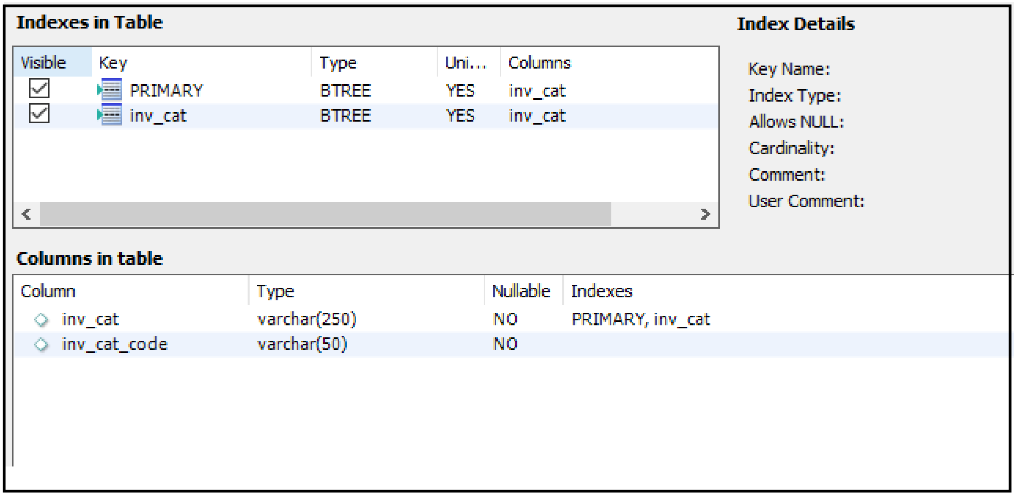
Columns in table

Column	Type	Nullable	Indexes
inv_id	int(11)	NO	PRIMARY
inv_price	int(50)	NO	
inv_name	varchar(250)	NO	
inv_amount	int(250)	NO	
inv_category	varchar(250)	YES	inv_category
inv_images_code	varchar(250)	YES	inv_images_code

9) inventory category tables



```sql
CREATE TABLE `inventory_category` (
  `inv_cat` varchar(250) NOT NULL,
  `inv_cat_code` varchar(50) NOT NULL,
  PRIMARY KEY (`inv_cat`),
  UNIQUE KEY `inv_cat` (`inv_cat`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	inv_cat
✓	inv_cat	BTREE	YES	inv_cat

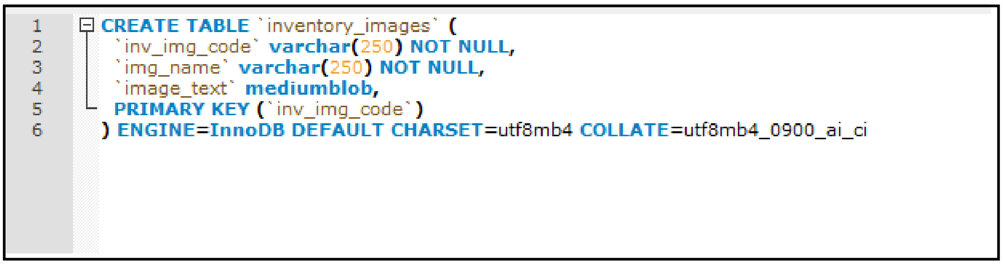
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

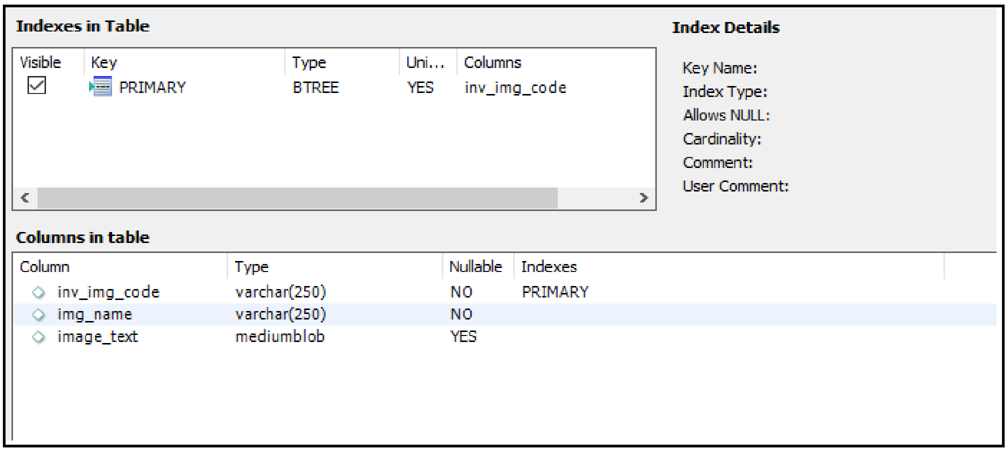
Columns in table

Column	Type	Nullable	Indexes
inv_cat	varchar(250)	NO	PRIMARY, inv_cat
inv_cat_code	varchar(50)	NO	

10) inventory_images tables



```
CREATE TABLE `inventory_images` (
  `inv_img_code` varchar(250) NOT NULL,
  `img_name` varchar(250) NOT NULL,
  `image_text` mediumblob,
  PRIMARY KEY (`inv_img_code`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	inv_img_code

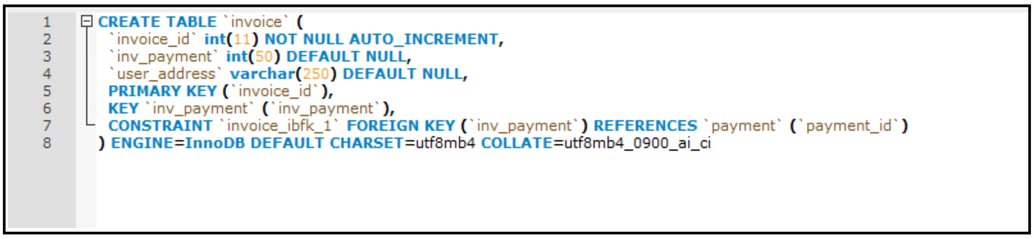
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

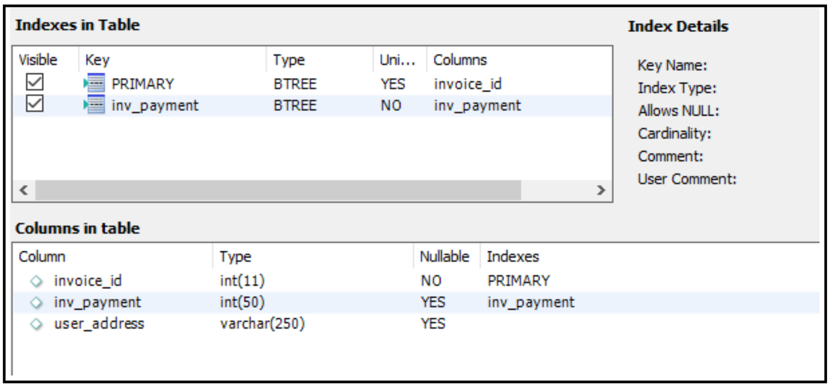
Columns in table

Column	Type	Nullable	Indexes
inv_img_code	varchar(250)	NO	PRIMARY
img_name	varchar(250)	NO	
image_text	mediumblob	YES	

11) invoice tables



```
CREATE TABLE `invoice` (
  `invoice_id` int(11) NOT NULL AUTO_INCREMENT,
  `inv_payment` int(50) DEFAULT NULL,
  `user_address` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`invoice_id`),
  KEY `inv_payment` (`inv_payment`),
  CONSTRAINT `invoice_ibfk_1` FOREIGN KEY (`inv_payment`) REFERENCES `payment` (`payment_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	invoice_id
✓	inv_payment	BTREE	NO	inv_payment

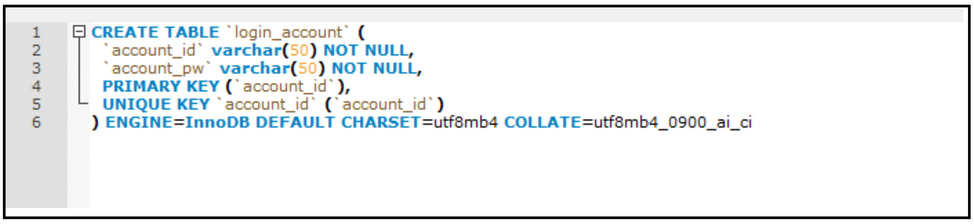
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

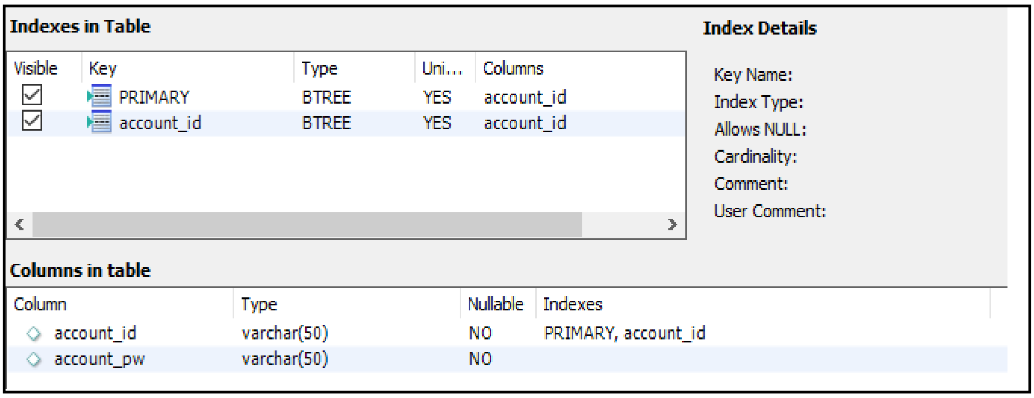
Columns in table

Column	Type	Nullable	Indexes
invoice_id	int(11)	NO	PRIMARY
inv_payment	int(50)	YES	inv_payment
user_address	varchar(250)	YES	

12) login_account tables



```
CREATE TABLE `login_account` (
  `account_id` varchar(50) NOT NULL,
  `account_pw` varchar(50) NOT NULL,
  PRIMARY KEY (`account_id`),
  UNIQUE KEY `account_id` (`account_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	account_id
✓	account_id	BTREE	YES	account_id

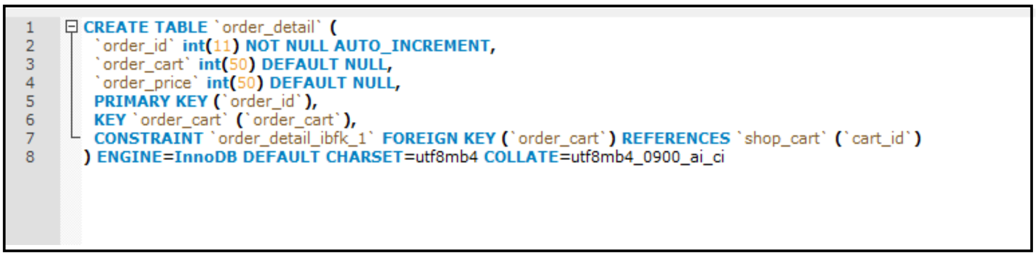
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

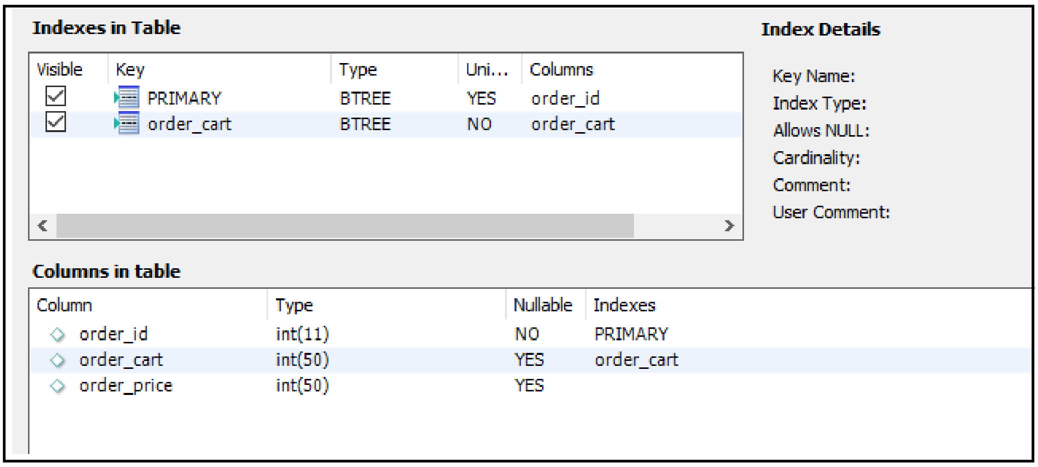
Columns in table

Column	Type	Nullable	Indexes
account_id	varchar(50)	NO	PRIMARY, account_id
account_pw	varchar(50)	NO	

13) order_detail tables



```
CREATE TABLE `order_detail` (
  `order_id` int(11) NOT NULL AUTO_INCREMENT,
  `order_cart` int(50) DEFAULT NULL,
  `order_price` int(50) DEFAULT NULL,
  PRIMARY KEY (`order_id`),
  KEY `order_cart` (`order_cart`),
  CONSTRAINT `order_detail_ibfk_1` FOREIGN KEY (`order_cart`) REFERENCES `shop_cart` (`cart_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
✓	PRIMARY	BTREE	YES	order_id
✓	order_cart	BTREE	NO	order_cart

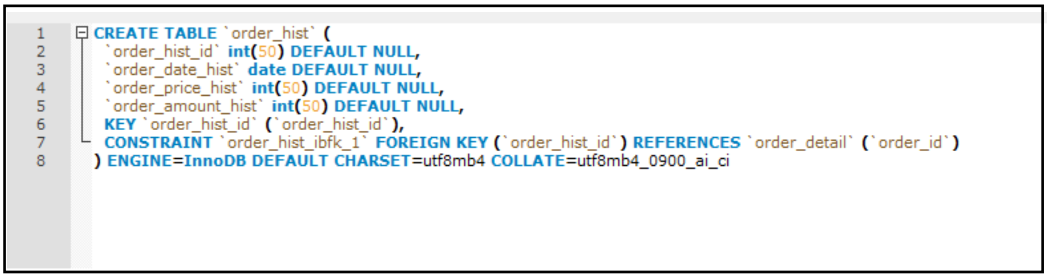
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

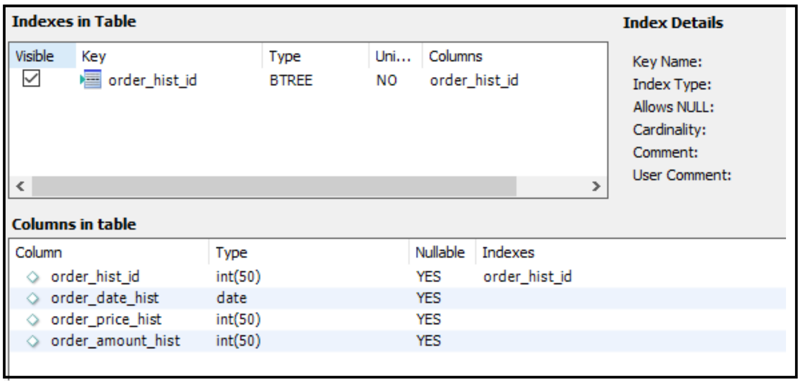
Columns in table

Column	Type	Nullable	Indexes
order_id	int(11)	NO	PRIMARY
order_cart	int(50)	YES	order_cart
order_price	int(50)	YES	

14) order_history tables



```
CREATE TABLE `order_hist` (
  `order_hist_id` int(50) DEFAULT NULL,
  `order_date_hist` date DEFAULT NULL,
  `order_price_hist` int(50) DEFAULT NULL,
  `order_amount_hist` int(50) DEFAULT NULL,
  KEY `order_hist_id` (`order_hist_id`),
  CONSTRAINT `order_hist_ibfk_1` FOREIGN KEY (`order_hist_id`) REFERENCES `order_detail` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	order_hist_id	BTREE	NO	order_hist_id

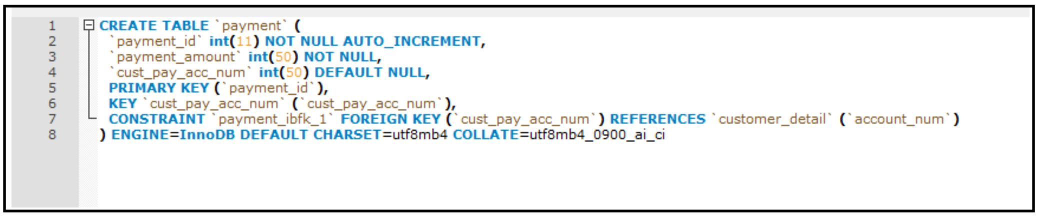
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

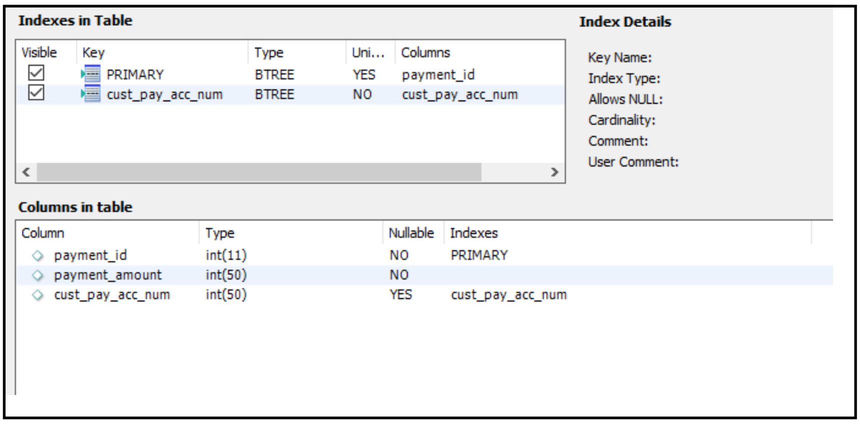
Columns in table

Column	Type	Nullable	Indexes
order_hist_id	int(50)	YES	order_hist_id
order_date_hist	date	YES	
order_price_hist	int(50)	YES	
order_amount_hist	int(50)	YES	

15) payment tables



```
CREATE TABLE `payment` (
  `payment_id` int(11) NOT NULL AUTO_INCREMENT,
  `payment_amount` int(50) NOT NULL,
  `cust_pay_acc_num` int(50) DEFAULT NULL,
  PRIMARY KEY (`payment_id`),
  KEY `cust_pay_acc_num` (`cust_pay_acc_num`),
  CONSTRAINT `payment_ibfk_1` FOREIGN KEY (`cust_pay_acc_num`) REFERENCES `customer_detail` (`account_num`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	PRIMARY	BTREE	YES	payment_id
☑	cust_pay_acc_num	BTREE	NO	cust_pay_acc_num

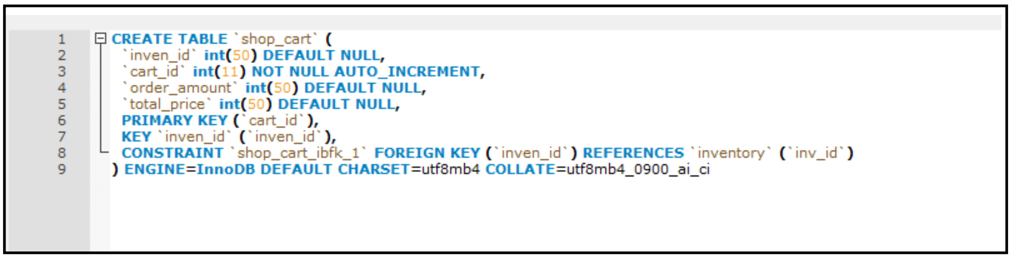
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

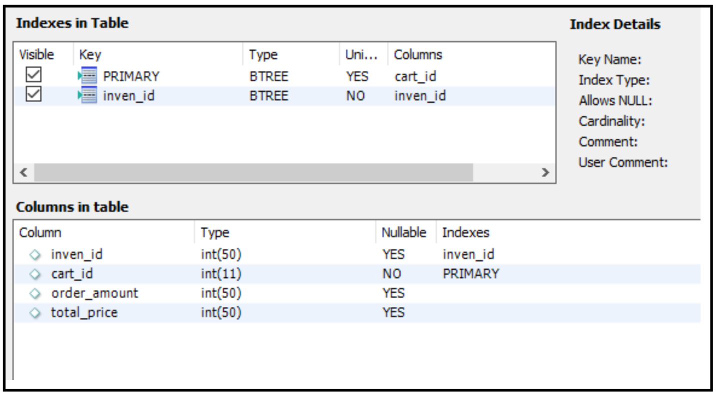
Columns in table

Column	Type	Nullable	Indexes
payment_id	int(11)	NO	PRIMARY
payment_amount	int(50)	NO	
cust_pay_acc_num	int(50)	YES	cust_pay_acc_num

16) shop_cart tables



```
CREATE TABLE `shop_cart` (
  `inven_id` int(50) DEFAULT NULL,
  `cart_id` int(11) NOT NULL AUTO_INCREMENT,
  `order_amount` int(50) DEFAULT NULL,
  `total_price` int(50) DEFAULT NULL,
  PRIMARY KEY (`cart_id`),
  KEY `inven_id` (`inven_id`),
  CONSTRAINT `shop_cart_ibfk_1` FOREIGN KEY (`inven_id`) REFERENCES `inventory` (`inv_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	PRIMARY	BTREE	YES	cart_id
☑	inven_id	BTREE	NO	inven_id

Index Details

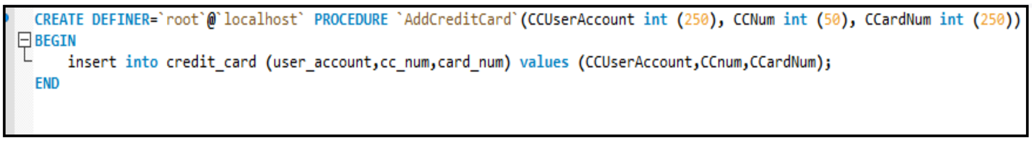
Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:

Columns in table

Column	Type	Nullable	Indexes
inven_id	int(50)	YES	inven_id
cart_id	int(11)	NO	PRIMARY
order_amount	int(50)	YES	
total_price	int(50)	YES	

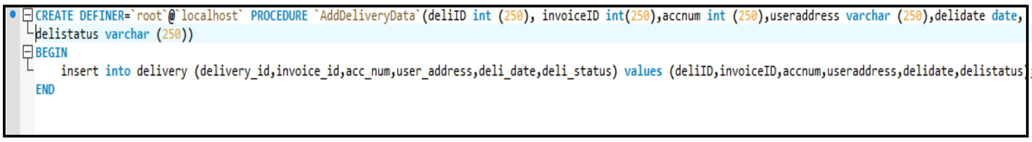
APPENDIX B: Stored Procedures

1) AddCreditCard



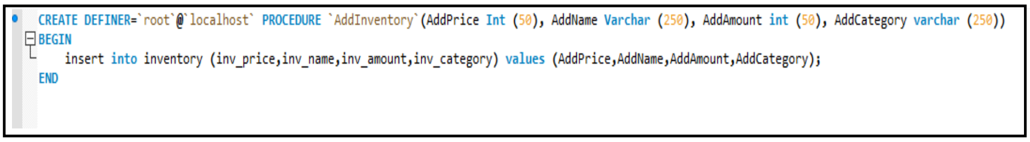
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddCreditCard`(CCUserAccount int (250), CCNum int (50), CCardNum int (250))
BEGIN
    insert into credit_card (user_account,cc_num,card_num) values (CCUserAccount,CCnum,CCardNum);
END
```

2) AddDeliveryData



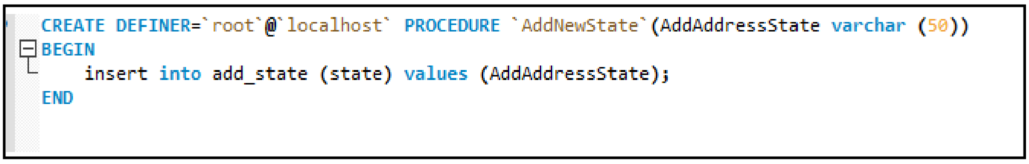
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddDeliveryData`(deliID int (250), invoiceID int(250),accnum int (250),useraddress varchar (250),delidate date,
delistatus varchar (250))
BEGIN
    insert into delivery (delivery_id,invoice_id,acc_num,user_address,deli_date,deli_status) values (deliID,invoiceID,accnum,useraddress,delidate,delistatus);
END
```

3) AddInventory



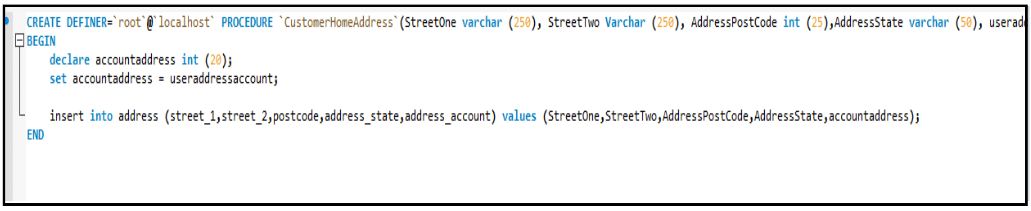
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddInventory`(AddPrice Int (50), AddName Varchar (250), AddAmount int (50), AddCategory varchar (250))
BEGIN
    insert into inventory (inv_price,inv_name,inv_amount,inv_category) values (AddPrice,AddName,AddAmount,AddCategory);
END
```

4) AddNewState



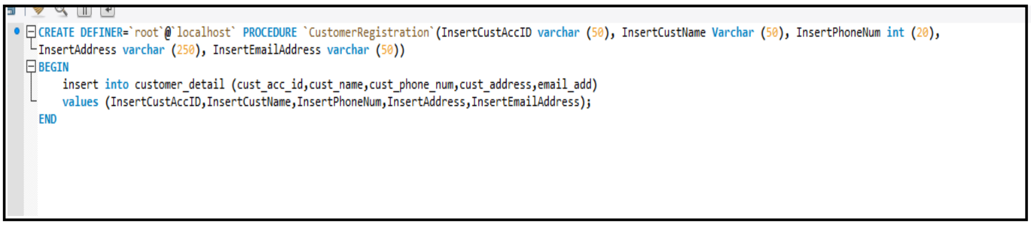
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddNewState`(AddAddressState varchar (50))
BEGIN
    insert into add_state (state) values (AddAddressState);
END
```

5) AddCustomerAddress



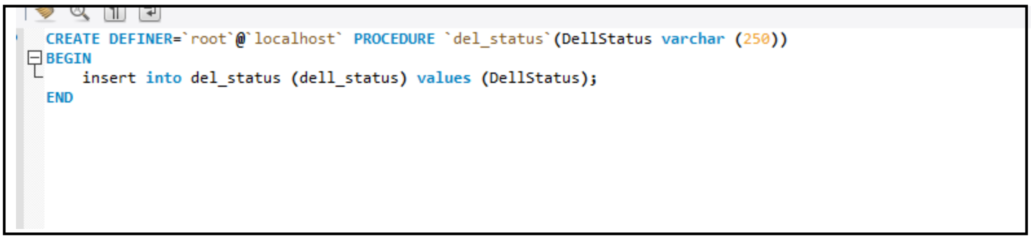
```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomerHomeAddress`(StreetOne varchar (250), StreetTwo Varchar (250), AddressPostCode int (25),AddressState varchar (50), userad
BEGIN
    declare accountaddress int (20);
    set accountaddress = useraddressaccount;

    insert into address (street_1,street_2,postcode,address_state,address_account) values (StreetOne,StreetTwo,AddressPostCode,AddressState,accountaddress);
END
```

6) CustomerRegistration



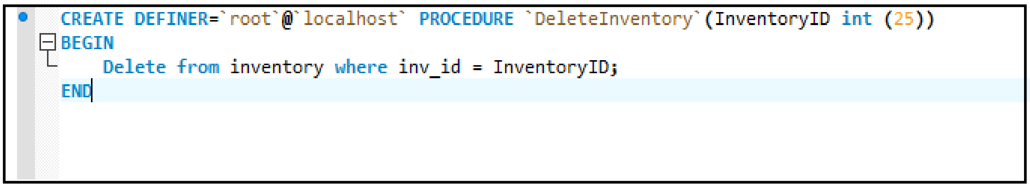
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomerRegistration`(InsertCustAccID varchar (50), InsertCustName Varchar (50), InsertPhoneNum int (20),
InsertAddress varchar (250), InsertEmailAddress varchar (50))
BEGIN
    insert into customer_detail (cust_acc_id,cust_name,cust_phone_num,cust_address,email_add)
    values (InsertCustAccID,InsertCustName,InsertPhoneNum,InsertAddress,InsertEmailAddress);
END
```

7) Del_Status



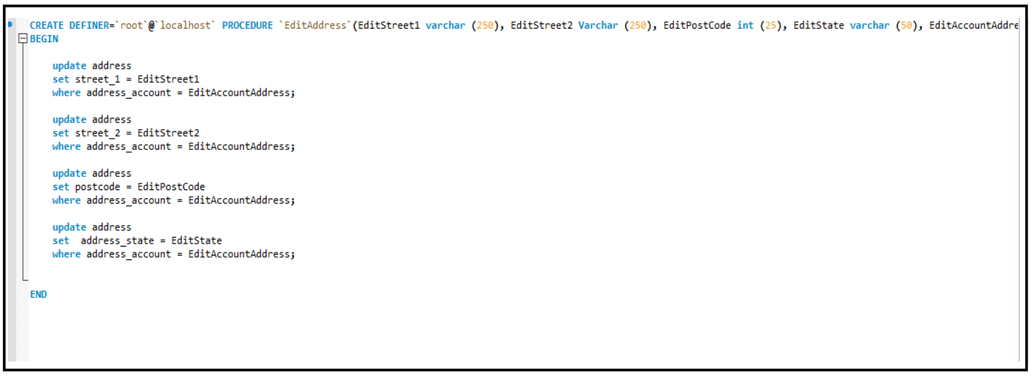
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `del_status`(DellStatus varchar (250))
BEGIN
    insert into del_status (dell_status) values (DellStatus);
END
```

8) DeleteInventory



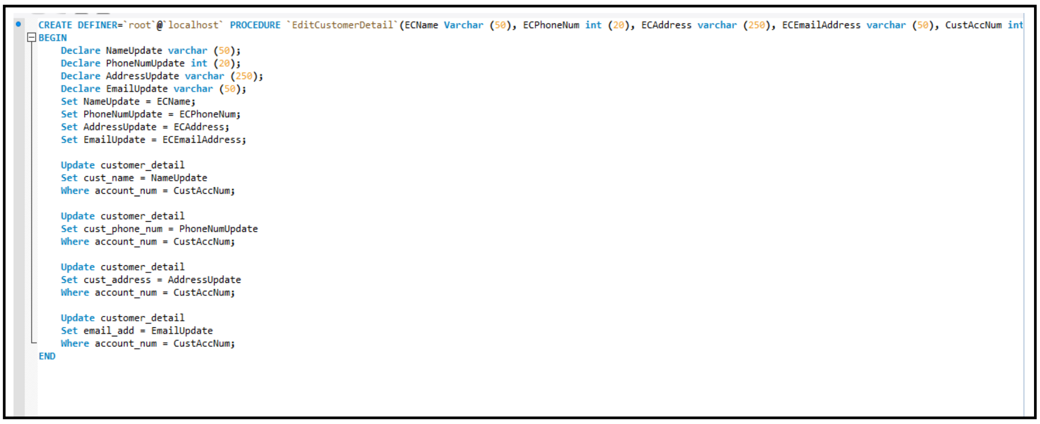
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteInventory`(InventoryID int (25))
BEGIN
    Delete from inventory where inv_id = InventoryID;
END
```

9) EditAddress



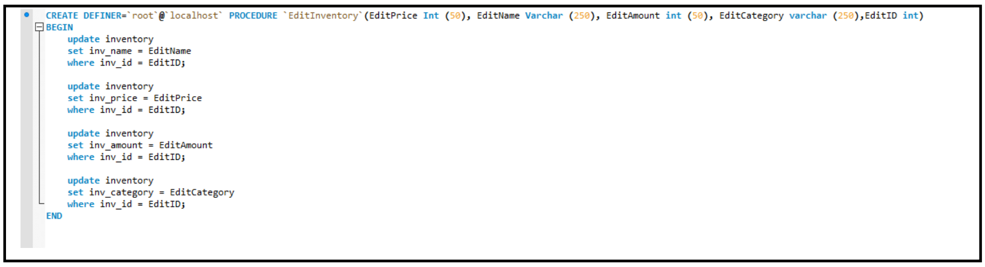
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EditAddress`(EditStreet1 varchar (250), EditStreet2 Varchar (250), EditPostCode int (25), EditState varchar (50), EditAccountAddre
BEGIN

    update address
    set street_1 = EditStreet1
    where address_account = EditAccountAddress;

    update address
    set street_2 = EditStreet2
    where address_account = EditAccountAddress;

    update address
    set postcode = EditPostCode
    where address_account = EditAccountAddress;

    update address
    set  address_state = EditState
    where address_account = EditAccountAddress;


END
```

10) EditCustomerDetail



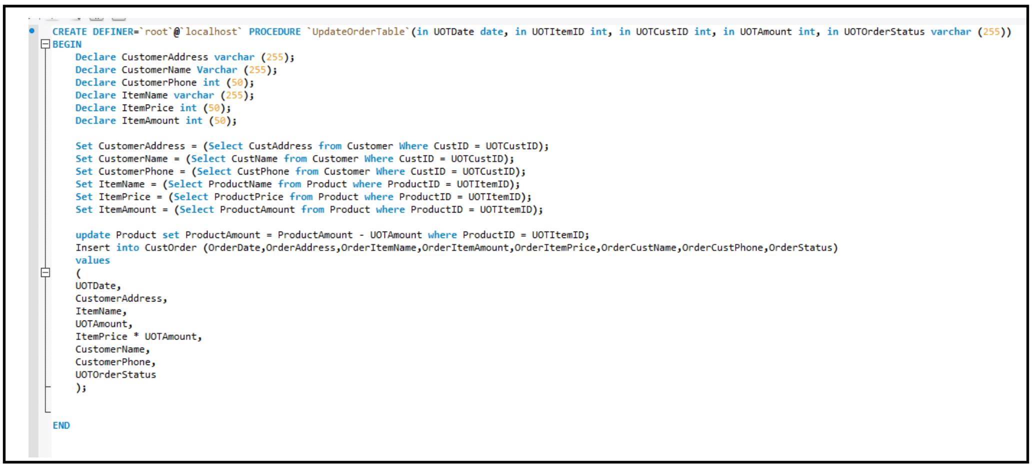
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EditCustomerDetail`(ECName Varchar (50), ECPhoneNum int (20), ECAddress varchar (250), ECEmailAddress varchar (50), CustAccNum int
BEGIN
    Declare NameUpdate varchar (50);
    Declare PhoneNumUpdate int (20);
    Declare AddressUpdate varchar (250);
    Declare EmailUpdate varchar (50);
    Set NameUpdate = ECName;
    Set PhoneNumUpdate = ECPhoneNum;
    Set AddressUpdate = ECAddress;
    Set EmailUpdate = ECEmailAddress;

    Update customer_detail
    Set cust_name = NameUpdate
    Where account_num = CustAccNum;

    Update customer_detail
    Set cust_phone_num = PhoneNumUpdate
    Where account_num = CustAccNum;

    Update customer_detail
    Set cust_address = AddressUpdate
    Where account_num = CustAccNum;

    Update customer_detail
    Set email_add = EmailUpdate
    Where account_num = CustAccNum;
END
```

11) EditInventory



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EditInventory`(EditPrice Int (50), EditName Varchar (250), EditAmount int (50), EditCategory varchar (250),EditID int)
BEGIN
    update inventory
    set inv_name = EditName
    where inv_id = EditID;

    update inventory
    set inv_price = EditPrice
    where inv_id = EditID;

    update inventory
    set inv_amount = EditAmount
    where inv_id = EditID;

    update inventory
    set inv_category = EditCategory
    where inv_id = EditID;
END
```

12) UpdateOrderTable



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateOrderTable`(in UOTDate date, in UOTItemID int, in UOTCustID int, in UOTAmount int, in UOTOrderStatus varchar (255))
BEGIN
    Declare CustomerAddress varchar (255);
    Declare CustomerName Varchar (255);
    Declare CustomerPhone int (50);
    Declare ItemName varchar (255);
    Declare ItemPrice int (50);
    Declare ItemAmount int (50);

    Set CustomerAddress = (Select CustAddress from Customer Where CustID = UOTCustID);
    Set CustomerName = (Select CustName from Customer Where CustID = UOTCustID);
    Set CustomerPhone = (Select CustPhone from Customer Where CustID = UOTCustID);
    Set ItemName = (Select ProductName from Product where ProductID = UOTItemID);
    Set ItemPrice = (Select ProductPrice from Product where ProductID = UOTItemID);
    Set ItemAmount = (Select ProductAmount from Product where ProductID = UOTItemID);

    update Product set ProductAmount = ProductAmount - UOTAmount where ProductID = UOTItemID;
    Insert into CustOrder (OrderDate,OrderAddress,OrderItemName,OrderItemAmount,OrderItemPrice,OrderCustName,OrderCustPhone,OrderStatus)
    values
    (
    UOTDate,
    CustomerAddress,
    ItemName,
    UOTAmount,
    ItemPrice * UOTAmount,
    CustomerName,
    CustomerPhone,
    UOTOrderStatus
    );

END
```

References

[1] Bhat, Dr. Shahid & Kansana, Keshav & Majid, Jenifur, (2016), “A Review Paper on E-Commerce”, Asian Journal of Technology & Management Research , [ISSN: 2249 –0892], Vol. 6 – Issue: 1, pp 16-21.

[2] Abdul Gaffar Khan (2016), “Electronic Commerce: A Study on Benefits and Challenges in an Emerging Economy”, Global Journal of Management and Business research: (B) Economics and Commerce, Online ISSN : 2249-4588, Vol 16 – Issue : 1, pp 18-22.

[3] Jessica young (2019). “Global E-commerce Sales Grow 18%”. Online Records: https://www.digitalcommerce360.com/article/global-ecommerce-sales/

[4] Guy Harrison, Steven Feuerstein (2006), *My SQL Stored Procedure Programming,* ISBN : 0596100892

[5] Il- Yeol Song (2000), “Database Design for Real – E-Commerce Systems”, IEEE Data Engineering Bulleting, Vol 23, No.1, pp 23-28

[6] Varshney, Upkar & Vetter, Ron. (2001), “A framework for the emerging mobile commerce applications”, Proceedings of the 34th Hawaii International Conference on System Sciences – 2001, pp 1-10.

[7] Hameed, K., Ahsan, K., & Yang, W. (2010), “Mobile Commerce and Applications: An Exploratory Study and Review”, Journal of Computing Volume 2 Issue 4, ISSN 2151-9617, pp 110-114

[8] Oluwaseye Omowa (2016), *Development of an E-Service App on the Android Platform.,* Oulu University of Applied Sciences

[9] Courtney B. Thaden (2010), *Analysis of Multi-Platform Mobile Application Development,* University of North Dakota. UMI: 1560009

[10] Wei, H., & Godfrey, T. (2006), Database Middleware and Web Services for Data Distribution and Integration in Distributed Heterogeneous Database Systems, *IKE*.

[11] van der Meer, Sven. (2002), *Middleware and Application Management Architecture*.

[12] Roland Balk (2016), *Database programming made easier*, University of Twente, Formal Methods & Tools group

[13] Hawick, Ken & A. James, H. (2019). “Middleware Issues for Mobile Business and Commerce.” https://www.researchgate.net/publication/239830008

[14] Seshasayee, B. (2008), *Middleware-based services for virtual cooperative mobile platforms*, Georgia Institute of Technology.

[15] Adam H. Mitz (2004). *The Design and Implementation of Database-Access Middleware for Live Object-Oriented Programming.* Report Number : WUCSE- 2004-21

Acknowledgements

First, I want to thanks Prof. Hyu Chan Park for giving me the chance to continue my master at Korea Maritime Ocean University. Without his constant support and guidance, I would never have the chance to study at KMOU, and I would never completed my thesis. Thank you for all the support and guidance that is given.

Secondly, I would like to thank Prof. Ok Keun Shin and Prof. Jang Se Lee for evaluating my thesis. Without your criticism and help my thesis will never be what it is today. Thank you for your criticism and help regarding my thesis.

Finally yet importantly, I would like to thank all my family and friends that is directly or indirectly involved with my thesis. Without your constant support, I would never be able to be where I am today. Thank you.

2019년 8월 Mohammad Haziq Maslan