The Good Read BookStore Database

Course Section: CS605.441.31
Fall, 2015

Prepared by

Jonathan Thompson

12/2/2015

Database Design Project Document

Table of Contents

1. Introduction

This report is for the Good Read Bookstore. An simple, straight forward online book store. The intent of this project that focuses on this hypothetical online book store is firstly and most importantly to learn through doing how to design and implement a database, even if on a small scale. But beyond this very practical motivation, personally I chose the topic of an online book store due to my volunteer efforts with a non-profit Christian books distributor. As this company is a non-profit company, personnel and funding is limited. Therefore I thought this project would allow me to gain good experience in the space of online book distribution so that I would be able to better help in the future.

1.1. Scope and Purpose of Document

The scope of this document intends to cover the full spectrum of the database for the Good Read online bookstore. This first begins with covering the related requirements. These requirements include both system requirements related to software and hardware, as well as the requirements related to the functionality of the database specifically. Next this documents outlines the design of the database in detail. This section will focus on the design rationale, the entity-relationship model, the relational model, security, and other pertinent topics. Additionally this document will cover the implementation description of the database. This section focuses on features and queries of the database. Lastly this document will conclude with a CRUD matrix and some concluding remarks.

1.2. Project Objective

The objective of this project is to produce a functioning, feasible database that fulfills the needed functionality for an online bookstore through design and implementation. Additionally an objective of this project is to use and gain experience in a main stream database management management system and design tool.

2. System Requirements

This section focuses on the requirements related to the functionality related to the Good Read online bookstore database. This includes any hardware requirements and software requirements needed for operation and support of the database. Additionally this section covers the functional requirements that are to be fulfilled by the database. Lastly this section will indicate the specific database that is to be implemented.

2.1. Hardware Requirements

Hardware requirements follow the hardware requirements put forth by Microsoft for a SQL server. The following are the minimum requirements:
Processor: x64 architecture with minimum 1.5 GHz processor
Memory: Minimum 4 GB RAM
Hard disk: SAS RAID 5 or RAID 10 hard disk array

It is noted that these minimum requirements have been tested for a load of 320 users. If more through put is needed to handle a greater load or if greater performance is required, the following recommended hardware requirements are suggested:
Processor: Quad-core x64 architecture with minimum 2 GHz processor
Memory: Minimum 16 GB RAM
Hard disk: SAS RAID 5 or RAID 10 hard disk array

2.2. Software Requirements

The following software requirements relate the OS requirements for specific instances of SQL server put forth by Microsoft. Only those for the principal SQL Server editions:

SQL Server Edition	OS
SQL Server Enterprise	Windows Server 2012 R2 Datacenter 64-bit Windows Server 2012 R2 Standard 64-bit Windows Server 2012 R2 Essentials 64-bit Windows Server 2012 R2 Foundation 64-bit Windows Server 2012 Datacenter 64-bit Windows Server 2012 Standard 64-bit Windows Server 2012 Essentials 64-bit Windows Server 2012 Foundation 64-bit
SQL Server Business Intelligence	Windows Server 2012 R2 Datacenter 64-bit Windows Server 2012 R2 Standard 64-bit Windows Server 2012 R2 Essentials 64-bit Windows Server 2012 R2 Foundation 64-bit Windows Server 2012 Datacenter 64-bit Windows Server 2012 Standard 64-bit Windows Server 2012 Essentials 64-bit Windows Server 2012 Foundation 64-bit

SQL Server Edition	OS
SQL Server Standard	Windows Server 2012 R2 Datacenter 64-bit Windows Server 2012 R2 Standard 64-bit Windows Server 2012 R2 Essentials 64-bit Windows Server 2012 R2 Foundation 64-bit Windows Server 2012 Datacenter 64-bit Windows Server 2012 Standard 64-bit Windows Server 2012 Essentials 64-bit Windows Server 2012 Foundation 64-bit Windows 8.1 64-bit Windows 8.1 Pro 64-bit Windows 8.1 Enterprise 64-bit Windows 8 64-bit Windows 8 Pro 64-bit Windows 8 Enterprise 64-bit Windows 10 64-bit

Additionally it is also required to have installed the .NET 3.5 SP1.
Lastly while these requirements are listed relative to Windows OS, similar 64 bit OS and packages are required for usage on a Unix\Linux based OS.

2.3. Functional Requirements

The following are requirements for the Good Read online bookstore database. The database shall have the functionality to support:

User Interface

1. Users to create and edit profile
2. Users to enter and use multiple forms of credit card payment
3. Users to enter and use multiple shipping information
4. Users to view their order history
5. Users to search for books based upon genre, author, publisher, or other book related information
6. Users to purchase books through placing an order

Book Inventory

1. The storage of books, maintaining information specific to the book such as title, publish date, cost etc.
2. The storage of publishers
3. The storage of authors
4. The storage of genres
5. The storage of books sales relative to a specific book
6. The storage of stock for each book

Order Information

1. The storage of all orders
2. The storage of order statuses of all orders
3. The ability to maintain shipping information
4. The ability for the user to select a shipping method

2.4. Database Requirements

For this project the database that was utilized was mySQL database version 5.6.26 Community Server.

3. Database Design Description

The database for the Good Reads Bookstore focuses on a simple, straight forward layout that divides the database coverage into 3 main areas: customer information, book information, and order information.

Customer information area has as entities CUSTOMER, CUSTOMER_PAYMENT, BILL_ADDRESS, SHIP_ADDRESS, and ORDER_HISTORY. CUSTOMER contains on attributes related to customer identification such as the customer's name, date of birth, and phone, among others. A customer can have multiple forms of credit card payment. This information is contained in the CUSTOMER_PAYMENT entity. The CUSTOMER_PAYMENT entity holds the information for credit card payment instances. A CUSTOMER also will have one billing address to be associated with the account. That is captured in the BILL_ADDRESS entity and contains attributes for an address used for billing. A customer may have many entered and saved shipping addresses for orders and this seen in the SHIP_ADDRESS entity. This entity contains attributes for addresses used for shipping for a customer. A customer also has an order history that stores their previous orders. This is seen in the ORDER_HISTORY entity.

Book information area has entities BOOK, AUTHOR, BOOK_AUTHOR, PUBLISHER, GENRE, BOOK_GENRE, and BOOK_ORDER. The BOOK entity contains attributes related to the descriptive information of a book, such as title, edition, and cost. A book can be written by multiple authors and an author can author many books. Then we have the AUTHOR entity and it contains attributes for information related to the author. BOOK_AUTHOR is an entity that associates an author to a book, allowing for multiple authors to be associated to a book and allowing for an author to be associated to multiple books. A book also must be published and for this there is a PUBLISHER entity that contains attributes which describe a publisher. Additionally a book can be described by its genre or genres. For this there is firstly a GENRE reference table

for various genres. BOOK_GENRE is an entity that associates genres to a book, allowing for a book to be described by many genres and for a genre to be associated to multiple books. A book is associated to an order of that book and the order of a specific book may be just one book ordered among many in a customer's order. To allow for this there is a BOOK_ORDER table that holds attributes for the order of a book and has the attribute count to indicate the number of the book ordered.

Order information area has entities FULL_ORDER, ORDER_STATUS, INVOICE, and SHIPPING_METHOD. An order comprises ordering multiple books, that is BOOK_ORDERS. It is placed by a customer, CUSTOMER, has the customer's shipping address, SHIPPING_ADDRESS, and is recorded in a customer's order history, ORDER_HISTORY. The FULL_ORDER entity contains attributes for the order of books for a customer. A FULL_ORDER is also related to an ORDER_STATUS entity and has attributes for the various statuses of an order, such as processed and delivered. Connected to the FULL_ORDER entity is a reference entity for shipping methods called SHIPPING_METHOD. Additionally an order produces invoices to the customer for payment. For this there is the INVOICE entity which contains information relating to the total cost of an order from the order and shipping costs. This entity is also directly related to customer payment. Related to the shipping information, SHIPPING_METHOD holds the attributes related to the shipping options coming from the FULL_ORDER.

3.1 Design Rationale

The database design for the Good Read Bookstore employed some design decisions that warrant some rationale. In order to do this, this section progresses through the design and highlights pertinent design choices and their rationale.

Within the book area, first to highlight is the fact that the BOOK entity is connected to the PUBLISHER entity via a non-identifying relationship. This is justified as a book may or may not be published but still exists as a book written by a certain author or authors. Thus the existence of a book does not strictly require a publisher. Also in this area are two intersection entities BOOK_AUTHOR and BOOK_GENRE which are each involved in two identifying relationships. For both of these intersection entities the identifying relationships are justified as each intersection entity is strongly dependent on the existence of each of the contributing entities in the relationship. For example the instance that represents a particular author and book could not exist without the existence of the author and book. It is

also noted that the choice to use both intersection entities is warranted as first a book can be written by multiple authors and each author can write many books. Secondly it is noted that a certain genre can be associated to multiple books and that one book can be described by being multiple genres. Finally the entity BOOK_ORDER is also an intersection entity between BOOK and ORDER which has two identifying relationships. This is justified as BOOK_ORDER represents a book that is being ordered as part of a larger order. That larger order, FULL_ORDER, is the composition of potentially many book orders. Thus this entity could not exist without a FULL_ORDER and a BOOK. Additionally this entity allows for a book to be involved in multiple orders and for an order to have many different books as part of the order.

Within the order area first noted is the FULL_ORDER entity, which is central in this area. The FULL_ORDER entity is in an identifying relationship with CUSTOMER. This is justified as the notion of an order could not exist independently of a customer. Indeed an order only exists because a customer exists to make an order. Another item to note is that ORDER_HISTORY is connected to FULL_ORDER via a non-identifying relationship. This is also justified as ORDER_HISTORY may contain 0 to many orders and thus can exist apart from FULL_ORDER. Additionally in this area is the ORDER_STATUS table which is fully dependent on the FULL_ORDER table and thus is related using an identifying relationship. But the minimum cardinality is zero allowing a FULL_ORDER to not need to produce an ORDER_STATUS immediately. Finally the INVOICE is the intersection entity of FULL_ORDER, and CUSTOMER_PAYMENT via non-identifying relationships. This allows for limiting the primary key migration into the INVOICE entity by using non-identifying relationships with non-null constraints on foreign keys.

3.2. E/R Model

This section covers the entity-relationship diagram produced for this database. The entity-relationship diagram follows IE form and crows foot notation.

3.2.1. Entities

BOOK
Entity that is for book instances. Each book sold in the bookstore will be entered in this entity. The primary key used is the ISBN number. The

BOOK entity has the attributes Title, Edition, Weight_oz, Publish_date, Cost, Stock_count, and Page_count. There is also a boolean attribute Hardcover and Softcover which specify whether the book has a hardcover or not. Additionally there is a boolean flag Reorder that is set to 0 default. The entity has as foreign keys Publisher_id.

PUBLISHER
Entity that is for publisher instances. All publishers that are related to books sold in the bookstore will be stored in this entity. The primary key is Publisher_id attribute. The attributes for this entity first is for the publisher name is called Name. The remaining attributes are related to the publisher's address and they are Street_name, Street_number, City, State, and Country.

AUTHOR
Entity that is for author instances. All authors that are related to books sold in the bookstore will be stored in this entity. The primary key attribute for this entity is Author_id. The other attributes are First_name, Last_name, and Middle_init.

BOOK_AUTHOR
Entity that is an intersection entity between BOOK and AUTHOR. This entity has instances that represent an author writing a specific book. This entity allows for the many-to-many relationship between BOOK and AUTHOR to be reconciled. This entity contains two attributes which together forms the primary key. They are Author_id and ISSBN.

GENRE
Entity that is a reference entity which holds the various genres that could describe a book. The only attributes are Genre_id,the primary key, and Description which gives a description of the specific instance. The instances in this table and how they relate to Genre_id are:
id = 1: Adventure
id = 2: Romance
id = 3: Mystery
id = 4: Science Fiction
id = 5: Horror
id = 6: Humor
id = 7: Children
id = 8: Action
id = 10: Science
id = 11: Religious
id = 12: Self-Improvement

id = 13: History
id = 14: Non-Fiction
id = 15: Education

BOOK_GENRE
Intersection entity that represents a book being related to a genre. This allows for a genre to be associated to multiple books and for one book to be associated to multiple genres. The only attributes are Genre_id and ISBN which together form the primary key.

BOOK_ORDER
This intersection entity is for instances of a book being ordered. This intersection entity allows for an ORDER to have multiple books and for a book to be associated to multiple orders. The attributes Order_id, Customer_id, and ISBN form the primary key. The entity has the Count attribute representing the number of copies of a book purchased, the Sale attribute which is for the cost of the book order, and Total_weight_oz which describes the full weight in ounces for the book or books ordered.

FULL_ORDER
Entity that is for instances of an order. A FULL_ORDER is composed of one to many BOOK_ORDERs. That is an order is related to potentially many ordered books. The primary key is composed of the attributes Order_id and Customer_id, where Customer_id is migrated from the table CUSTOMER. The attributes History_id, Ship_address_id, and Method_id are both foreign keys from the entities ORDER_HISTORY, SHIP_ADDRESS, and SHIPPING_METHOD, respectively. The attribute Order_date is to be populated with the date of order. The attribute Total_books is to hold the total number of books ordered. The attribute Total_weight is the total weight in ounces for the full order. Finally the attribute Book_cost is for the total cost for all the books purchased that does not include tax or shipping.

SHIPPING_METHOD
A reference entity for the selection of a shipping method. The attribute Method_id is the primary key. The attribute Cost_per_oz is the cost in cents per ounce for the shipping method. The attribute Description describes the shipping method relative to Method_id. The instances in this table and how they relate to Method_id are:
id = 1: Ground
id = 2: Overnight
id = 3: Priority

ORDER_HISTORY
Entity to maintain a history of the orders for a customer. The only two attributes form the primary key. They are History_id and Customer_id, where Customer_id is migrated from the CUSTOMER entity.

ORDER_STATUS
Entity that is information related to the status of an order. The primary key is formed from the attributes Status_id, Order_id, and Customer_id. The next set of attributes are boolean attributes indicating when a shipping event has occurred. These attributes are Delivered, Shipped, and Processed. The final set of attributes hold the dates when these events have happened. These are Delivered_date, Shipped_date, and Processed_date.

INVOICE
Entity for instances of an invoice produced from an order. An invoice is primarily for payment information. The primary key is the attribute Invoice_id and the attributes Order_id, Customer_id, and Payment_id are foreign keys. The attribute Payment_received is a boolean to indicate the order cost was paid. The attribute Total_cost is the total value of the order that includes the tax cost, and the shipping cost. The attribute Ship_cost is the total cost for shipping of the order. The attribute Tax_cost is the tax cost for the order.

CUSTOMER
Entity to hold instances of a customer. The primary key is the attribute Customer_id. The attributes First_name, Last_name, and Middle_init are attributes for the customer name. The attribute DOB is for the date of birth of the customer. The Creation_date is an attribute to hold the date when the customer profile was created. The Email attribute holds the email address of the customer. The attribute Primary_phone is for the main contact phone number of the customer. The Password attribute is for the holding the password of the customer.

SHIP_ADDRESS
Entity for instances of shipping addresses for a customer. The primary key is the Ship_address_id. The attributes Street_address, City, State, Zip, and Unit_number form the shipping address for a customer. The attribute Customer_id is a foreign key.

BILL_ADDRESS
Entity for instances of a billing address for a customer. The primary key is the Bill_address_id. The attributes Street_address, City, State, Zip, and Unit_number form the billing address for a customer. The attribute Customer_id is a foreign key.

CUSTOMER_PAYMENT
Entity for instances of credit card payments for a customer. The primary key is the attribute is Payment_id. Customer_id is a foreign key. The attribute Credit_card_type holds the type of credit card. The attributes First_name, Middle_init, and Last_name are for the name on the credit card. The attribute Expire_date is for the credit card expiration date. The attribute Security_code is for the security code on the card. The attribute Credit_card_number is for the number of the credit card.

3.2.2. Relationships

This section seeks to provide a detailed description of the relationships in the Good Read Bookstore database. The convention used for each relationship is Entity1-to-Entity2. This corresponds to the relationship between Entity1 and Entity2.

PUBLISHER-to-BOOK
This relationship represents that a publisher publishes a book, where the parent is PUBLISHER and the child is BOOK. It is a non-identifying relationship as a book will still exist prior to publication. BOOK has a zero-to-many relationship with PUBLISHER as a publisher can publisher zero to many books. PUBLISHER as has one-to-one constraint as a BOOK instance that is to be sold can only be published by one publisher.

AUTHOR-to-BOOK_AUTHOR
This relationship represents that an author authors, either singly or jointly, a book. The parent entity is AUTHOR and the child entity is BOOK_AUTHOR. This is an identifying relationship with the BOOK_AUTHOR entity having a one-to-many constraint as an author can author one to many books. The AUTHOR entity has a one-to-one constraint with BOOK_AUTHOR.

BOOK-to-BOOK_AUTHOR
This relationship represents that a book is authored by one to many authors. This is an identifying relationship where BOOK where is parent and BOOK_AUTHOR is the child. BOOK_AUTHOR has a one-to-many

constraint with BOOK as a book may have one to many authors. BOOK has a one-to-one constraint with BOOK_AUTHOR.

GENRE-to-BOOK_GENRE
This relationship represents that a genre describes one to many books. This is an identifying relationship that has GENRE as the parent and BOOK_GENRE as the child. BOOK_GENRE has a one-to-many constraint with GENRE showing that one genre can describe one to many books. GENRE has a constraint of one-to-one showing that the BOOK_GENRE entity is associated to one genre.

BOOK-to-BOOK_GENRE
This relationship represents that a book can be described by many genres. This is an identifying relationship that has BOOK as the parent and BOOK_GENRE as the child. BOOK has a constraint of one-to-many with BOOK_GENRE as one book can be described by many genres. BOOK_GENRE has a structural constraint of one-to-one.

BOOK-to-BOOK_ORDER
This relationship represents that a book is part of an order for that book. More specifically that a certain book can be part of many orders for that specific book. This relationship is an identifying relationship with the parent being BOOK and the child being BOOK_ORDER as an order for a certain book could not exist without the book existing. The constraint for BOOK_ORDER relative to BOOK is zero to many as a book may be involved in zero-to-many orders for that specific book. The constraint for BOOK is one-to-one as a book order for a certain book can be for that one specific book.

BOOK_ORDER-to-FULL_ORDER
This relationship represents that an order is composed of book orders. This is an identifying order as a book order only exists if there is an order placed that orders that book. The constraint on BOOK_ORDER relative to BOOK is one-to-many as an order must order at least one book but can also order many books. The constraint on BOOK is one-to-one. In this relationship FULL_ORDER is the parent and BOOK_ORDER is the child.

FULL_ORDER-to-SHIPPING_METHOD
The relationship represents that an order has a shipping method associated to it. This is a non-identifying relationship as a shipping method exists independent of an order. The constraint on SHIP_METHOD relative to FULL_ORDER is zero-to-one. This signifies that at the time of order an

FULL_ORDER need not specify a shipping method but if it does it can have no more than one shipping method. The constraint on SHIP_METHOD is one-to-many as a shipping method may be assigned to zero orders or to many orders. The parent is SHIPPING_METHOD and the child is FULL_ORDER in the relationship.

FULL_ORDER-to-ORDER_STATUS
This relationship represents that an order has an order status. This is an identifying relationship as an ORDER_STATUS cannot exist without a FULL_ORDER. The parent is FULL_ORDER and the child is ORDER_STATUS with the structural constraint on ORDER_STATUS being zero-to-one. This allows for an order to not immediately need for an order status to be produced but has that an order can only produce one order status instance. The constraint on FULL_ORDER relative to ORDER_STATUS is one-to-one.

FULL_ORDER-to-INVOICE
This relation represents that an order produces an invoice. This is a non-identifying relationship with FULL_ORDER being the parent and INVOICE being the child. The structural constraint on INVOICE relative to FULL_ORDER is one-to-many, showing that an order may produce one or many invoices to be paid. The structural constraint on FULL_ORDER relative to INVOICE is one-to-one showing that each invoice must be related to only one order.

FULL_ORDER-to-CUSTOMER
This relationship represents that an order is placed by a customer. This is an identifying relationship since there must be a customer to place an order. The parent in the relationship is CUSTOMER and the child is FULL_ORDER. The structural constraint on FULL_ORDER relative to CUSTOMER is zero-to-many as a customer may place zero to many orders. The constraint on CUSTOMER is one-to-one showing that an order is related to only one customer.

FULL_ORDER-to-ORDER_HISTORY
This relationship represents that an order is part of the order history of a customer. This is a non-identifying relationship with the parent being ORDER_HISTORY and the child being FULL_ORDER. The structural constraint on ORDER_HISTORY relative to FULL_ORDER is one-to-one since an order can only be related to the order history of one customer. The constraint on FULL_ORDER is zero-to-many as an order history of a customer may have zero or many orders.

FULL_ORDER-to-SHIP_ADDRESS
This relationship represents that an order is shipped to a shipping address. This is a non-identifying relationship as both entities exists independently of the other. The parent is SHIPPING_ADDRESS and the child is FULL_ORDER. The structural constraint on SHIP_ADDRESS relative to FULL_ORDER is one-to-one as an order can only be sent to one selected ship address. The constraint on FULL_ORDER relative to SHIP_ADDRESS is zero-to-many as a certain stored shipping address may be used for zero orders or used for many orders.

ORDER_HISTORY-to-CUSTOMER
This relationship represents that a customer has an order history. This is an identifying relationship as an order history of a customer requires there be a customer. The parent is CUSTOMER and the child is ORDER_HISTORY. The structural constraint on CUSTOMER relative to ORDER_HISTORY is one-to-one as an order history must have one and only one customer associated to it. The constraints on ORDER_HISTORY is zero-to-one showing that a customer may have no order history but at max can only have one history of orders.

CUSTOMER-to-SHIP_ADDRESS
The relationship represents that a customer has shipping addresses stored. This a non-identifying relationship since the address and customer both exist independent of one another. The parent is CUSTOMER and the child is SHIP_ADDRESS. The structural constraint on SHIP_ADDRESS relative to CUSTOMER is zero-to-many, showing that customer may have many shipping addresses stored but is not required to store any initially. The constraints on CUSTOMER is one-to-one, requiring a shipping address to be associated to only one customer.

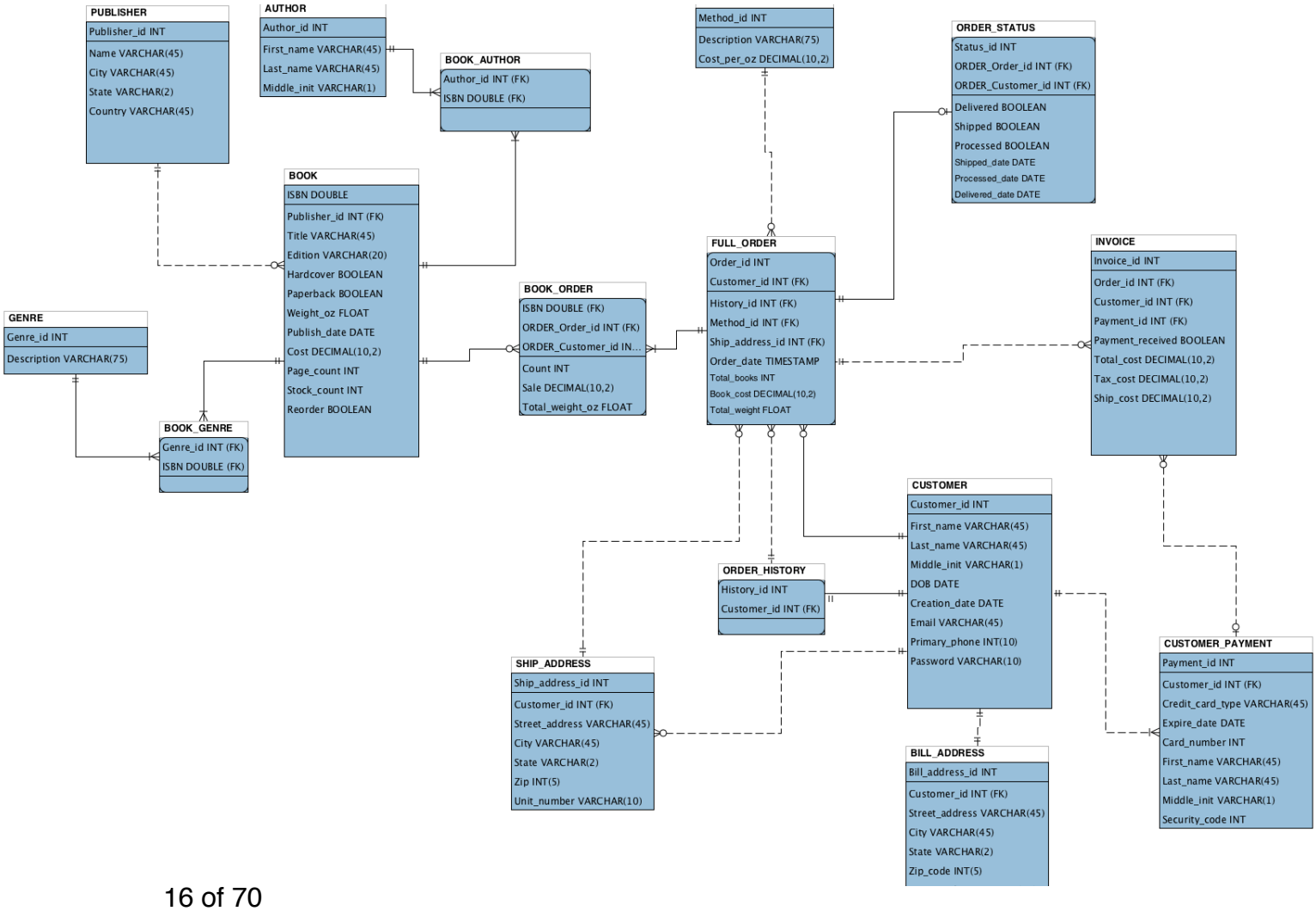
CUSTOMER-to-BILL_ADDRESS
This relationship represents that a customer has a billing address stored. This a non-identifying relationship since the address and customer both exist independent of one another. The parent is CUSTOMER and the child is BILL_ADDRESS. The structural constraint on BILL_ADDRESS relative to CUSTOMER is one-to-one, requiring a customer to have only one billing address. Likewise the constraint on CUSTOMER is one-to-one.

CUSTOMER-to-CUSTOMER_PAYMENT
This relationship represents that a customer has credit card payment methods. This is a non-identifying relationship with the parent being CUSTOMER and the child CUSTOMER_PAYMENT. The structural

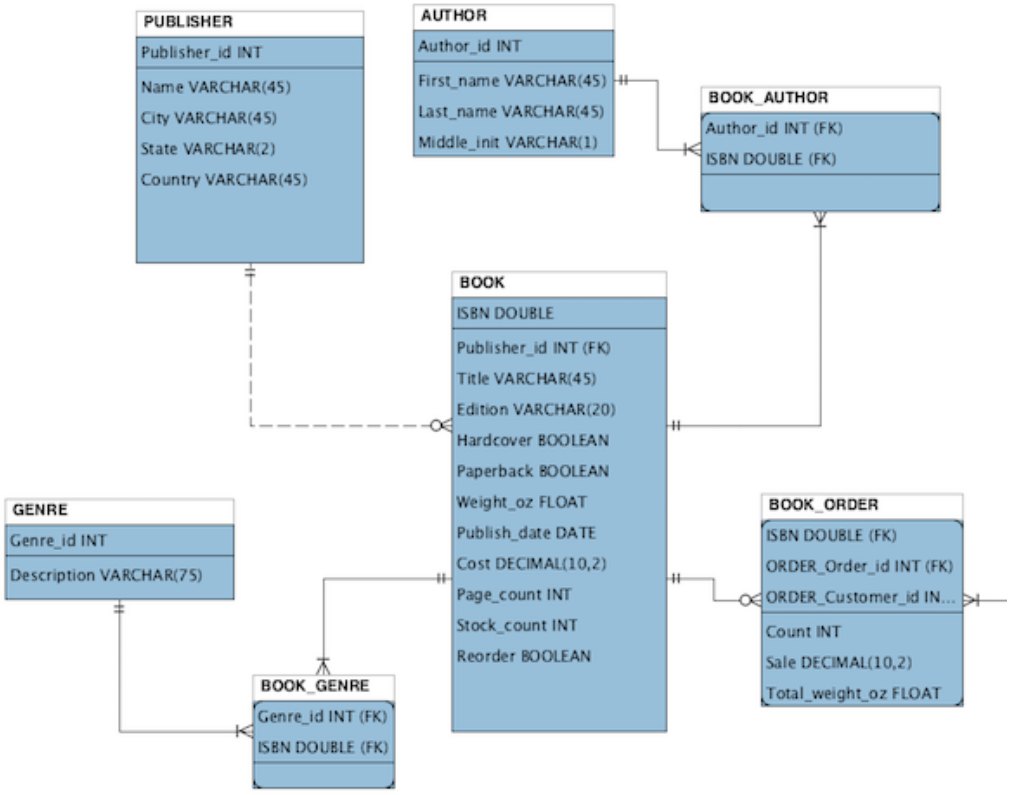
constraint on CUSTOMER_PAYMENT relative to CUSTOMER is one-to-many showing that a customer is required to have one credit card payment entered and is allowed to have many. The constraint on CUSTOMER is one-to-one showing that a credit card payment can only be associated with one customer.

INVOICE-to-CUSTOMER_PAYMENT
This relationship represents that an invoice is paid by a customer's credit card payment method. This is a non-identifying relationship as both are independent in existence from one another. The parent is CUSTOMER_PAYMENT and the child is INVOICE. The structural constraint on CUSTOMER_PAYMENT relative to INVOICE is one-to-one showing that an invoice must be paid for in its entirety by one payment method of the customer. The structural constraint on INVOICE is zero-to-many as a customer's credit card payment method may pay zero invoices or many invoices.

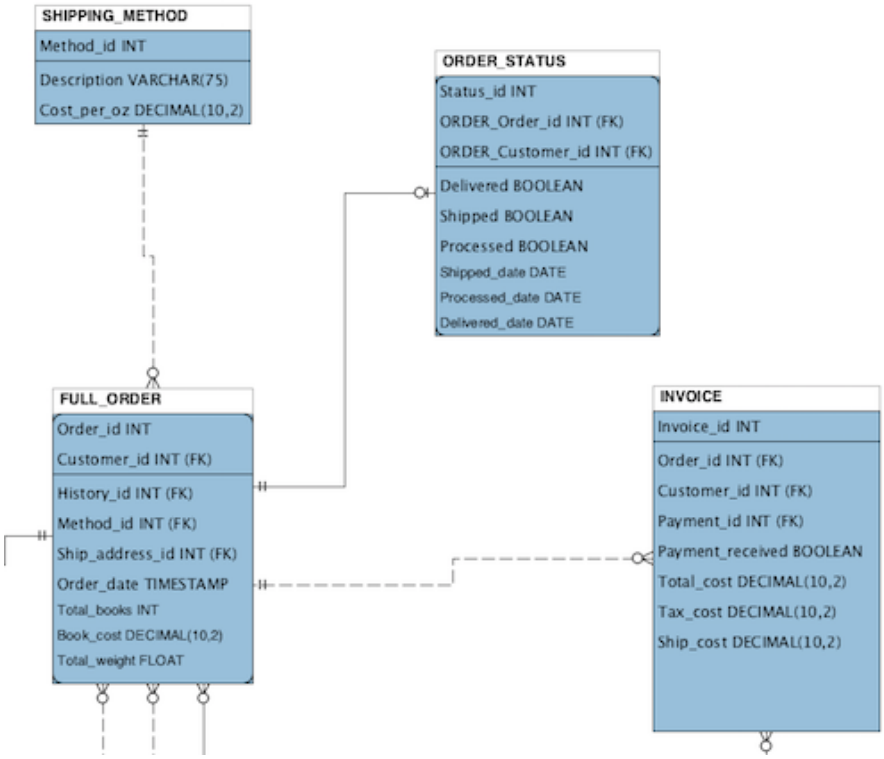
3.2.3. E/R Diagram



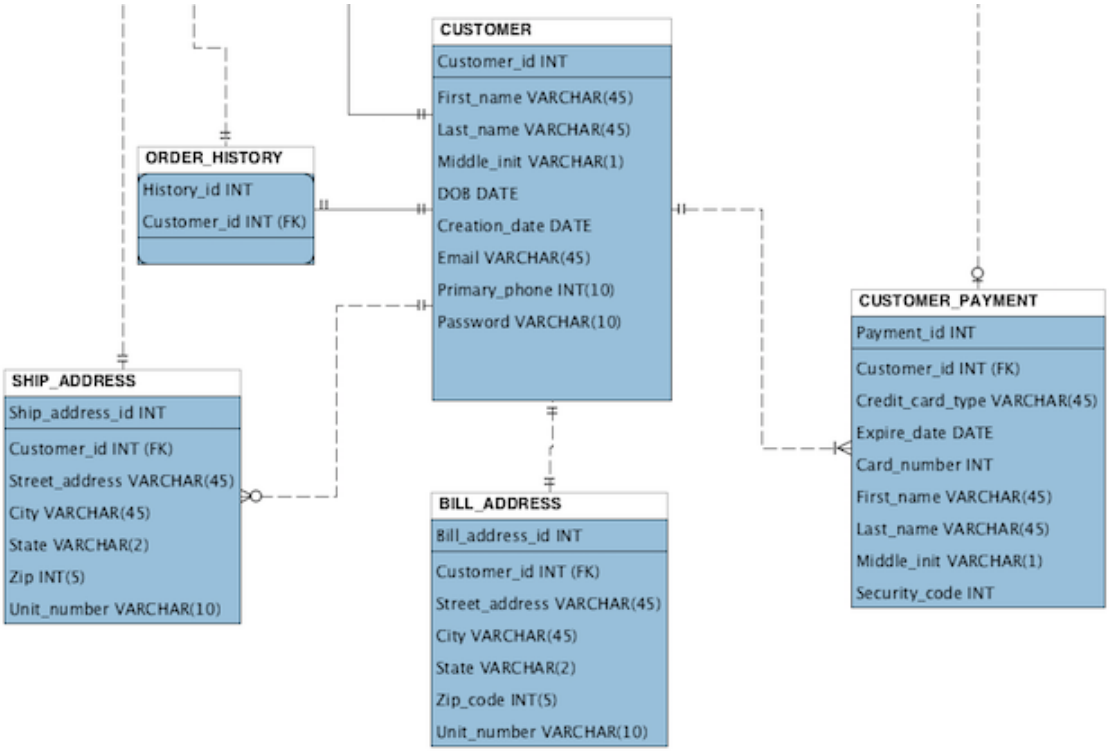
Book Section



Order Sections



Customer Sections



3.3. Relational Model

The following section covers the relational model employed in the Good Read Bookstore database. This section first presents the data dictionary that describes in detail all the attributes in the database. The following section covers the integrity rules for the database. Next the operations involved are described. The remaining sections focuses on security, backup, and other possible E/R designs that were considered.

3.3.1 Data Dictionary

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
GENRE	Genre_id	Primary Key - represents genre type	Int	NA	Primary Key	Y	9 numeric, positive digits
BOOK_GENRE	Genre_id	Migrated foreign key from GENRE	Int	NA	Part of Primary Key	Y	9 numeric, positive digits
BOOK_GENRE	ISBN	Migrated foreign key from BOOK	Double	NA	Part of Primary Key	Y	9 numeric, positive digits
PUBLISHER	Publisher_id	Primary Key to identify a publisher	Int	NA	Primary Key	Y	9 numeric, positive digits
PUBLISHER	Name	The name of the publishing company	VARCHAR	45	None	Y	All character values
PUBLISHER	Street_ address	Street address for publisher	VARCHAR	45	None	Y	All character values
PUBLISHER	City	City name	VARCHAR	45	None	Y	All character values
PUBLISHER	State	State abbreviation	VARCHAR	2	None	N	A-Z
PUBLISHER	Country	Country	VARCHAR	45	None	Y	A-Z,a-z
AUTHOR	Author_id	Primary Key to identify author	Int	NA	Primary Key	Y	9 numeric, positive digits
AUTHOR	First_ name	First name of author	VARCHAR	45	None	Y	A-Z,a-z
AUTHOR	Last_ name	Last name of author	VARCHAR	45	None	Y	A-Z,a-z

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
BOOK	Edition	Edition of book	VARCHAR	20	None	Y	All characters
BOOK	Hardcover	Flag to indicate if book has a hardcover	Boolean	NA	None	N	T/F
BOOK	Paperback	Flag to indicate if book has a softcover	Boolean	NA	None	N	T/F
BOOK	Weight_oz	Weight of book in ounces	Float	NA	None	Y	9 numeric, positive digits including decimal point
BOOK	Publish_date	Date book version was published	Date	NA	None	Y	Date format
BOOK	Cost	Cost of book in US currency	Decimal	10,2	None	Y	9 numeric, positive digits including decimal point
BOOK	Page_count	Number of pages in book	Int	NA	None	N	9 numeric, positive digits
BOOK_ORDER	Order_id	Migrated foreign key from ORDER.	Int	NA	Part of Primary Key	Y	9 numeric, positive digits
BOOK_ORDER	ISBN	Migrated foreign key from ORDER.	Int	NA	Part of Primary Key	Y	9 numeric, positive digits
BOOK_ORDER	Book_sales_id	Migrated foreign key from ORDER.	Int	NA	Referential Integrity	Y	9 numeric, positive digits

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
FULL_ORDER	Order_date	Date and time order was placed	TIME STAMP	NA	None	Y	Calendar date and times
FULL_ORDER	Total_weight	Total weight of full order in ounces	Float	NA	None	N	9 numeric, positive digits including decimal point
FULL_ORDER	Book_cost	Amount of sale price due to book cost	Decimal	10,2	None	N	9 numeric, positive digits including decimal point
ORDER_ HISTORY	History_id	Unique identifier for order history	Int	NA	Part of primary key	Y	9 numeric, positive digits
ORDER_ HISTORY	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Part of primary key	Y	9 numeric, positive digits
SHIP_ ADDRESS	Ship_address -id	Unique identifier for shipping address	Int	NA	Primary key	Y	9 numeric, positive digits
SHIP_ ADDRESS	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
SHIP_ ADDRESS	Street_ address	Street address for shipping	VARCHAR	45	None	Y	Alpha-numeric and special character
SHIP_ ADDRESS	City	City for shipping	VARCHAR	45	None	Y	Alpha-numeric and special character

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
BOOK_ORDER	Count	Number of copies ordered for a book	Int	NA	None	Y	9 numeric, positive digits
BOOK_ORDER	Sale	Amount of sale for book copies	Decimal	10,2	None	Y	9 numeric, positive digits including decimal point
FULL_ORDER	Order_id	Unique identifier for an order	Int	NA	Part of primary key. Unique	Y	9 numeric, positive digits
FULL_ORDER	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Part of primary key. Unique	Y	9 numeric, positive digits
FULL_ORDER	History_id	Migrated foreign key from ORDER_ HISTORY	Int	NA	Referential Integrity	Y	9 numeric, positive digits
FULL_ORDER	Method_id	Migrated foreign key from SHIPPING_ METHOD	Int	NA	Referential Integrity	Y	9 numeric, positive digits
FULL_ORDER	Ship_address _id	Migrated foreign key from SHIP_ ADDRESS	Int	NA	Referential Integrity	Y	9 numeric, positive digits
FULL_ORDER	Total_books	Total number of all books ordered	Int	NA	NONE	N	9 numeric, positive digits

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
ORDER_ STATUS	Status_id	Unique identifier for the status of an order	Int	NA	Part of primary key	Y	9 numeric, positive digits
ORDER_ STATUS	Order_id	Migrated foreign key from ORDER	Int	NA	Part of primary key	Y	9 numeric, positive digits
ORDER_ STATUS	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Part of primary key	Y	9 numeric, positive digits
ORDER_ STATUS	Shipping_id	Migrated foreign from Shipping_INFO	Int	NA	Referential Integrity	Y	9 numeric, positive digits
ORDER_ STATUS	Delivered	Flag to indicate if order has been delivered	Boolean	NA	None	N	T/F
ORDER_ STATUS	Shipped	Flag to indicate if order has been shipped	Boolean	NA	None	N	T/F
ORDER_ STATUS	Processed	Flag to indicate if order has been processed	Boolean	NA	None	N	T/F
INVOICE	Invoice_id	Unique identifier for an invoice	Int	NA	Primary Key	Y	9 numeric, positive digits
INVOICE	Order_id	Migrated foreign key from ORDER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
INVOICE	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
INVOICE	Payment_id	Migrated foreign from CUSTOMER_ PAYMENT	Int	NA	Referential Integrity	Y	9 numeric, positive digits

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
INVOICE	Payment_ received	Flag to indicate payment was received for an order	Boolean	NA	None	Y	T/F
INVOICE	Total_cost	Total cost of an order	Decimal	10,2	None	N	9 numeric, positive digits including decimal point
CUSTOMER	Customer_id	Unique identifier for a customer	Int	NA	Primary Key	Y	9 numeric, positive digits
CUSTOMER	First_name	First name of customer	VARCHAR	45	None	N	A-Z,a-z
CUSTOMER	Last_name	Last name of customer	VARCHAR	45	None	N	A-Z,a-z
CUSTOMER	Middile_int	Middle initial of customer	VARCHAR	1	None	N	A-Z
CUSTOMER	DOB	Date of birth of customer	DATE	NA	None	N	Date values
CUSTOMER	Creation_ date	Date of customer profile creation	DATE	NA	None	N	Date values
CUSTOMER	Email	Customer email address	VARCHAR	45	None	N	Alpha-numeric and special character

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
CUSTOMER	Primary_ phone	Migrated foreign key from SHIPPING_ INFO	Int	10	None	N	10 numeric, non-negative digits
CUSTOMER	Password	Password for customer profile	VARCHAR	10	None	Y	Alpha-numeric and special character
BILL_ ADDRESS	Bill_address _id	Unique identifier for customer bill address	Int	NA	Primary Key	Y	9 numeric, positive digits
BILL_ ADDRESS	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
BILL_ ADDRESS	Street_ address	Street address for bill address	VARCHAR	45	None	Y	Alpha-numeric and special character
BILL_ ADDRESS	City	City of billing address	VARCHAR	45	None	Y	Alpha-numeric and special character
BILL_ ADDRESS	State	State of billing address	VARCHAR	2	None	Y	A-Z
BILL_ ADDRESS	Zip_code	Zip code for billing address	Int	5	None	Y	10 numeric, non-negative
BILL_ ADDRESS	Unit_number	Unit number for billing address	VARCHAR	10	None	N	Alpha-numeric

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
CUSTOMER_ PAYMENT	Payment_id	Unique identifier for customer payment	Int	NA	Primary Key	Y	9 numeric, positive digits
CUSTOMER_ PAYMENT	Customer_id	Migrated foreign key from CUSTOMER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
CUSTOMER_ PAYMENT	Credit_card _type	Type of credit card	VARCHAR	45	None	Y	A-Z,a-z
CUSTOMER_ PAYMENT	Expire_date	Date of expiration on credit card	DATE	NA	None	Y	Date values
CUSTOMER_ PAYMENT	Card_number	Credit card number	Int	NA	None	Y	10 numeric, non-negative digits
CUSTOMER_ PAYMENT	First_name	First name on credit card	VARCHAR	45	None	Y	A-Z,a-z
CUSTOMER_ PAYMENT	Last_name	Last name on credit card	VARCHAR	45	None	Y	A-Z,a-z
CUSTOMER_ PAYMENT	Middile_init	Middle initial on credit card	VARCHAR	1	None	Y	A-Z
CUSTOMER_ PAYMENT	Security_ code	Security code on credit card	Int	NA	None	Y	10 numeric, non-negative digits

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
AUTHOR	Middle_init	Middle initial of author	VARCHAR	1	None	N	A-Z
BOOK_ AUTHOR	Author_id	Migrated foreign key from AUTHOR	Int	NA	Part of Primary Key	Y	9 numeric, positive digits
BOOK_ AUTHOR	ISBN	Migrated foreign key from BOOK	Int	NA	Part of Primary Key	Y	9 numeric, positive digits
BOOK	ISBN	Unique identifier for book	Double	NA	Primary Key	Y	9 numeric, positive digits
BOOK	Publisher_id	Migrated foreign key from PUBLISHER	Int	NA	Referential Integrity	Y	9 numeric, positive digits
BOOK	Title	Title of book	VARCHAR	45	None	Y	All characters
GENRE	Description	Description of genre according to id	VARCHAR	75	None	Y	All characters
BOOK_ORDER	Total_weight _oz	Weight in ounces of books	FLOAT	NA	None	Y	9 numeric, positive digits with decimal

Table Name	Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
SHIP_ ADDRESS	State	State for shipping	VARCHAR	2	None	Y	A-Z
SHIP_ ADDRESS	Zip	Zip for shipping	Int	5	None	Y	Non-negative numeric
SHIP_ ADDRESS	Unit_number	Unit number for shipping	VARCHAR	10	None	N	Alpha-numeric
SHIP_METHOD	Method_id	Unique identifier that represents shipping method type	Int	NA	Primary Key	Y	9 numeric, positive digits
SHIP_METHOD	Description	Description shipping method according to id	VARCHAR	75	None	Y	A-Z, a-z
SHIP_METHOD	Cost_per_oz	Cost of shipping. Cents per oz	Decimal	10,2	None	Y	9 numeric, positive digits
ORDER_ STATUS	Shipped_date	Date the order shipped	Date	NA	None	N	Date inputs
ORDER_ STATUS	Processed_ date	Date the order was processed	Date	NA	None	N	Date inputs
ORDER_ STATUS	Delivered_ date	Date the order was delivered	Date	NA	None	N	Date inputs
INVOICE	Tax_cost	Cost of tax for full order	Decimal	10,2	None	N	9 numeric, positive digits
INVOICE	Ship_cost	Cost of shipping for full order	Decimal	10,2	None	N	9 numeric, positive digits

3.3.2. Integrity Rules

Mandatory fields required for data entry have the added constraint of ‘Not Null’. This requires that these fields be filled upon entry of the data. For those fields that are mandatory from a processing perspective but filled via trigger operation do not have the constraint of ‘Not Null’ as these fields will be filled programmatically.

Certain fields employ a specific data formatting that is necessary for proper representation and/or calculation of the data. For fields that are to represent a monetary amount the datatype of DECIMAL(10,2) is used in order to represent both the dollar and cents of an amount. For fields where a date or data/time value is needed, such as customer creation, either a DATA datatype or a TIMESTAMP type was used, with certain fields taking on the default date/time of the current data/time of the system. For Boolean flags that represent whether or not an even has happened TINYINT(1) was used. For fields that are specific in length and type, such as for credit card number, the appropriate datatype was used and had the necessary length restriction inserted. Then for a credit card value the datatype would be BIGINT(16). All other data values are listed with their appropriate data types. As such there is limited need for conversion or additional data formatting.

In order to handle and ensure referential integrity each table has along with its foreign key constraints, constraints on how it is to handle the update and deletes. As a uniform decision across the database it was decided to employ the standard of cascading all updates to foreign keys and restricting on all deletions. This allows for continued operation when foreign key is just updated but does not allow for cascading deletion or other such actions. Additionally since the RDBMD ensures existence and uniques of keys, any update to a foreign key must first pass these tests.

Finally it is noted that for many of the fields where dates, lengths, and input values are needed to be checked that a CHECK constraint would have been used. Unfortunately mySQL does not support CHECK constraint operation so these were not explicitly used. But in a normal situation this type of constraint would be used liberally to verify data input. Additionally the database application would also have some of this responsibility, or at least drive the process.

3.3.3. Operational Rules

Certain operational rules apply the operations of the database. These include:

1. The system will not allow a customer to add to their order a book whose stock count is 0
2. The system will not allow a customer perform a new order if their previous order is outstanding
3. The system will not allow a customer's account to be deleted if there are outstanding orders unpaid or unfulfilled
4. System will not allow an order to proceed until a valid form of payment is entered
5. System will not allow an order to proceed until form of payment with passed expiration date is updated or replaced
6. System will customers with the same name to be entered into the database since each customer will have a unique identification number
7. System does not allow for partial payments of orders
8. System allows for users to have the same address
9. System will only produce one invoice per order
10. Customers can produce and be associated to multiple orders

3.3.4. Operations

The following describes the general, smooth use case of a customer creating an account and ordering 2 books. The following operation will involve the eventual database application that will be paired with the database. This section however will not specify how and when said database application will be in effect.

A new customer enters the Good Read Online BookStore and creates a new customer account. This causes an insert into the CUSTOMER, SHIP_ADDRESS, BILL_ADDRESS, and CUSTOMER_PAYMENT tables with the user entering the necessary data. Upon creation of the CUSTOMER account the system creates an ORDER_HISTORY insert that is associated to the user. The user searches the database for books via genre, title, author, and/or publisher using the database queries. Upon selection of a book, a new entry is inserted into FULL_ORDER and BOOK_ORDER with fields for the FULL_ORDER id inserted into the new BOOK_ORDER row and cumulative book values inserted in FULL_ORDER row. The user chooses a second book to purchase and another entry is inserted into BOOK_ORDER and

the cumulative value fields for the cost of the books and total book weight are updated. The user selects the desired shipping address through use of the query on the SHIP_ADDRESS table and the entry for FULL_ORDER is updated. The user then selects their shipping method of choice by using a query of the SHIP_METHOD table and upon selection of a shipping method the FULL_ORDER row is updated with the appropriate shipping method id. Upon submission of the order an entry is inserted into ORDER_STATUS with default values inserted for all non-key attributes. Additionally an invoice is created by inserting a new entry in the INVOICE table with all appropriate fields populated. The customer selects their desired credit card for payment through use of the query on the CUSTOMER_PAYMENT table. Upon selection of payment the INVOICE table is updated for payment id and the flag for payment being received. Finally the user verifies that the stored Billing Address is to be used. If not then the user is prompted to enter a new bill address which will delete the old entry and create a new one.

3.4. Security

The security of the database for the Good Read Bookstore is built upon a phased approach that involves security emplacements and monitoring. The most sensitive aspects of the database is related to customer information, specifically the tables CUSTOMER, SHIP_ADDRESS, BILL_ADDRESS, and CUSTOMER_PAYMENT. Additionally the business and inventory tables are important to business operations and while not considered sensitive, the integrity of these tables need to have a level of protection. The following security measures are implemented in order to secure these two areas of data:

1. Privileges - The database will implement different levels of privileges that restrict or allow access to certain operations and data within the database. Privileges will consist of 3 different levels: Level-1 has only basic write and read privileges. These read and write privileges are restricted to viewing book related information and writing user information. This level will be applied to user accounts. Level-2 has all the privileges of level-1 but has write privileges to insert data into the inventory tables such as BOOK and AUTHOR, but does not allow any financial read or write access. This level is used for mid-tier system administrators. Level-3 has all the previous levels access but allows read and write access to financial information. Additionally only this level has

the authority to drop or add tables. This level is reserved for super system administrators or DBAs.

2. Encryption - To protect customer information, all customer tables will be encrypted using AES encryption and a key length of 128bits. These include CUSTOMER, SHIP_ADDRESS, BILL_ADDRESS, and CUSTOMER_PAYMENT. The key will be stored on the database server, that will have network security and a firewall. Additionally the key will be stored and accessible via password only to those with level-3 access.

3. Monitoring - The database will employ second party monitoring to monitor database activities to determine if any breaches or issues in protocol have been identified. Any security breaches that are identified will be escalated to the DBA.

3.5. Database Backup and Recovery

The database will employ a recovery manager tool to facilitate the backup and recovery of the database. The schedule for database backup will be:
1. Full database backup every 24 hours. To include OS, application, and all other database pertinent data.
2. Differential backup every 4 hours
3. Transaction log backup every 1 hour

This process will be tested weekly to ensure the appropriate backups are occurring and that these backups are able to be used for recovery. All backups will be stored to disk and transferred to tape for storage. Backups will be stored for 1 year. Additionally the usability of the database will be monitored and tested during the backup process.

The recovery process will be detailed and implemented monthly to produce a non-production database. During the process of an actual database restore the website will be down and a default webpage explaining site maintenance is being implemented will be displayed.

3.6. Using Database Design or CASE Tool

To design and implement the Good Read Bookstore database I primarily used mySQL Workbench. Through the use of this design tool I was able to first produce a database entity-relationship diagram that displayed all the tables, attributes with datatypes, the relationships with structural constraints, and all key assignments with constraints. Additionally this tool allowed for integrated implementation that was seamless.

In addition to using mySQL Workbench, in order to properly and incrementally implement the database design, the version control framework Git was used. This tool allowed for complete backup, tracking, and recovery of all necessary design and implementation points.

3.7. Other Possible E/R Relationships

In addition to the existing design for the Good Read Book Store other design alternatives were considered and still remain as possibilities for extension to the database design and functionality. Below outlines these design alternatives in brief detail:

1. RETURNS - This table is to show all books and/or orders that were returned. This would include the reason for the return. This table would have a relationship with FULL_ORDER, BOOK_ORDER, and CUSTOMER. There would also be functionality needed to either restock the returned book or log the book as being damaged.

2. USED_BOOKS - This table would be an inventory of the books in stock that are damaged or used. These books would be sold at a reduced price and would have their own inventory keeping. This table would have a relationship with BOOK and RETURNS. These books would only be those books that are sold new in the database but were found to be defective or returned.

3. INVOICES - This table would be a history of invoices for a CUSTOMER and would allow for invoices to be modified or updated, as in the case of a returned book. This table would have a relationship with INVOICE.

4. REVIEWS - This table would be to hold customer feedback concerning a book. This table would hold values such as ‘Review’, ‘wouldRecommend’, and, ‘Score’. This table would be in a relationship with CUSTOMER, BOOK, and AUTHOR.

5. Tables for employee information - This section of the database would focus on the employees and business specifics of running the store for a personnel perspective. It was decided that the book store would initially be a very small operation not need such tables as EMPLOYEES or DEPENDENTS. Yet in the future as the company grows this section would certainly be necessary.

4. Implementation Description

The following section outlines the implementation details of the Good Read Bookstore first outlining the data dictionary, then the advanced features of the database, and finally database queries. It should be noted that this section assumes a joint functionality of a database application that will be processing much of the input and output data from the user and/or the database. Where important the functionality of the database application as it pertains to the database implementation will be mentioned.

4.1. Data Dictionary

Field	Type	Null	Key	Default	Extras	Table
Author_id	int(11)	No	PRI		auto_ increment	AUTHOR
First_name	varchar(45)	No				AUTHOR
Last_name	varchar(45)	No				AUTHOR
Middle_init	varchar(1)	Yes				AUTHOR
Bill_ address_id	int(11)	No	PRI		auto_ increment	BILL_ ADDRESS
Customer_ id	int(11)	No	MUL			BILL_ ADDRESS
Street_ address	varchar(45)	No				BILL_ ADDRESS
City	varchar(45)	No				BILL_ ADDRESS
State	varchar(2)	No				BILL_ ADDRESS
Zip_code	int(5) unsigned	No				BILL_ ADDRESS
Unit_ number	varchar(10)	Yes				BILL_ ADDRESS
ISBN	double	No	PRI			BOOK
Publisher _id	int(11)	No	MUL			BOOK
Title	varchar(45)	Yes				BOOK
Edition	varchar(20)	Yes				BOOK
Hardcover	tinyint(1)	Yes				BOOK
Paperback	tinyint(1)	Yes				BOOK
Weight_oz	float unsigned	No				BOOK
Publish_ date	date	No				BOOK

Field	Type	Null	Key	Default	Extras	Table
Cost	decimal (10,2)	No				BOOK
Page_count	int(10) unsigned	Yes				BOOK
Stock_ count	int(11)	No				BOOK
Reorder	tinyint(1)	Yes		0		BOOK
Author_id	int(11)	No	PRI			BOOK_ AUTHOR
ISBN	double	No	PRI			BOOK_ AUTHOR
Genre_id	int(11)	No	PRI			BOOK_ AUTHOR
ISBN	double	No	PRI			BOOK_ GENRE
Order_id	int(11)	No	PRI			BOOK_ GENRE
ISBN	double	No	PRI			BOOK_ ORDER
Customer_ id	int(11)	No	PRI			BOOK_ ORDER
Count	int(10)	No				BOOK_ ORDER
Sale	decimal (10,2)	Yes				BOOK_ ORDER
Total_ weight_oz	float	Yes				BOOK_ ORDER
Customer_ id	int(11)	NO	PRI			CUSTOM ER
First_name	varchar(45)	Yes				CUSTOM ER
Last_name	varchar(45)	Yes				CUSTOM ER
Middle_init	varchar(1)	Yes				CUSTOM ER
DOB	date	Yes				CUSTOM ER
Creation_ date	timestamp	Yes		Current_ Timestamp		CUSTOM ER
Email	varchar(45)	Yes				CUSTOM ER
Primary _phone	bigint(10)	Yes				CUSTOM ER
Password	varchar(10)	No				CUSTOM ER
Payment_id	int(11)	No	PRI		auto_ increment	CUSTOM ER_PAY MENT

Field	Type	Null	Key	Default	Extras	Table
Customer_id	int(11)	No	MUL			CUSTOMER_PAYMENT
Credit_card_type	varchar(45)	No				CUSTOMER_PAYMENT
Expire_date	date	No				CUSTOMER_PAYMENT
Card_number	bigint(16) unsigned	No				CUSTOMER_PAYMENT
First_name	varchar(45)	No				CUSTOMER_PAYMENT
Last_name	varchar(45)	No				CUSTOMER_PAYMENT
Middle_init	varchar(1)	Yes				CUSTOMER_PAYMENT
Security_code	int(10) unsigned	No				CUSTOMER_PAYMENT
Order_id	int(11)	No	PRI			FULL_ORDER
History_id	int(11)	No	MUL			FULL_ORDER
Method_id	int(11)	No	MUL			FULL_ORDER
Ship_address_id	int(11)	No	MUL			FULL_ORDER
Customer_id	int(11)	No	PRI			FULL_ORDER
Order_date	date	Yes			current_timestamp	FULL_ORDER
Total_books	int(11)	Yes			0	FULL_ORDER
Book_cost	decimal (10,2)	Yes			0.00	FULL_ORDER
Total_weight	float	Yes			0	FULL_ORDER
Genre_id	int(11)	No	PRI		auto_increment	GENRE
Description	varchar(75)	No				GENRE
Invoice_id	int(11)	No	PRI		auto_increment	INVOICE
Order_id	int(11)	No	MUL			INVOICE

Field	Type	Null	Key	Default	Extras	Table
Customer_id	int(11)	No				INVOICE
Payment_id	int(11)	Yes	MUL			INVOICE
Payment_received	tinyint(1)	No		0		INVOICE
Ship_cost	decimal(10,2)	Yes				INVOICE
Tax_cost	decimal(10,2)	Yes				INVOICE
Total_cost	decimal(10,2) unsigned	Yes				INVOICE
History_id	int(11)	No	PRI		auto_increment	ORDER_HISTORY
Customer_id	int(11)	No	PRI			ORDER_HISTORY
Status_id	int(11)	No	PRI		auto_increment	ORDER_STATUS
Order_id	int(11)	No	PRI			ORDER_STATUS
Customer_id	int(11)	No	PRI			ORDER_STATUS
Delivered	tinyint(1)	Yes		0		ORDER_STATUS
Shopped	tinyint(1)	Yes		0		ORDER_STATUS
Processed	tinyint(1)	Yes		0		ORDER_STATUS
Delivered_date	date	Yes				ORDER_STATUS
Shipped_date	date	Yes				ORDER_STATUS
Processed_date	date	Yes				ORDER_STATUS
Publisher_id	int(11)	No	PRI		auto_increment	PUBLISHER
Name	varchar(45)	No				PUBLISHER
City	varchar(45)	No				PUBLISHER
State	varchar(2)	Yes				PUBLISHER
Country	varchar(45)	No				PUBLISHER
Ship_address_id	int(11)	No	PRI		auto_increment	SHIP_ADDRESS
Customer_id	int(11)	No	MUL			SHIP_ADDRESS
Street_address	varchar(45)	No				SHIP_ADDRESS

Field	Type	Null	Key	Default	Extras	Table
City	varchar(45)	No				SHIP_ADDRESS
State	varchar(2)	No				SHIP_ADDRESS
Zip	int(5) unsigned	No				SHIP_ADDRESS
Unit_number	varchar(10)	Yes				SHIP_ADDRESS
Method_id	int(11)	No	PRI			SHIPPING_METHOD
Description	varchar(75)	No				SHIPPING_METHOD
Cost_per_oz	decimal (10,2)	No				SHIPPING_METHOD

4.2. Advanced Features - The Good Read Book database implements some advanced features to implement database functionality and ensure business rules are enforced. These advance features consist of triggers and stored procedures. The following section describes both such features and provides implementation code but leaves out some of the extraneous coding syntax.

Triggers

In order to verify that the credit card expiration date is not expired a trigger is used to verify the credit card is still valid. This trigger is implemented before an insertion and before an update. The trigger is as follows:

```
CREATE TRIGGER `CUSTOMER_PAYMENT_BEFORE_INSERT`
BEFORE INSERT ON `CUSTOMER_PAYMENT`
FOR EACH ROW
BEGIN
        IF NEW.Expire_date < CURRENT_TIMESTAMP() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Credit
        Card Expiration Date Invalide';
END IF;
END
```

In order for customer orders to be stored in an order history each customer in CUSTOMER must first have an entry in ORDER_HISTORY before an order can be placed. In order to facilitate this, after insertion of a new

customer an entry is populated in the ORDER_HISTORY table. The trigger is as follows:

```
CREATE TRIGGER `CUSTOMER_AFTER_INSERT` AFTER INSERT
ON `CUSTOMER`
FOR EACH ROW
BEGIN
	INSERT INTO ORDER_HISTORY(Customer_id)
	VALUES(NEW.Customer_id);
END
```

An order from the Good Read Bookstore consists of one to many BOOK_ORDER entries that comprise one FULL_ORDER entry. Each BOOK_ORDER entry is to one or more copies of a specific book. When a customer orders a book there must be enough in stock to facilitate the order. A trigger checks to ensure that the stock of the book supports the order. If after the order the stock is left at zero, the trigger sets the Reorder flag to 1. Additionally the trigger populates the Total_weight, Total_books, and Book_cost of the FULL_ORDER table. These values reflect the weight, total number of books, and total book cost of all books in the full order. This functionality is implemented on update and insertion of a table. The triggers for insertion and update are as follows:

```
CREATE TRIGGER `BOOK_ORDER_BEFORE_INSERT` BEFORE
INSERT ON `BOOK_ORDER`
FOR EACH ROW
BEGIN
	Declare stk INT;
	Declare cst DECIMAL(10,2);
	Declare wgt FLOAT;
	Declare buy TINYINT(1);
	Set stk = (SELECT Stock_count FROM BOOK WHERE
	BOOK.ISBN = NEW.ISBN);
	Set cst = (SELECT Cost From BOOK WHERE BOOK.ISBN =
	NEW.ISBN);
	Set wgt = (SELECT Weight_oz From BOOK WHERE
	BOOK.ISBN = NEW.ISBN);
	Set buy = 0;
	IF (stk - NEW.Count < 0) Then
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
		'Count requested exceeds stock';
	ELSE
		IF (stk - NEW.Count = 0) Then
		Set buy = 1;
		END IF;
```

```
                Set NEW.Sale = cst*NEW.Count;
                Set New.Total_weight_oz = wgt*NEW.Count;
                Update BOOK
                Set Stock_count = Stock_count - NEW.Count,
                Reorder = buy
                WHERE BOOK.ISBN = NEW.ISBN;
                Update FULL_ORDER
                Set Total_books = Total_books + NEW.Count,
                Book_cost = Book_cost + NEW.Sale,
                Total_weight = Total_weight + NEW.Total_weight_oz
                WHERE FULL_ORDER.Order_id = NEW.Order_id AND
                FULL_ORDER.Customer_id = NEW.Customer_id;
        END if;
END

CREATE TRIGGER `BOOK_ORDER_BEFORE_UPDATE` BEFORE
UPDATE ON `BOOK_ORDER`
FOR EACH ROW
BEGIN
        Declare stk INT;
        Declare cst DECIMAL(10,2);
        Declare wgt FLOAT;
        Declare buy TINYINT(1);
        Set stk = (SELECT Stock_count FROM BOOK WHERE
        BOOK.ISBN = NEW.ISBN);
        Set cst = (SELECT Cost From BOOK WHERE BOOK.ISBN =
        NEW.ISBN);
        Set wgt = (SELECT Weight_oz From BOOK WHERE
        BOOK.ISBN = NEW.ISBN);
        Set buy = 0;
        IF (stk - NEW.Count + OLD.Count < 0) Then
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
                'Count requested exceeds stock';
        ELSE
                IF (stk - NEW.Count = 0) Then
                Set buy = 1;
                END IF;
                Set NEW.Sale = cst*NEW.Count;
                Set New.Total_weight_oz = wgt*NEW.Count;
                Update BOOK
                Set Stock_count = Stock_count - NEW.Count +
                OLD.Count,
                Set Reorder = buy
```

```
                WHERE BOOK.ISBN = NEW.ISBN;
                Update FULL_ORDER
                Set Total_books = Total_books + NEW.Count -
                OLD.Count,
                Book_cost = Book_cost + NEW.Sale - OLD.Sale,
                Total_weight = Total_weight + NEW.Total_weight_oz -
                OLD.Total_weight_oz
                WHERE FULL_ORDER.Order_id = NEW.Order_id AND
                FULL_ORDER.Customer_id = NEW.Customer_id;
        END if;
END
```

Stored Procedures

Once an order is placed by a customer the database application will generate an invoice for the customer. Part of this generation is a stored procedure that will calculate the shipping cost based upon the shipping method chose, the tax cost based upon the book cost, and the total cost all together for the order. Since this online book store will be conducted in the state of Maryland, the Good Read Book Store will use 6% sales tax. The stored procedure receives as input the Order_id from FULL_ORDER. The stored procedure is as follows:

```
CREATE PROCEDURE `Invoice_Populate`(IN orderID INT)
BEGIN
        DECLARE ShipR, ShipC, TotalC, TaxC DECIMAL(10,2);
        DECLARE shipID, CustId INT;
        SET shipID = (SELECT Method_id FROM FULL_ORDER
        WHERE FULL_ORDER.Order_id = orderID);
        SET ShipR = (SELECT Cost_per_oz FROM
        SHIPPING_METHOD WHERE Method_id = shipID);
        SET ShipC = ShipR*(SELECT Total_weight FROM
        FULL_ORDER WHERE FULL_ORDER.Order_id = orderID);
        SET TaxC = .06*(SELECT Book_cost FROM FULL_ORDER
        WHERE FULL_ORDER.Order_id = orderID);
        SET TotalC = TaxC + ShipC + (SELECT Book_cost FROM
        FULL_ORDER WHERE FULL_ORDER.Order_id = orderID);
        SET CustID = (SELECT Customer_id FROM FULL_ORDER
        WHERE FULL_ORDER.Order_id = orderID);
        INSERT INTO INVOICE(Order_id, Customer_id, Ship_cost,
        Tax_cost, Total_cost) VALUES(orderID, CustID, ShipC, TaxC,
        TotalC);
END
```

The Good Read Bookstore also will need to produce monthly sales reports to determine if they are meeting sales quotas and for future projections. The following stored procedures takes two inputs that are variable characters for month and year. These values are cast into unsigned integers and used in the query to filter based upon the desired month and year for displaying the total books sold, the order dates, the total cost of the books, and the total cost of the order. If either input entry is 'NULL', then that respective variable receives the current month or year as its value. The code is as follows:

```
CREATE PROCEDURE `Monthly_Sales`(IN month VARCHAR(2), IN
year VARCHAR(4))
BEGIN
	DECLARE monthVal, yearVal INT;
	IF month IS NULL THEN
		SET monthVal = MONTH(CURRENT_TIMESTAMP);
	ELSE
		SET monthVal = CAST(month as UNSIGNED);
	END IF;
	IF year IS NULL THEN
		SET yearVal = YEAR(CURRENT_TIMESTAMP);
	ELSE
		SET yearVal = CAST(year as UNSIGNED);
	END IF;
	SELECT Order_date, Total_books, Book_cost, Total_cost
	FROM INVOICE JOIN FULL_ORDER ON INVOICE.Order_id =
	FULL_ORDER.Order_id
	WHERE MONTH(Order_date) = monthVal AND
	YEAR(Order_date) = yearVal;
END
```

4.3. **Queries** - The Good Read Bookstore will support queries of the database. These queries will be implemented by users, system administrators, or database administrators. According to each user the need, purpose, and sophistication will vary.

End users will be interacting with the database via the database application and will never actually enter mySQL query syntax. Rather through their interaction with the database application, the bookstore database will be queried through the user input via database language framework such as JDBC. Therefore the queries implemented will be written in mySQL but the inputs that specifically determine the query will be administered and received by the database application. This type of interface will most likely be a webpage form that sends the user inputs to the backend

controller code that implements the JDBC logic to query the database. The results would also then be parsed and displayed by the controller code to the webpage. Additionally all inputs and outputs would be verified at the controller level.

System administrators would be workers who help maintain the general operations of the website. This would include database updates for inventory, password resets, ensuring orders are fulfilled, and maintaining up-to-date order statuses. Of course this is not a complete list of responsibilities but a sample. System administrators also are not required to interact with the database via mySQL code but via the database application but with the privileges to interact with the database to fulfill their responsibilities. Therefore the interaction and interface will be similar to end users however the terminal or account they will be using will have the appropriate privileges to perform these operations.

Lastly the database administrator will be a super user that is fully qualified to use mySQL coding interactions with the database and as such would not be using any specific database application functionality that is unique to database administrators. Of course the DBA would be able to implement all database application interactions that system administrators and end users implement.

The queries that follow are example queries and are presented as they would be implemented and seen in mySQL code reflecting the final query implemented after user inputs are applied by the database application language, such as JDBC. Those values that inserted by the application language will be bold and in blue font.

4.3.1. Customer Bills and Orders

1. Select a specific customer's bills:
SELECT * FROM INVOICE WHERE Customer_id = **1**;

2. Select all customer bills that are unpaid:
SELECT * FROM INVOICE WHERE Payment_received = 0;

3. Select a specific customer's bills that are unpaid:
SELECT * FROM INVOICE WHERE Customer_id = **1** AND Payment_received = 0;

4. Show all bills for a customer in a certain month and year:
SELECT INVOICE.Invoice_id, INVOICE.Order_id, INVOICE.Customer_id, INVOICE.Payment_id, INVOICE.Payment_received, INVOICE.Ship_cost, INVOICE.Tax_cost, INVOICE.Total_cost FROM INVOICE JOIN FULL_ORDER ON INVOICE.Order_id = FULL_ORDER.Order_id WHERE INVOICE.Customer_id = **1**
AND MONTH(Order_date) = **11** AND YEAR(Order_date) = **2015**;

5. Show all processed orders from a customer that are unshipped:
SELECT * FROM ORDER_STATUS WHERE Customer_id = **1** AND Shipped = 0 AND Processed = 1;

6. Show all processed orders that are unshipped listed according to customer and from oldest to newest:
SELECT * FROM ORDER_STATUS WHERE Shipped = 0 AND Processed = 1 GROUP BY Customer_id ORDER BY Processed_date;

7. List all customers by the number of books purchased from most books purchased to least books purchased:
SELECT CUSTOMER.Customer_id, SUM(FULL_ORDER.Total_books) AS 'Books Purchased' FROM FULL_ORDER JOIN CUSTOMER ON FULL_ORDER.Customer_id = CUSTOMER.Customer_id GROUP BY CUSTOMER.Customer_id ORDER BY SUM(FULL_ORDER.Total_books) DESC;

4.3.2. Customer Purchase History

1. List all books in a customer order:
SELECT Order_id, BOOK.ISBN, BOOK.Title, BOOK.Cost, BOOK.Weight_oz, Count FROM BOOK_ORDER JOIN BOOK ON BOOK_ORDER.ISBN = BOOK.ISBN
WHERE Customer_id = **1**;

2. List total number of books purchased by a customer:
SELECT Customer_id, SUM(Count) From BOOK_ORDER WHERE Customer_id = **1**;

3. List genres by name purchased by a customer:
SELECT Customer_id, GENRE.Description FROM BOOK_ORDER JOIN BOOK_GENRE ON BOOK_ORDER.ISBN = BOOK_GENRE.ISBN JOIN GENRE ON

BOOK_GENRE.Genre_id = GENRE.Genre_id WHERE Customer_id = 1;

4. List authors of all books purchased by a customer:
SELECT Customer_id, AUTHOR.First_name, AUTHOR. Middle_init, AUTHOR.Last_name FROM BOOK_ORDER JOIN BOOK_AUTHOR ON
BOOK_ORDER.ISBN = BOOK_AUTHOR.ISBN JOIN AUTHOR ON BOOK_AUTHOR.Author_id = AUTHOR.Author_id WHERE Customer_id =1;

4.3.3. Book Purchase History

1. Show the books that have been purchased and how many times each has been purchased:
SELECT BOOK.Title, BOOK.ISBN, SUM(BOOK_ORDER.Count) AS 'Number Purchased' FROM
BOOK_ORDER JOIN BOOK ON BOOK_ORDER.ISBN = BOOK.ISBN GROUP BY BOOK.ISBN;

2. Show the books that have been purchased in a certain month and year, and how many times each has been purchased:
SELECT DATE(FULL_ORDER.Order_date) AS Date,BOOK.Title, BOOK.ISBN, SUM(BOOK_ORDER.Count) AS 'Number Purchased' FROM
BOOK_ORDER JOIN BOOK ON BOOK_ORDER.ISBN = BOOK.ISBN JOIN FULL_ORDER ON FULL_ORDER.Order_id = BOOK_ORDER.Order_id
WHERE MONTH(FULL_ORDER.Order_date) = 11 and YEAR(FULL_ORDER.Order_date)=2015 GROUP BY BOOK.ISBN;

3. Show all genres reflected in book purchases and how many times each has been purchased:
SELECT GENRE.Description, SUM(BOOK_ORDER.COUNT) AS 'Number Purchased' FROM BOOK_ORDER JOIN BOOK_GENRE ON BOOK_ORDER.ISBN =
BOOK_GENRE.ISBN JOIN GENRE ON BOOK_GENRE.Genre_id = Genre.Genre_id GROUP BY Genre.Genre_id;

4. Show all genres reflected in book purchases in a certain month and year, and how many times each has been purchased:
SELECT DATE(FULL_ORDER.Order_date) AS Date, GENRE.Description, SUM(BOOK_ORDER.COUNT) AS 'Number

```
Purchased' FROM BOOK_ORDER JOIN BOOK_GENRE ON
BOOK_ORDER.ISBN =
BOOK_GENRE.ISBN JOIN GENRE ON BOOK_GENRE.Genre_id =
Genre.Genre_id JOIN FULL_ORDER ON FULL_ORDER.Order_id =
BOOK_ORDER.Order_id
WHERE MONTH(FULL_ORDER.Order_date)=11 AND
YEAR(FULL_ORDER.Order_date)=2015 GROUP BY Genre.Genre_id;
```

5. Show all authors who have books that have been purchased and how many books have been purchased:

```
SELECT AUTHOR.First_name, AUTHOR.Middle_init,
Author.Last_name,
SUM(BOOK_ORDER.COUNT) AS 'Number Purchased' FROM
BOOK_ORDER JOIN BOOK_AUTHOR ON BOOK_ORDER.ISBN =
BOOK_AUTHOR.ISBN JOIN AUTHOR ON
BOOK_AUTHOR.Author_id = AUTHOR.Author_id GROUP BY
AUTHOR.Author_id;
```

6. Show all authors who have books that have been purchased in a certain month and year, and how many books have been purchased:

```
SELECT DATE(FULL_ORDER.Order_date) AS Date,
AUTHOR.First_name, AUTHOR.Middle_init, Author.Last_name,
SUM(BOOK_ORDER.COUNT) AS 'Number Purchased' FROM
BOOK_ORDER JOIN BOOK_AUTHOR ON BOOK_ORDER.ISBN =
BOOK_AUTHOR.ISBN JOIN AUTHOR ON
BOOK_AUTHOR.Author_id = AUTHOR.Author_id JOIN
FULL_ORDER ON
FULL_ORDER.Order_id = BOOK_ORDER.Order_id WHERE
MONTH(FULL_ORDER.Order_date)=11 AND
YEAR(FULL_ORDER.Order_date)=2015
GROUP BY AUTHOR.Author_id;
```

7. List genres by book purchases in descending order:

```
SELECT GENRE.Description, SUM(BOOK_ORDER.COUNT) AS
'Number Purchased' FROM BOOK_ORDER JOIN BOOK_GENRE ON
BOOK_ORDER.ISBN = BOOK_GENRE.ISBN JOIN GENRE ON
BOOK_GENRE.Genre_id = Genre.Genre_id GROUP BY Genre.Genre_id
ORDER BY SUM(BOOK_ORDER.COUNT) DESC;
```

4.3.4. List Books

1. List all books by genre:

SELECT BOOK.ISBN, BOOK.Title, BOOK.Edition, BOOK.Hardcover, BOOk.Paperback, BOOK.Weight_oz, BOOK.Publish_date, BOOK.Cost, BOOK.Page_count FROM BOOK JOIN BOOK_GENRE ON BOOK.ISBN = BOOK_GENRE.ISBN JOIN GENRE ON BOOK_GENRE.Genre_id = GENRE.Genre_id WHERE Genre.Description = '**Adventure**';

2. List all books by genre sorted by price:
SELECT BOOK.ISBN, BOOK.Title, BOOK.Edition, BOOK.Hardcover, BOOk.Paperback, BOOK.Weight_oz, BOOK.Publish_date, BOOK.Cost, BOOK.Page_count FROM BOOK JOIN BOOK_GENRE ON BOOK.ISBN = BOOK_GENRE.ISBN JOIN GENRE ON BOOK_GENRE.Genre_id = GENRE.Genre_id WHERE Genre.Description = '**Adventure**' ORDER BY BOOK.Cost;

3. List all Books by author:
SELECT BOOK.ISBN, BOOK.Title, BOOK.Edition, BOOK.Hardcover, BOOk.Paperback, BOOK.Weight_oz, BOOK.Publish_date, BOOK.Cost, BOOK.Page_count FROM BOOK JOIN BOOK_AUTHOR ON BOOK.ISBN = BOOK_AUTHOR.ISBN JOIN AUTHOR ON BOOK_AUTHOR.Author_id = AUTHOR.Author_id WHERE AUTHOR.First_name = '**Dr**' AND Last_name = '**Seuss**';

4. Find book by title:
SELECT BOOK.ISBN, BOOK.Title, BOOK.Edition, BOOK.Hardcover, BOOk.Paperback, BOOK.Weight_oz, BOOK.Publish_date, BOOK.Cost, BOOK.Page_count FROM BOOK WHERE BOOK.Title = '**IT**';

5. Find all books by title and sorted by edition:
SELECT BOOK.ISBN, BOOK.Title, BOOK.Edition, BOOK.Hardcover, BOOk.Paperback, BOOK.Weight_oz, BOOK.Publish_date, BOOK.Cost, BOOK.Page_count FROM BOOK WHERE BOOK.Title = '**IT**' ORDER BY BOOK.Edition;

4.4 **Use Case** - The following section shows a use case of a customer purchasing a book and how the database functions operation during this process. Specifically this section will focus on the triggers and stored procedures involved.

4.4.1. Trigger and Stored Procedure Use Case

A customer first begins by purchasing a book and the database application firstly produces an insertion for the FULL_ORDER table. This line is to hold the BOOK_ORDER rows for each book purchased.

ISBN	Title	Weight_oz	Cost	Stock_count	Reorder
9780345391803	The Hitch Hiker's Guide to the Galaxy	11.6	15.00	1	0
9780394800011	The Cat in the Hat	9.1	7.00	2	0
9780394800165	Green Eggs and Ham	8.8	9.00	0	1
9780439708180	Harry Potter and the Sorcerer's Stone	7.2	5.00	2	0
9780440245919	A Time to Kill	12.6	5.00	1	0
9780451169518	IT	16	5.00	3	0
9780553380163	A Brief History of Time	11.4	20.00	2	0

7 rows in set (0.00 sec)

The customer wants to buy ‘The Hitch Hiker’s Guide to the Galaxy’ and from the BOOK table above, we can see that there is only 1 book left in the inventory.

```
mysql> SELECT * FROM FULL_ORDER;
```

Order_id	History_id	Method_id	Ship_address_id	Customer_id	Order_date	Total_books	Book_cost	Total_weight
1	1	1	1	1	2015-11-22 23:20:48	2	20.00	24.2
2	3	2	4	3	2015-01-23 20:48:06	2	18.00	17.6
3	2	3	3	2	2015-10-29 13:13:36	1	5.00	16
4	1	1	1	1	2015-11-29 17:43:57	1	5.00	7.2
5	3	2	4	3	2015-12-01 18:12:02	2	22.00	20.7
6	2	2	3	2	2015-12-02 18:40:28	0	0.00	0

6 rows in set (0.01 sec)

The customer chooses to make a book order. The database application populates a FULL_ORDER entry as seen above. The values for Total_books, Book_cost, and Total_weight are all initially set at 0.

```
mysql> INSERT INTO BOOK_ORDER (Order_id,ISBN,Customer_id,Count) VALUES ('6', '9780345391803', '2', '1');
Query OK, 1 row affected (0.01 sec)
```

The customer chooses ‘The Hitch Hiker’s Guide to the Galaxy’ as book to purchase and the database application would use the insert statement above to create a new row in BOOK_ORDER associated to Order_id 6. Upon insertion the trigger is activated and it first checks to see if the stock can accommodate the number of copies requested for the book. In this case there is 1 left and the customer only wants 1 so the purchase is allowed. But since the stock count will go to zero the trigger will set the reorder flag to 1. Additionally the trigger will update the values in FULL_ORDER that were initialized to zero. This is seen below.

```
mysql> SELECT * FROM FULL_ORDER;
+----------+------------+----------+----------------+-------------+---------------------+-------------+-----------+--------------+
| Order_id | History_id | Method_id | Ship_address_id | Customer_id | Order_date          | Total_books | Book_cost | Total_weight |
+----------+------------+----------+----------------+-------------+---------------------+-------------+-----------+--------------+
|        1 |          1 |        1 |              1 |           1 | 2015-11-22 23:20:48 |           2 |     20.00 |         24.2 |
|        2 |          3 |        2 |              4 |           3 | 2015-01-23 20:48:06 |           2 |     18.00 |         17.6 |
|        3 |          2 |        3 |              3 |           2 | 2015-10-29 13:13:36 |           1 |      5.00 |           16 |
|        4 |          1 |        1 |              1 |           1 | 2015-11-29 17:43:57 |           1 |      5.00 |          7.2 |
|        5 |          3 |        2 |              4 |           3 | 2015-12-01 18:12:02 |           2 |     22.00 |         20.7 |
|        6 |          2 |        2 |              3 |           2 | 2015-12-02 18:40:28 |           1 |     15.00 |         11.6 |
+----------+------------+----------+----------------+-------------+---------------------+-------------+-----------+--------------+
6 rows in set (0.01 sec)
```

Additionally the trigger updates the values in BOOK_ORDER as well, which is seen below.

```
mysql> SELECT * FROM BOOK_ORDER;
+----------+---------------+-------------+-------+-------+-----------------+
| Order_id | ISBN          | Customer_id | Count | Sale  | Total_weight_oz |
+----------+---------------+-------------+-------+-------+-----------------+
|        1 | 9780345391803 |           1 |     1 | 15.00 |            11.6 |
|        5 | 9780345391803 |           3 |     1 | 15.00 |            11.6 |
|        6 | 9780345391803 |           2 |     1 | 15.00 |            11.6 |
|        5 | 9780394800011 |           3 |     1 |  7.00 |             9.1 |
|        2 | 9780394800165 |           3 |     2 | 18.00 |            17.6 |
|        4 | 9780439708180 |           1 |     1 |  5.00 |             7.2 |
|        1 | 9780440245919 |           1 |     1 |  5.00 |            12.6 |
|        3 | 9780451169518 |           2 |     1 |  5.00 |              16 |
+----------+---------------+-------------+-------+-------+-----------------+
8 rows in set (0.01 sec)
```

And upon viewing the BOOK table we see the appropriate changes are made to Stock and Reorder, where Stock is now 0 and Reorder is 1.

```
+---------------+-------------------------------------+-----------+-------+-------------+---------+
| ISBN          | Title                               | Weight_oz | Cost  | Stock_count | Reorder |
+---------------+-------------------------------------+-----------+-------+-------------+---------+
| 9780345391803 | The Hitch Hiker's Guide to the Galaxy |      11.6 | 15.00 |           0 |       1 |
| 9780394800011 | The Cat in the Hat                  |       9.1 |  7.00 |           2 |       0 |
| 9780394800165 | Green Eggs and Ham                  |       8.8 |  9.00 |           0 |       1 |
| 9780439708180 | Harry Potter and the Sorcerer's Stone |       7.2 |  5.00 |           2 |       0 |
| 9780440245919 | A Time to Kill                      |      12.6 |  5.00 |           1 |       0 |
| 9780451169518 | IT                                  |        16 |  5.00 |           3 |       0 |
| 9780553380163 | A Brief History of Time             |      11.4 | 20.00 |           2 |       0 |
+---------------+-------------------------------------+-----------+-------+-------------+---------+
7 rows in set (0.00 sec)
```

Finally the stored procedure Invoice_Populate is called to produce the necessary entry in INVOICE for the new order.

```
mysql> CALL Invoice_Populate(6);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM INVOICE;
+------------+----------+-------------+------------+------------------+-----------+----------+------------+
| Invoice_id | Order_id | Customer_id | Payment_id | Payment_received | Ship_cost | Tax_cost | Total_cost |
+------------+----------+-------------+------------+------------------+-----------+----------+------------+
|          5 |        1 |           1 |          1 |                1 |      8.47 |     1.20 |      29.67 |
|          6 |        2 |           3 |          3 |                1 |     22.00 |     1.08 |      41.08 |
|          7 |        3 |           2 |          2 |                1 |     12.00 |     0.30 |      17.30 |
|          9 |        4 |           1 |       NULL |                0 |      2.52 |     0.30 |       7.82 |
|         10 |        5 |           3 |       NULL |                0 |     25.88 |     1.32 |      49.20 |
|         11 |        6 |           2 |       NULL |                0 |     14.50 |     0.90 |      30.40 |
+------------+----------+-------------+------------+------------------+-----------+----------+------------+
6 rows in set (0.00 sec)
```

4.4.2. Queries Examples

List all customers by the number of books purchased from most books purchased to least books purchased:

```
mysql> SELECT CUSTOMER.Customer_id, SUM(FULL_ORDER.Total_books) AS 'Books Purchased' FROM
 FULL_ORDER JOIN CUSTOMER ON
    -> FULL_ORDER.Customer_id = CUSTOMER.Customer_id GROUP BY CUSTOMER.Customer_id ORDER
BY SUM(FULL_ORDER.Total_books) DESC;
+-------------+-----------------+
| Customer_id | Books Purchased |
+-------------+-----------------+
|           3 |               4 |
|           1 |               3 |
|           2 |               2 |
+-------------+-----------------+
3 rows in set (0.02 sec)
```

List authors of all books purchased by a customer:

```
mysql> SELECT Customer_id, AUTHOR.First_name, AUTHOR. Middle_init, AUTHOR.Last_name FROM
BOOK_ORDER JOIN BOOK_AUTHOR ON
    -> BOOK_ORDER.ISBN = BOOK_AUTHOR.ISBN JOIN AUTHOR ON BOOK_AUTHOR.Author_id = AUTHOR.A
uthor_id WHERE Customer_id =1;
+-------------+------------+-------------+-----------+
| Customer_id | First_name | Middle_init | Last_name |
+-------------+------------+-------------+-----------+
|           1 | Douglas    | NULL        | Adams     |
|           1 | Eoin       | NULL        | Colfer    |
|           1 | J          | K           | Rowling   |
|           1 | John       | NULL        | Grisham   |
+-------------+------------+-------------+-----------+
4 rows in set (0.01 sec)
```

List genres by book purchases in descending order:

```
mysql> SELECT GENRE.Description, SUM(BOOK_ORDER.COUNT) AS 'Number Purchased' FROM BOOK_OR
DER JOIN BOOK_GENRE ON BOOK_ORDER.ISBN = BOOK_GENRE.ISBN JOIN GENRE ON BOOK_GENRE.Genre_i
d = GENRE.Genre_id GROUP BY Genre.Genre_id ORDER BY SUM(BOOK_ORDER.COUNT) DESC;
+-------------+------------------+
| Description | Number Purchased |
+-------------+------------------+
| Adventure   |                4 |
| Mystery     |                4 |
| Children    |                4 |
| Humor       |                3 |
| Education   |                3 |
| Horror      |                1 |
+-------------+------------------+
6 rows in set (0.00 sec)
```

5. CRUD Matrix

The following Create, Read, Update, and Delete Matrix reflects the interactions between entities and the database application functions that are employed.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16
F1												CR UD	CR UD	CR UD	CR UD	
F2	R	R	R	CR UD	CR UD	CR UD										
F3		CR UD			CR UD											
F4	CR UD			RU												
F5			CR UD	R		CR UD										
F6								RU	CR UD							
F7								R						R		CR UD
F8								CR UD	R							RU D
F9				RU			CR UD	RU		R	R	R				
F10	R	R	R	R	R	R	R									
F11								RU			R	CR UD				
F12											R			CR UD		RU
F13								RU	R							
F14	R	R	R	R	R	R	R	R		R	R					
F15											RU					

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16
F16								R			R				CR UD	
F17										CR UD	R					

5.1. List of Entity Types

E1 - PUBLISHER
E2 - BOOK
E3 - AUTHOR
E4 - GENRE
E5 - BOOK
E6 - BOOK_AUTHOR
E7 - BOOK_GENRE
E8 - BOOK_ORDER
E9 - FULL_ORDER
E10 - SHIPPING_METHOD
E11 - ORDER_HISTORY
E12 - CUSTOMER
E13 - SHIPPING_ADDRESS
E14 - BILLING_ADDRESS
E15 - CUSTOMER_PAYMENT
E16 - ORDER_STATUS
E17 - INVOICE

5.2. List of Functions

F1 - Insert/Update/Delete customer information
F2 - Insert/Update/Delete book information
F3 - Insert/Update/Delete author information
F4 - Insert/Update/Delete publisher information
F5 - Insert/Update/Delete genre information
F6 - Insert/Update/Delete shipping method information
F7 - Insert/Update/Delete invoice
F8 - Insert/Update full order when customer purchases a book
F9 - Insert/Update book orders for customer related to a full order
F10 - Searches books for user to select a purchase
F11 - Chooses a shipping address for an order
F12 - Chooses a method of payment for an order
F13 - Chooses a shipping method for an order
F14 - Shows the purchase history of a customer
F15 - Validates customer login
F16 - Insert/Update/Delete/Read order status
F17 - Insert/Update/Delete order history of a customer

6. Concluding Remarks

The database project was an interesting one but really highlighted the depth of the complexity and scope of material required in database work. Additionally I can very much appreciate the need for thorough and deep conceptual design work. While that process seemed like a long one, I can see how that time spent was an investment in future implementation work. Further I can see the extreme importance of a good RDBMS tool. While I used mySQL WorkBench, next time and for future work I would rather choose to use PostgreSQL. This tool seems much more powerful and professional than mySQL and mySQL Workbench.

I felt during this database project I had a particular strength in the coding aspect of the project. I enjoyed coding the queries, triggers, and procedures and would look forward to future learning in this regard. I am sure much of my coding was not the best or most efficient but I plan on developing further in this area, particularly in conjunction with a programming framework like JDBC.

What was challenging was the design of the database in regards to the functionality of the database and the functionality of the database application. Knowing where one ended and the other began was a difficult one. Perhaps as I gain more experience in database design this aspect will become easier. Further I hope the more advanced aspects of database design will become more obvious in the design process.

Lastly while I feel I gained a lot of experience in database design and querying, I do feel I need more experience in database management. I can see how managing an operational database that is a very large is a big challenge and involves a much deeper set of skill sets. From optimization, to security, to management that allows operations, I feel this area is a large one to learn and probably only comes through years of hands on practice under expert database administrators.

Appendices – The following section firstly shows the pertinent SQL statements for object creation, data insertion, and select statements for sample data viewing. While the object creation and select statements are complete, the insert statements listed represent only a sample of the insertion statements as those shown essentially reflect the form found in each insertion statement. Finally the list of figures is presented.

A. DDL, INSERT, SELECT Statements

Database Object Creation:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
```

```
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema ThompsonBookStoreDB
-- -----------------------------------------------------
-- Bookstore for class
DROP SCHEMA IF EXISTS `ThompsonBookStoreDB` ;

-- -----------------------------------------------------
-- Schema ThompsonBookStoreDB
--
-- Bookstore for class
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `ThompsonBookStoreDB` DEFAULT
CHARACTER SET utf8 COLLATE utf8_general_ci ;
USE `ThompsonBookStoreDB` ;

-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`PUBLISHER`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`PUBLISHER` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`PUBLISHER` (
  `Publisher_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Name` VARCHAR(45) NOT NULL COMMENT '',
  `City` VARCHAR(45) NOT NULL COMMENT '',
  `State` VARCHAR(2) NULL COMMENT '',
  `Country` VARCHAR(45) NOT NULL COMMENT '',
  PRIMARY KEY (`Publisher_id`)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`BOOK`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`BOOK` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`BOOK` (
  `ISBN` DOUBLE NOT NULL COMMENT '',
  `Publisher_id` INT NOT NULL COMMENT '',
  `Title` VARCHAR(45) NULL COMMENT '',
  `Edition` VARCHAR(20) NULL COMMENT '',
```

```
  `Hardcover` TINYINT(1) NULL COMMENT '',
  `Paperback` TINYINT(1) NULL COMMENT '',
  `Weight_oz` FLOAT UNSIGNED NOT NULL COMMENT '',
  `Publish_date` DATE NOT NULL COMMENT '',
  `Cost` DECIMAL(10,2) NOT NULL COMMENT '',
  `Page_count` INT UNSIGNED NULL COMMENT '',
  `Stock_count` INT NOT NULL COMMENT '',
  `Reorder` TINYINT(1) NULL DEFAULT 0 COMMENT '',
  PRIMARY KEY (`ISBN`)  COMMENT '',
  INDEX `fk_BOOK_PUBLISHER1_idx` (`Publisher_id` ASC)  COMMENT '',
  CONSTRAINT `fk_BOOK_PUBLISHER1`
    FOREIGN KEY (`Publisher_id`)
    REFERENCES `ThompsonBookStoreDB`.`PUBLISHER` (`Publisher_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`AUTHOR`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`AUTHOR` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`AUTHOR` (
  `Author_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `First_name` VARCHAR(45) NOT NULL COMMENT '',
  `Last_name` VARCHAR(45) NOT NULL COMMENT '',
  `Middle_init` VARCHAR(1) NULL COMMENT '',
  PRIMARY KEY (`Author_id`)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`GENRE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`GENRE` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`GENRE` (
  `Genre_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Description` VARCHAR(75) NOT NULL COMMENT '',
  PRIMARY KEY (`Genre_id`)  COMMENT '')
ENGINE = InnoDB
```

```
COMMENT = 'This is a reference entity that holds genres related to books. \n\nid = 1: Adventure\nid = 2: Romance\nid = 3: Mystery\nid = 4: Science Fiction\nid = 5: Horror \nid = 6: Humor\nid = 7: Children\nid = 8: Action\nid = 10: Science\nid = 11: Religious \nid = 12: Self-Improvement\nid = 13: History\nid = 14: Non-Fiction\nid = 15: Education';


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`CUSTOMER`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`CUSTOMER` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`CUSTOMER` (
  `Customer_id` INT NOT NULL COMMENT '',
  `First_name` VARCHAR(45) NULL COMMENT '',
  `Last_name` VARCHAR(45) NULL COMMENT '',
  `Middle_init` VARCHAR(1) NULL COMMENT '',
  `DOB` DATE NULL COMMENT '',
  `Creation_date` DATE NULL COMMENT '',
  `Email` VARCHAR(45) NULL COMMENT '',
  `Primary_phone` INT(10) NULL COMMENT '',
  `Password` VARCHAR(10) NOT NULL COMMENT '',
  PRIMARY KEY (`Customer_id`)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`ORDER_HISTORY`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`ORDER_HISTORY` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`ORDER_HISTORY` (
  `History_id` INT NOT NULL COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  PRIMARY KEY (`History_id`, `Customer_id`)  COMMENT '',
  INDEX `fk_ORDER_HISTORY_CUSTOMER1_idx` (`Customer_id` ASC)
COMMENT '',
  CONSTRAINT `fk_ORDER_HISTORY_CUSTOMER1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`SHIP_ADDRESS`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`SHIP_ADDRESS` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`SHIP_ADDRESS` (
  `Ship_address_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  `Street_address` VARCHAR(45) NOT NULL COMMENT '',
  `City` VARCHAR(45) NOT NULL COMMENT '',
  `State` VARCHAR(2) NOT NULL COMMENT '',
  `Zip` INT(5) UNSIGNED NOT NULL COMMENT '',
  `Unit_number` VARCHAR(10) NULL COMMENT '',
  PRIMARY KEY (`Ship_address_id`)  COMMENT '',
  INDEX `fk_SHIP_ADDRESS_CUSTOMER1_idx` (`Customer_id` ASC)  COMMENT
'',
  CONSTRAINT `fk_SHIP_ADDRESS_CUSTOMER1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`SHIPPING_METHOD`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`SHIPPING_METHOD` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`SHIPPING_METHOD` (
  `Method_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Description` VARCHAR(75) NOT NULL COMMENT '',
  `Cost_per_oz` DECIMAL(10,2) NOT NULL COMMENT '',
  PRIMARY KEY (`Method_id`)  COMMENT '')
ENGINE = InnoDB
COMMENT = 'This is a reference entity for shipping methods:\nid = 1: Ground\nid = 2:
Overnight\nid = 3: Priority';


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`FULL_ORDER`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`FULL_ORDER` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`FULL_ORDER` (
  `Order_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `History_id` INT NOT NULL COMMENT '',
  `Method_id` INT NOT NULL COMMENT '',
  `Ship_address_id` INT NOT NULL COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  `Order_date` TIMESTAMP NOT NULL COMMENT '',
  `Total_books` INT NULL DEFAULT 0 COMMENT '',
  `Book_cost` DECIMAL(10,2) NULL DEFAULT 0.00 COMMENT '',
  `Total_weight` FLOAT NULL DEFAULT 0 COMMENT '',
  PRIMARY KEY (`Order_id`, `Customer_id`)  COMMENT '',
  INDEX `fk_ORDER_ORDER_HISTORY1_idx` (`History_id` ASC)  COMMENT '',
  INDEX `fk_ORDER_CUSTOMER1_idx` (`Customer_id` ASC)  COMMENT '',
  INDEX `fk_ORDER_SHIP_ADDRESS1_idx` (`Ship_address_id` ASC)  COMMENT
'',
  INDEX `fk_ORDER_SHIPPING_METHOD1_idx` (`Method_id` ASC)  COMMENT '',
  CONSTRAINT `fk_ORDER_ORDER_HISTORY1`
    FOREIGN KEY (`History_id`)
    REFERENCES `ThompsonBookStoreDB`.`ORDER_HISTORY` (`History_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_ORDER_CUSTOMER1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_ORDER_SHIP_ADDRESS1`
    FOREIGN KEY (`Ship_address_id`)
    REFERENCES `ThompsonBookStoreDB`.`SHIP_ADDRESS` (`Ship_address_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_ORDER_SHIPPING_METHOD1`
    FOREIGN KEY (`Method_id`)
    REFERENCES `ThompsonBookStoreDB`.`SHIPPING_METHOD` (`Method_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `ThompsonBookStoreDB`.`BOOK_ORDER`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`BOOK_ORDER` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`BOOK_ORDER` (
  `ISBN` DOUBLE NOT NULL COMMENT '',
  `Count` INT UNSIGNED NOT NULL COMMENT '',
  `Sale` DECIMAL(10,2) UNSIGNED NOT NULL COMMENT '',
  `Total_weight_oz` FLOAT NULL COMMENT '',
  `ORDER_Order_id` INT NOT NULL COMMENT '',
  `ORDER_Customer_id` INT NOT NULL COMMENT '',
  INDEX `fk_BOOK_INSTANCE_BOOK1_idx` (`ISBN` ASC)  COMMENT '',
  PRIMARY KEY (`ISBN`, `ORDER_Order_id`, `ORDER_Customer_id`)  COMMENT
'',
  INDEX `fk_BOOK_ORDER_ORDER1_idx` (`ORDER_Order_id` ASC,
`ORDER_Customer_id` ASC)  COMMENT '',
  CONSTRAINT `fk_BOOK_INSTANCE_BOOK1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `ThompsonBookStoreDB`.`BOOK` (`ISBN`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_BOOK_ORDER_ORDER1`
    FOREIGN KEY (`ORDER_Order_id` , `ORDER_Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`FULL_ORDER` (`Order_id` ,
`Customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`ORDER_STATUS`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`ORDER_STATUS` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`ORDER_STATUS` (
  `Status_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Delivered` TINYINT(1) NULL COMMENT '',
  `Shipped` TINYINT(1) NULL COMMENT '',
  `Processed` TINYINT(1) NULL COMMENT '',
  `ORDER_Order_id` INT NOT NULL COMMENT '',
  `ORDER_Customer_id` INT NOT NULL COMMENT '',
  `Shipped_date` DATE NULL COMMENT '',
```

```
  `Processed_date` DATE NULL COMMENT '',
  `Delivered_date` DATE NULL COMMENT '',
  PRIMARY KEY (`Status_id`, `ORDER_Order_id`, `ORDER_Customer_id`)
COMMENT '',
  INDEX `fk_ORDER_STATUS_ORDER1_idx` (`ORDER_Order_id` ASC,
`ORDER_Customer_id` ASC)  COMMENT '',
  CONSTRAINT `fk_ORDER_STATUS_ORDER1`
    FOREIGN KEY (`ORDER_Order_id` , `ORDER_Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`FULL_ORDER` (`Order_id` ,
`Customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`BILL_ADDRESS`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`BILL_ADDRESS` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`BILL_ADDRESS` (
  `Bill_address_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  `Street_address` VARCHAR(45) NOT NULL COMMENT '',
  `City` VARCHAR(45) NOT NULL COMMENT '',
  `State` VARCHAR(2) NOT NULL COMMENT '',
  `Zip_code` INT(5) UNSIGNED NOT NULL COMMENT '',
  `Unit_number` VARCHAR(10) NULL COMMENT '',
  PRIMARY KEY (`Bill_address_id`)  COMMENT '',
  INDEX `fk_BILL_ADDRESS_CUSTOMER1_idx` (`Customer_id` ASC)  COMMENT
'',
  CONSTRAINT `fk_BILL_ADDRESS_CUSTOMER1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` ;
```

```
CREATE TABLE IF NOT EXISTS
`ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (
  `Payment_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  `Credit_card_type` VARCHAR(45) NOT NULL COMMENT '',
  `Expire_date` DATE NOT NULL COMMENT '',
  `Card_number` INT UNSIGNED NOT NULL COMMENT '',
  `First_name` VARCHAR(45) NOT NULL COMMENT '',
  `Last_name` VARCHAR(45) NOT NULL COMMENT '',
  `Middle_init` VARCHAR(1) NULL COMMENT '',
  `Security_code` INT UNSIGNED NOT NULL COMMENT '',
  PRIMARY KEY (`Payment_id`)  COMMENT '',
  INDEX `fk_CUSTOMER_PAYMENT_CUSTOMER1_idx` (`Customer_id` ASC)
COMMENT '',
  CONSTRAINT `fk_CUSTOMER_PAYMENT_CUSTOMER1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`BOOK_AUTHOR`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`BOOK_AUTHOR` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`BOOK_AUTHOR` (
  `Author_id` INT NOT NULL COMMENT '',
  `ISBN` DOUBLE NOT NULL COMMENT '',
  PRIMARY KEY (`Author_id`, `ISBN`)  COMMENT '',
  INDEX `fk_BOOK_AUTHOR_BOOK1_idx` (`ISBN` ASC)  COMMENT '',
  CONSTRAINT `fk_BOOK_AUTHOR_AUTHOR1`
    FOREIGN KEY (`Author_id`)
    REFERENCES `ThompsonBookStoreDB`.`AUTHOR` (`Author_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_BOOK_AUTHOR_BOOK1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `ThompsonBookStoreDB`.`BOOK` (`ISBN`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`BOOK_GENRE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`BOOK_GENRE` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`BOOK_GENRE` (
  `Genre_id` INT NOT NULL COMMENT '',
  `ISBN` DOUBLE NOT NULL COMMENT '',
  PRIMARY KEY (`Genre_id`, `ISBN`)  COMMENT '',
  INDEX `fk_BOOK_GENRE_BOOK1_idx` (`ISBN` ASC)  COMMENT '',
  CONSTRAINT `fk_BOOK_GENRE_GENRE1`
    FOREIGN KEY (`Genre_id`)
    REFERENCES `ThompsonBookStoreDB`.`GENRE` (`Genre_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_BOOK_GENRE_BOOK1`
    FOREIGN KEY (`ISBN`)
    REFERENCES `ThompsonBookStoreDB`.`BOOK` (`ISBN`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ThompsonBookStoreDB`.`INVOICE`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ThompsonBookStoreDB`.`INVOICE` ;

CREATE TABLE IF NOT EXISTS `ThompsonBookStoreDB`.`INVOICE` (
  `Invoice_id` INT NOT NULL AUTO_INCREMENT COMMENT '',
  `Order_id` INT NOT NULL COMMENT '',
  `Customer_id` INT NOT NULL COMMENT '',
  `Payment_id` INT NULL COMMENT '',
  `Payment_received` TINYINT(1) NOT NULL COMMENT '',
  `Total_cost` DECIMAL(10,2) UNSIGNED NOT NULL COMMENT '',
  `Book_cost` DECIMAL(10,2) NOT NULL COMMENT '',
  `Tax_cost` DECIMAL(10,2) NULL COMMENT '',
  `Ship_cost` DECIMAL(10,2) NULL COMMENT '',
  PRIMARY KEY (`Invoice_id`)  COMMENT '',
```

```
  INDEX `fk_Invoice_ORDER1_idx` (`Order_id` ASC, `Customer_id` ASC)
COMMENT '',
  INDEX `fk_Invoice_CUSTOMER_PAYMENT1_idx` (`Payment_id` ASC)
COMMENT '',
  CONSTRAINT `fk_Invoice_ORDER1`
    FOREIGN KEY (`Order_id` , `Customer_id`)
    REFERENCES `ThompsonBookStoreDB`.`FULL_ORDER` (`Order_id` ,
`Customer_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Invoice_CUSTOMER_PAYMENT1`
    FOREIGN KEY (`Payment_id`)
    REFERENCES `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (`Payment_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Examples of Data Insertion

Customer Insert:

```
INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`, `First_name`,
`Last_name`, `DOB`, `Email`, `Primary_phone`, `Password`) VALUES ('4', 'Kimi', 'Lee',
'1990-08-27', 'kimi@email.com', '1112223333', 'sushi');
INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER` (`Customer_id`, `First_name`,
`Last_name`, `Middle_init`, `DOB`, `Email`, `Password`) VALUES ('5', 'Joe', 'Little', 'B',
'2007-06-12', 'joey@dinosaur.com', 'candy');
```

CustomerPayment Insert:

```
INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (`Payment_id`,
`Customer_id`, `Credit_card_type`, `Expire_date`, `Card_number`, `First_name`,
`Last_name`, `Middle_init`, `Security_code`) VALUES ('2', '2', 'MASTERCARD',
'2018-02-22', '3333444455556666', 'Mary', 'Mo', 'A', '222');
UPDATE `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` SET `Middle_init`=NULL
WHERE `Payment_id`='1';
INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (`Payment_id`,
`Customer_id`, `Credit_card_type`, `Expire_date`, `Card_number`, `First_name`,
`Last_name`, `Middle_init`, `Security_code`) VALUES ('3', '3', 'VISA', '2016-01-03',
'1000200030004000', 'Cletus', 'Parker', 'S', '111');
```

INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (`Payment_id`, `Customer_id`, `Credit_card_type`, `Expire_date`, `Card_number`, `First_name`, `Last_name`, `Security_code`) VALUES ('4', '4', 'DISCOVER', '2020-02-10', '4444555566667777', 'Kimi', 'Lee', '666');
INSERT INTO `ThompsonBookStoreDB`.`CUSTOMER_PAYMENT` (`Payment_id`, `Customer_id`, `Credit_card_type`, `Expire_date`, `Card_number`, `First_name`, `Last_name`, `Middle_init`, `Security_code`) VALUES ('5', '5', 'MASTERCARD', '2018-03-04', '9999888877776666', 'Joe', 'Little', 'B', '123');

Shipping Address Insert:
INSERT INTO `ThompsonBookStoreDB`.`SHIP_ADDRESS` (`Ship_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip`) VALUES ('3', '2', '123', 'Baltimore', 'MD', '27777');
INSERT INTO `ThompsonBookStoreDB`.`SHIP_ADDRESS` (`Ship_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip`) VALUES ('4', '3', '456', 'Westminster', 'MD', '33333');
INSERT INTO `ThompsonBookStoreDB`.`SHIP_ADDRESS` (`Ship_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip`, `Unit_number`) VALUES ('5', '4', '12334', 'Ball', 'MD', '12344', '5');
INSERT INTO `ThompsonBookStoreDB`.`SHIP_ADDRESS` (`Ship_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip`) VALUES ('6', '5', '678', 'Rockville', 'MD', '77777');

Billing Address Insert:
INSERT INTO `ThompsonBookStoreDB`.`BILL_ADDRESS` (`Bill_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip_code`) VALUES ('2', '2', '123 Pine Rd', 'Baltimore', 'MD', '27777');
INSERT INTO `ThompsonBookStoreDB`.`BILL_ADDRESS` (`Bill_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip_code`) VALUES ('3', '3', '456 Oak St', 'Westminster', 'MD', '33333');
INSERT INTO `ThompsonBookStoreDB`.`BILL_ADDRESS` (`Bill_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip_code`, `Unit_number`) VALUES ('4', '4', '12334 Hickory', 'Ball', 'MD', '12344', '5');
INSERT INTO `ThompsonBookStoreDB`.`BILL_ADDRESS` (`Bill_address_id`, `Customer_id`, `Street_address`, `City`, `State`, `Zip_code`) VALUES ('5', '5', '678 Pecan', 'Rockville', 'MD', ‘77777');

Book Order Insert:
INSERT INTO `ThompsonBookStoreDB`.`BOOK_ORDER` (`Order_id`, `ISBN`, `Customer_id`, `Count`) VALUES ('2', '9780394800165', '3', ‘2');

<u>SELECT statements:</u>

SELECT * FROM AUTHOR;
SELECT * FROM BILL_ADDRESS;

```
SELECT * FROM BOOK;
SELECT * FROM BOOK_AUTHOR;
SELECT * FROM BOOK_GENRE;
SELECT * FROM CUSTOMER;
SELECT * FROM CUSTOMER_PAYMENT;
SELECT * FROM FULL_ORDER;
SELECT * FROM GENRE;
SELECT * FROM INVOICE;
SELECT * FROM ORDER_HISTORY;
SELECT * FROM ORDER_STATUS;
SELECT * FROM PUBLISHER;
SELECT * FROM SHIP_ADDRESS;
SELECT * FROM SHIPPING_METHOD;
```

B. Data Dictionary Index

Field	Table
Author_id	AUTHOR
Author_id	BOOK_ AUTHOR
Bill_ address_id	BILL_ ADDRESS
Book_cost	FULL_ ORDER
Card_ number	CUSTOMER_PAYMENT
City	BILL_ ADDRESS
City	PUBLISHER
City	SHIP_ ADDRESS
Cost	BOOK
Cost_per_oz	SHIPPING_METHOD
Count	BOOK_ ORDER
Country	PUBLISHER
Creation_ date	CUSTOMER
Credit_card_type	CUSTOMER_PAYMENT

Field	Table
Customer_ id	BILL_ ADDRESS
Customer_ id	BOOK_ ORDER
Customer_ id	CUSTOMER
Customer_ id	FULL_ ORDER
Customer_ id	INVOICE
Customer_ id	ORDER_ HISTORY
Customer_ id	ORDER_ STATUS
Customer_ id	SHIP_ ADDRESS
Customer_id	CUSTOMER_PAYMENT
Delivered	ORDER_ STATUS
Delivered_ date	ORDER_ STATUS
Description	GENRE
Description	SHIPPING_METHOD
DOB	CUSTOMER
Edition	BOOK
Email	CUSTOMER
Expire_ date	CUSTOMER_PAYMENT
First_name	AUTHOR
First_name	CUSTOMER
First_name	CUSTOMER_PAYMENT
Genre_id	BOOK_ AUTHOR

Field	Table
Genre_id	GENRE
Hardcover	BOOK
History_id	FULL_ ORDER
History_id	ORDER_ HISTORY
Invoice_id	INVOICE
ISBN	BOOK
ISBN	BOOK_ AUTHOR
ISBN	BOOK_ GENRE
ISBN	BOOK_ ORDER
Last_name	AUTHOR
Last_name	CUSTOMER
Last_name	CUSTOMER_PAYMENT
Method_id	FULL_ ORDER
Method_id	SHIPPING_METHOD
Middle_init	AUTHOR
Middle_init	CUSTOMER
Middle_init	CUSTOMER_PAYMENT
Name	PUBLISHER
Order_date	FULL_ ORDER
Order_id	BOOK_ GENRE
Order_id	FULL_ ORDER
Order_id	INVOICE
Order_id	ORDER_ STATUS

Field	Table
Page_count	BOOK
Paperback	BOOK
Password	CUSTOMER
Payment _received	INVOICE
Payment_id	CUSTOMER_PAYMENT
Payment_id	INVOICE
Primary _phone	CUSTOMER
Processed	ORDER_ STATUS
Processed_ date	ORDER_ STATUS
Publish_ date	BOOK
Publisher _id	BOOK
Publisher _id	PUBLISHER
Reorder	BOOK
Sale	BOOK_ ORDER
Security_ code	CUSTOMER_PAYMENT
Ship_address _id	FULL_ ORDER
Ship_address _id	SHIP_ ADDRESS
Ship_cost	INVOICE
Shipped_ date	ORDER_ STATUS
Shopped	ORDER_ STATUS

Field	Table
State	BILL_ ADDRESS
State	PUBLISHER
State	SHIP_ ADDRESS
Status_id	ORDER_ STATUS
Stock_ count	BOOK
Street_ address	BILL_ ADDRESS
Street_ address	SHIP_ ADDRESS
Tax_cost	INVOICE
Title	BOOK
Total_ weight_oz	BOOK_ ORDER
Total_books	FULL_ ORDER
Total_cost	INVOICE
Total_weight	FULL_ ORDER
Unit_ number	BILL_ ADDRESS
Unit_ number	SHIP_ ADDRESS
Weight_oz	BOOK
Zip	SHIP_ ADDRESS
Zip_code	BILL_ ADDRESS

References

1. *Hardware and Software Requirements for Installing SQL Server 2016;* https://msdn.microsoft.com/en-us/library/ms143506.aspx*;* September 2016
2. Gheorghiu, Radu*;* SqlBak Blog: Backup and Monitoring SQL Server databases from the Web; https://sqlbak.com/blog/sql-server-backup-and-restore/; October 2013
3. Database Security, Wikipedia; https://en.wikipedia.org/wiki/Database_security; January 2015
4. Akhtar, Ali Navid; Buchholtz, Jeff; Ryan, Michael; Setty, Kumar; Database Backup and Recovery Best Practices. http://www.isaca.org/Journal/archives/2012/Volume-1/Pages/Database-Backup-and-Recovery-Best-Practices.aspx; 2012