A·P·U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

APU's E-Bookstore

LECTURER NAME:	ABDALLAH S.M. ALNATSHA
MODULE NAME:	INTRODUCTION TO DATABASE SYSTEMS
MODULE CODE:	CT042-3-1-IDB
HAND OUT DATE:	12 JUNE 2017
HAND IN DATE:	21 AUGUST 2017
INTAKE CODE:	UC1F1611
STUDENTS:	**MUSTAFA AHMED ABDULJABBAR** (TP043972) **MUKABAK ORAZBEK** (TP040205) **YOUSSEF THARAWAT** (TP040164)

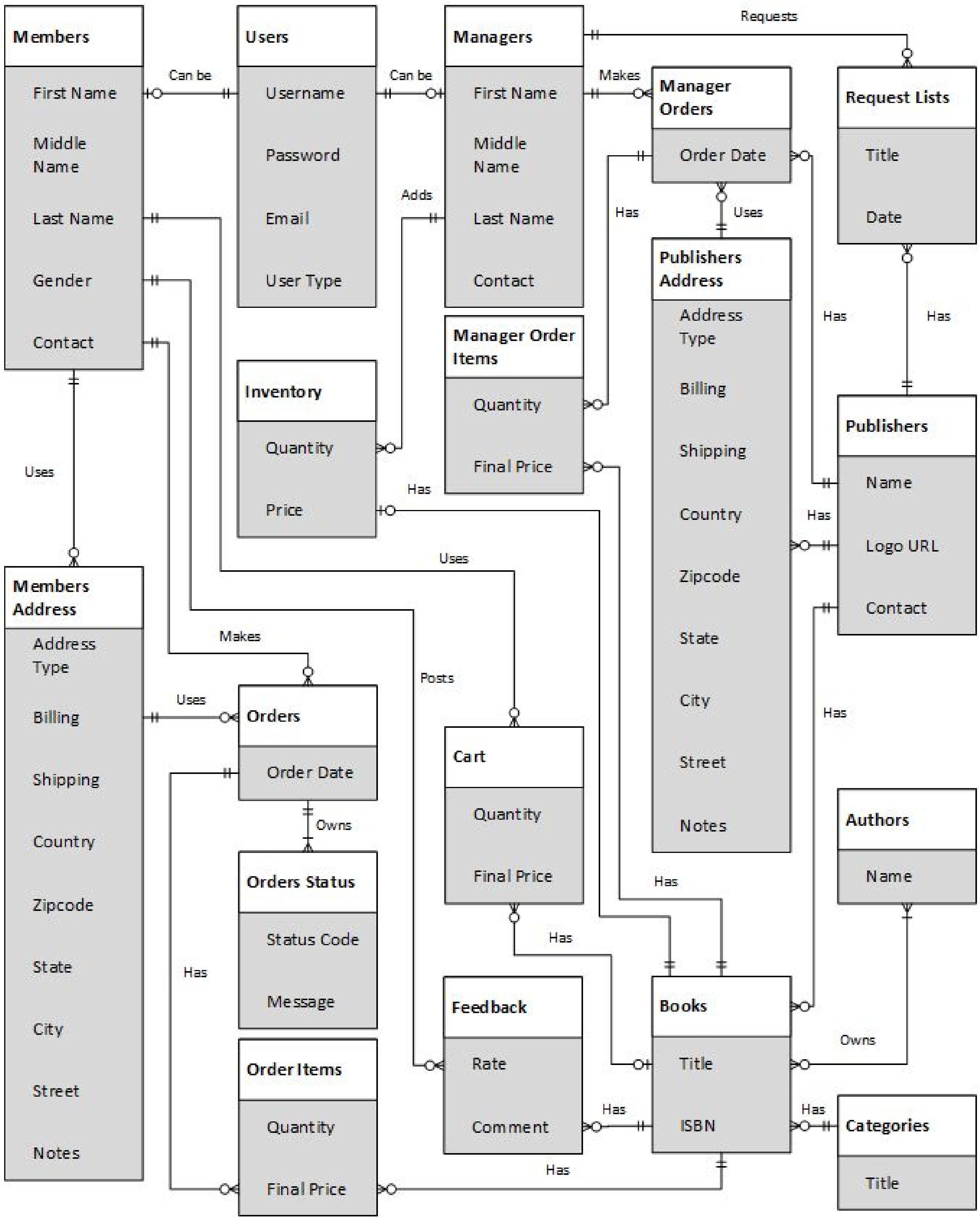
Table Of Contents

1.0 Introduction

As the traditional book store in APU is inefficient, the availability of books and reading material for purchase is insufficient. Student and staffs only have the option of a small bookshop within the enterprise. Therefore, the university decided to establish an e-bookstore. This will allow students to view and purchase the book online. Therefore, my team and I developed the database for the e-bookstore's online system.

Additionally, for the database management software we selected MySQL Community Edition 5.7, we used phpMyadmin to manage the database by interface. We have made the database easy for front-end developers to use as a data source as everything has been organized and correlated.

2.0 ERD


Members
First Name
Middle Name
Last Name
Gender
Contact
Users
Username
Password
Email
User Type
Managers
First Name
Middle Name
Last Name
Contact
Manager Orders
Order Date
Request Lists
Title
Date
Publishers Address
Address Type
Billing
Shipping
Country
Zipcode
State
City
Street
Notes
Publishers
Name
Logo URL
Contact
Manager Order Items
Quantity
Final Price
Inventory
Quantity
Price
Members Address
Address Type
Billing
Shipping
Country
Zipcode
State
City
Street
Notes
Orders
Order Date
Orders Status
Status Code
Message
Order Items
Quantity
Final Price
Cart
Quantity
Final Price
Feedback
Rate
Comment
Books
Title
ISBN
Authors
Name
Categories
Title
Can be
Can be
Makes
Requests
Adds
Has
Uses
Has
Has
Has
Has
Uses
Uses
Makes
Uses
Posts
Owns
Has
Has
Has
Has
Owns
Has
Has
Has

3.0 Business Rules

- Only registered users can be member or manager.

Each registered user can be zero or one manager, and each manager is one and only one registered user. Each registered user can be zero or one member, and each member is one and only one registered user.

- Only managers can request list of books from publisher.

Each manager has zero or many relationship to requested lists, and each requested list is made by only and only one manager.

- Only managers can order books from publishers (“manager_orders” table).

Each manager has zero or many relationship to orders, and each order is made by only and only one manager. Each publisher can get zero or many orders, and each order has one and only one publisher.

- Only managers can add books to catalog.

Each manager has zero or many relationship to inventory, and each inventory is added by only and only one manager. Each inventory is made for only and only one book, and each book has zero or one relationship with inventory.

- Only members can order books from library catalog (“orders” table).

Each member has zero or many relationship to orders, and each order is made by only and only one member.

- Only members can write comments for books

Each member has zero or many relationship to feedbacks, and each feedback is posted by only and only one member. Each book has zero or many feedbacks, and each feedback is written for one and only one book.

4.0 Normalization Process

4.1 books

4.1.1 Unnormalized Form (UNF)

- category
- isbn
- publisher
- title
- author
- author2
- author3
- quantity
- price
- manager_added_by

4.1.2 First step of normalization (1NF)

A. Decomposition of information
   a. Passed: nothing to decompose into multiple columns

B. Row Accessibility
   a. Auto increment ID Column added as Primary key.

C. No repeating groups
   a. author1,2,3 removed to be two tables called authors, book_authors

D. No redundant columns
   a. Passed: no redundant column found.

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The raw will be determined by the ID created in NF1.

4.2 Managers

4.2.1 Unnormalized Form (UNF)

- name
- mobile_contact
- username
- password
- usertype

4.2.2 First step of normalization (1NF)

E. Decomposition of information
   a. Name decomposed it into firstname,middlename,lastname

F. Row Accessibility
   a. Auto incremented ID primary key.

G. No repeating groups
   a. Passed

H. No redundant columns
   a. name removed because firstname,middlename,lastname

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The raw will be determined by the ID created in NF1.

4.3 Members

4.3.1 Unnormalized Form (UNF)

- fullname
- gender
- mobile_number
- shipping_address
- billing_address
- username
- password
- email

4.3.2 First step of normalization (1NF)

I. Decomposition of information
   a. Name decomposed it into firstname,middlename,lastname
   b. Shipping address and billing address have been decomposed into country, zipcode, state, city and street.

J. Row Accessibility
   a. Auto incremented ID primary key.

K. No repeating groups
   a. Passed: No repetition group founded.

L. No redundant columns
   a. name column will be removed because it's redundant.

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The rwa will be determined by the ID created in NF1.

4.4 Publishers

4.4.1 Unnormalized Form (UNF)

- name
- logoURL
- mobile_number
- shipping_address
- billing_address

4.4.2 First step of normalization (1NF)

M. Decomposition of information
   a. Shipping address and billing address have been decomposed into country, zipcode, state, city and street.

N. Row Accessibility
   a. Auto incremented ID primary key.

O. No repeating groups
   a. Passed: No repetition group founded.

P. No redundant columns
   a. Passed: No redundant group founded.

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The rwa will be determined by the ID created in NF1.

4.5 Orders

4.5.1 Unnormalized Form (UNF)

- user_billing_address
- user_shipping_address
- date
- member_name
- status
- item
- quantity
- price
- book

4.5.2 First step of normalization (1NF)

Q. Decomposition of information

Shipping address and billing address have been decomposed into country, zipcode, state, city and street.

R. Row Accessibility
   a. Auto incremented ID primary key.

S. No repeating groups
   a. Passed: No repetition group founded.

T. No redundant columns
   a. Passed: No redundant group founded.

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The raw will be determined by the ID created in NF1.

4.6 request_list

4.5.1 Unnormalized Form (UNF)

- created_at_date
- publisher
- manager
- list_title
- book_name
- book_quantity

4.5.2 First step of normalization (1NF)

U. Decomposition of information
   a. No decomposition of information done in this table

V. Row Accessibility
   a. Auto incremented ID primary key.

W. No repeating groups
   a. Passed: No repetition group founded.

X. No redundant columns
   a. Passed: No redundant group founded.

4.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.4 Third step of normalization (3NF)

Step 4.4.1 Find transitive dependencies

- The raw will be determined by the ID created in NF1.

4.7 manager_orders (publisher_invoice)

4.7.1 Unnormalized Form (UNF)

- user_billing_address
- user_shipping_address
- date
- publisher
- status
- item
- quantity
- price
- book

4.7.2 First step of normalization (1NF)

Y. Decomposition of information

a. Passed: No Decomposition required.

Z. Row Accessibility

a. ID column will be created as a Primary Key.

AA. No repeating groups

a. Passed: No repetition group founded.

BB. No redundant columns

a. Passed: No redundant group founded.

4.7.3 Second step of normalization (2NF)

4.3.1 Find partial dependencies (Functional dependencies)

- Since the identification table is added which is ID as a primary thus no partial dependencies.

4.7.4 Third step of normalization (3NF)

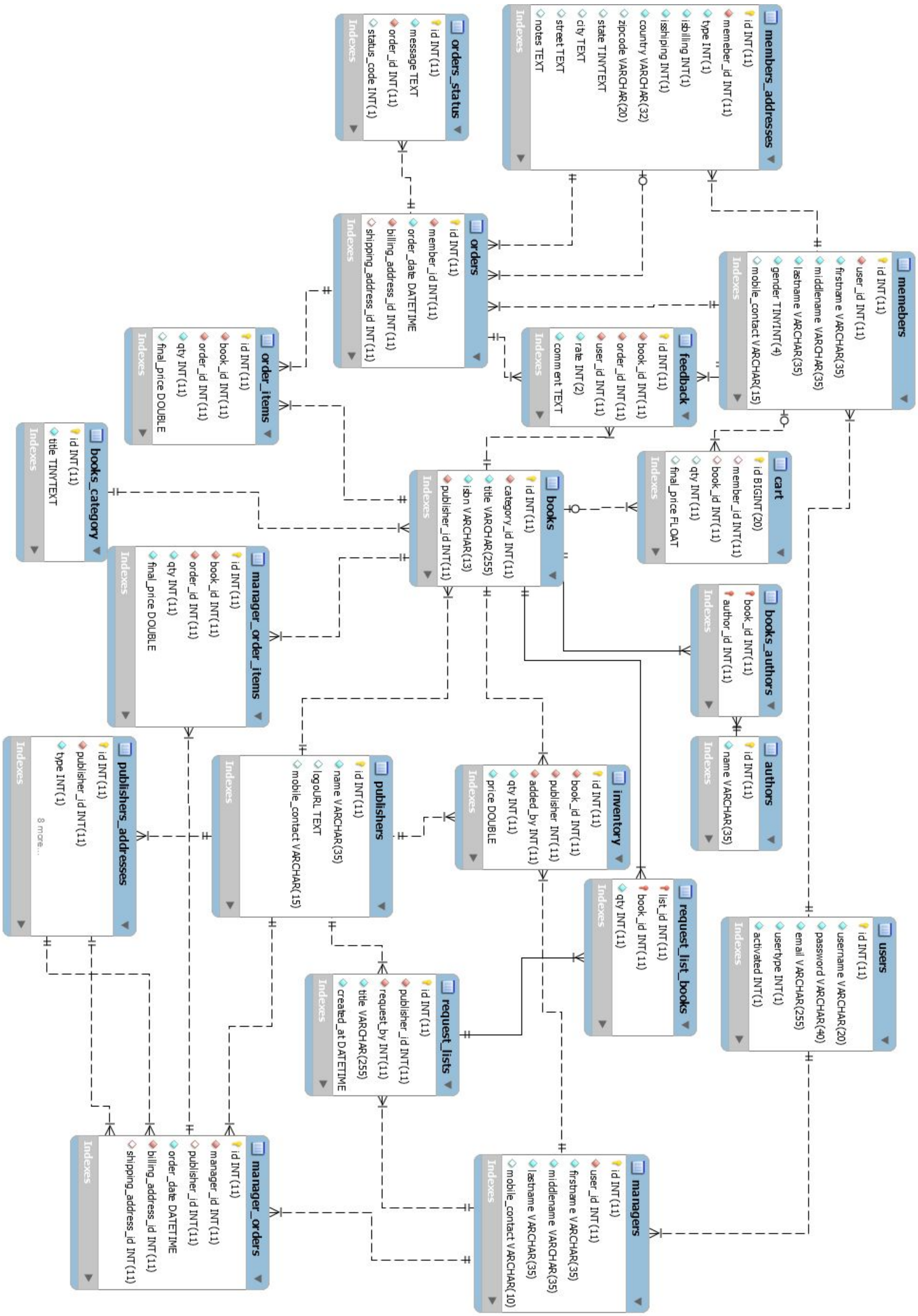
Step 4.4.1 Find transitive dependencies

- The raw will be determined by the ID created in NF1.

Further Normalizations

As it mention below in Database Schema section a further normalization occurred to higher level of decomposition for the database structure to ensure data consistency and adding constraints on database management system level to ensure no anomalies will occur in Creation/Updating/Removing across.

5.0 Database Schema



6.0 Data Dictionary

6.1 authors

Column	Type	Null
id *(Primary)*	int(11)	No
name	varchar(35)	No

6.2 books

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
category_id	int(11)	No		books_category -> id
title	varchar(255)	No		
isbn	varchar(13)	No		
publisher_id	int(11)	No		publishers -> id

6.3 books_authors

Column	Type	Null	Default	Links to
book_id *(Primary)*	int(11)	No		books -> id
author_id *(Primary)*	int(11)	No		authors -> id

6.4 books_category

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
title	tinytext	No		

6.5 cart

Column	Type	Null	Default	Links to
id *(Primary)*	bigint(20)	No		
member_id	bigint(20)	No		memebers -> id
book_id	bigint(20)	No		books -> id
qty	int(11)	No		
final_price	float	No		

6.6 feedback

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
book_id	int(11)	No		books -> id
order_id	int(11)	No		orders -> id
user_id	int(11)	No		memebers -> id
rate	int(2)	No	0	
comment	text	No		

6.7 inventory

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
book_id	int(11)	No		books -> id
publisher	int(11)	No	0	publishers -> id
added_by	int(11)	No	0	managers -> id
qty	int(11)	No	0	
price	double	No	0	

6.8 manager_order_items

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
book_id	int(11)	No		books -> id
order_id	int(11)	No		manager_orders -> id
qty	int(11)	No	1	
final_price	double	No		

6.9 manager_orders

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
manager_id	int(11)	No		managers -> id
publisher_id	int(11)	Yes	*NULL*	publishers -> id
order_date	datetime	No		
billing_address_id	int(11)	No		publishers_addresses -> id
shipping_address_id	int(11)	Yes	*NULL*	publishers_addresses -> id

6.10 managers

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
user_id	int(11)	No		users -> id
firstname	varchar(35)	No		
middlename	varchar(35)	No		
lastname	varchar(35)	No		
mobile_contact	varchar(15)	No		

6.11 members_addresses

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
memeber_id	int(11)	No		memebers -> id
type	int(1)	No	1	
isbilling	int(1)	No	1	
isshiping	int(1)	No	1	
country	varchar(32)	No		
zipcode	varchar(20)	Yes	*NULL*	
state	tinytext	Yes	*NULL*	
city	text	Yes	*NULL*	
street	text	Yes	*NULL*	
notes	text	Yes	*NULL*	

6.12 members

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
user_id	int(11)	No		users -> id
firstname	varchar(35)	No		
middlename	varchar(35)	No		
lastname	varchar(35)	No		
gender	tinyint(4)	No		
mobile_contact	varchar(15)	No		

6.13 order_items

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
book_id	int(11)	No		books -> id
order_id	int(11)	No		orders -> id
qty	int(11)	No	1	
final_price	double	No		

6.14 orders

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
member_id	int(11)	No		memebers -> id
order_date	datetime	No		
billing_address_id	int(11)	No		members_addresses -> id
shipping_address_id	int(11)	NO		members_addresses -> id

6.15 orders_status

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
message	text	No		
status_code	int(1)	No	0	
order_id	int(11)	No		orders -> id

6.16 publishers

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
name	varchar(35)	No		
logoURL	text	Yes	*NULL*	
mobile_contact	varchar(15)	No		

6.17 publishers_addresses

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
publisher_id	int(11)	No		publishers -> id
type	int(1)	No	1	
isbilling	int(1)	No	1	
isshiping	int(1)	No	1	
country	varchar(32)	No		
zipcode	varchar(20)	Yes	*NULL*	
state	tinytext	Yes	*NULL*	
city	text	Yes	*NULL*	
street	text	Yes	*NULL*	
notes	text	Yes	*NULL*	

6.18 request_list_books

Column	Type	Null	Default	Links to
list_id *(Primary)*	int(11)	No		request_lists -> id
book_id *(Primary)*	int(11)	No		books -> id
qty	int(11)	No	1	

6.19 request_lists

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
publisher_id	int(11)	No		publishers -> id
request_by	int(11)	No		managers -> id
title	varchar(255)	No		
created_at	datetime	No		

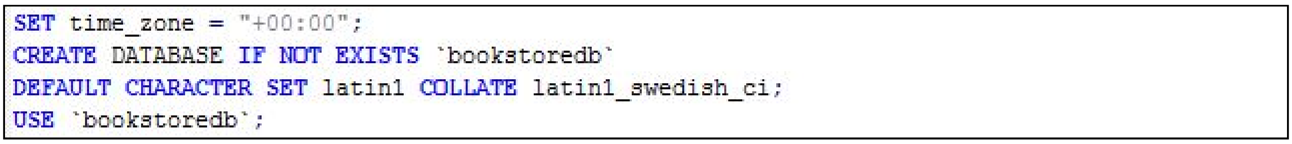
6.20 users

Column	Type	Null	Default	Links to
id *(Primary)*	int(11)	No		
username	varchar(20)	No		
password	varchar(40)	No		
email	varchar(255)	No		
usertype	int(1)	No	0	
activated	int(1)	No	0	

7.0 Data Definition Language (DDL)

The DDL commands below are built on MySQL

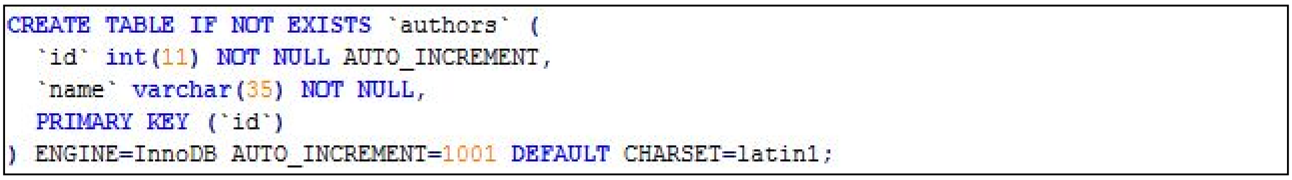
7.1 Create Database



```
SET time_zone = "+00:00";
CREATE DATABASE IF NOT EXISTS `bookstoredb`
DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
USE `bookstoredb`;
```

Figure 7.1: DDL command for creating database "bookstoredb".

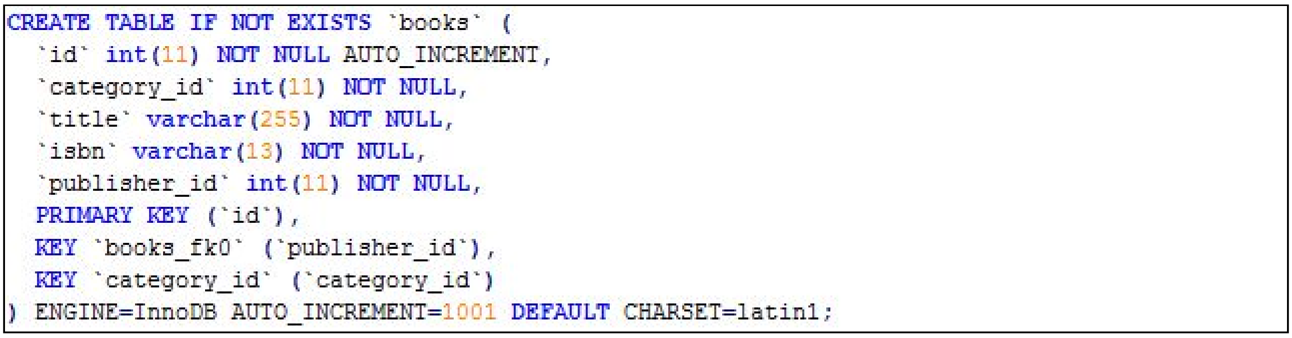
7.2 Create Table Authors



```
CREATE TABLE IF NOT EXISTS `authors` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(35) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1;
```

Figure 7.2: This diagram represents DDL commands for creating new table "authors".

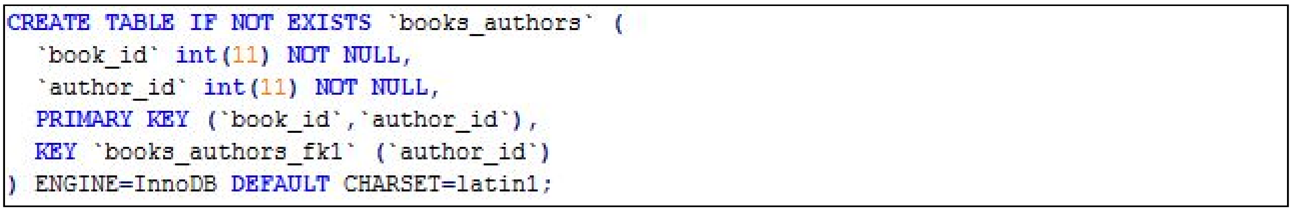
7.3 Create Table Books



```
CREATE TABLE IF NOT EXISTS `books` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `isbn` varchar(13) NOT NULL,
  `publisher_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `books_fk0` (`publisher_id`),
  KEY `category_id` (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1;
```

Figure 7.3: Basic MySQL commands for creating "books" table.

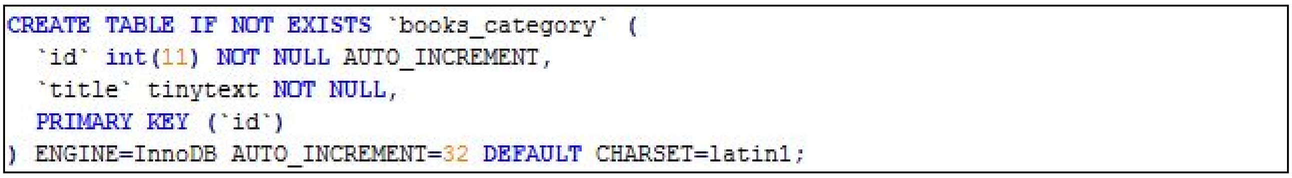
7.4 Create Table Books Authors



```
CREATE TABLE IF NOT EXISTS `books_authors` (
  `book_id` int(11) NOT NULL,
  `author_id` int(11) NOT NULL,
  PRIMARY KEY (`book_id`,`author_id`),
  KEY `books_authors_fk1` (`author_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 7.4: This DDL commands used for creating "books_authors" table.

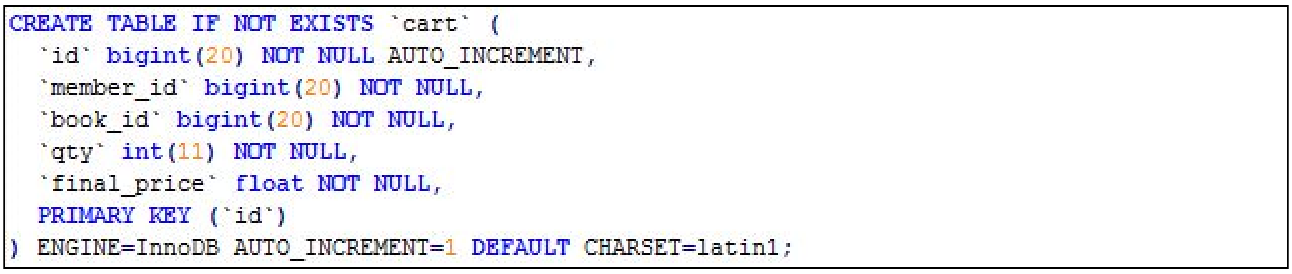
7.5 Create Table Books Category



```
CREATE TABLE IF NOT EXISTS `books_category` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` tinytext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=latin1;
```

Figure 7.5: Table at the above represents DDL commands for creating specific table.

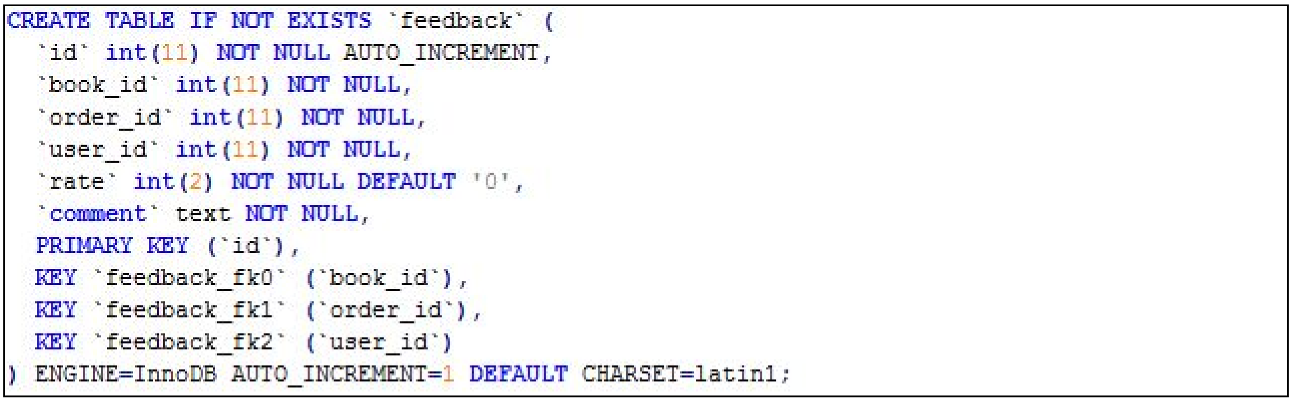
7.6 Create Table Cart



```
CREATE TABLE IF NOT EXISTS `cart` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `member_id` bigint(20) NOT NULL,
  `book_id` bigint(20) NOT NULL,
  `qty` int(11) NOT NULL,
  `final_price` float NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.6: This DDL commands designed for creating table “cart”.

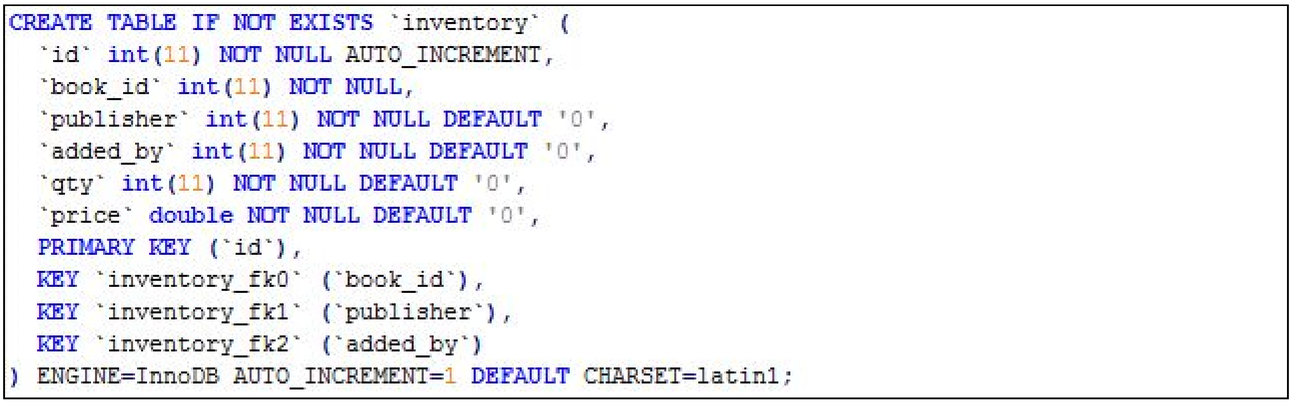
7.7 Create Table Feedback



```
CREATE TABLE IF NOT EXISTS `feedback` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `book_id` int(11) NOT NULL,
  `order_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `rate` int(2) NOT NULL DEFAULT '0',
  `comment` text NOT NULL,
  PRIMARY KEY (`id`),
  KEY `feedback_fk0` (`book_id`),
  KEY `feedback_fk1` (`order_id`),
  KEY `feedback_fk2` (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.7: Create a table “feedback”.

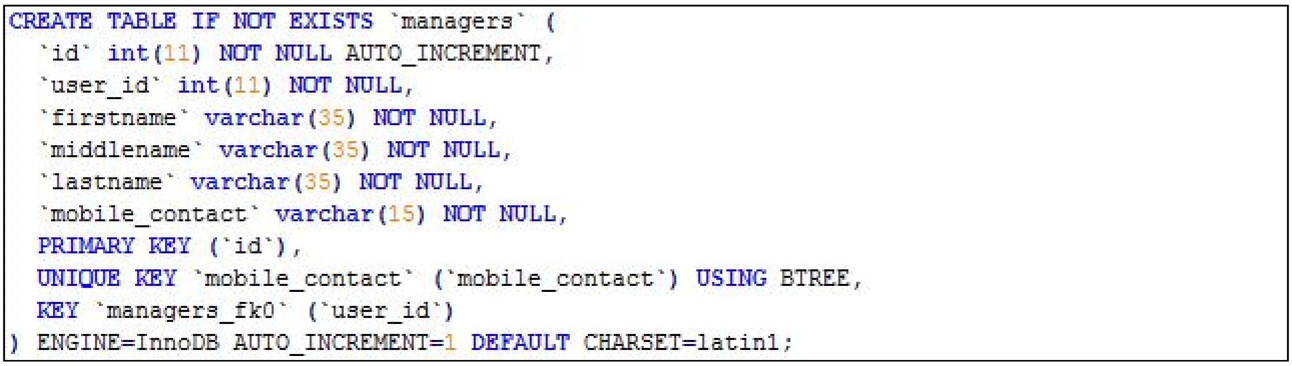
7.8 Create Table Inventory



```
CREATE TABLE IF NOT EXISTS `inventory` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `book_id` int(11) NOT NULL,
  `publisher` int(11) NOT NULL DEFAULT '0',
  `added_by` int(11) NOT NULL DEFAULT '0',
  `qty` int(11) NOT NULL DEFAULT '0',
  `price` double NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  KEY `inventory_fk0` (`book_id`),
  KEY `inventory_fk1` (`publisher`),
  KEY `inventory_fk2` (`added_by`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.8: To generate a table "inventory", our group members used following DDL commands.

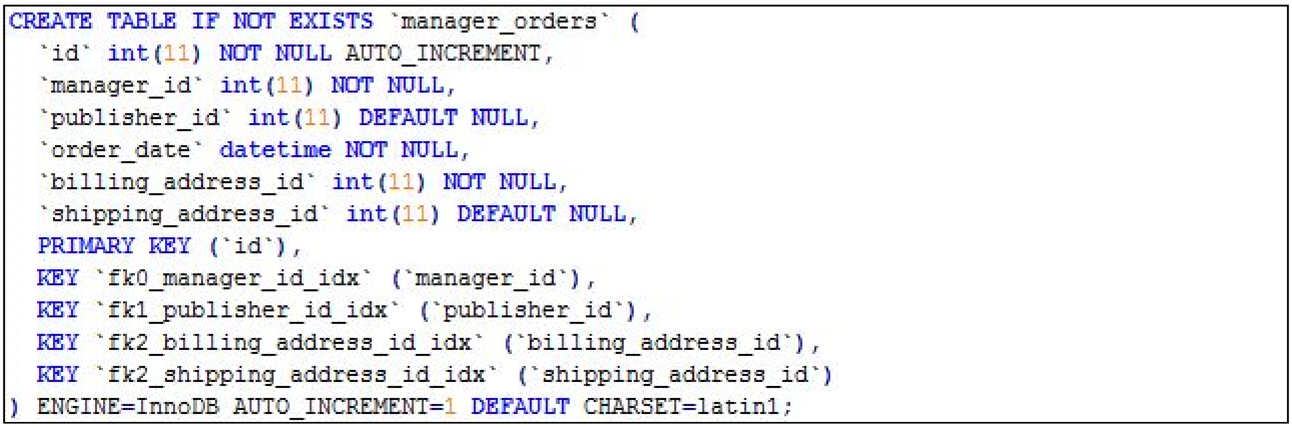
7.9 Create Table Managers



```
CREATE TABLE IF NOT EXISTS `managers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `firstname` varchar(35) NOT NULL,
  `middlename` varchar(35) NOT NULL,
  `lastname` varchar(35) NOT NULL,
  `mobile_contact` varchar(15) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mobile_contact` (`mobile_contact`) USING BTREE,
  KEY `managers_fk0` (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.9: One of the example of DDL commands for creating table "managers".

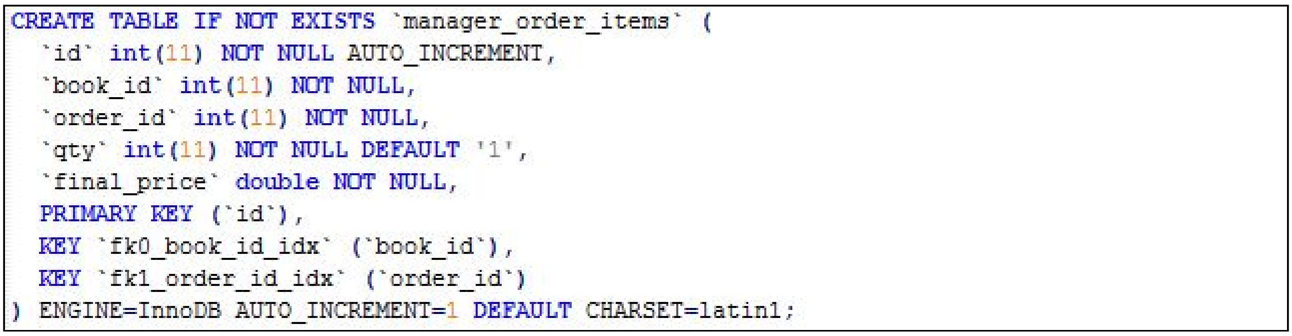
7.10 Create Table Manager Orders



```
CREATE TABLE IF NOT EXISTS `manager_orders` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `manager_id` int(11) NOT NULL,
  `publisher_id` int(11) DEFAULT NULL,
  `order_date` datetime NOT NULL,
  `billing_address_id` int(11) NOT NULL,
  `shipping_address_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk0_manager_id_idx` (`manager_id`),
  KEY `fk1_publisher_id_idx` (`publisher_id`),
  KEY `fk2_billing_address_id_idx` (`billing_address_id`),
  KEY `fk2_shipping_address_id_idx` (`shipping_address_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.10: This DDL commands used for creating "manager_orders" table.

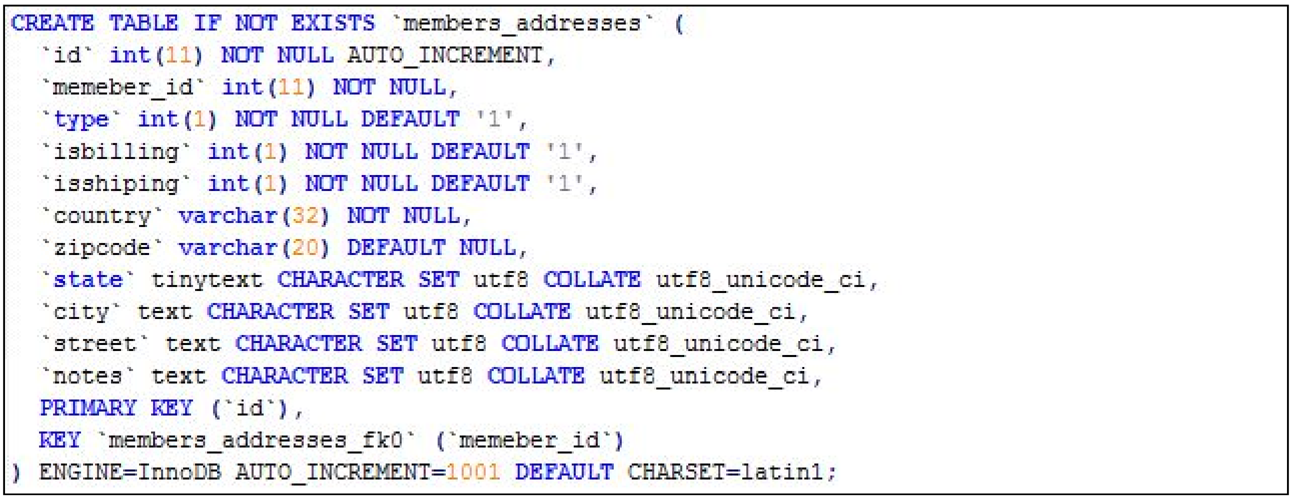
7.11 Create Table Manager Order Items



```
CREATE TABLE IF NOT EXISTS `manager_order_items` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `book_id` int(11) NOT NULL,
  `order_id` int(11) NOT NULL,
  `qty` int(11) NOT NULL DEFAULT '1',
  `final_price` double NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk0_book_id_idx` (`book_id`),
  KEY `fk1_order_id_idx` (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.11: The SQL code at the above used for creating "manager_order_items" table.

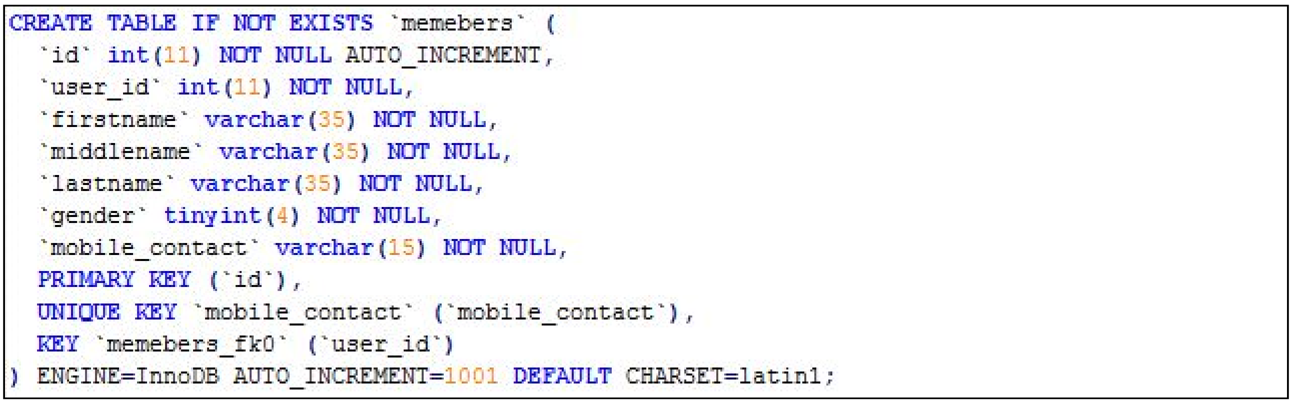
7.12 Create Table Members Addresses



```
CREATE TABLE IF NOT EXISTS `members_addresses` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `memeber_id` int(11) NOT NULL,
  `type` int(1) NOT NULL DEFAULT '1',
  `isbilling` int(1) NOT NULL DEFAULT '1',
  `isshiping` int(1) NOT NULL DEFAULT '1',
  `country` varchar(32) NOT NULL,
  `zipcode` varchar(20) DEFAULT NULL,
  `state` tinytext CHARACTER SET utf8 COLLATE utf8_unicode_ci,
  `city` text CHARACTER SET utf8 COLLATE utf8_unicode_ci,
  `street` text CHARACTER SET utf8 COLLATE utf8_unicode_ci,
  `notes` text CHARACTER SET utf8 COLLATE utf8_unicode_ci,
  PRIMARY KEY (`id`),
  KEY `members_addresses_fk0` (`memeber_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1;
```

Figure 7.12: This DDL commands designed for creating table "members_addresses".

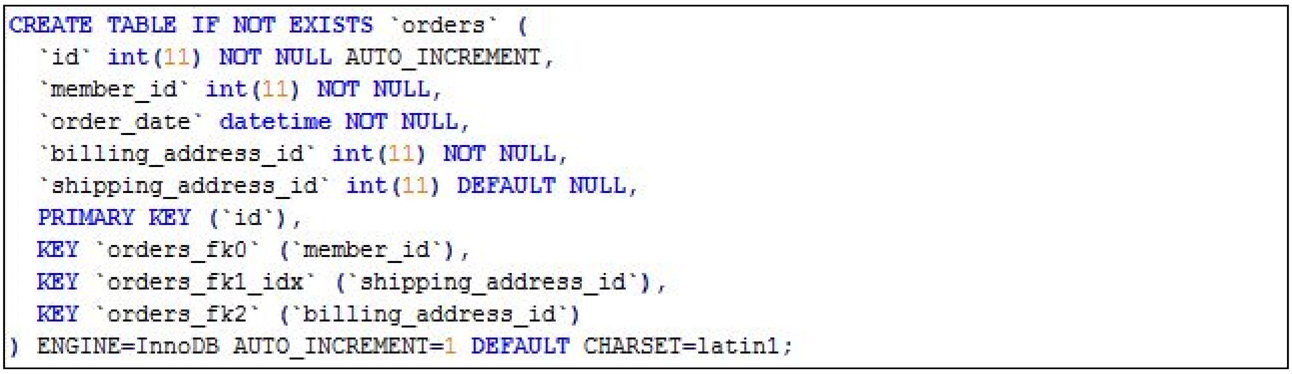
7.13 Create Table Members



```
CREATE TABLE IF NOT EXISTS `memebers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `firstname` varchar(35) NOT NULL,
  `middlename` varchar(35) NOT NULL,
  `lastname` varchar(35) NOT NULL,
  `gender` tinyint(4) NOT NULL,
  `mobile_contact` varchar(15) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mobile_contact` (`mobile_contact`),
  KEY `memebers_fk0` (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1;
```

Figure 7.13: This DDL commands used for creating "members" table.

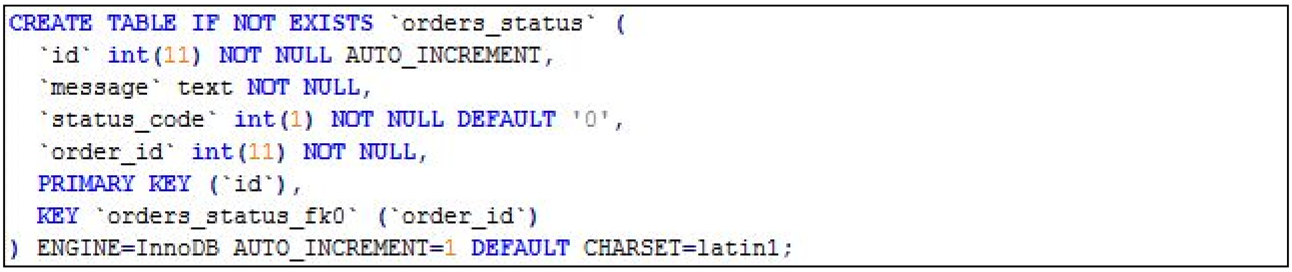
7.14 Create Table Orders



```
CREATE TABLE IF NOT EXISTS `orders` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `member_id` int(11) NOT NULL,
  `order_date` datetime NOT NULL,
  `billing_address_id` int(11) NOT NULL,
  `shipping_address_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `orders_fk0` (`member_id`),
  KEY `orders_fk1_idx` (`shipping_address_id`),
  KEY `orders_fk2` (`billing_address_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.14: Via implementing following DDL commands, group members created table "orders".

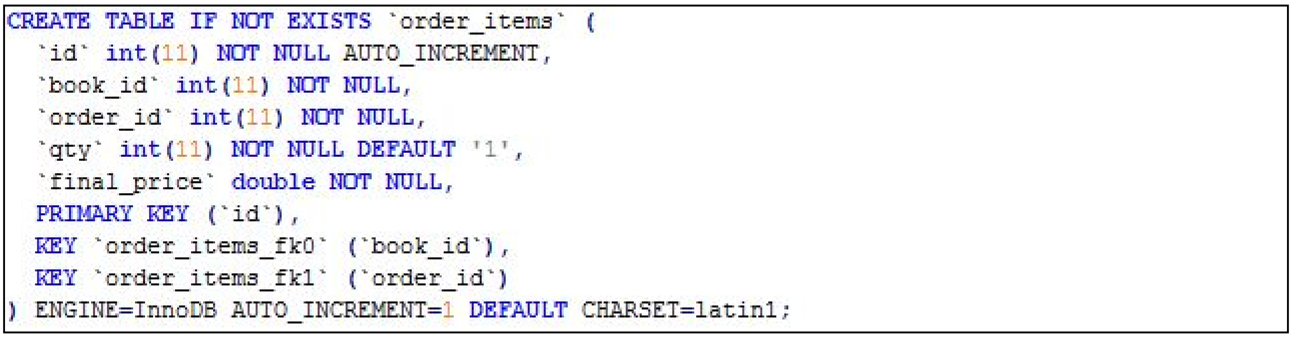
7.15 Create Table Orders Status



```
CREATE TABLE IF NOT EXISTS `orders_status` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `message` text NOT NULL,
  `status_code` int(1) NOT NULL DEFAULT '0',
  `order_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `orders_status_fk0` (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.15: This DDL commands used for creating "orders_status" table.

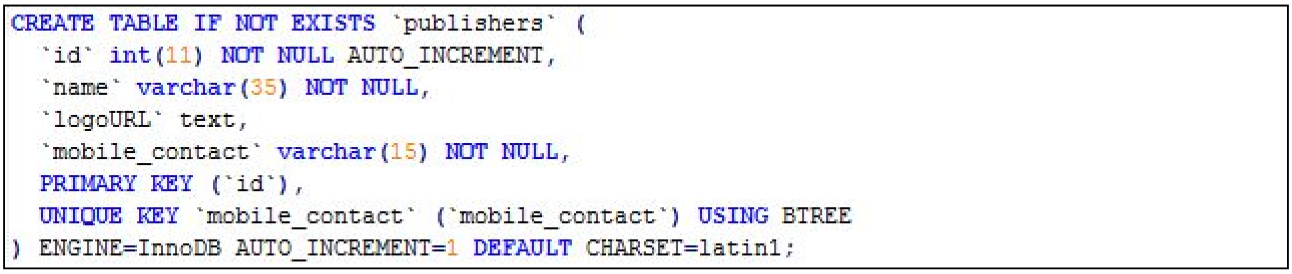
7.16 Create Table Order Items



```
CREATE TABLE IF NOT EXISTS `order_items` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `book_id` int(11) NOT NULL,
  `order_id` int(11) NOT NULL,
  `qty` int(11) NOT NULL DEFAULT '1',
  `final_price` double NOT NULL,
  PRIMARY KEY (`id`),
  KEY `order_items_fk0` (`book_id`),
  KEY `order_items_fk1` (`order_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.16: This DDL commands allows to users to create "books_authors" table.

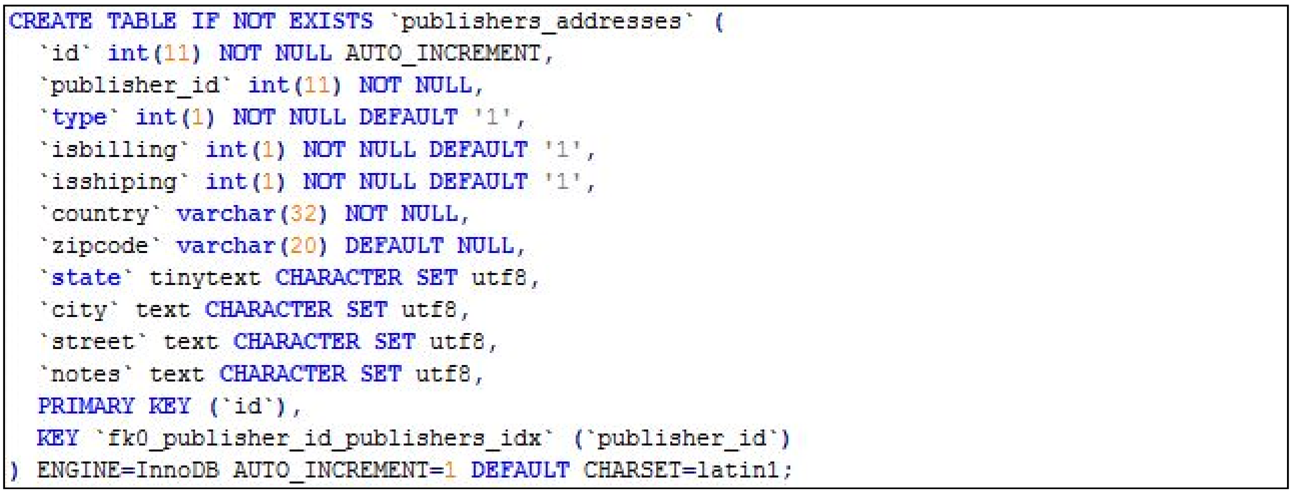
7.17 Create Table Publishers



```
CREATE TABLE IF NOT EXISTS `publishers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(35) NOT NULL,
  `logoURL` text,
  `mobile_contact` varchar(15) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mobile_contact` (`mobile_contact`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.17: This DDL commands used for creating "publishers" table.

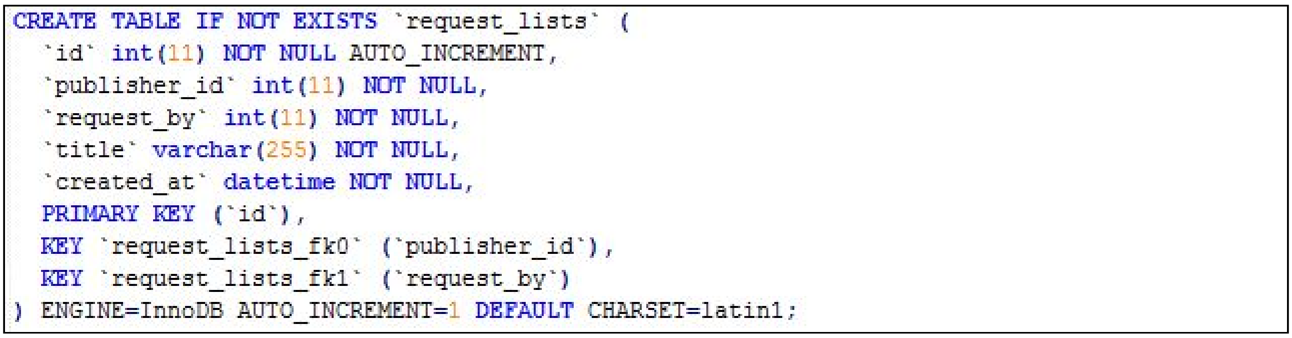
7.18 Create Table Publishers Address



```
CREATE TABLE IF NOT EXISTS `publishers_addresses` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `publisher_id` int(11) NOT NULL,
  `type` int(1) NOT NULL DEFAULT '1',
  `isbilling` int(1) NOT NULL DEFAULT '1',
  `isshiping` int(1) NOT NULL DEFAULT '1',
  `country` varchar(32) NOT NULL,
  `zipcode` varchar(20) DEFAULT NULL,
  `state` tinytext CHARACTER SET utf8,
  `city` text CHARACTER SET utf8,
  `street` text CHARACTER SET utf8,
  `notes` text CHARACTER SET utf8,
  PRIMARY KEY (`id`),
  KEY `fk0_publisher_id_publishers_idx` (`publisher_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.18: One of the example of creating tables in DDL.

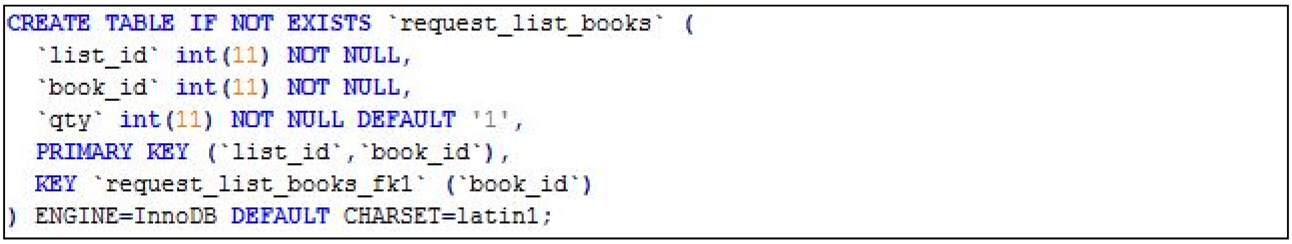
7.19 Create Table Request Lists



```
CREATE TABLE IF NOT EXISTS `request_lists` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `publisher_id` int(11) NOT NULL,
  `request_by` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `created_at` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `request_lists_fk0` (`publisher_id`),
  KEY `request_lists_fk1` (`request_by`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Figure 7.19: This DDL commands designed for creating "request_lists" table.

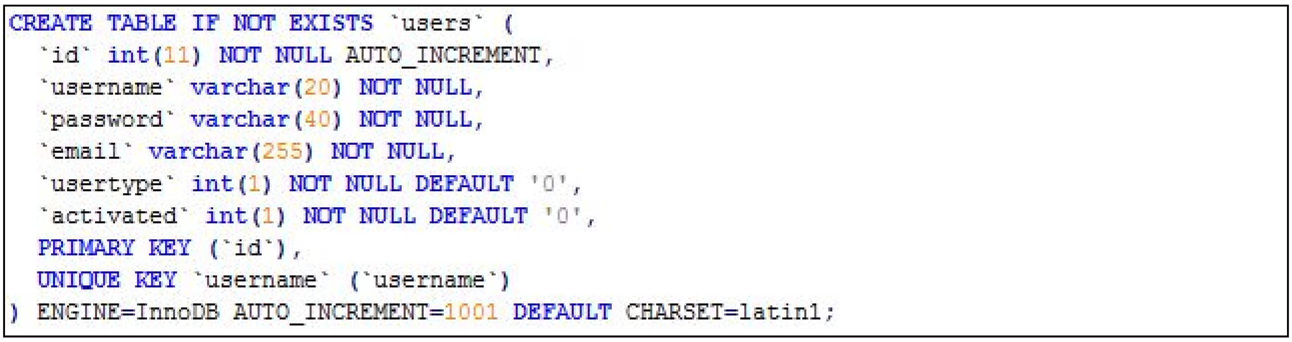
7.20 Create Table Request List Books



```
CREATE TABLE IF NOT EXISTS `request_list_books` (
  `list_id` int(11) NOT NULL,
  `book_id` int(11) NOT NULL,
  `qty` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`list_id`,`book_id`),
  KEY `request_list_books_fk1` (`book_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 7.20: This DDL commands written for creating specific tables in DDL.

7.21 Create Table Users

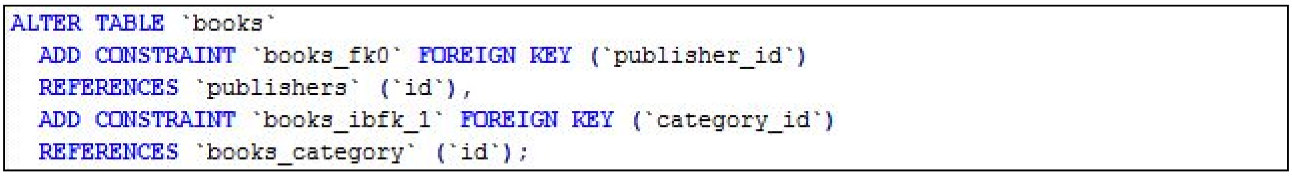


```
CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) NOT NULL,
  `password` varchar(40) NOT NULL,
  `email` varchar(255) NOT NULL,
  `usertype` int(1) NOT NULL DEFAULT '0',
  `activated` int(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1;
```

Figure 7.21: This DDL commands used for creating "users" table.

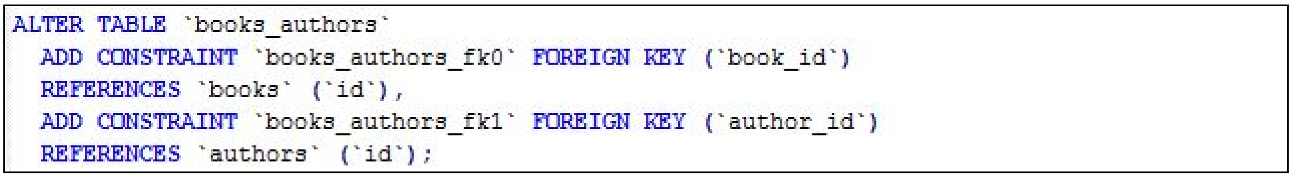
7.22 Adding CRUD Table Constraints

This constraints takes preventive actions when Insert/Update/Delete occurs to ensure that the entry is not redundant and relative to the reference table.



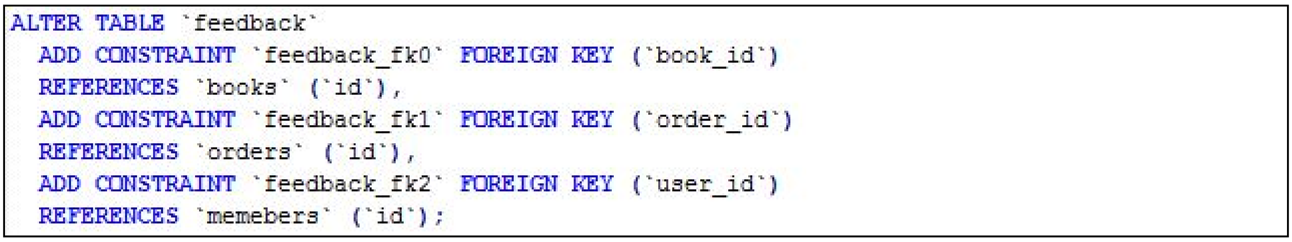
```
ALTER TABLE `books`
  ADD CONSTRAINT `books_fk0` FOREIGN KEY (`publisher_id`)
  REFERENCES `publishers` (`id`),
  ADD CONSTRAINT `books_ibfk_1` FOREIGN KEY (`category_id`)
  REFERENCES `books_category` (`id`);
```

Figure 7.22: Update table "books".



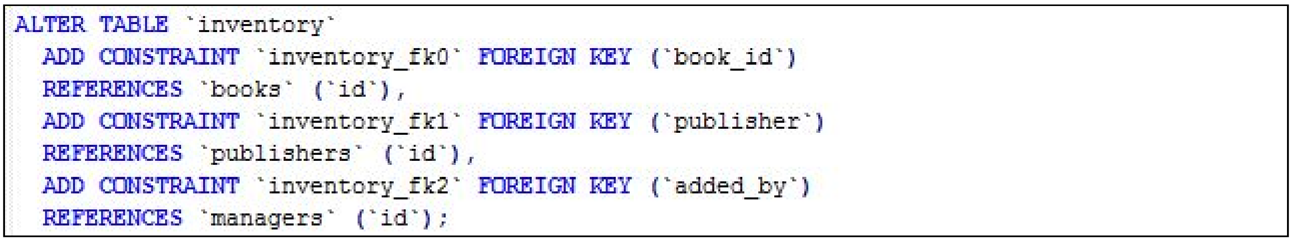
```
ALTER TABLE `books_authors`
  ADD CONSTRAINT `books_authors_fk0` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`),
  ADD CONSTRAINT `books_authors_fk1` FOREIGN KEY (`author_id`)
  REFERENCES `authors` (`id`);
```

Figure 7.23: Update table "books_authors".



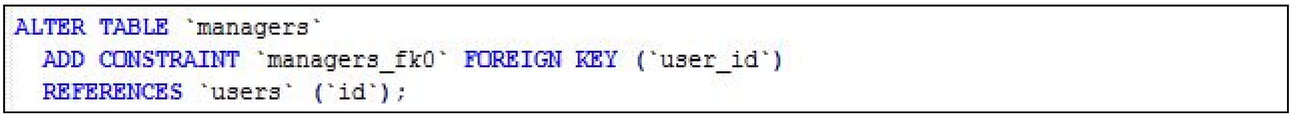
```
ALTER TABLE `feedback`
  ADD CONSTRAINT `feedback_fk0` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`),
  ADD CONSTRAINT `feedback_fk1` FOREIGN KEY (`order_id`)
  REFERENCES `orders` (`id`),
  ADD CONSTRAINT `feedback_fk2` FOREIGN KEY (`user_id`)
  REFERENCES `memebers` (`id`);
```

Figure 7.24: Update table "feedback".



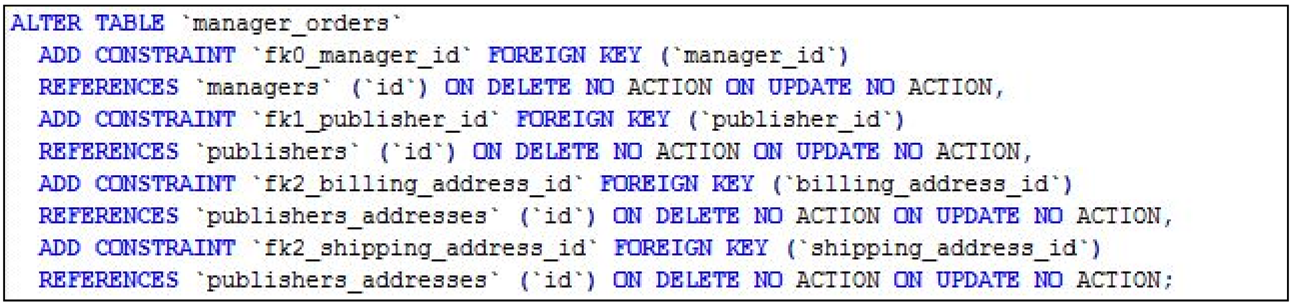
```
ALTER TABLE `inventory`
  ADD CONSTRAINT `inventory_fk0` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`),
  ADD CONSTRAINT `inventory_fk1` FOREIGN KEY (`publisher`)
  REFERENCES `publishers` (`id`),
  ADD CONSTRAINT `inventory_fk2` FOREIGN KEY (`added_by`)
  REFERENCES `managers` (`id`);
```

Figure 7.25: Update table "inventory".



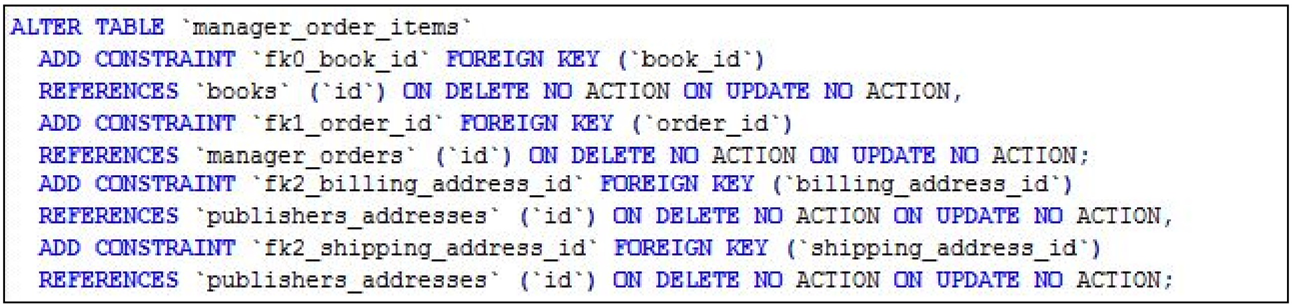
```
ALTER TABLE `managers`
  ADD CONSTRAINT `managers_fk0` FOREIGN KEY (`user_id`)
  REFERENCES `users` (`id`);
```

Figure 7.26: Update table “managers”.



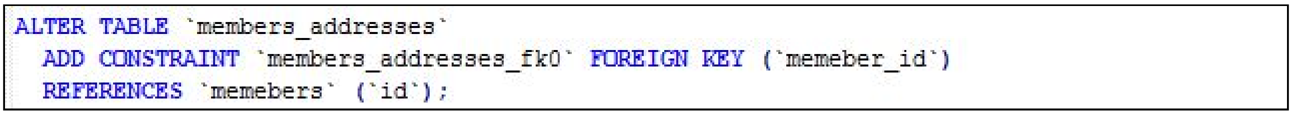
```
ALTER TABLE `manager_orders`
  ADD CONSTRAINT `fk0_manager_id` FOREIGN KEY (`manager_id`)
  REFERENCES `managers` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk1_publisher_id` FOREIGN KEY (`publisher_id`)
  REFERENCES `publishers` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk2_billing_address_id` FOREIGN KEY (`billing_address_id`)
  REFERENCES `publishers_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk2_shipping_address_id` FOREIGN KEY (`shipping_address_id`)
  REFERENCES `publishers_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Figure 7.27: Update table “manager_orders”.



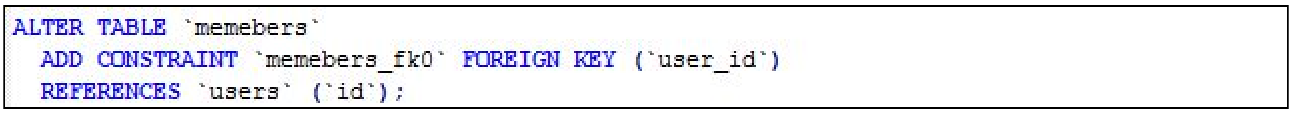
```
ALTER TABLE `manager_order_items`
  ADD CONSTRAINT `fk0_book_id` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk1_order_id` FOREIGN KEY (`order_id`)
  REFERENCES `manager_orders` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
  ADD CONSTRAINT `fk2_billing_address_id` FOREIGN KEY (`billing_address_id`)
  REFERENCES `publishers_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk2_shipping_address_id` FOREIGN KEY (`shipping_address_id`)
  REFERENCES `publishers_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Figure 7.28: Update table “manager_order_items”.



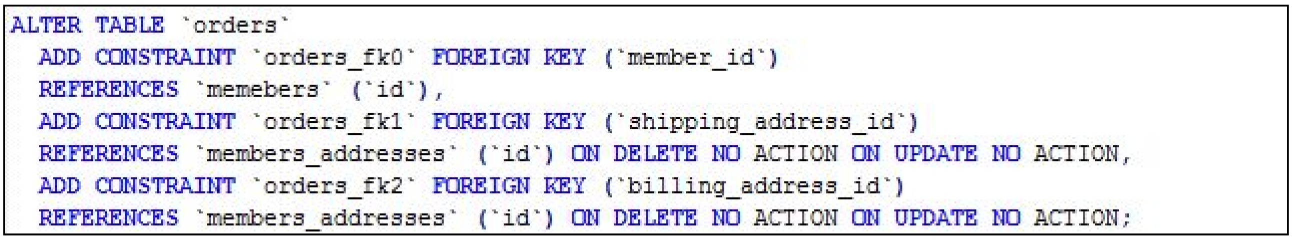
```
ALTER TABLE `members_addresses`
  ADD CONSTRAINT `members_addresses_fk0` FOREIGN KEY (`memeber_id`)
  REFERENCES `memebers` (`id`);
```

Figure 7.29: Update table “members_addresses”.



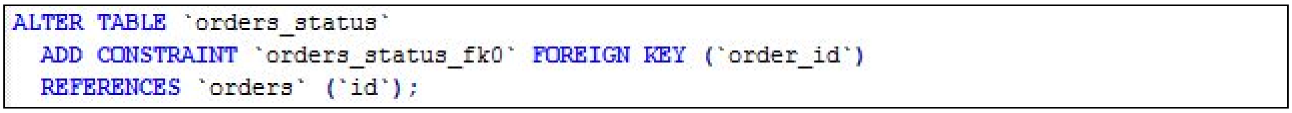
```
ALTER TABLE `memebers`
  ADD CONSTRAINT `memebers_fk0` FOREIGN KEY (`user_id`)
  REFERENCES `users` (`id`);
```

Figure 7.30: Update table “members”.



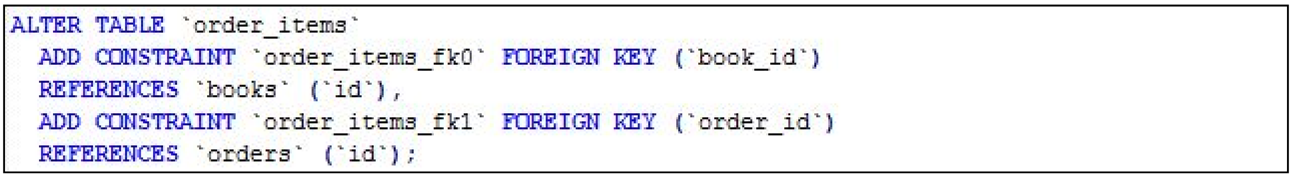
```
ALTER TABLE `orders`
  ADD CONSTRAINT `orders_fk0` FOREIGN KEY (`member_id`)
  REFERENCES `memebers` (`id`),
  ADD CONSTRAINT `orders_fk1` FOREIGN KEY (`shipping_address_id`)
  REFERENCES `members_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `orders_fk2` FOREIGN KEY (`billing_address_id`)
  REFERENCES `members_addresses` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Figure 7.31: Update table “orders”.



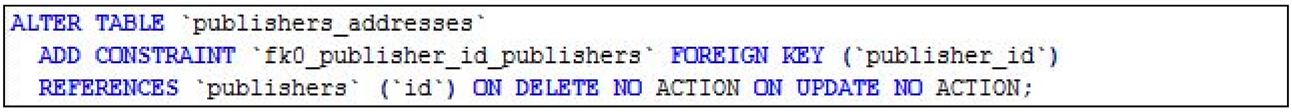
```
ALTER TABLE `orders_status`
  ADD CONSTRAINT `orders_status_fk0` FOREIGN KEY (`order_id`)
  REFERENCES `orders` (`id`);
```

Figure 7.32: Update table “orders_status”.



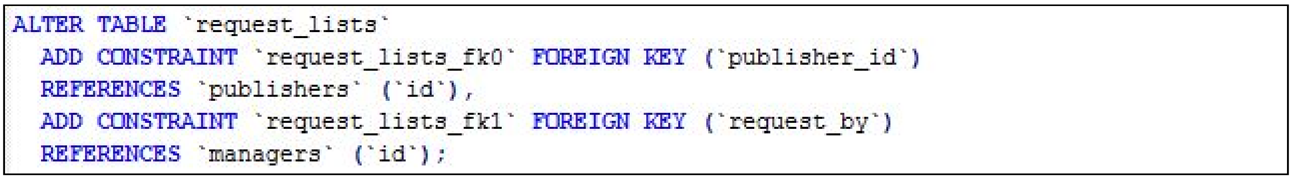
```
ALTER TABLE `order_items`
  ADD CONSTRAINT `order_items_fk0` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`),
  ADD CONSTRAINT `order_items_fk1` FOREIGN KEY (`order_id`)
  REFERENCES `orders` (`id`);
```

Figure 7.33: Update table “order_items”.



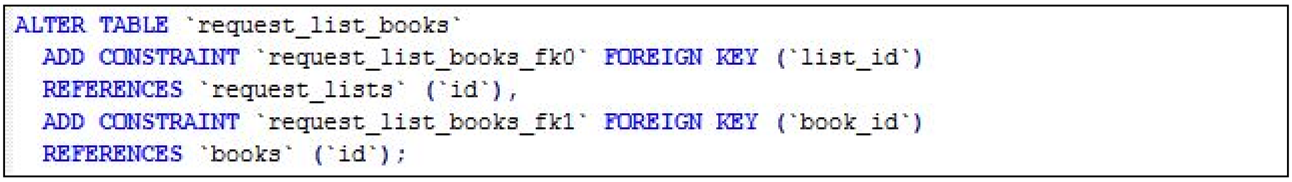
```
ALTER TABLE `publishers_addresses`
  ADD CONSTRAINT `fk0_publisher_id_publishers` FOREIGN KEY (`publisher_id`)
  REFERENCES `publishers` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Figure 7.34: Update table “publishers_addresses”.



```
ALTER TABLE `request_lists`
  ADD CONSTRAINT `request_lists_fk0` FOREIGN KEY (`publisher_id`)
  REFERENCES `publishers` (`id`),
  ADD CONSTRAINT `request_lists_fk1` FOREIGN KEY (`request_by`)
  REFERENCES `managers` (`id`);
```

Figure 7.35: Update table “request_lists”.



```
ALTER TABLE `request_list_books`
  ADD CONSTRAINT `request_list_books_fk0` FOREIGN KEY (`list_id`)
  REFERENCES `request_lists` (`id`),
  ADD CONSTRAINT `request_list_books_fk1` FOREIGN KEY (`book_id`)
  REFERENCES `books` (`id`);
```

Figure 7.36: Update table “request_list_books”.

8.0 Data Manipulation Language (DML)

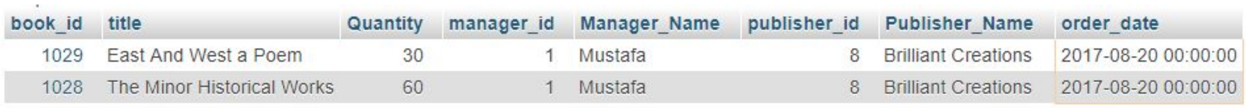
8.1 Question 1

QUERY (YOUSSEF THARAWAT)

```
SELECT manager_order_items.book_id, books.title, manager_order_items.qty AS Quantity,
manager_orders.manager_id, managers.firstname AS Manager_Name
,manager_orders.publisher_id, publishers.name AS Publisher_Name, manager_orders.order_date

FROM manager_order_items
INNER JOIN manager_orders ON manager_orders.id = manager_order_items.order_id
INNER JOIN books ON books.id = manager_order_items.book_id
INNER JOIN publishers ON publishers.id = manager_orders.publisher_id
INNER JOIN managers ON managers.id = manager_orders.manager_id
WHERE TIMESTAMPDIFF(DAY,manager_orders.order_date ,NOW()) < 30
```

RESULTS



book_id	title	Quantity	manager_id	Manager_Name	publisher_id	Publisher_Name	order_date
1029	East And West a Poem	30	1	Mustafa	8	Brilliant Creations	2017-08-20 00:00:00
1028	The Minor Historical Works	60	1	Mustafa	8	Brilliant Creations	2017-08-20 00:00:00

DESCRIPTION

The Query selects book Id, book title, book quantity renames the column to Quantity, Manager Id, Manager first name renames it to Manager_Name, publisher Id, publisher name and renames it to Publisher_Name and finally order date. The columns and attributes have been selected from manager_order_items and by joining manager_order table, books table, publishers table and managers table. Then the query connects the manager_orders table matching it up using the manager_orders.id with the manager_order_items.order_id which links the tables based on the order Id. Moreover, connecting the table books and using the book's id in books table and the book's id in manager_order_items to match the tables together. Then the query connects publishers table and using the publisher's id in publishers table and publisher's id in the manager orders table to match the tables together so I am able to use the attributes of publishers table such as publisher's name. Furthermore, connecting managers table using manager id from managers table and manager id from manager orders table to match up the tables together.

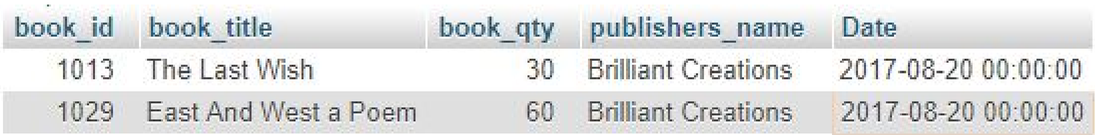
In order to calculate based on monthly basis, where is used to filter the data giving the criteria using a function called TIMESTAMPDIFF which takes 3 parameters, first is the unit of measurement which converts the two other parameters to the unit specified which is 'Day' in my case. The two other parameters which are the ORDER_DATE - NOW() (A function that captures the current date) and then if the value is less than 30 days then display it.

8.2 Question 2

QUERY (YOUSSEF THARAWAT)

```
SELECT books.id AS book_id ,books.title AS book_title ,request_list_books.qty AS book_qty ,
publishers.name AS publishers_name, request_lists.created_at AS 'Date'
FROM `request_list_books`
INNER JOIN request_lists ON request_lists.id = request_list_books.list_id
INNER JOIN books ON books.id = request_list_books.book_id
INNER JOIN publishers ON publishers.id = request_lists.publisher_id
WHERE TIMESTAMPDIFF(DAY,request_lists.created_at,NOW()) < 30
```

RESULTS



book_id	book_title	book_qty	publishers_name	Date
1013	The Last Wish	30	Brilliant Creations	2017-08-20 00:00:00
1029	East And West a Poem	60	Brilliant Creations	2017-08-20 00:00:00

DESCRIPTION

The Query selects book Id, book title and renames it to book_title, books quantity and renames it to book_qty, publishers name and renames it to publishers_name and created_at and renames it to Date. The columns and attributes have been selected from request_list_books and by joining request_lists table, books table and publishers table. The table request_lists has been connected using the request_lists.id to match up with the request_list_books.id. The table books has been connected and using the books.id and the request_list_books.book_id it matches up both tables. The table publishers has been connected and using the publishers.id and the request_lists.publisher_id it matches up both tables.

In order to calculate based on monthly basis, where is used to filter the data giving the criteria using a function called TIMESTAMPDIFF which takes 3 parameters, first is the unit of measurement which converts the two other parameters to the unit specified which is ‘Day’ in my case. The two other parameters which are the request_lists.created_at - NOW() (A function that captures the current date) and then if the value is less than 30 days then display it.

8.3 Question 3

QUERY (YOUSSEF THARAWAT)

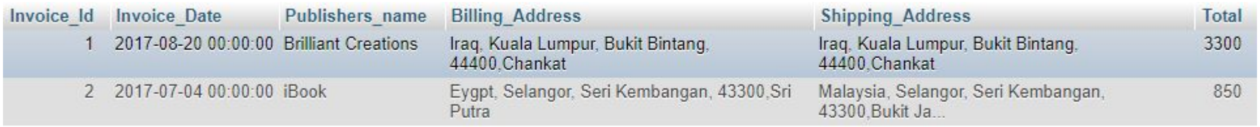
```
SELECT manager_orders.id AS Invoice_Id, manager_orders.order_date AS Invoice_Date,
publishers.name AS Publishers_name,

(SELECT CONCAT(publishers_addresses.country,', ',publishers_addresses.state,',
',publishers_addresses.city,', ',publishers_addresses.zipcode,',',publishers_addresses.street)
FROM publishers_addresses WHERE publishers_addresses.id =
manager_orders.billing_address_id,

(SELECT CONCAT(publishers_addresses.country,', ',publishers_addresses.state,',
',publishers_addresses.city,', ',publishers_addresses.zipcode,',',publishers_addresses.street)
FROM publishers_addresses WHERE publishers_addresses.id =
manager_orders.shipping_address_id,

(SELECT SUM(final_price*qty) FROM manager_order_items WHERE
manager_order_items.order_id = manager_orders.id) as Total
FROM manager_orders
INNER JOIN publishers ON publishers.id = manager_orders.publisher_id
ORDER BY manager_orders.id ASC
```

RESULTS



Invoice_Id	Invoice_Date	Publishers_name	Billing_Address	Shipping_Address	Total
1	2017-08-20 00:00:00	Brilliant Creations	Iraq, Kuala Lumpur, Bukit Bintang, 44400,Chankat	Iraq, Kuala Lumpur, Bukit Bintang, 44400,Chankat	3300
2	2017-07-04 00:00:00	iBook	Eygpt, Selangor, Seri Kembangan, 43300,Sri Putra	Malaysia, Selangor, Seri Kembangan, 43300,Bukit Ja...	850

DESCRIPTION

The Query selects manager_orders.id and renames it as Invocie_Id, manager_orders.order_date and renames it as Invoice_Date and publishers.name and renames it as Publishers_name from manager_orders table and publishers table. There are three sub-queries that are used from the logic of the query, in order to differentiate the address is it a billing address or a shipping address. CONCAT function has been used to combine the attributes together to form a single string with commas in between.

The SELECT function in the first sub-query has embedded a CONCAT function that combines all the attributes of the publisher's address from the publisher addresses table. However, the WHERE filters the selection by matching up the publisher's address.id with the manager_orders.billing_address_id so it only shows the billing address and the same sub-query but for shipping address so the query can distinguish which address to show based on the publisher’s address id. The third sub-query is for the total calculation.

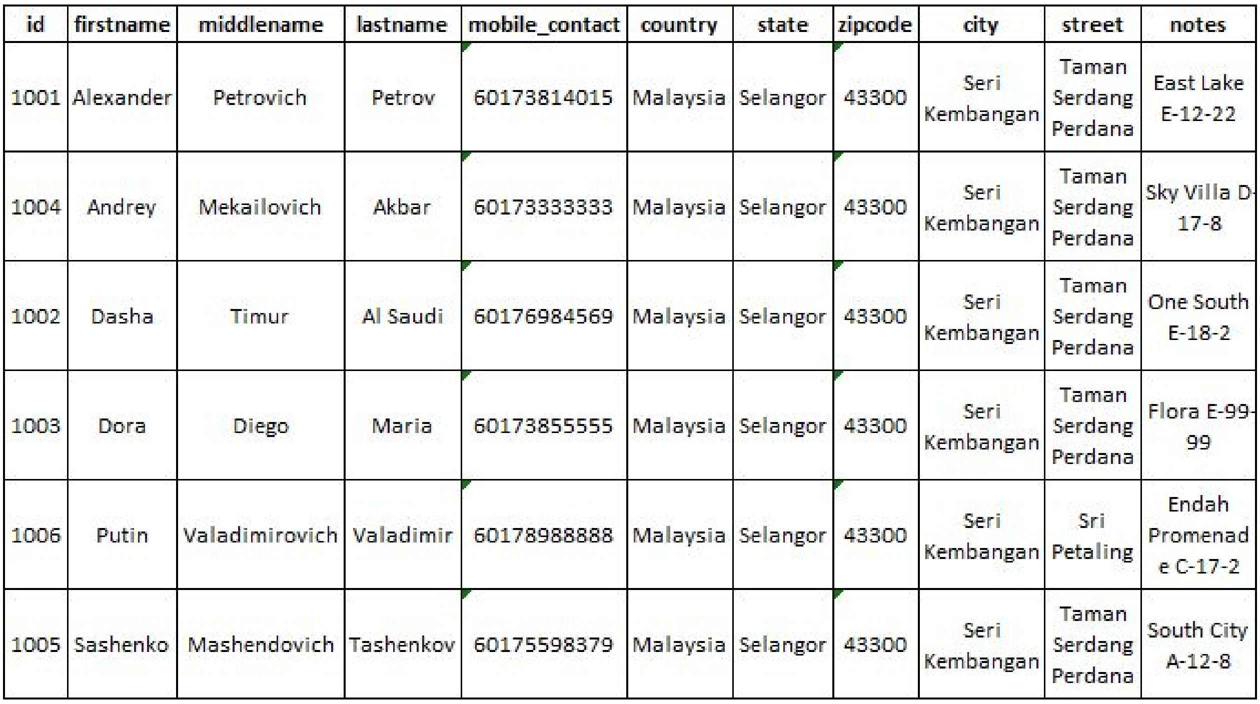
The query first matches up the ORDER_ITEMS and the ORDER using manager_order_items.order_id = manager_orders.id. Therefore, each order item will be matched up with its corresponding order. Moreover, the SUM function sums the final_price, which is the price of each item multiplied by the quantity. The query is order by an ascending order of Invoice_id using MANAGER_ORDERS.ID ASC

8.4 Question 4

QUERY (MUKABAK ORAZBEK)

```
SELECT users.id, memebers.firstname, memebers.middlename, memebers.lastname,
memebers.mobile_contact, members_addresses.country, members_addresses.state,
members_addresses.zipcode, members_addresses.city, members_addresses.street,
members_addresses.notes
FROM users
INNER JOIN memebers ON users.id = memebers.user_id
INNER JOIN members_addresses ON memebers.id = members_addresses.memeber_id
WHERE users.usertype = 1
ORDER BY memebers.firstname ASC
```

RESULTS



id	firstname	middlename	lastname	mobile_contact	country	state	zipcode	city	street	notes
1001	Alexander	Petrovich	Petrov	60173814015	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	East Lake E-12-22
1004	Andrey	Mekailovich	Akbar	60173333333	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	Sky Villa D-17-8
1002	Dasha	Timur	Al Saudi	60176984569	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	One South E-18-2
1003	Dora	Diego	Maria	60173855555	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	Flora E-99-99
1006	Putin	Valadimirovich	Valadimir	60178988888	Malaysia	Selangor	43300	Seri Kembangan	Sri Petaling	Endah Promenade C-17-2
1005	Sashenko	Mashendovich	Tashenkov	60175598379	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	South City A-12-8

DESCRIPTION

This query selects all members id, first name, middle name, last name, full address, contact number from 3 different tables via using join method, then displays data by alphabetical order (A-Z, 0-9, etc.) based on first name of users. According to the official website of Mysql.com (2017), there are main 2 ways of ordering data via alphabetically, query at the above included ascending (ASC) way, another way of ordering data is descending (DESC) way, which is reverse of first one (Z-A, 9-0, etc).

Important:

Users.usertype = 1 (Member).

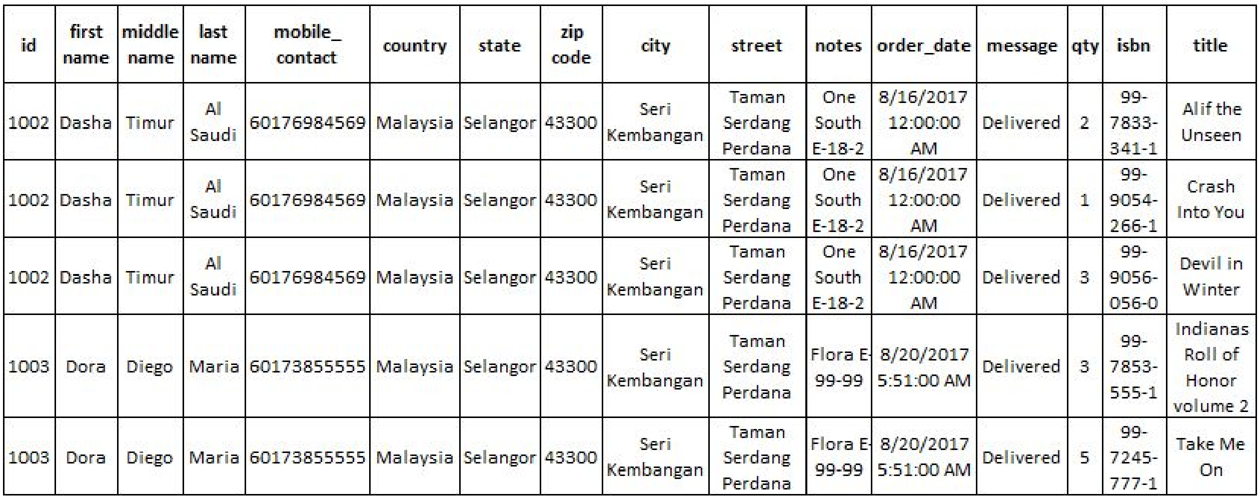
Users.usertype = 2 (Manager).

8.5 Question 5

QUERY (MUKABAK ORAZBEK)

```
SELECT users.id, memebers.firstname, memebers.middlename, memebers.lastname,
memebers.mobile_contact, members_addresses.country, members_addresses.state,
members_addresses.zipcode, members_addresses.city, members_addresses.street,
members_addresses.notes, orders.order_date, orders_status.message, order_items.qty,
books.isbn, books.title
FROM users
INNER JOIN memebers ON users.id = memebers.user_id
INNER JOIN members_addresses ON memebers.id = members_addresses.memeber_id
INNER JOIN orders ON memebers.id = orders.member_id
INNER JOIN orders_status ON orders.id = orders_status.order_id
INNER JOIN order_items ON orders.id = order_items.order_id
INNER JOIN books ON order_items.book_id = books.id
WHERE users.usertype = 1 AND orders_status.status_code = 3
ORDER BY books.title ASC
```

RESULTS



id	first name	middle name	last name	mobile_ contact	country	state	zip code	city	street	notes	order_date	message	qty	isbn	title
1002	Dasha	Timur	Al Saudi	60176984569	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	One South E-18-2	8/16/2017 12:00:00 AM	Delivered	2	99-7833-341-1	Alif the Unseen
1002	Dasha	Timur	Al Saudi	60176984569	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	One South E-18-2	8/16/2017 12:00:00 AM	Delivered	1	99-9054-266-1	Crash Into You
1002	Dasha	Timur	Al Saudi	60176984569	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	One South E-18-2	8/16/2017 12:00:00 AM	Delivered	3	99-9056-056-0	Devil in Winter
1003	Dora	Diego	Maria	60173855555	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	Flora E-99-99	8/20/2017 5:51:00 AM	Delivered	3	99-7853-555-1	Indianas Roll of Honor volume 2
1003	Dora	Diego	Maria	60173855555	Malaysia	Selangor	43300	Seri Kembangan	Taman Serdang Perdana	Flora E-99-99	8/20/2017 5:51:00 AM	Delivered	5	99-7245-777-1	Take Me On

DESCRIPTION

This query selects purchased (delivered) items via using 7 different tables, then displays ordered user's id, first name, middle name, last name, full address, contact number, date, status, book title, isbn, quantity.

Important:

Orders_status.status_code = 0 (Order Created).

Orders_status.status_code = 1 (Processing).

Orders_status.status_code = 2 (In Transit).

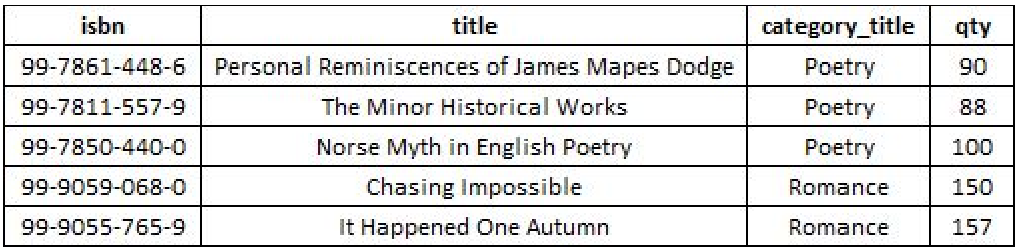
Orders_status.status_code = 3 (Delivered).

8.6 Question 6

QUERY (MUKABAK ORAZBEK)

```
SELECT books.isbn, books.title, books_category.title, inventory.qty
FROM books
INNER JOIN books_category ON books.category_id = books_category.id
INNER JOIN inventory ON books.id = inventory.book_id
ORDER BY books_category.title ASC
```

RESULTS



isbn	title	category_title	qty
99-7861-448-6	Personal Reminiscences of James Mapes Dodge	Poetry	90
99-7811-557-9	The Minor Historical Works	Poetry	88
99-7850-440-0	Norse Myth in English Poetry	Poetry	100
99-9059-068-0	Chasing Impossible	Romance	150
99-9055-765-9	It Happened One Autumn	Romance	157

DESCRIPTION

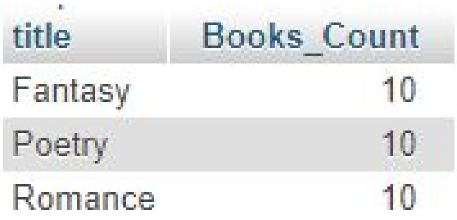
This query selects all books from books table, then displays isbn, title, category, quantity of selected books. And list of books on the screen ordered by category of it in alphabetic format. The query uses different 3 tables via join method. Namely: table books, table books_category, table inventory. The isbn and title attributes are located in books table. The category title is belongs to books_category table, and books quantity is stored in inventory table. In this case, query identifies books category title via using books.category_id = books_category.id, and the quantity of selected item identifies by books.id = inventory.book_id, and this magic trick known as join table method. According to the author of MySQL Bible (Steve, 2002), join table method plays main role in database management system, via implementing this kind of methods to their projects, programmers could get core idea to develop perfect management system.

8.7 Question 7

QUERY (MUSTAFA AHMED)

```
SELECT books_category.title,COUNT(*) as Books_Count
FROM books
INNER JOIN books_category ON books.category_id = books_category.id
GROUP BY books_category.title
```

RESULT



title	Books_Count
Fantasy	10
Poetry	10
Romance	10

DESCRIPTION

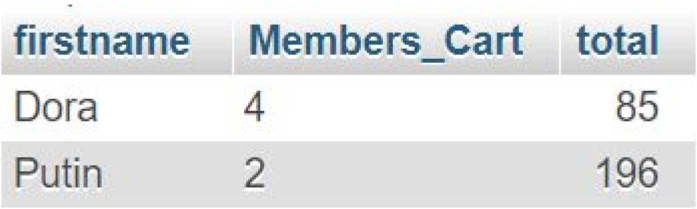
This query will aggregate all books into its categories respectively, Where it demands joining in two tables due to the nature of cardinality of many books belong to single category (1-N).

8.8 Question 8

QUERY (MUSTAFA AHMED)

```
SELECT memebers.firstname,SUM(qty) AS Members_Cart,SUM(final_price*qty) AS total
FROM `cart`
INNER JOIN memebers ON cart.member_id = memebers.id
GROUP BY memebers.firstname
```

RESULT



firstname	Members_Cart	total
Dora	4	85
Putin	2	196

DESCRIPTION

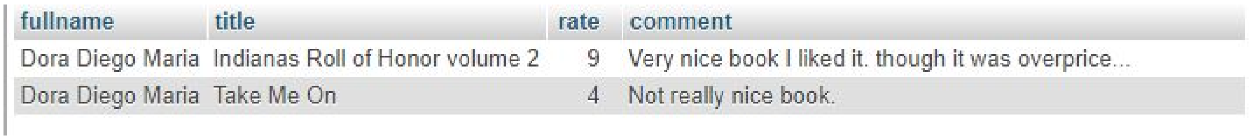
This query will sum all prices multiplied by the quantities and display quantity to each different member, which requires to join members table into this table to group by name and display it accordingly.

8.9 Question 9

QUERY (MUSTAFA AHMED)

```
SELECT  CONCAT(memebers.firstname," ",   memebers.middlename, " ", memebers.lastname)
AS fullname,books.title,feedback.rate,feedback.comment FROM feedback
INNER JOIN books ON feedback.book_id = books.id
INNER JOIN memebers ON feedback.user_id = memebers.id
```

RESULT



fullname	title	rate	comment
Dora Diego Maria	Indianas Roll of Honor volume 2	9	Very nice book I liked it. though it was overprice...
Dora Diego Maria	Take Me On	4	Not really nice book.

DESCRIPTION

This query returns full name of the customer by concatenating three separate columns and the rated book as well by joining two different tables which is *books* and *members* to retrieve the name and book title on *book_id* and *member_id* respectively.

9.0 Conclusion

In this project, students have prepared database management system for university library. In order to create a perfect database system for managing library, developers implemented more than 20 different tables for it. It means overall data is divided to small parts systematically, and number of parts equal to number of tables. Each table responsible for only part of data based on its structure, and in result, following separated design helps to increase speed of access and level of security. Also during the designing process, students considered interface requirements for different platforms. After considering all requirements, the proposed database system become more reliable, and responds to all requested hosts from any platform or any devices.

10.0 Workload Matrix

	Youssef Thawarwat (TP040164)	**Mustafa Ahmad (TP043972)**	**Orazbak Mukabak (TP040205)**
Introduction	33%	33%	33%
ER Modeling	33%	33%	33%
Business Rules	33%	33%	33%
Normalization Process	33%	33%	33%
Database Schema	33%	33%	33%
Data Dictionary	33%	33%	33%
DDL	33%	33%	33%
DML	33%	33%	33%
Overall	33%	33%	33%
Signatures			

11.0 References

Steve, S. (2002) *MySQL Bible: Chapter 14 Performance Tuning.* Wiley Publishing, Inc. New York.

MySQL.com. (2017) *MySQL 5.7 Reference Manual: Sorting Rows.* [Online] Available at: https://dev.mysql.com/doc/refman/5.7/en/sorting-rows.html.
[Accessed: 20/08/2017].

CalebTheVideoMaker2 (www.youtube.com). (2012) *MySQL Complete Series: Videos 1 – 43*. [Online] Available at: https://www.youtube.com/watch?v=6pbxQQG25Jw.
[Accessed: 19/08/2017].

Steven, J. (2013) *WebScaleSQL: MySQL For Facebook-Sized Databases.* [Online] Available at: http://www.zdnet.com/article/webscalesql-mysql-for-facebook-sized-databases.
[Accessed: 21/08/2017].

NEXTSTEP4IT (www.nextstep4it.com). (2014) *DDL (Data Definition Language) Commands in MySQL.* [Online] Available at: https://www.nextstep4it.com/ddl-statements-in-mysql.
[Accessed: 21/08/2017].