LIBRARY MANAGEMENT SYSTEM

DATABASE DESIGN & SQL

FSDM 2023S

Team Members
Harkirat Singh – C0897852
Sarpreet – C0894124
Jagjot Singh Chopra – C0897833
Chandanjot Singh – C0896984
Piyumika Samarasuriyage – C0900440

Submitted to
Prof. Sagara Samarawickrama

Table of Contents

1. INTRODUCTION

This Database Design project is intended to provide a complete database structure for a Library management System. Database design includes three design steps as

- Conceptual Design
- Logical Design
- Physical Design

Conceptual design is the high level entity relationship that is drawn during the data analysis from available resources. It is independent from any database management system. Evolving from the conceptual design, logical design models the entity relationship diagram in more detail including entities, attributes, and relationships. Here,

- Entities are the people, places, things, or concepts about which the data must be recorded. For example, books and reservations can be considered as prominent entities in a library management system.
- Attributes are properties of entities such as the book title, ISBN and author.
- Relationships are dependencies between or among entities.

Physical design is the transformation of the logical data model representing entities, attributes and relationships into a physically implemented system having database tables, columns and foreign keys representing relationships.

This report is intended to deliver the design process from the logical design to physical design of the library management system along with the illustrations of database theories involved.

2. LOGICAL DESIGN

2.1 Entities of the Library Management System

- Books
- Authors
- Publishers
- Sections
- Languages
- Category (Genres)
- Users
- Roles
- Reservations
- Books Issued (or Borrowing)
- BookAuthor
- BookCopies

2.2 Description of the Entities

Entity	Description
BOOKS	Represents individual books available in the library. Contains information such as BookID, Title, ISBN, Publication Year, Number of Copies, LanguageID (foreign key to Languages), CategoryID (foreign key to Category), SectionID (foreign key to Sections), PublisherID (foreign key to Publishers), etc
AUTHORS	Stores details about the authors of the books. Includes attributes like AuthorID, Name, Biography, etc
PUBLISHERS	Stores details about book publishers. Attributes might include PublisherID, Name, Address, etc.
SECTIONS	Represents different sections or areas such as (Fiction, Non-Fiction, Business, Self-Help etc), within the library where books are categorized. Contains attributes like SectionID and SectionName.
LANGUAGES	Contains information about the languages in which books are available. Attributes may include LanguageID and LanguageName.
CATEGORY (GENRES)	Contains information about book genres or categories such as (Romance, Philosophy, Comics etc). Attributes may include CategoryID and CategoryName.
BOOK_AUTHOR	Stores information about different books and their authors. Contain attributes like BookID, AuthorID etc.
USERS	Represents registered users of the library. Contains attributes like UserID, Username, Password, Email and RoleID as foreign key.

ROLES	Stores information about different user roles within the library system, such as Member, Librarian, Admin, etc. Attributes might include RoleID and RoleName.
RESERVATIONS	Stores details of book reservations made by users. Contains attributes like ReservationID, UserID (foreign key to Users), CopyID (foreign key to BookCopies), Reservation Date, etc.
BOOKS_ISSUED	Tracks the borrowing activity of users. Includes attributes like IssueID, UserID (foreign key to Users), CopyID (foreign key to BookCopies), Issue Date, Due Date, Return Date, etc.
BOOK_COPIES	Tracks the copies of available books. Includes attributes like CopyID, BookID (foreign key to Books).

2.3 Primary Key(s) of Entities

Entity	Primary Key
BOOKS	BOOK_ID
AUTHORS	AUTHOR_ID
PUBLISHERS	PUBLISHER_ID
SECTIONS	SECTION_ID
LANGUAGES	LANGUAGE_ID
CATEGORY (Genres)	CATEGORY_ID
USERS	USER_ID
ROLES	ROLE_ID
RESERVATIONS	RESERVATION_ID
BOOKS_ISSUED	ISSUE_ID
BOOK_AUTHOR	BOOK_ID, AUTHOR_ID
BOOK_COPIES	COPY_ID

2.4 Required/Mandatory and Optional Attributes of Entities

Entity	Mandatory Attributes	Optional Attributes
BOOKS	**Book ID**, Title, ISBN, Publication Year, Number of Copies, Language ID, Category ID, Section ID, Publisher ID	Description, Cover Image, etc.
AUTHORS	**Author ID**, Name	Biography, Birth Date, Death Date, etc.
PUBLISHERS	**Publisher ID**, Name	Address, Contact Info, etc.
SECTIONS	**Section ID**, Section Name	
LANGUAGES	**Language ID**, Language Name	
CATEGORY (Genres)	**Category ID**, Category Name	
USERS	**User ID**, Username, Password, Email, Role ID	Contact Info, Address, etc.
ROLES	**Role ID**, Role Name	
RESERVATIONS	**Reservation ID**, User ID, Book ID, Reservation Date	Expiry Date (if the reservation is cancelled or expires), etc.
BOOKS_ISSUED	**Issue ID**, User ID, Book ID, Issue Date, Due Date	Return Date (if returned), Fine Amount (if applicable), etc.
BOOK_AUTHOR	**Book ID, Author ID**	
BOOK_COPIES	**Copy ID**, Book ID	

2.5 Relationships between Entities

Books can have multiple authors, and authors can write multiple books. This many-to-many relationship is represented using the "BookAuthor" association table.

Each record in the "BookAuthor" table contains a combination of BookID and AuthorID, linking a specific book to its corresponding author(s).

A Book can belong to one Publisher, and a Publisher can publish multiple Books. (1-to-many relationship)

A Book can be in one Section, and a Section can have multiple Books. (1-to-many relationship)

A Book can be available in one Language, and a language can have multiple Books. (1-to-many relationship)

A Book can belong to one Category (Genre), and a Category can have multiple Books. (1-to-many relationship)

Users can have one Role, and a Role can be associated with multiple Users. (1-to-many relationship)

Users can make multiple Reservations, and each Reservation is associated with one User. (1-to-many relationship)

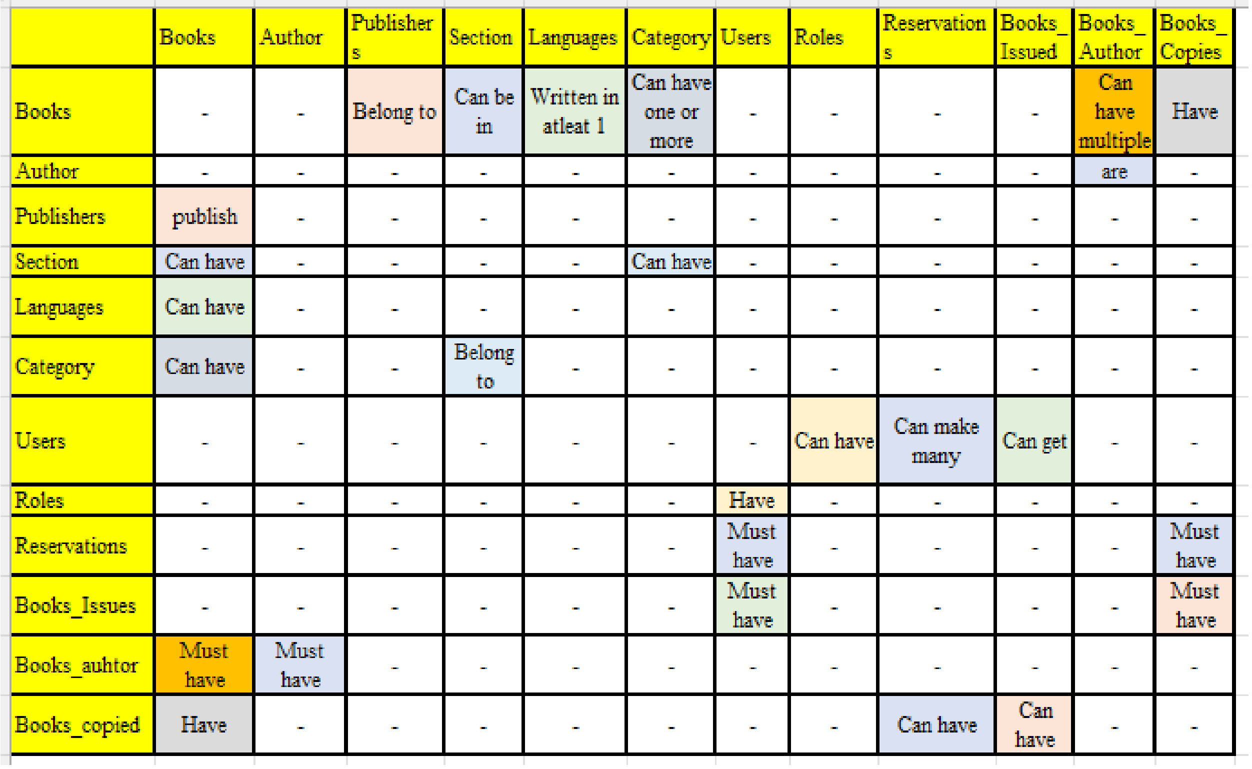
Users can borrow multiple Books (Books Issued), and one User can borrow each book. (1-to-many relationship)

A book must have at least one copy and many copies may belong to same book (1-to-many relationship)

For a book copy, there can be multiple reservations and each Reservation is associated with a copy of the book (1-to-many relationship)

A copy of a book can be issued to many user, each reservation is concerned with a copy of the book (1-to-many relationship).

2.6 Relationship Matrix



	Books	Author	Publishers	Section	Languages	Category	Users	Roles	Reservations	Books_Issued	Books_Author	Books_Copies
Books	-	-	Belong to	Can be in	Written in atleat 1	Can have one or more	-	-	-	-	Can have multiple	Have
Author	-	-	-	-	-	-	-	-	-	-	are	-
Publishers	publish	-	-	-	-	-	-	-	-	-	-	-
Section	Can have	-	-	-	-	Can have	-	-	-	-	-	-
Languages	Can have	-	-	-	-	-	-	-	-	-	-	-
Category	Can have	-	-	Belong to	-	-	-	-	-	-	-	-
Users	-	-	-	-	-	-	-	Can have	Can make many	Can get	-	-
Roles	-	-	-	-	-	-	Have	-	-	-	-	-
Reservations	-	-	-	-	-	-	Must have	-	-	-	-	Must have
Books_Issues	-	-	-	-	-	-	Must have	-	-	-	-	Must have
Books_auhtor	Must have	Must have	-	-	-	-	-	-	-	-	-	-
Books_copied	Have	-	-	-	-	-	-	-	Can have	Can have	-	-

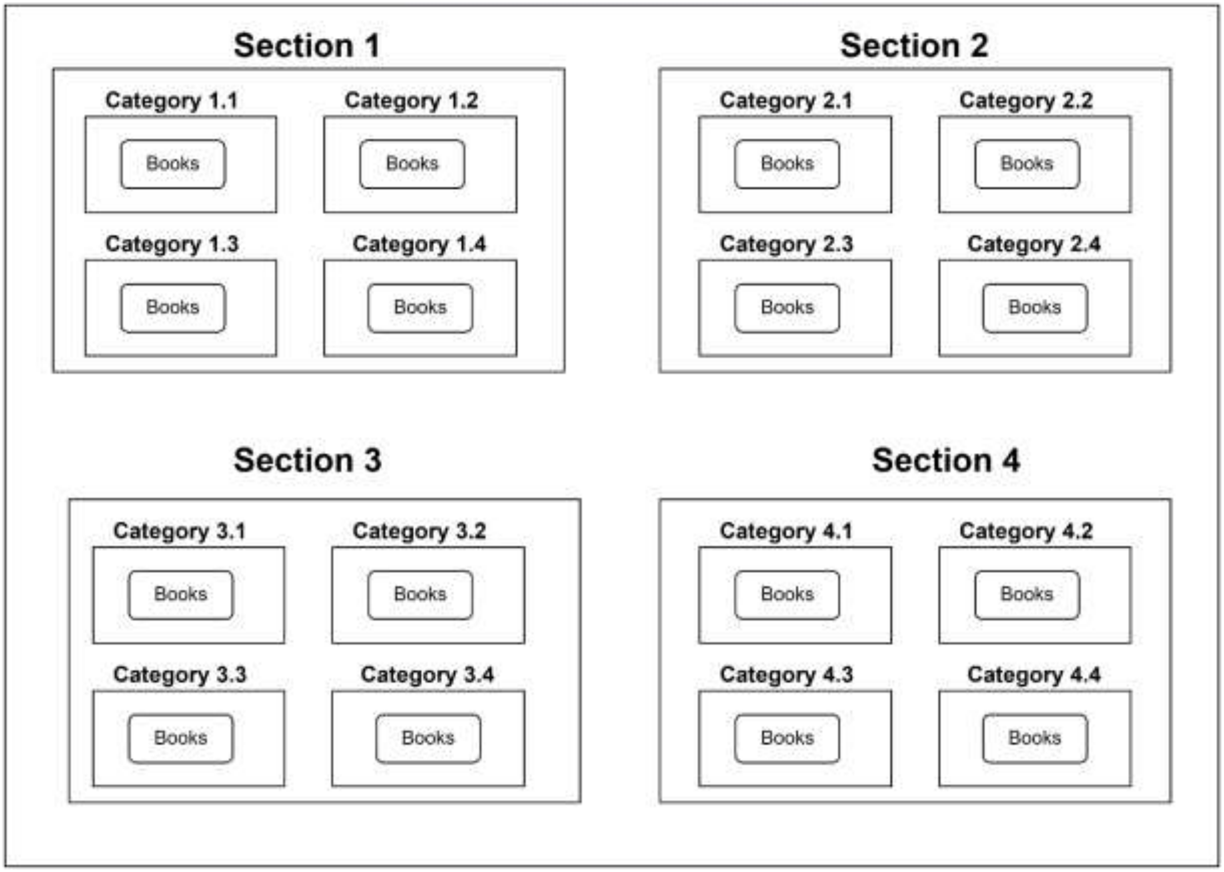
3. ENTITY RELATIONSHIP DIAGRAM

3.1 Library Management System Architecture

Before Moving towards ER- Diagram, let’s understand the **architecture of library management system**.

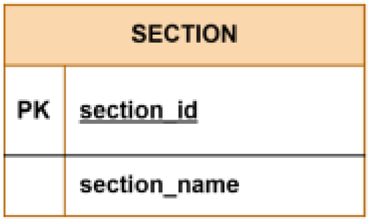
1. The library is divided into further sections. For Example: - Fiction Section, Non- Fiction Section, Children’s Section, Science and Technology Section, Travel Section, Religion and Philosophy Section etc.
2. These sections are also divided into categories because books in the library are divided according to its categories as well as section.
   For example: -
   - The Science and Technology Section is divided into physics category, chemistry category, computer science and engineering and many more.
   - The Religion and Philosophy Section is divided into Christianity category, Buddhism category, Islam Category, Sikhism Category etc.
3. Books are placed in the library according to its category as well as its section.

Library Architecture

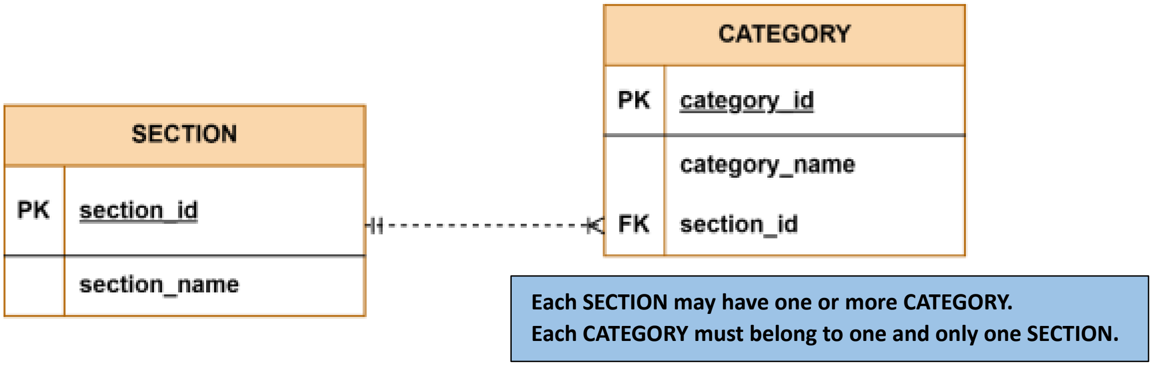


3.2 ER-Diagram Design Process

- Library has many sections. So, **SECTION Entity** is created which contains section id[PK] and section name attributes as shown below.

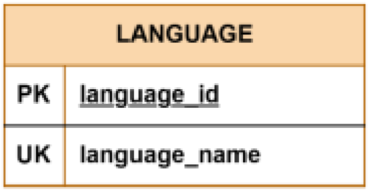


- Every section is divided into further categories. So **CATEGORY Entity** is created which contains category id [PK], category name and section id [FK] attributes as shown below. We clearly know there is **relationship between SECTION and CATEGORY** because category comes under the section.

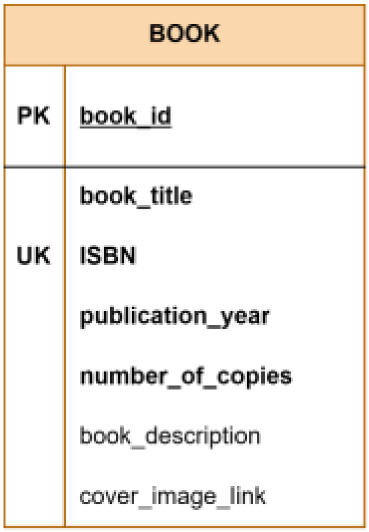


Each SECTION may have one or more CATEGORY.
Each CATEGORY must belong to one and only one SECTION.

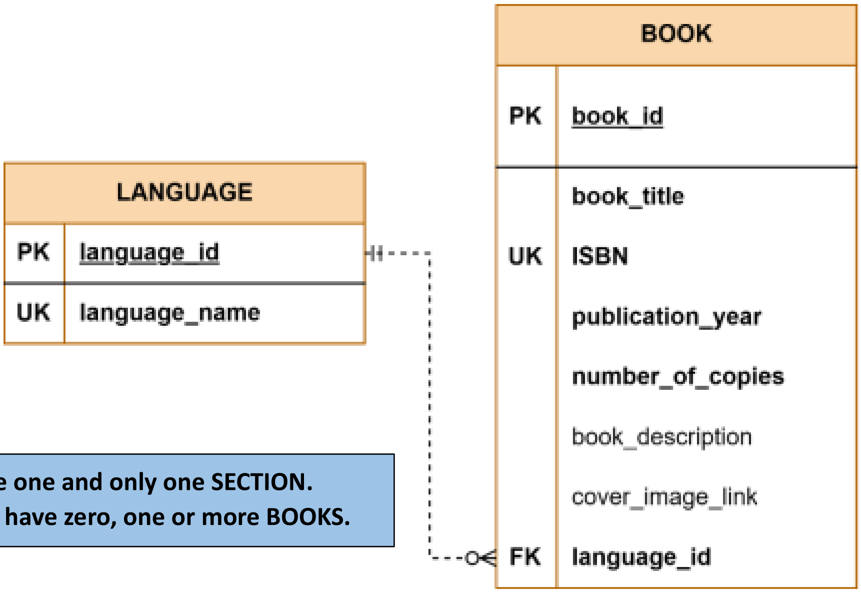
- Books are in so many different languages. So there is need to make **LANGUAGE Entity** which contain language id[PK] and language name[UK] attributes to identiy the book language as shown below. This language Entity will be use in book entity which we will make later on.



- Now let's make **BOOK Entity** from the basic level which contains book id[PK], book title, ISBN[UK], publication year, number of copies, book description, cover image link attributes as shown below.

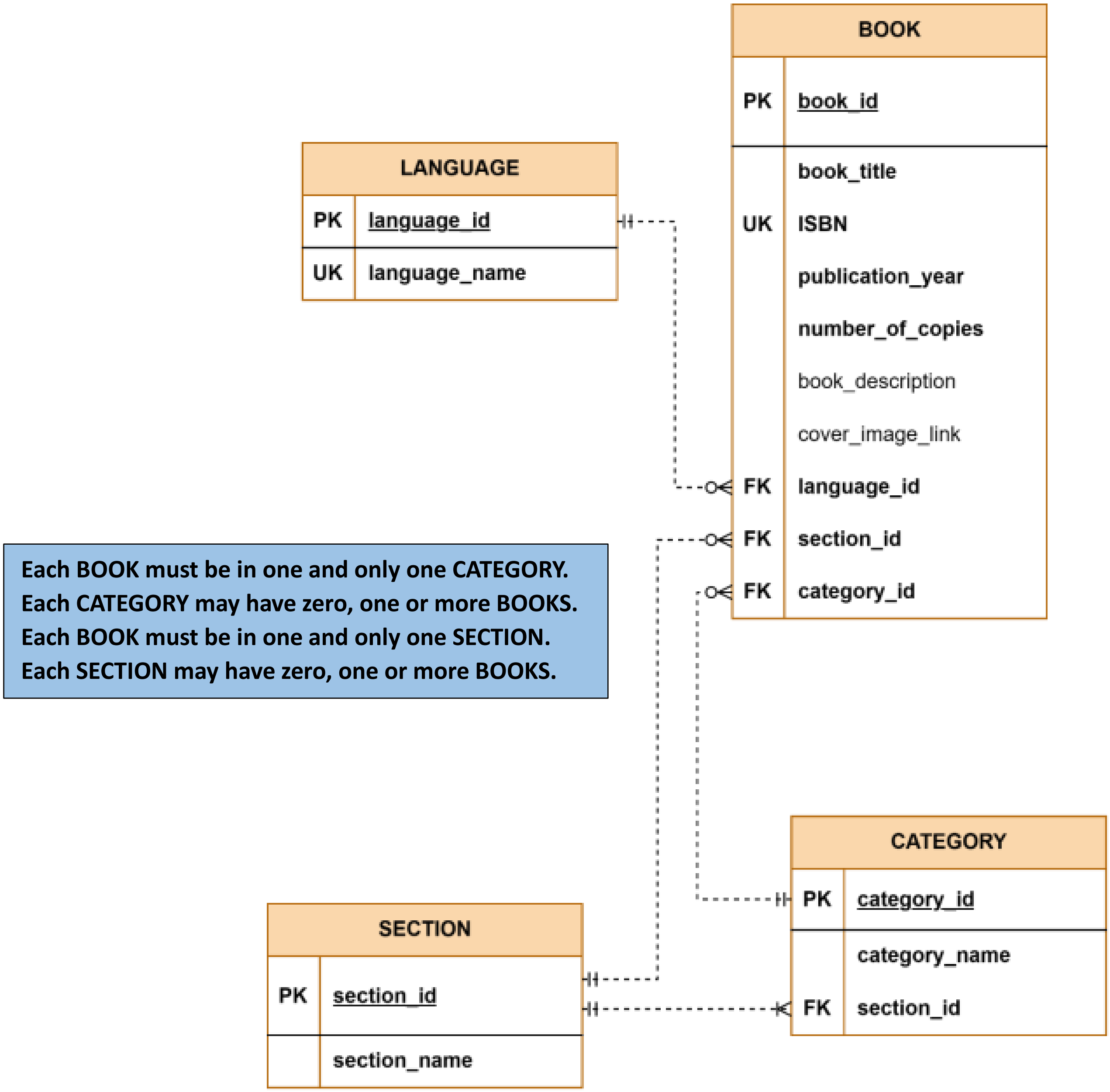


To identify book language, let’s make **connection between BOOK entity and LANGUAGE entity** as shown below.



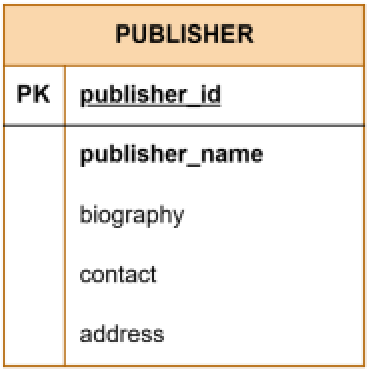
Each BOOK must have one and only one SECTION.
Each LANGUAGE may have zero, one or more BOOKS.

We knew that in the library, books are placed according to category as well as section wise. So we have to **make relationships between BOOK and CATEGORY, BOOK and SECTION** as shown below.

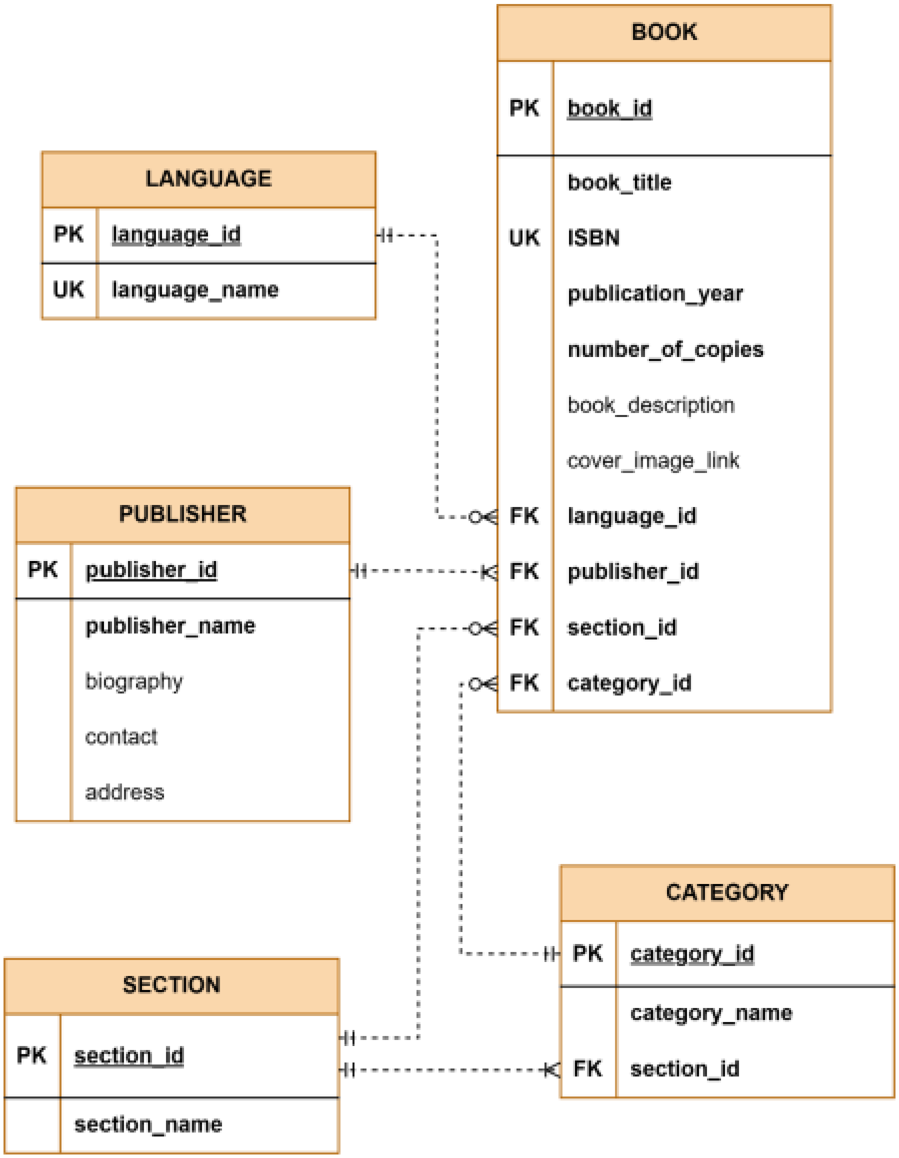


Each BOOK must be in one and only one CATEGORY.
Each CATEGORY may have zero, one or more BOOKS.
Each BOOK must be in one and only one SECTION.
Each SECTION may have zero, one or more BOOKS.

- Now each book has a publisher, so we have to make **PUBLISHER entity** which contain publisher id [PK], publisher name, biography, contact and address attributes as shown below.

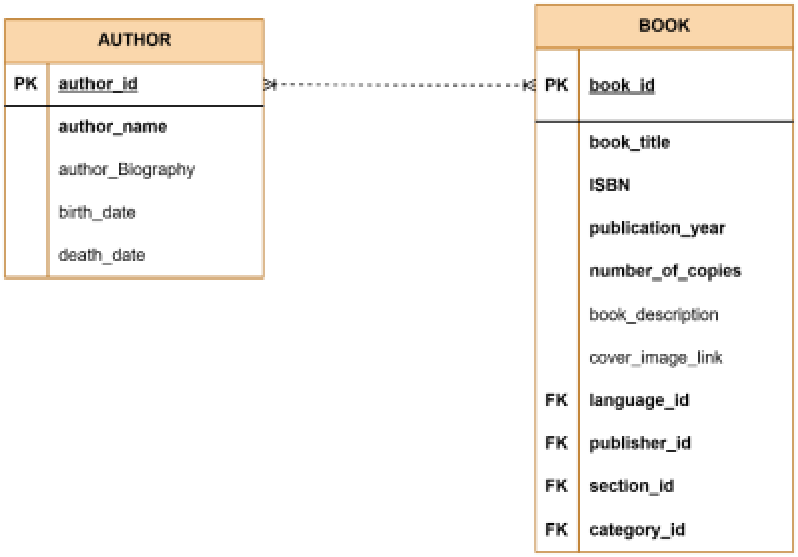


And let’s make a **connection between PUBLISHER and BOOK entity**. So, books can easily be identified by publisher name easily.

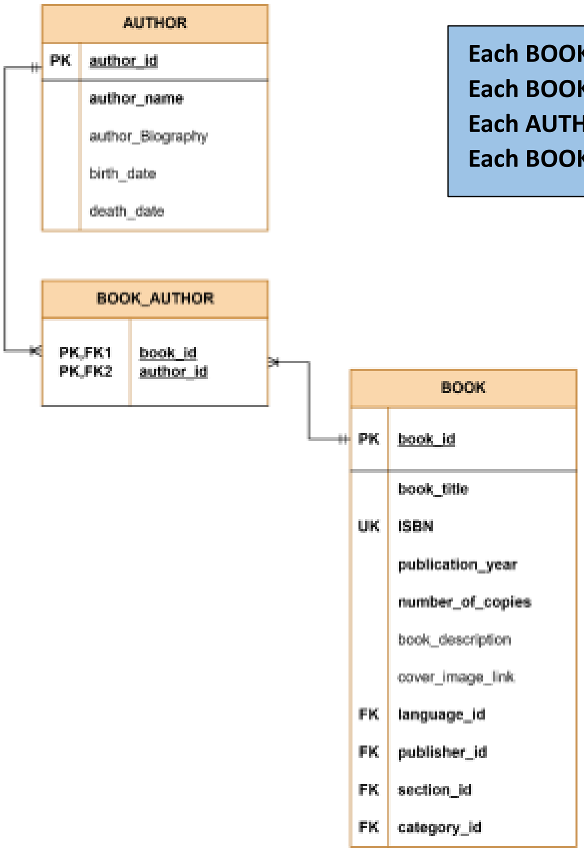


Each BOOK must have one and only one PUBLISHER.
Each PUBLISHER may publish one or more BOOKS.

- Books have authors. So, there must be an **AUTHOR entity**. Each book may have author more than one. So, there are **many to many relationships** between author entity and book entity.

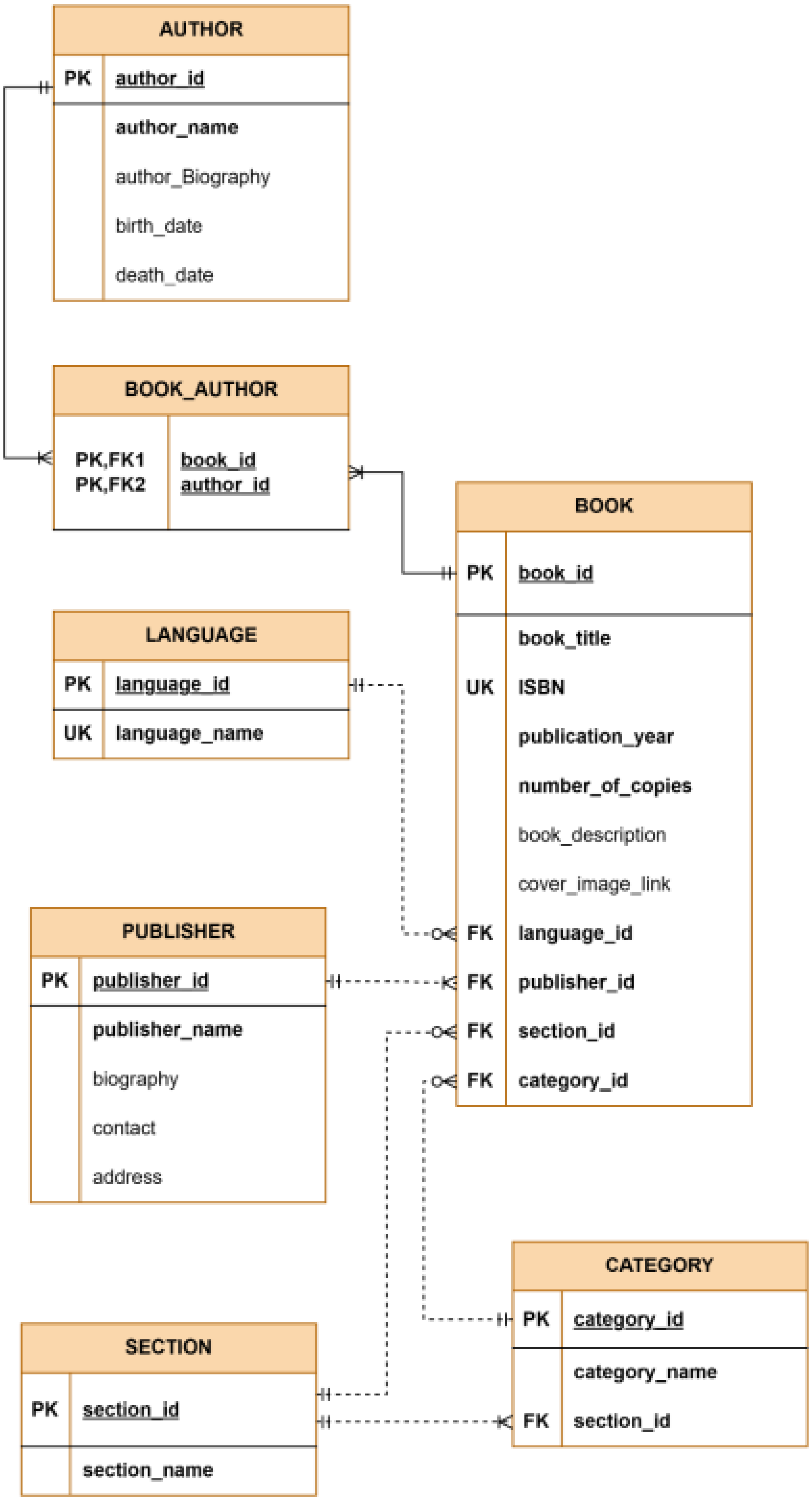


Now, **we have eliminated many to many relationships**, So, we need to **make new entity BOOK_AUTHOR** which works as junction entity. It will have concatenated UID from the books and author entities.

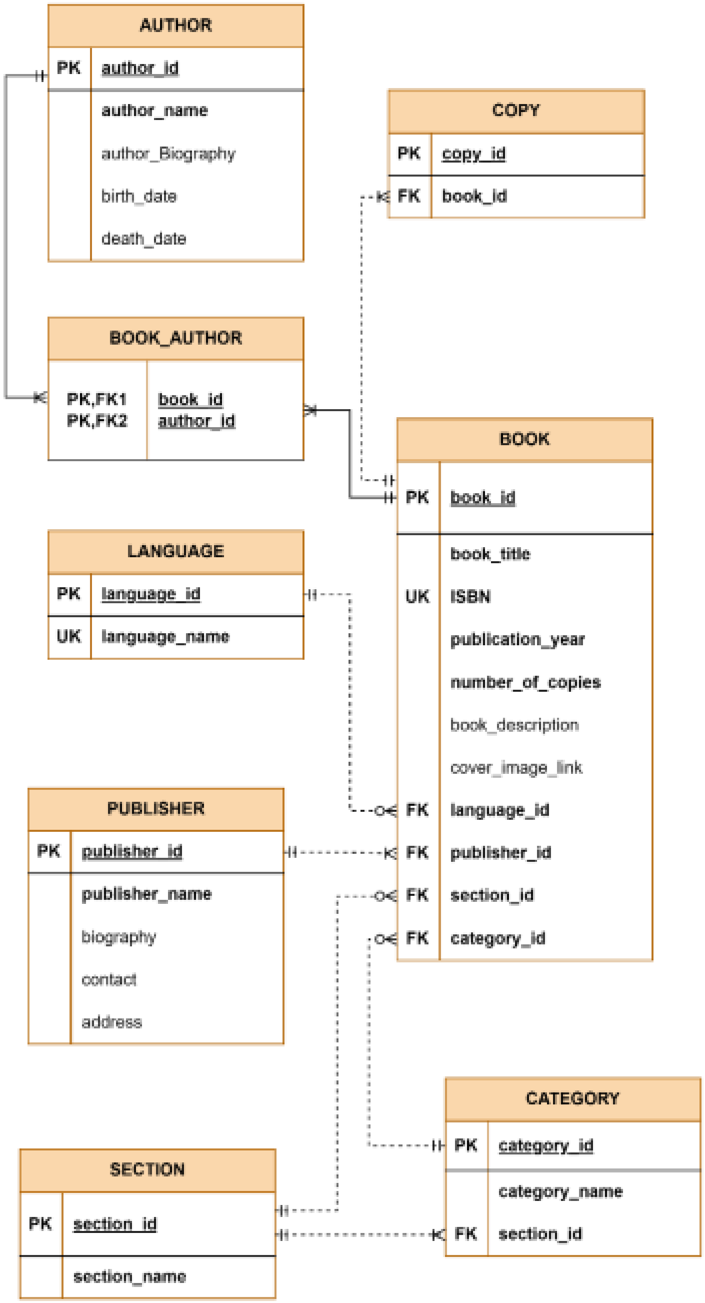


Each BOOK may have one or more BOOK_AUTHOR.
Each BOOK_AUTHOR must be assigned to one and only one BOOK.
Each AUTHOR may have one or more BOOK_AUTHOR.
Each BOOK_AUTHOR must be assigned to one and only one AUTHOR.

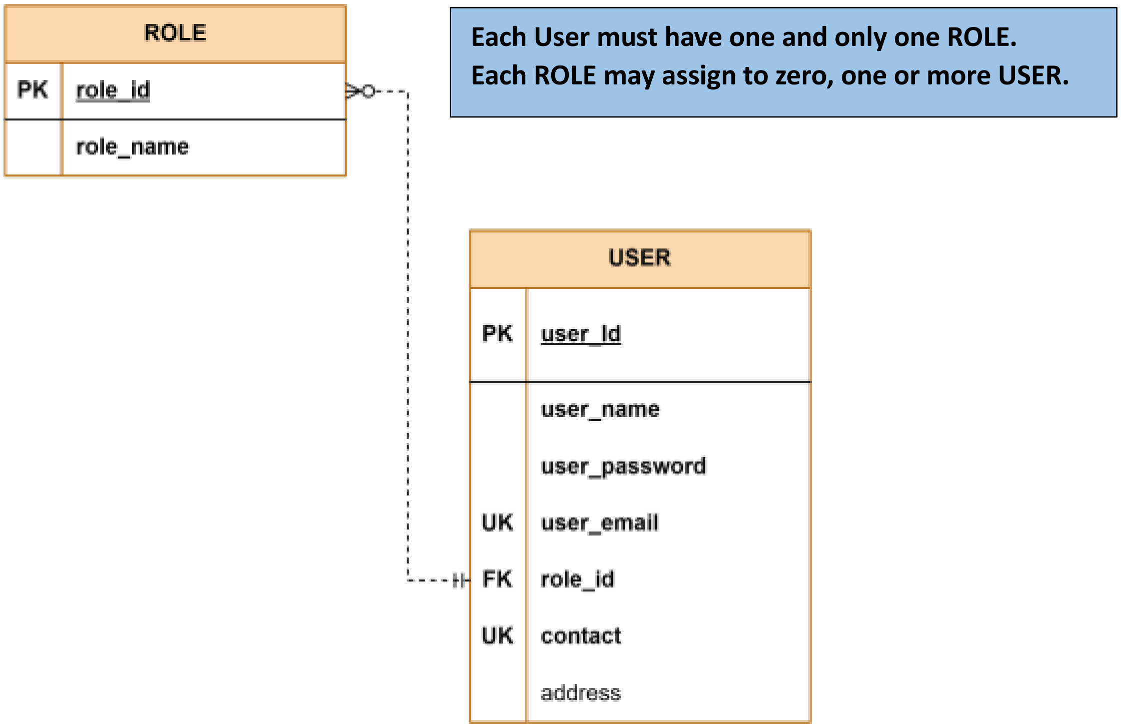
Our Latest ER Diagram is:



Now, **in the library we have multiple copies of a particular book**. So, there is need to make **book COPY entity** which contain copy id [PK] and book id [FK] attributes as shown below. The **purpose of making this book COPY entity** is to identify which book copy is taken by user. If a user does something wrong with a book copy, such as damaging, then it is easy for library staff to identify or trace the user who did it.

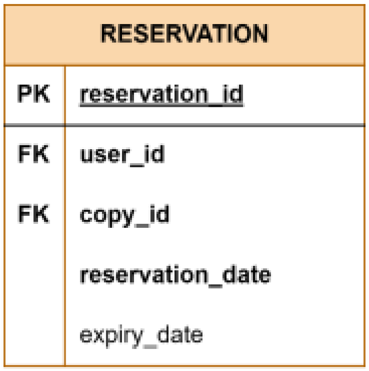


- In the library, to store user data, we will create a **USER entity** and the user may be staff or customer. To identify that we will create a **ROLE entity** which identify that user is staff or normal user in the library as shown below.

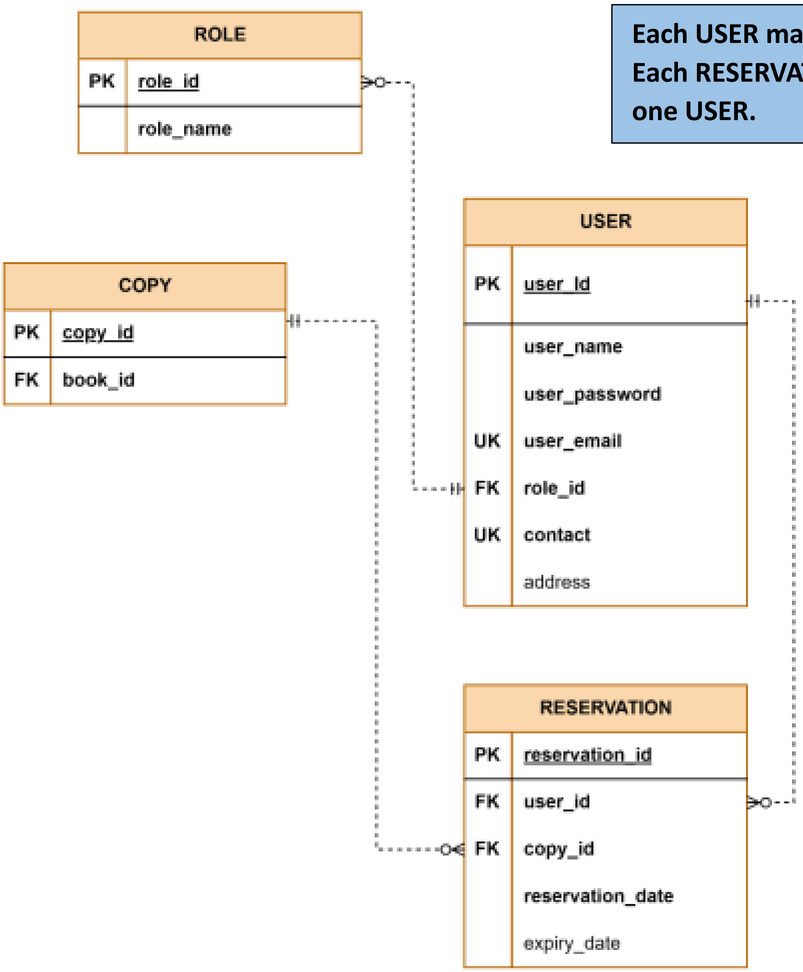


Each User must have one and only one ROLE.
Each ROLE may assign to zero, one or more USER.

- **For book reservation facility for user**, there is need to create a **RESERVATION entity** which contain reservation id, user id, copy id, reservation date and expiry date attributes as shown below.



So, there is a connection between reservation and user entity which helps to know which user reserved the book copy. And the relationship between reservation and book copy entity tells which book copy is reserved for that user as shown below.



ROLE	
PK	role_id
	role_name

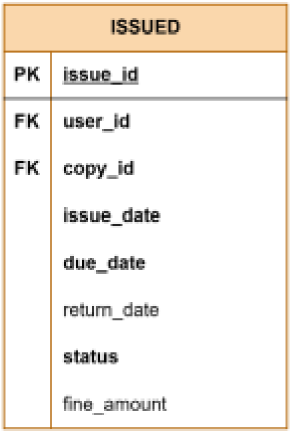
COPY	
PK	copy_id
FK	book_id

USER	
PK	user_Id
	user_name
	user_password
UK	user_email
FK	role_id
UK	contact
	address

RESERVATION	
PK	reservation_id
FK	user_id
FK	copy_id
	reservation_date
	expiry_date

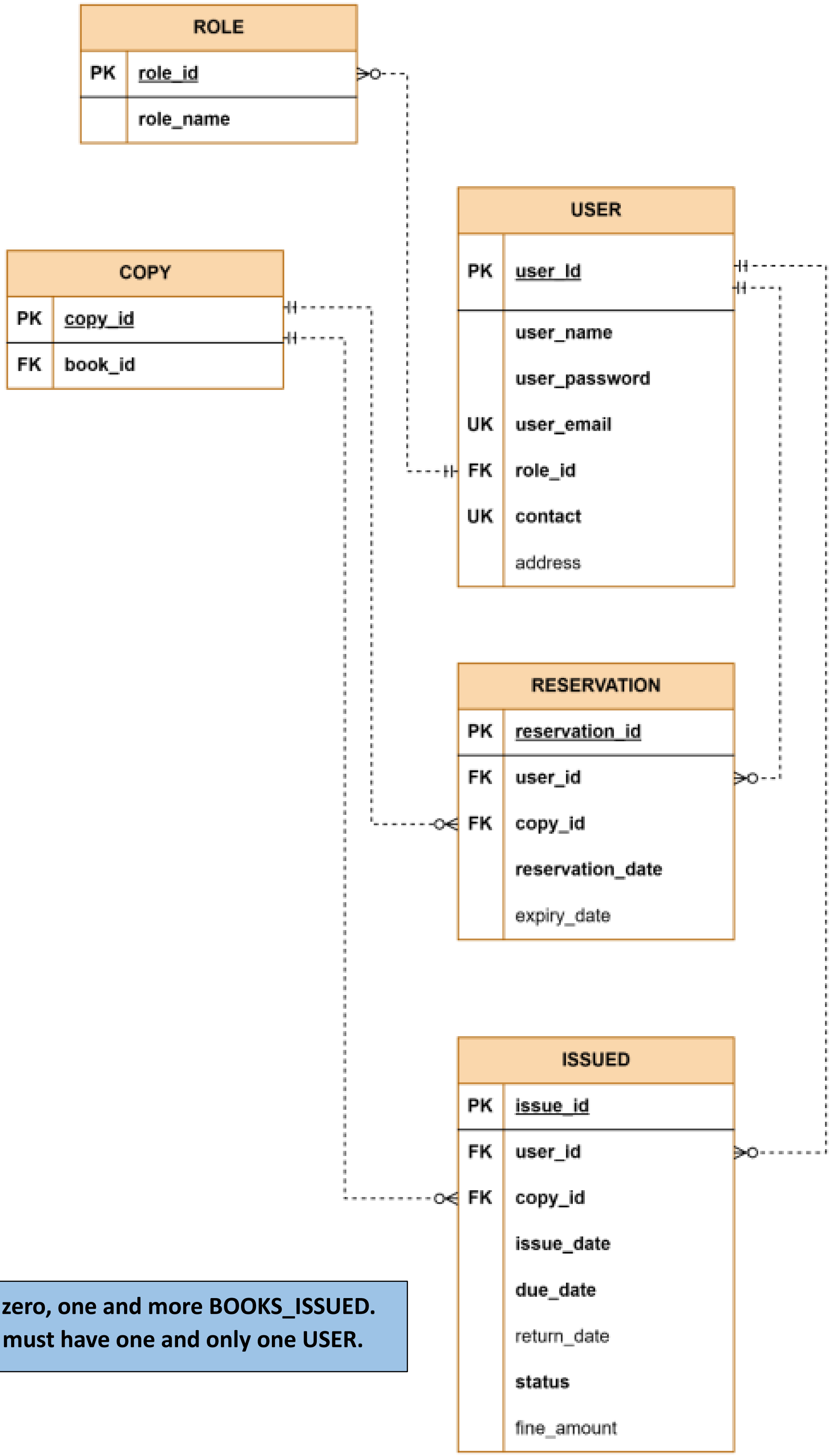
Each USER may have zero, one and more RESERVATIONS. Each RESERVATIONS must be associate with one and only one USER.

- **For storing book issued data**, we are creating **book ISSUED entity** which contain issue id [PK], user id [FK], copy id [FK] , issue date, due date, return date, status, fine amount attributes as shown below.



ISSUED	
PK	issue_id
FK	user_id
FK	copy_id
	issue_date
	due_date
	return_date
	status
	fine_amount

Let's make connection between issued entity and user entity, copy entity and user entity to identify which book copy is issued to which user. as shown below.



ROLE	
PK	role_id
	role_name

USER	
PK	user_Id
	user_name
	user_password
UK	user_email
FK	role_id
UK	contact
	address

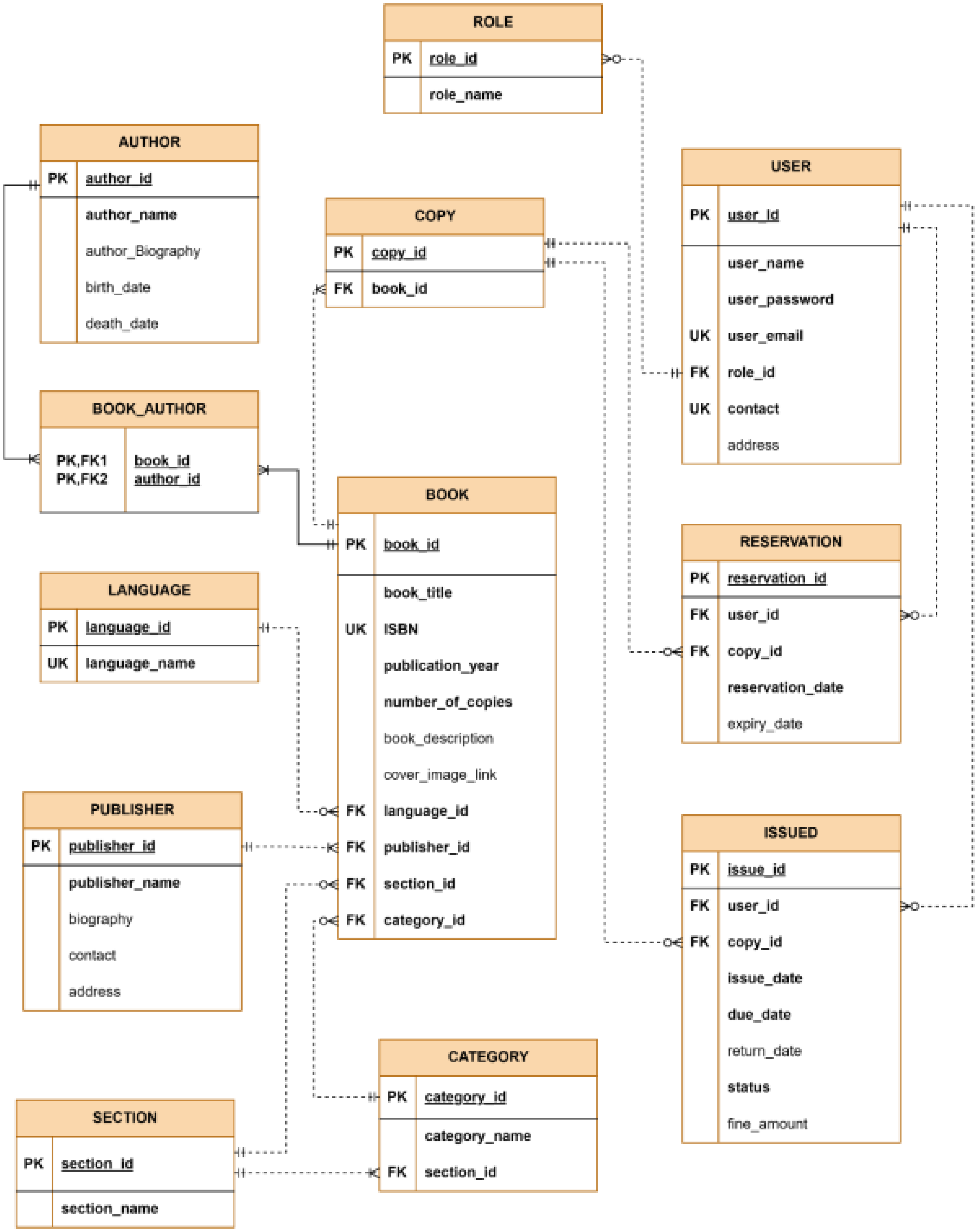
COPY	
PK	copy_id
FK	book_id

RESERVATION	
PK	reservation_id
FK	user_id
FK	copy_id
	reservation_date
	expiry_date

ISSUED	
PK	issue_id
FK	user_id
FK	copy_id
	issue_date
	due_date
	return_date
	status
	fine_amount

Each USER may have zero, one and more BOOKS_ISSUED.
Each BOOKS_ISSUED must have one and only one USER.

3.3 Final ER Diagram



4. NORMALIZATION

Normalization is the process of evaluating and modifying database table structure into more organized form ensuring higher efficiency in querying and maintenance. The purpose of Normalization is to eliminate redundant data which would eventually remove anomalies thereby improving the efficiency of the database design making it a good database design.

Redundancy is the storage of the same data in more than one table which makes the database inconsistent for queries and difficult to maintain. During the normalization process, redundant data is eliminated by splitting tables with redundant data into multiple tables without redundancy.

Normalization is performed by applying rules called Normal Forms to tables. There are 3 common normal forms known as First Normal Form (1NF), Second Normal Form (2NF) and Third Normal Form (3NF). Each normal form is a step-by-step guide towards a more refined and structured database design. In addition to these 3 forms, Boyce-Codd normal form (BCNF) (sometimes referred to as 3.5NF) is also used for this purpose. A table is said to be in one of the normal forms if it meets the criteria required by that form.

First normal form (1NF)

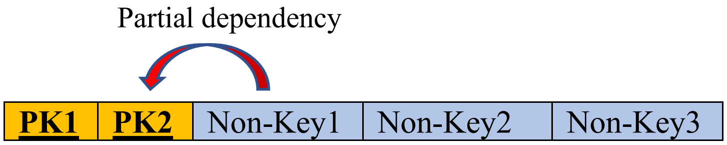
A table is in first normal form (1NF) when:

- No repeating groups: A table does not contain two or more columns with similar data.
- No multivalued columns: No columns with multiple values
- A primary key has been identified.
- All columns are dependent on primary key.

Second normal form (2NF)

A table is in second normal form (2NF) when:

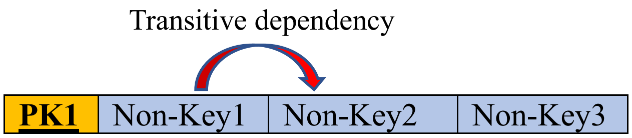
- It is in first normal form (1NF).
- No partial dependencies: Each non-key attribute depends on the entire primary key not part of the primary key. (Therefore, 2NF possibly applies only to tables with concatenated primary keys.)
  Ex: Let the Primary key be PK1 + PK2 and Non-Key columns be Non-Key1, Non-Key2 and Non-Key3. If Non-Key1 solely depends only on PK2it is known as partial dependency.



Third normal form (3NF)

A table is in third normal form (3NF) when:

- It is in second normal form (2NF).
- No non-key dependencies (transitive dependencies): Non-key dependency occurs when a non-key attribute determines the value of another non-key attribute. If the table needs to be in its 3rd normal form, each non-key should solely depend on primary key only.
  Ex: Let the Primary key be PK1 and Non-Key columns be Non-Key1, Non-Key2 and Non-Key3. If Non-Key1 column determines the value of Non-Key2 column it is known as transitive dependency (non-key dependency).



Let's consider each table in the database and normalize them using 1NF, 2NF and 3NF.

4.1 BOOK

book_id	book_title	ISBN	publication_year	number_of_copies
12345	‘Harry Potter’	9860747532745	1985	100000
12346	‘Pride And Prejudice’	9780192827609	1790	99000

book_description	cover_image_link	*language_id*	*publisher_id*	*section_id*	*category_id*
‘Series of seven…’		1	2246	3	10
‘Pride And Preju…’		1	2247	2	11

BOOK (**book_id**, book_title, ISBN, publication_year, number_of_copies, book_description, cover_image_link, *language_id, publisher_id, section_id, category_id*)

4.1.1 Evaluate BOOK table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **BOOK** is in 1NF.

4.1.2 Evaluate BOOK table for 2NF

- The table is now in 1NF.
- As there is only a single primary key each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **BOOK** is in 2NF.

4.1.3 Evaluate BOOK table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **BOOK** is in 3NF.

The BOOK table is normalized already.

4.2 AUTHOR

AUTHOR				
author_id	author_name	author_biography	birth_date	death_date
1	‘J.K. Rowling’	‘British Author..’	1965-07-31	NULL
2	‘Jane Austen’	‘English Nov..’	1775-12-16	1817-07-18

AUTHOR (**author_id**, author_name, author_biography, birth_date, death_date)

4.2.1 Evaluate AUTHOR table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **author_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **AUTHOR** is in 1NF.

4.2.2 Evaluate AUTHOR table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **AUTHOR** is in 2NF.

4.2.3 Evaluate AUTHOR table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **AUTHOR** is in 3NF.

The AUTHOR table is normalized already.

4.3 BOOK_AUTHOR

BOOK_AUTHOR	
book_id	***author_id***
12345	1
12346	2

BOOK_AUTHOR (*book_id*, *author_id*)

4.3.1 Evaluate BOOK_AUTHOR table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A concatanated primary key is identified as **book_id, author_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **BOOK_AUTHOR** is in 1NF.

4.3.2 Evaluate BOOK_AUTHOR table for 2NF

- The table is now in 1NF.
- The table has a concatenated primary key only. There are not any non-key columns. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **BOOK_AUTHOR** is in 2NF.

4.3.3 Evaluate BOOK_AUTHOR table for 3NF

- The table is now in 2NF.
- The table has a concatenated primary key only. There are not any non-key columns. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **BOOK_AUTHOR** is in 3NF.

Further to above details, BOOK_AUTHOR is an intersection table created at the ERD design stage in order to resolve many to many relationships between BOOK and AUTHOR entities.

It can be concluded that the BOOK_AUTHOR table is normalized already.

4.4 PUBLISHER

PUBLISHER				
publisher_id	publisher_name	biography	contact	address
2246	‘Canada Publish’	‘Publishing over ..’	999-99999	‘123, Brunel..’
2324	‘Novel Printers’	‘Printers to Nation’	123-12345	’96, Bakers st.’

PUBLISHER (**publisher_id**, publisher_name, biography, contact, address)

4.4.1 Evaluate PUBLISHER table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **publisher_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **PUBLISHER** is in 1NF.

4.4.2 Evaluate PUBLISHER table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **PUBLISHER** is in 2NF.

4.4.3 Evaluate PUBLISHER table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns can be found.

Therefore, it can be concluded that the table **PUBLISHER** is in 3NF.

The PUBLISHER table is normalized already.

4.5 SECTION

SECTION	
section_id	section_name
1	‘Science’
2	‘Fiction’
3	‘Fantasy’

SECTION (**section_id**, section_name)

4.5.1 Evaluate SECTION table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **section_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **SECTION** is in 1NF.

4.5.2 Evaluate SECTION table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **SECTION** is in 2NF.

4.5.3 Evaluate SECTION table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no dependency between non-key columns.

Therefore, it can be concluded that the table **SECTION** is in 3NF.

The SECTION table is normalized already.

4.6 LANGUAGE

LANGUAGE	
language_id	Language_name
1	‘English’
2	‘French’

LANGUAGE (**language_id**, language_name)

4.6.1 Evaluate LANGUAGE table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **language_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **LANGUAGE** is in 1NF.

4.6.2 Evaluate LANGUAGE table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **LANGUAGE** is in 2NF.

4.6.3 Evaluate LANGUAGE table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns can be found.

Therefore, it can be concluded that the table **LANGUAGE** is in 3NF.

The LANGUAGE table is normalized already.

4.7 CATEGORY

CATEGORY		
category_id	category_name	*section_id*
‘PH678’	‘Physics’	1
‘CH123’	‘Chemistry’	1

CATEGORY (**category_id**, category_name, *section_id*)

4.7.1 Evaluate CATEGORY table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **category_id**
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **CATEGORY** is in 1NF.

4.7.2 Evaluate CATEGORY table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **CATEGORY** is in 2NF.

4.7.3 Evaluate CATEGORY table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns can be found.

Therefore, it can be concluded that the table **CATEGORY** is in 3NF.

The CATEGORY table is normalized already.

4.8 USER

USER						
<u>user_id</u>	user_name	user_password	user_email	*role_id*	contact	address
146	‘Monica S..’	‘123abc’	‘mo@g..’	4	12345678	‘8, …’
25	‘Kumar ..’	‘bh765’	‘kum@..’	2	15989465	‘121,.’

USER (**<u>user_id</u>**, user_name, user_password, user_email, *role_id*, contact, address)

4.8.1 Evaluate USER table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **<u>user_id</u>**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **USER** is in 1NF.

4.8.2 Evaluate USER table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **USER** is in 2NF.

4.8.3 Evaluate USER table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns can be found.

Therefore, it can be concluded that the table **USER** is in 3NF.

The USER table is normalized already.

4.9 ROLE

ROLE	
role_id	role_name
1	‘Librarian’
2	‘Reader’

ROLE (**role_id**, role_name)

4.9.1 Evaluate ROLE table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **role_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **ROLE** is in 1NF.

4.9.2 Evaluate ROLE table for 2NF

- The table is now in 1NF.
- The non-key column depends on the entire primary key. Therefore, no partial dependency between non-key column and part of the primary key can be found.

Therefore, it can be concluded that the table **ROLE** is in 2NF.

4.9.3 Evaluate ROLE table for 3NF

- The table is now in 2NF.
- There is only one non-key column. Therefore, none of the non-keys determine the value of another non-key. The non-key depends solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **ROLE** is in 3NF.

The ROLE table is normalized already.

4.10 RESERVATION

RESERVATION				
reservation_id	*user_id*	*book_id*	reservation_date	expiry_date
‘JUN134’	25	12345	‘2023-05-04’	
‘MAY256’	175	12345	‘2019-06-01’	

RESERVATION (**reservation_id**, *user_id, book_id*, reservation_date, expiry_date)

4.10.1 Evaluate RESERVATION table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **reservation_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **RESERVATION** is in 1NF.

4.10.2 Evaluate RESERVATION table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **RESERVATION** is in 2NF.

4.10.3 Evaluate RESERVATION table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **RESERVATION** is in 3NF.

The RESERVATION table is normalized already.

4.11 BOOKS_ISSUED

BOOKS_ISSUED							
issue_id	*user_id*	*book_id*	issue_date	due_date	return_date	status	fine_amount
156	146	12345	‘2023-05-01’	‘2023…’			
176	25	12346	‘2020-12-01’	‘2020...’			

BOOKS_ISSUED (**issue_id,** *user_id, book_id*, issue_date, due_date, return_date, status, fine_amount)

4.11.1 Evaluate BOOKS_ISSUED table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **issue_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **BOOKS_ISSUED** is in 1NF.

4.11.2 Evaluate BOOKS_ISSUED table for 2NF

- The table is now in 1NF.
- Each non-key column depends on the entire primary key. Therefore, no partial dependency between non-key columns and part of the primary key.

Therefore, it can be concluded that the table **BOOKS_ISSUED** is in 2NF.

4.11.3 Evaluate BOOKS_ISSUED table for 3NF

- The table is now in 2NF.
- None of the non-keys determine the value of another non-key. All the non-keys depend solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **BOOKS_ISSUED** is in 3NF.

The BOOKS_ISSUED table is normalized already.

4.12 BOOK_COPIES

BOOK_COPIES	
copy_id	*book_id*
‘VS123’	12345
‘VS345’	12345

BOOK_COPIES (**copy_id,** *book_id*)

4.12.1 Evaluate BOOK_COPIES table for 1NF

- The table has no repeating groups.
- The table has no multivalued columns.
- A primary key is identified as **copy_id**.
- All columns are dependent on the primary key.

As all the criteria for 1NF is already satisfied, it can be concluded that the table **BOOK_COPIES** is in 1NF.

4.12.2 Evaluate BOOK_COPIES table for 2NF

- The table is now in 1NF.
- The non-key column depends on the entire primary key. Therefore, no partial dependency between non-key column and part of the primary key.

Therefore, it can be concluded that the table **BOOK_COPIES** is in 2NF.

4.12.3 Evaluate BOOK_COPIES table for 3NF

- The table is now in 2NF.
- There is only one non-key column. Therefore, none of the non-keys determine the value of another non-key. The non-key depends solely on the primary key. Therefore, no transitive dependency between non-key columns.

Therefore, it can be concluded that the table **BOOK_COPIES** is in 3NF.

The BOOK_COPIES table is normalized already.

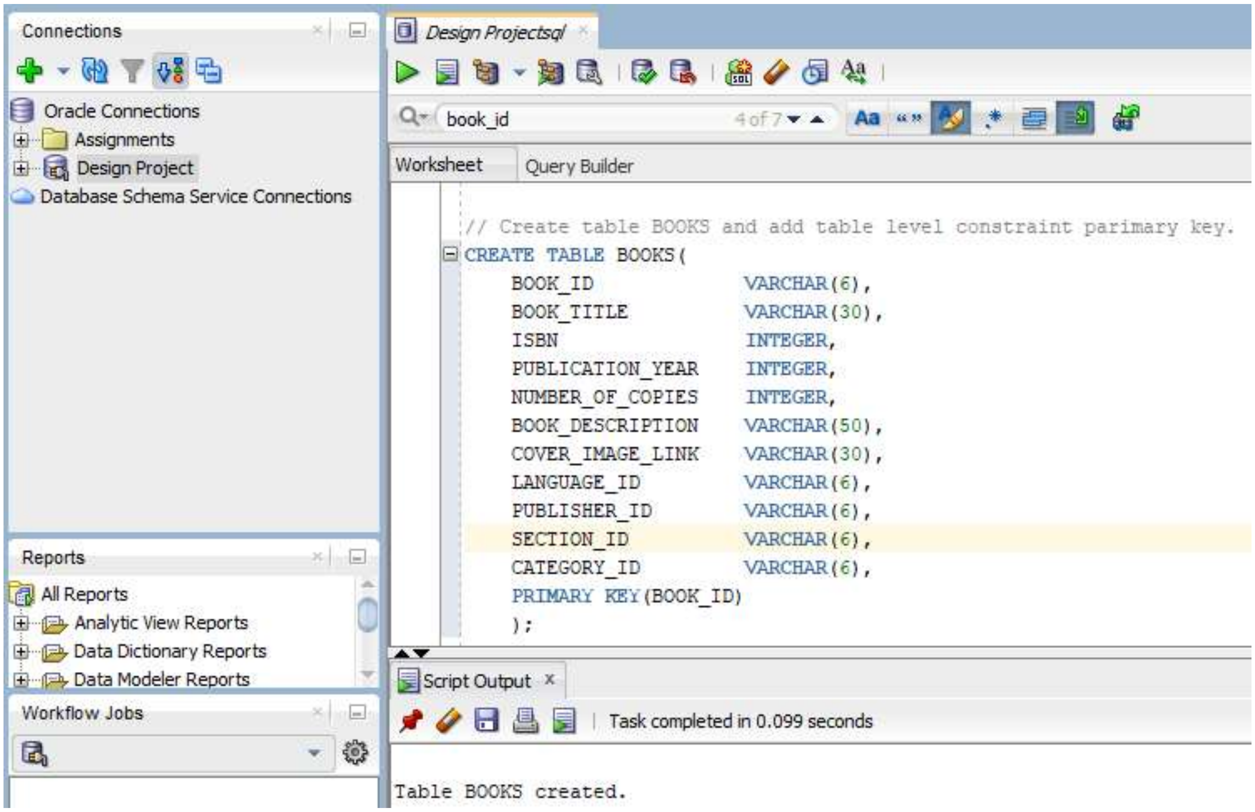
5. CONSTRAINTS

5.1 PRIMARY KEY CONSTRAINT – Table level constraint

The primary key identifies which column or set of columns act as the unique identifier for each row in the table. As the value in this column or the combined value of these set of columns must uniquely identify each row, it cannot be NULL. Therefore, when we set the PRIMARY KEY constraint, we can see the value for that column has been set to NOT NULL. Further, a table definition can have no more than one primary key constraint. So, it is a table level constraint.

Primary key constraint can be added at table level as part of the CREATE TABLE command as well as an ALTER TABLE command later.

CREATE TABLE and add constraint PRIMARY KEY.



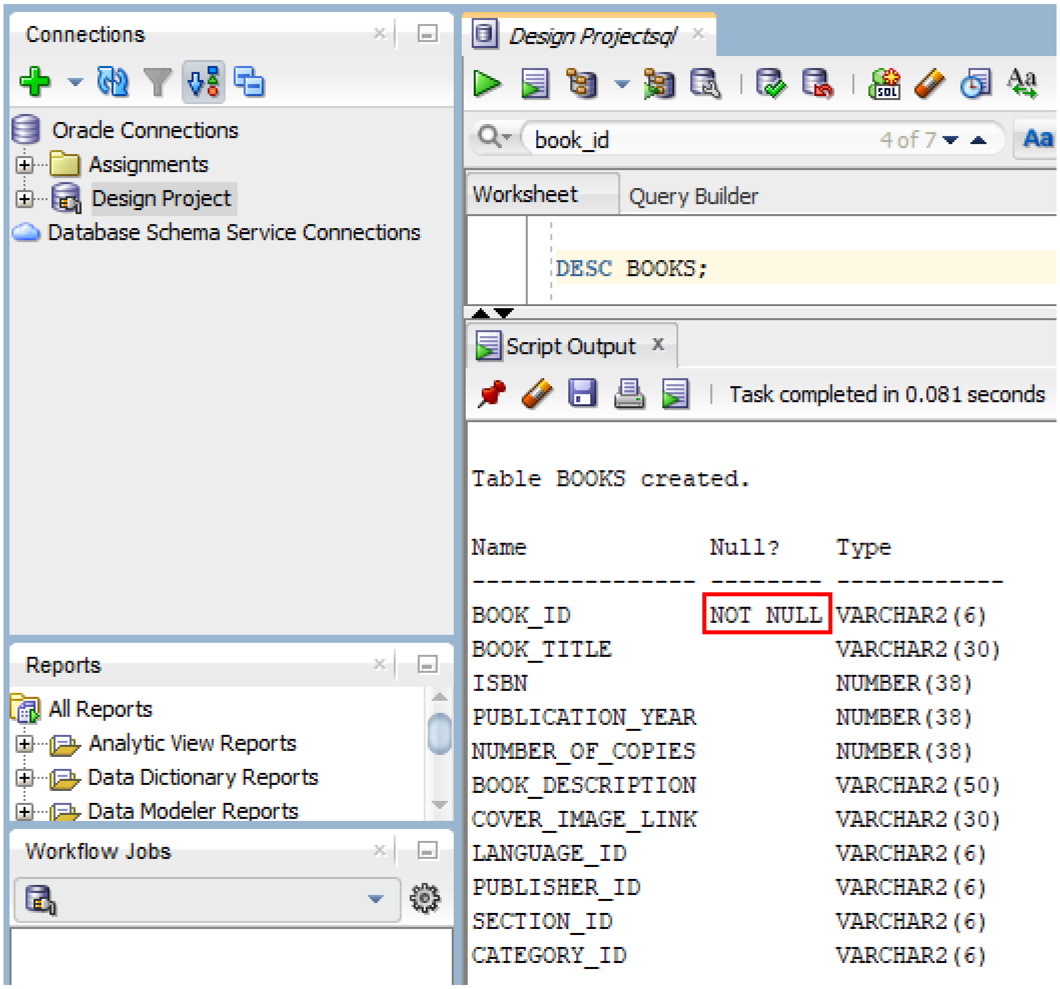
```
// Create table BOOKS and add table level constraint parimary key.
CREATE TABLE BOOKS(
    BOOK_ID             VARCHAR(6),
    BOOK_TITLE          VARCHAR(30),
    ISBN                INTEGER,
    PUBLICATION_YEAR    INTEGER,
    NUMBER_OF_COPIES    INTEGER,
    BOOK_DESCRIPTION    VARCHAR(50),
    COVER_IMAGE_LINK    VARCHAR(30),
    LANGUAGE_ID         VARCHAR(6),
    PUBLISHER_ID        VARCHAR(6),
    SECTION_ID          VARCHAR(6),
    CATEGORY_ID         VARCHAR(6),
    PRIMARY KEY(BOOK_ID)
    );
```

Task completed in 0.099 seconds

```
Table BOOKS created.
```

This way we can add a primary key constraint to BOOK_ID column.

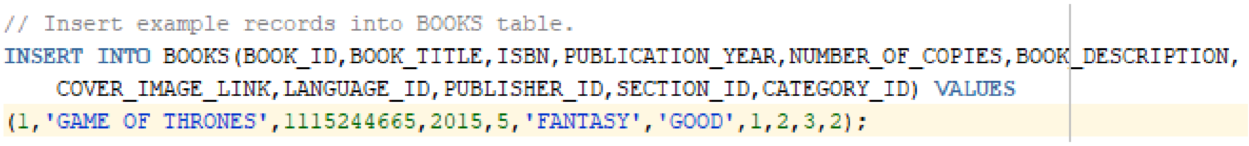
The description of the BOOKS table will now display NOT NULL under BOOK_ID column as we explained above.



```
DESC BOOKS;
```

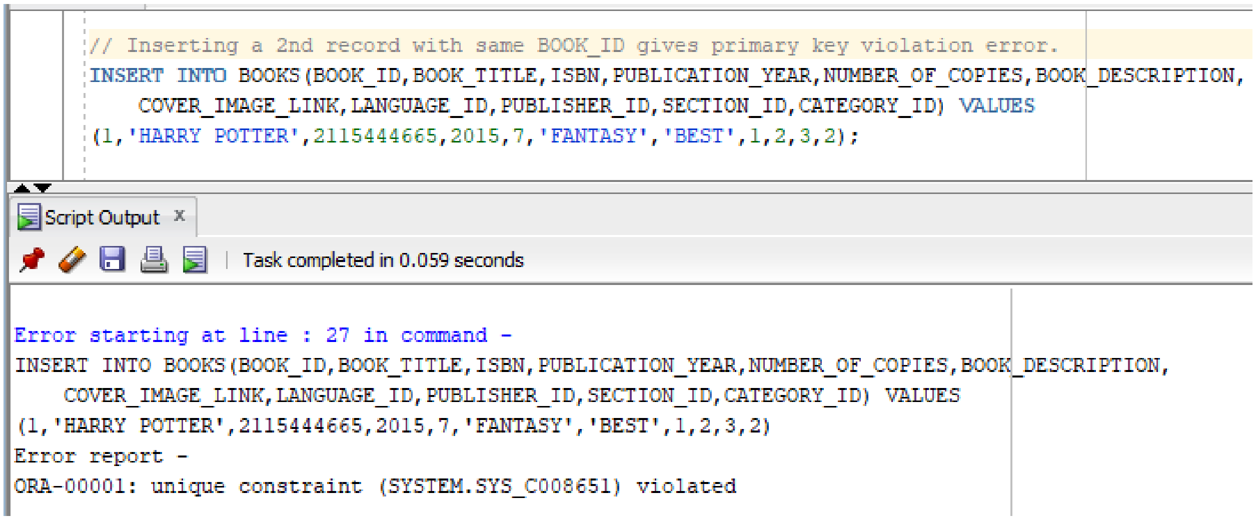
```
Table BOOKS created.

Name                Null?    Type
------------------- -------- ------------
BOOK_ID             NOT NULL VARCHAR2(6)
BOOK_TITLE                   VARCHAR2(30)
ISBN                         NUMBER(38)
PUBLICATION_YEAR             NUMBER(38)
NUMBER_OF_COPIES             NUMBER(38)
BOOK_DESCRIPTION             VARCHAR2(50)
COVER_IMAGE_LINK             VARCHAR2(30)
LANGUAGE_ID                  VARCHAR2(6)
PUBLISHER_ID                 VARCHAR2(6)
SECTION_ID                   VARCHAR2(6)
CATEGORY_ID                  VARCHAR2(6)
```

First row inserted with BOOK_ID 1.



```
// Insert example records into BOOKS table.
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(1,'GAME OF THRONES',1115244665,2015,5,'FANTASY','GOOD',1,2,3,2);
```

Inserting a second record with same BOOK_ID gives primary key constraint violation error.

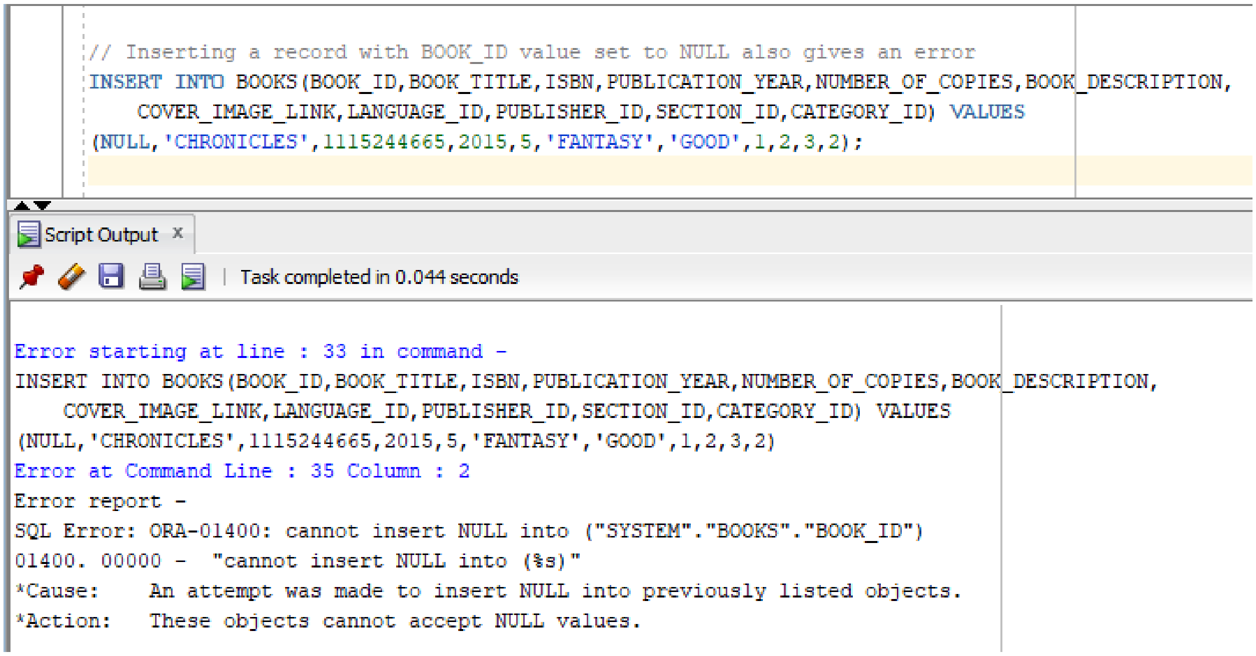


```
// Inserting a 2nd record with same BOOK_ID gives primary key violation error.
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(1,'HARRY POTTER',2115444665,2015,7,'FANTASY','BEST',1,2,3,2);
```

Task completed in 0.059 seconds

```
Error starting at line : 27 in command -
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(1,'HARRY POTTER',2115444665,2015,7,'FANTASY','BEST',1,2,3,2)
Error report -
ORA-00001: unique constraint (SYSTEM.SYS_C008651) violated
```

Inserting a record with BOOK_ID value set to NULL also gives an error as the primary key column cannot hold NULL values as well.



```
// Inserting a record with BOOK_ID value set to NULL also gives an error
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(NULL,'CHRONICLES',1115244665,2015,5,'FANTASY','GOOD',1,2,3,2);
```

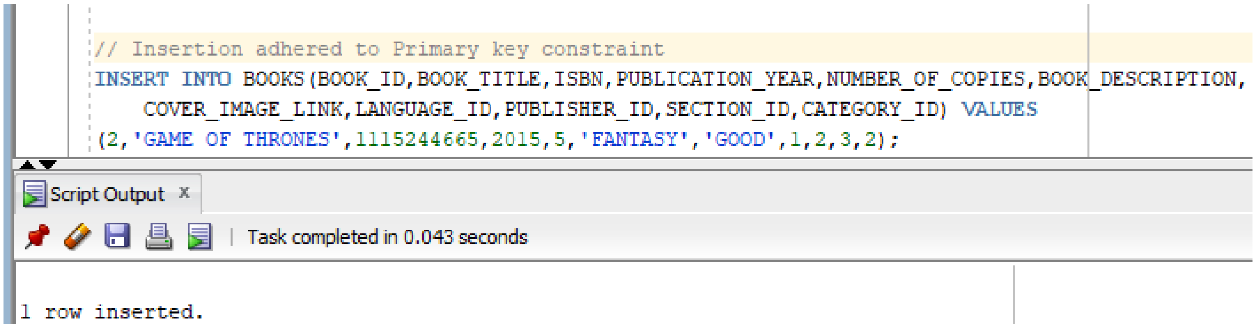
Task completed in 0.044 seconds

```
Error starting at line : 33 in command -
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(NULL,'CHRONICLES',1115244665,2015,5,'FANTASY','GOOD',1,2,3,2)
Error at Command Line : 35 Column : 2
Error report -
SQL Error: ORA-01400: cannot insert NULL into ("SYSTEM"."BOOKS"."BOOK_ID")
01400. 00000 -  "cannot insert NULL into (%s)"
*Cause:    An attempt was made to insert NULL into previously listed objects.
*Action:   These objects cannot accept NULL values.
```

Therefore, if there is no PRIMARY KEY constraint on the BOOKS table:

- users will insert duplicate rows.
- users can add NULL values for the primary key column which hinders unique identification of the row.

When we do insertion of records adhered to primary key constraint it will work fine without violating the constraint defined.



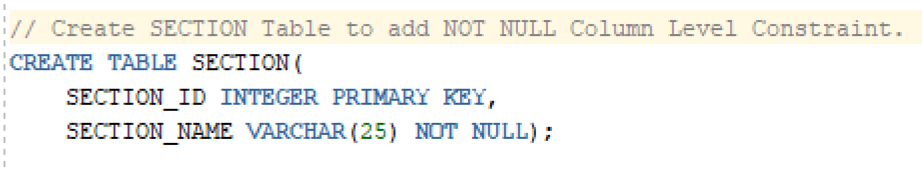
```
// Insertion adhered to Primary key constraint
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(2,'GAME OF THRONES',1115244665,2015,5,'FANTASY','GOOD',1,2,3,2);
```

```
1 row inserted.
```

5.2 NOT NULL CONSTRAINT – Column level constraint

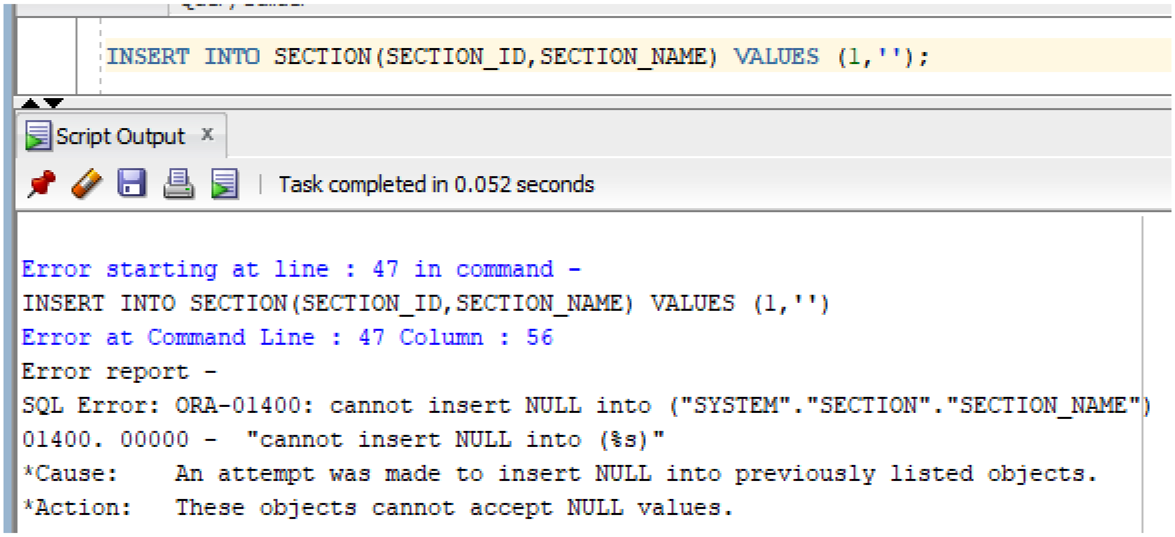
If a value for a particular column is unknown, a special value called NULL is used. A NOT NULL constraint can be used at the CREATE TABLE stage to imply that a particular column cannot have NULL value. That column does not allow entry of unknown values. Either the value for that column should be a default value or a user entered value.

Adding NOT NULL constraint to column "SECTION_NAME" at CREATE TABLE command.



```
// Create SECTION Table to add NOT NULL Column Level Constraint.
CREATE TABLE SECTION(
    SECTION_ID INTEGER PRIMARY KEY,
    SECTION_NAME VARCHAR(25) NOT NULL);
```

It does not allow to enter a NULL value to the Section Name column.



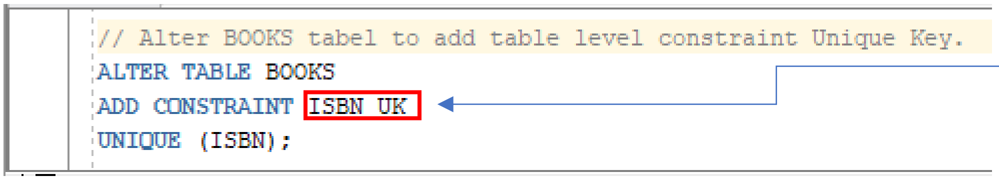
```
INSERT INTO SECTION(SECTION_ID,SECTION_NAME) VALUES (1,'');
```

```
Error starting at line : 47 in command -
INSERT INTO SECTION(SECTION_ID,SECTION_NAME) VALUES (1,'')
Error at Command Line : 47 Column : 56
Error report -
SQL Error: ORA-01400: cannot insert NULL into ("SYSTEM"."SECTION"."SECTION_NAME")
01400. 00000 -  "cannot insert NULL into (%s)"
*Cause:    An attempt was made to insert NULL into previously listed objects.
*Action:   These objects cannot accept NULL values.
```

5.3 UNIQUE Constraint – Table Level Constraint

This constraint identifies a column as having unique values for the UNIQUE column in each row in the table. It is like PRIMARY KEY with the exception that it allows NULL Values. Further, there can be one or more unique key constraints to a table unlike in primary key constraint. Unlike the Primary Key, the Unique keys are not used to uniquely identify each row, that is why they allow "NULL values" which are unknown but unique.

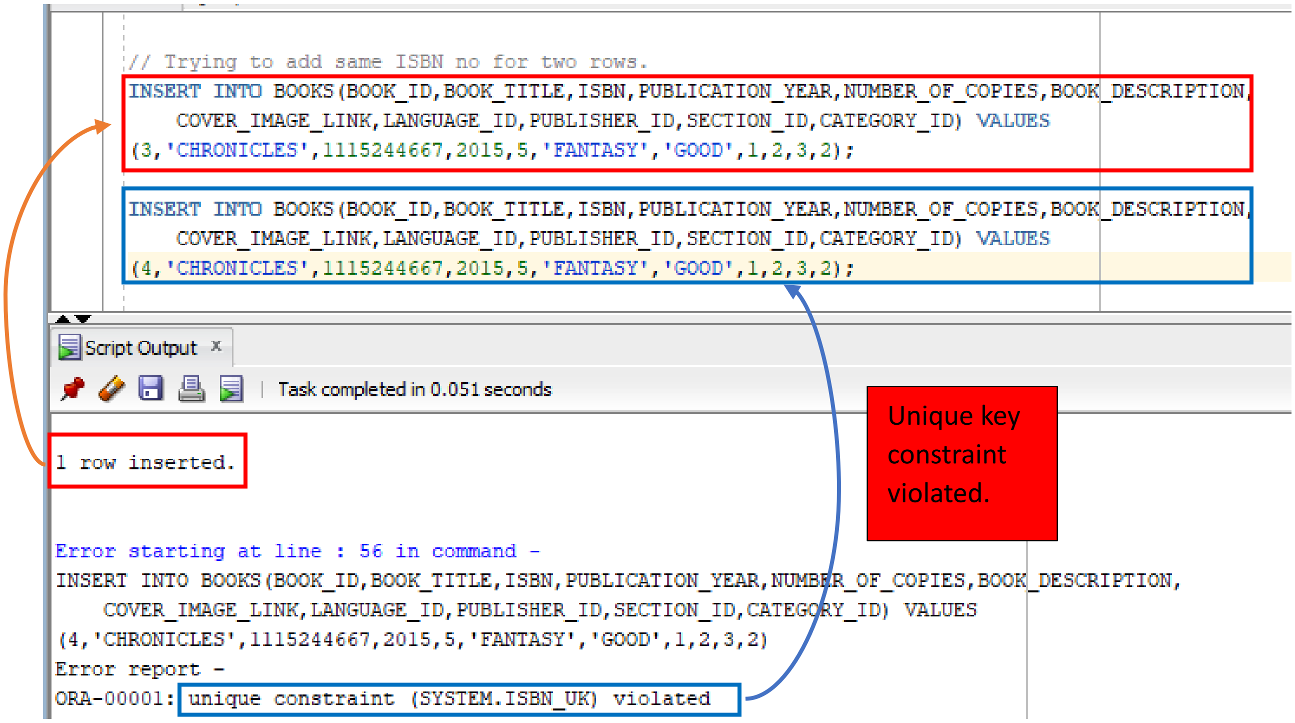
Adding UNIQUE KEY constraint to the Telephone column.



```
// Alter BOOKS tabel to add table level constraint Unique Key.
ALTER TABLE BOOKS
ADD CONSTRAINT ISBN_UK
UNIQUE (ISBN);
```

It is recommended to specify a constraint name for the unique constraint.

Try to add the same ISBN number for 2 rows.



```
// Trying to add same ISBN no for two rows.
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(3,'CHRONICLES',1115244667,2015,5,'FANTASY','GOOD',1,2,3,2);

INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(4,'CHRONICLES',1115244667,2015,5,'FANTASY','GOOD',1,2,3,2);
```

```
1 row inserted.


Error starting at line : 56 in command -
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(4,'CHRONICLES',1115244667,2015,5,'FANTASY','GOOD',1,2,3,2)
Error report -
ORA-00001: unique constraint (SYSTEM.ISBN_UK) violated
```

Record added.

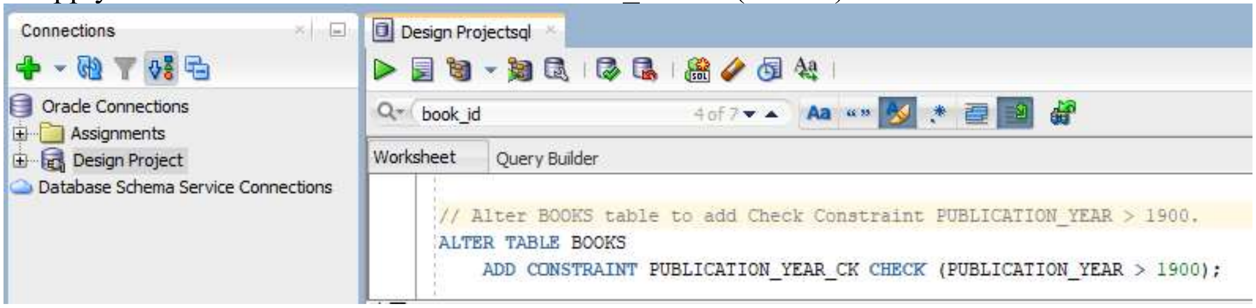
Unique key constraint violated.

It gives Unique Key Constraint Violation error because the ISBN column is protected by UNIQUE KEY constraint which requires a unique value for each row.

5.4 CHECK CONSTRAINT - Column Level Constraint

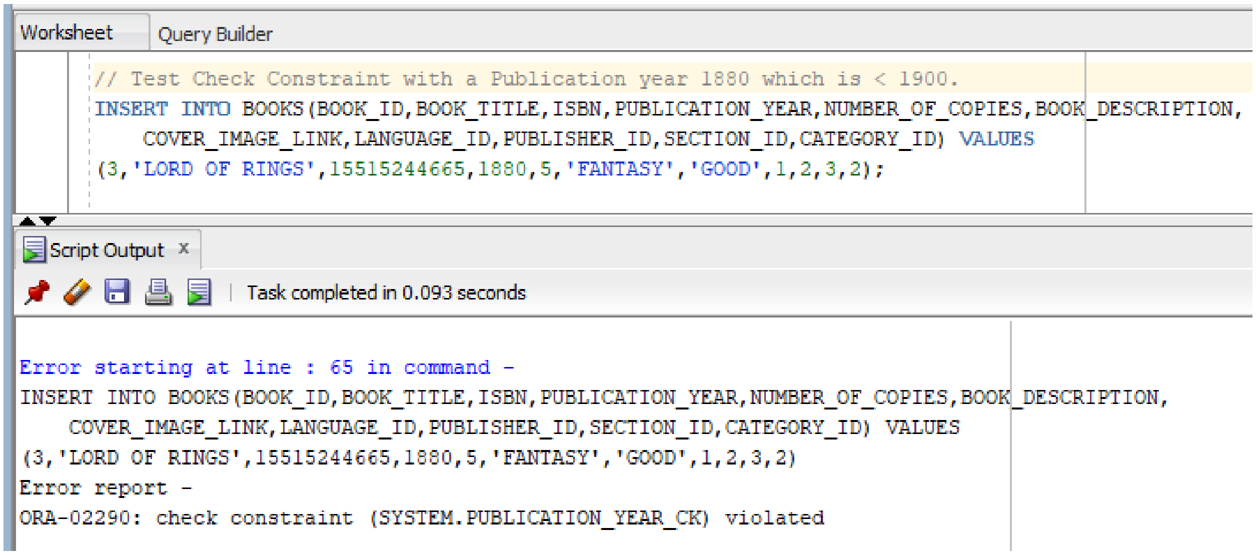
Check Constraint imposes a condition (business rule) on the data that can be entered into a column. Any attempt to modify the column (ex. INSERT, UPDATE) is permitted only if the CHECK CONSTRAINT is met. If not, the attempt is denied with a constraint violation error message.

Let’s apply CHECK Constraint to PUBLICATION_YEAR (> 1900) to BOOKS table.



```
// Alter BOOKS table to add Check Constraint PUBLICATION_YEAR > 1900.
ALTER TABLE BOOKS
    ADD CONSTRAINT PUBLICATION_YEAR_CK CHECK (PUBLICATION_YEAR > 1900);
```

Test with INSERT: If we try to INSERT a record with Publication year <= 1900 it throws a check constraint violation error.

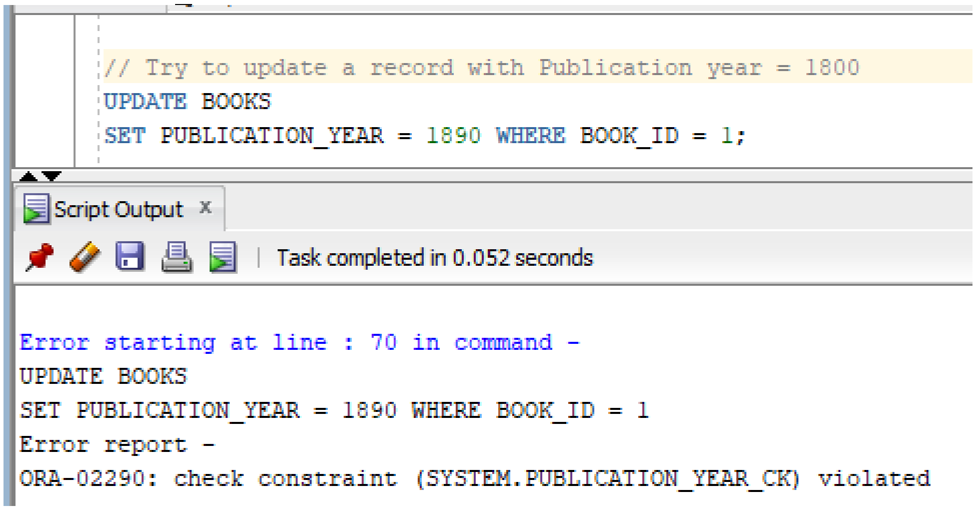


```
// Test Check Constraint with a Publication year 1880 which is < 1900.
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(3,'LORD OF RINGS',15515244665,1880,5,'FANTASY','GOOD',1,2,3,2);
```

Task completed in 0.093 seconds

```
Error starting at line : 65 in command -
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(3,'LORD OF RINGS',15515244665,1880,5,'FANTASY','GOOD',1,2,3,2)
Error report -
ORA-02290: check constraint (SYSTEM.PUBLICATION_YEAR_CK) violated
```

Test with UPDATE: Let’s try to update the record of BOOK_ID 1 in the table with PUBLICATION_YEAR = 1890.



```
// Try to update a record with Publication year = 1800
UPDATE BOOKS
SET PUBLICATION_YEAR = 1890 WHERE BOOK_ID = 1;
```

Task completed in 0.052 seconds

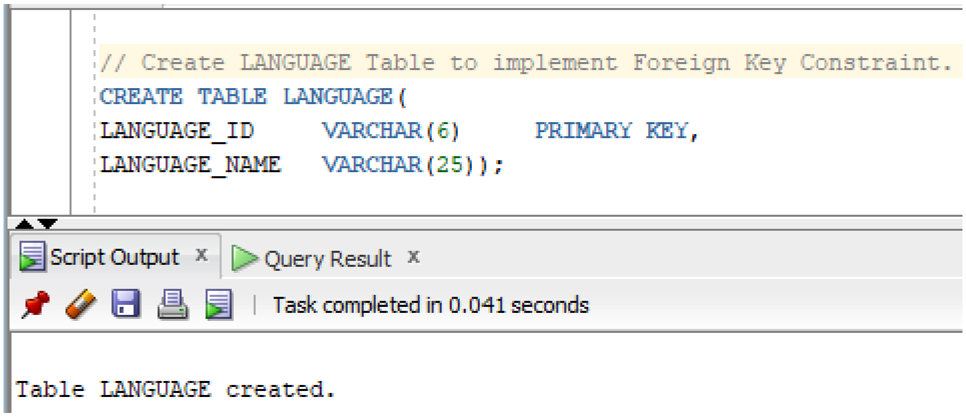
```
Error starting at line : 70 in command -
UPDATE BOOKS
SET PUBLICATION_YEAR = 1890 WHERE BOOK_ID = 1
Error report -
ORA-02290: check constraint (SYSTEM.PUBLICATION_YEAR_CK) violated
```

Update is not allowed as the “PUBLICATION_YEAR_CK” check constraint is violated.

5.5 FOREIGN KEY Constraint (Referential Integrity)

- When a logical database model is converted into a physical design, the relationships between entities are implemented as foreign key constraints.
- Foreign key is one or more columns in the child table (table where many instances are there for one instance of the other table) that contain values that match the primary key of the parent table.
- Foreign key constraint ensures that for each row in the child table where there is non-null value in the foreign key column, there should be a matching row in the parent table. Otherwise, that child row cannot exist.
- The Foreign key constraint is defined in the child table.
- The corresponding foreign key and primary key columns must have identical data type.

Let’s add the FOREIGN KEY constraint for the 2 tables BOOKS and LANGUAGE. BOOKS table has the foreign key LANGUAGE_ID which is the Primary key of the LANGUAGE table. LANGUAGE table is the parent and BOOKS is the child.

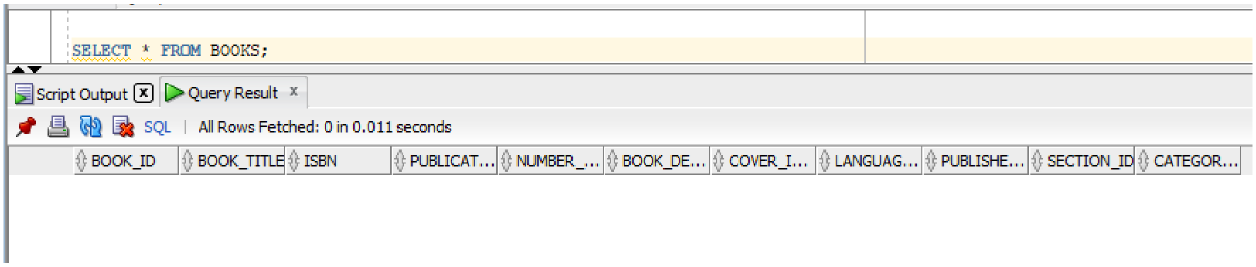


```
// Create LANGUAGE Table to implement Foreign Key Constraint.
CREATE TABLE LANGUAGE(
LANGUAGE_ID      VARCHAR(6)      PRIMARY KEY,
LANGUAGE_NAME    VARCHAR(25));
```

```
Task completed in 0.041 seconds

Table LANGUAGE created.
```

DROP the BOOKS table and re-create with above discussed constraints.

BOOKS

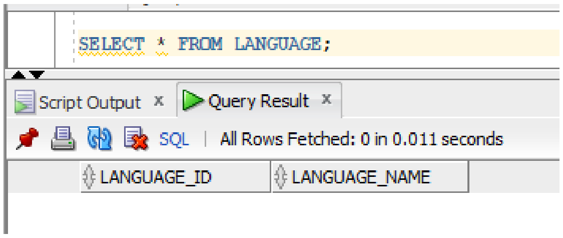


```
SELECT * FROM BOOKS;
```

```
All Rows Fetched: 0 in 0.011 seconds
```

BOOK_ID	BOOK_TITLE	ISBN	PUBLICAT...	NUMBER_...	BOOK_DE...	COVER_I...	LANGUAG...	PUBLISHE...	SECTION_ID	CATEGOR...

LANGUAGE

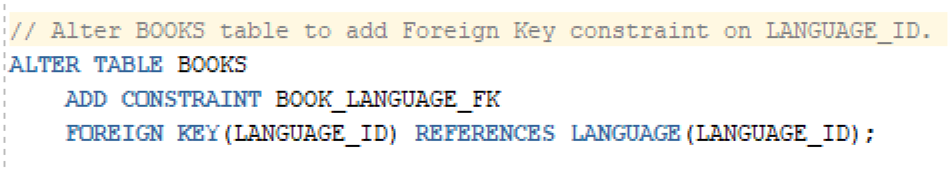


```
SELECT * FROM LANGUAGE;
```

```
All Rows Fetched: 0 in 0.011 seconds
```

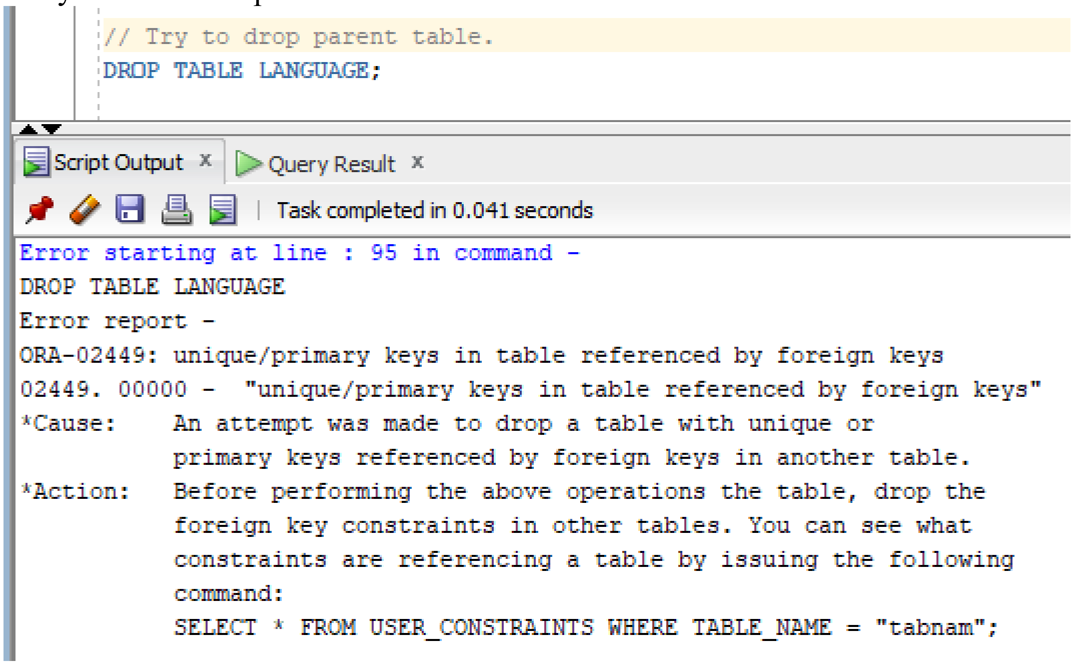
LANGUAGE_ID	LANGUAGE_NAME

Alter the BOOKS table to add Foreign Key constraint on LANGUAGE_ID.



```
// Alter BOOKS table to add Foreign Key constraint on LANGUAGE_ID.
ALTER TABLE BOOKS
    ADD CONSTRAINT BOOK_LANGUAGE_FK
    FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID);
```

Let's try to DROP the parent table LANGUAGE.



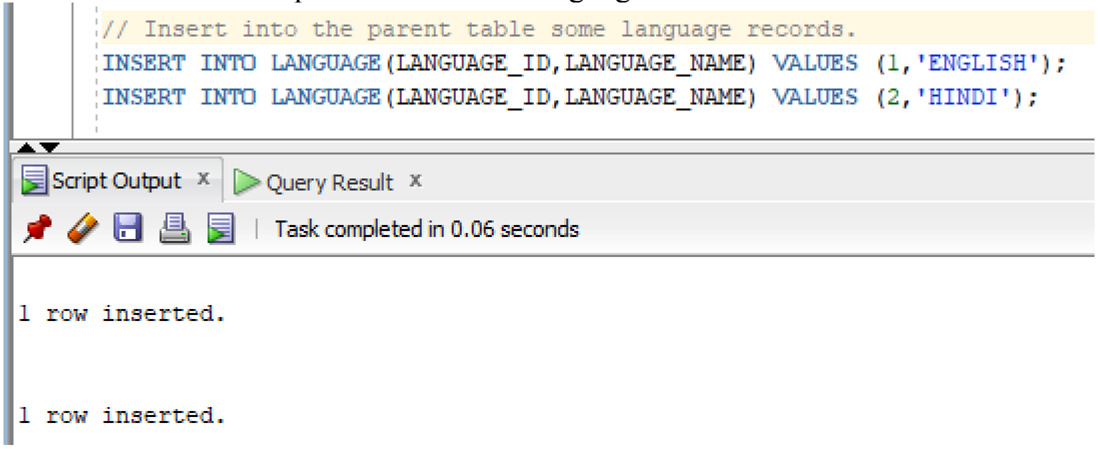
```
// Try to drop parent table.
DROP TABLE LANGUAGE;
```

```
Task completed in 0.041 seconds
```

```
Error starting at line : 95 in command -
DROP TABLE LANGUAGE
Error report -
ORA-02449: unique/primary keys in table referenced by foreign keys
02449. 00000 -  "unique/primary keys in table referenced by foreign keys"
*Cause:    An attempt was made to drop a table with unique or
           primary keys referenced by foreign keys in another table.
*Action:   Before performing the above operations the table, drop the
           foreign key constraints in other tables. You can see what
           constraints are referencing a table by issuing the following
           command:
           SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";
```

Dropping the parent table is prohibited due to the foreign key constraint defined.

Let's INSERT INTO the parent table come language records.

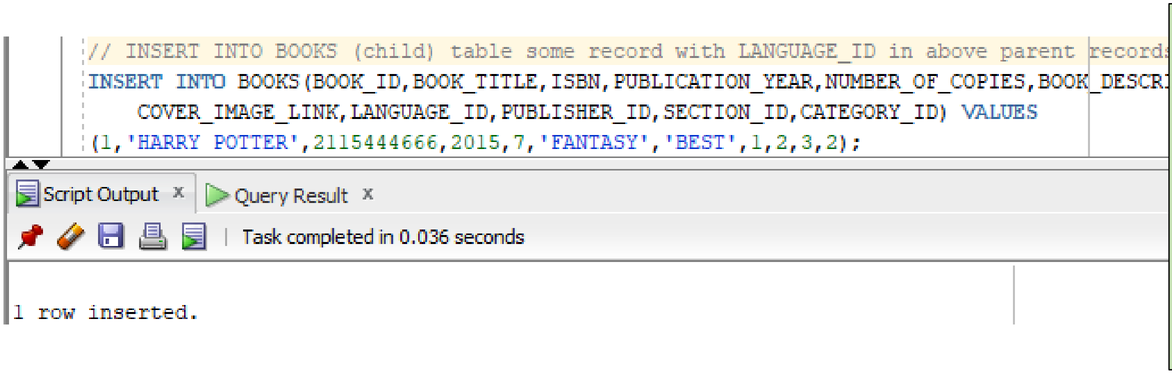


```
// Insert into the parent table some language records.
INSERT INTO LANGUAGE(LANGUAGE_ID,LANGUAGE_NAME) VALUES (1,'ENGLISH');
INSERT INTO LANGUAGE(LANGUAGE_ID,LANGUAGE_NAME) VALUES (2,'HINDI');
```

```
Task completed in 0.06 seconds
```

```
1 row inserted.


1 row inserted.
```

Let's INSERT INTO BOOKS (child) table some record with LANGUAGE_ID in above parent records.



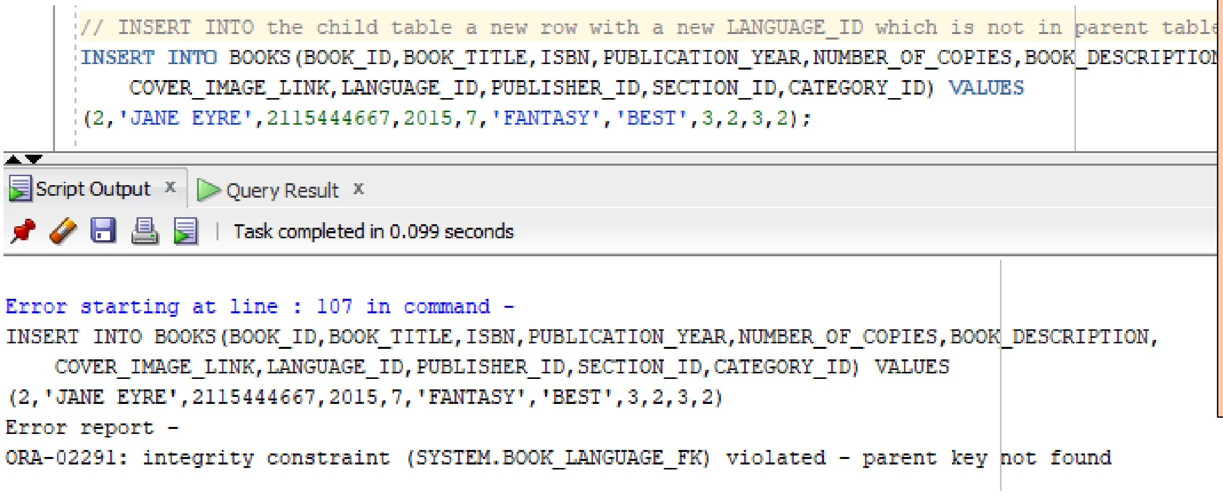
```
// INSERT INTO BOOKS (child) table some record with LANGUAGE_ID in above parent records
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRI
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(1,'HARRY POTTER',2115444666,2015,7,'FANTASY','BEST',1,2,3,2);
```

Task completed in 0.036 seconds

```
1 row inserted.
```

As this child record has an existing parent record in the parent table it is successfully inserted into child table.

Let's INSERT INTO the child table a new row with a new LANGUAGE_ID which is not in parent table.



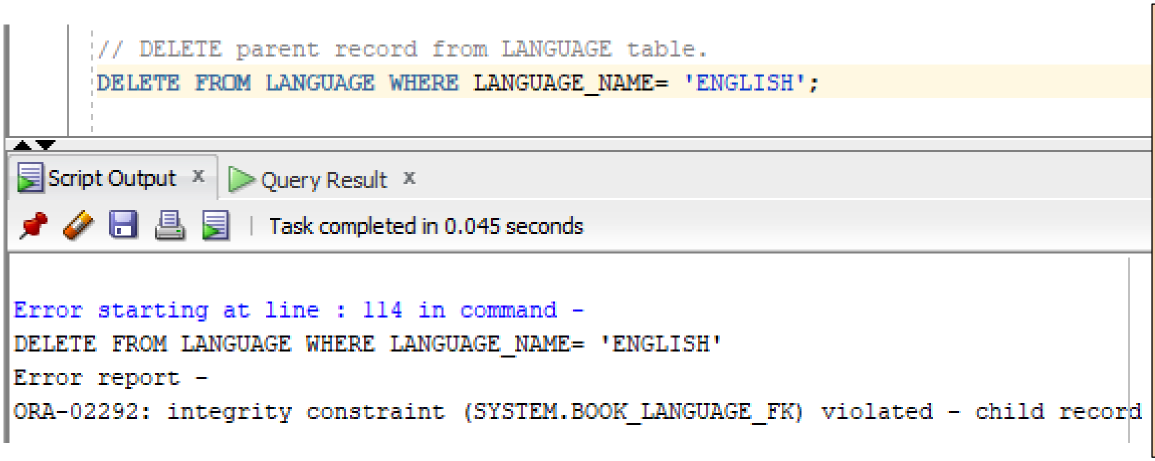
```
// INSERT INTO the child table a new row with a new LANGUAGE_ID which is not in parent table
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(2,'JANE EYRE',2115444667,2015,7,'FANTASY','BEST',3,2,3,2);
```

Task completed in 0.099 seconds

```
Error starting at line : 107 in command -
INSERT INTO BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
    COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID) VALUES
(2,'JANE EYRE',2115444667,2015,7,'FANTASY','BEST',3,2,3,2)
Error report -
ORA-02291: integrity constraint (SYSTEM.BOOK_LANGUAGE_FK) violated - parent key not found
```

It violates the Foreign key constraint as there is no relevant parent record found in the parent table.

Let's try to DELETE parent record from the parent table, which has a related child record in the child table.



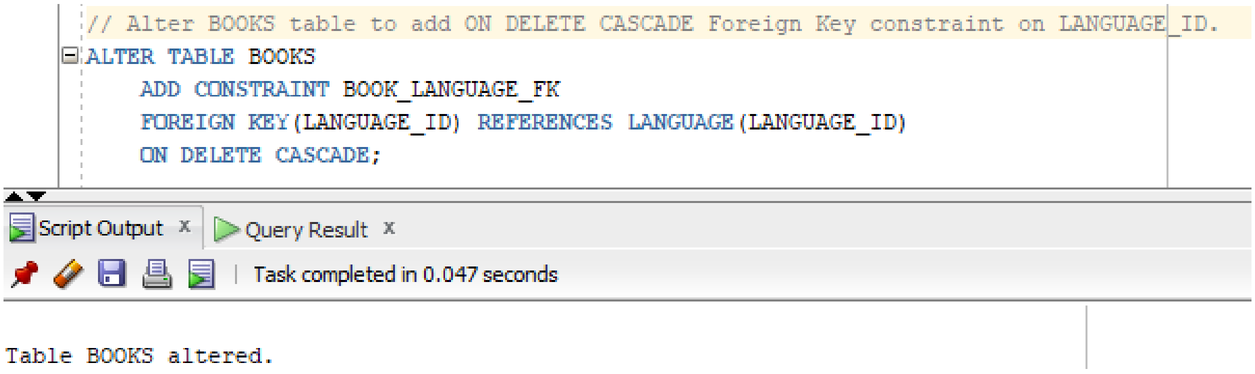
```
// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';
```

Task completed in 0.045 seconds

```
Error starting at line : 114 in command -
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH'
Error report -
ORA-02292: integrity constraint (SYSTEM.BOOK_LANGUAGE_FK) violated - child record
```

It violates the Foreign key constraint as there is a dependent child record existing in another table.

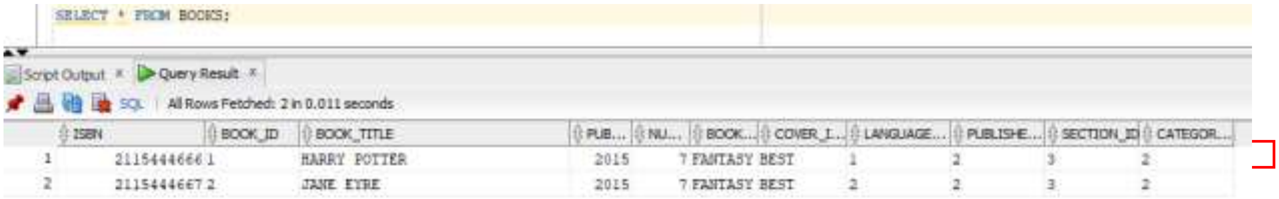
We can use **ON DELETE CASCADE** to specify the database to delete corresponding child records when a parent record is deleted.



```
// Alter BOOKS table to add ON DELETE CASCADE Foreign Key constraint on LANGUAGE_ID.
ALTER TABLE BOOKS
    ADD CONSTRAINT BOOK_LANGUAGE_FK
    FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID)
    ON DELETE CASCADE;
```

```
Task completed in 0.047 seconds

Table BOOKS altered.
```

BOOKS

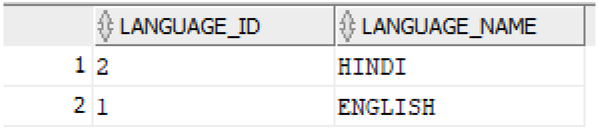


```
SELECT * FROM BOOKS;
```

All Rows Fetched: 2 in 0.011 seconds

	ISBN	BOOK_ID	BOOK_TITLE	PUB...	NU...	BOOK...	COVER_I...	LANGUAGE...	PUBLISHE...	SECTION_ID	CATEGOR...
1	2115444666	1	HARRY POTTER	2015	7	FANTASY	BEST	1	2	3	2
2	2115444667	2	JANE EYRE	2015	7	FANTASY	BEST	2	2	3	2

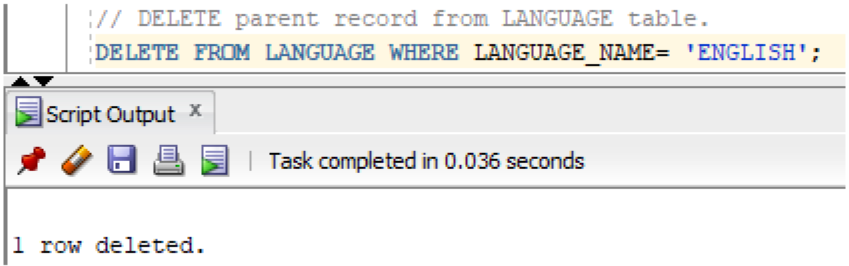
LANGUAGE



	LANGUAGE_ID	LANGUAGE_NAME
1	2	HINDI
2	1	ENGLISH

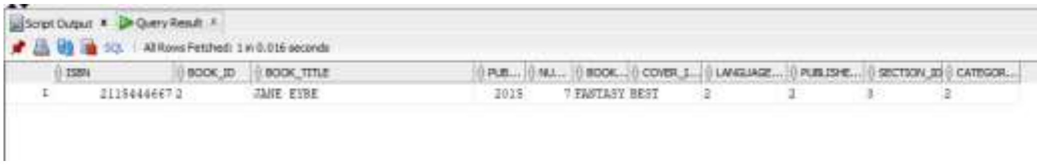
Now if we delete a parent record having child records in other tables, it will delete itself after deleting all child records as well.

Let's delete Language English from LANGUAGE table.



```
// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';
```

```
Task completed in 0.036 seconds

1 row deleted.
```

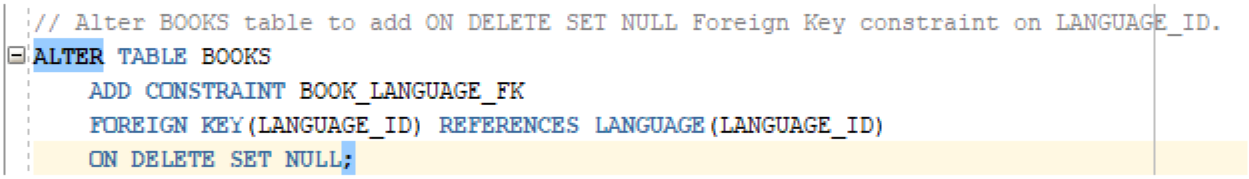


All Rows Fetched: 1 in 0.016 seconds

	ISBN	BOOK_ID	BOOK_TITLE	PUB...	NU...	BOOK...	COVER_I...	LANGUAGE...	PUBLISHE...	SECTION_ID	CATEGOR...
1	2115444667	2	JANE EYRE	2015	7	FANTASY	BEST	2	2	3	2

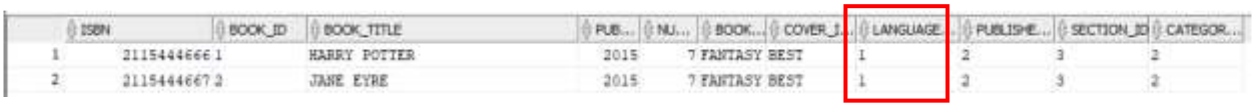
Now the BOOKS table has no HARRY POTTER record. It was the corresponding child record for English language record.

We can use **ON DELETE SET NULL** to delete a primary key row, instead of deleting all the foreign key rows, set the related foreign key columns to NULL.



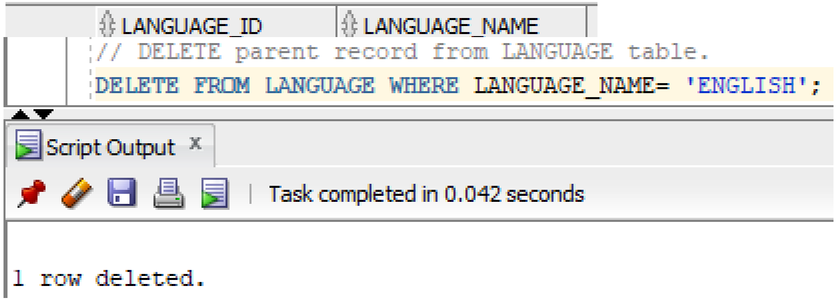
```
// Alter BOOKS table to add ON DELETE SET NULL Foreign Key constraint on LANGUAGE_ID.
ALTER TABLE BOOKS
    ADD CONSTRAINT BOOK_LANGUAGE_FK
    FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID)
    ON DELETE SET NULL;
```

BOOKS



	ISBN	BOOK_ID	BOOK_TITLE	PUB...	NU...	BOOK...	COVER_I...	LANGUAGE...	PUBLISHE...	SECTION_ID	CATEGOR...
1	2115444666	1	HARRY POTTER	2015	7	FANTASY	BEST	1	2	3	2
2	2115444667	2	JANE EYRE	2015	7	FANTASY	BEST	1	2	3	2

LANGUAGE



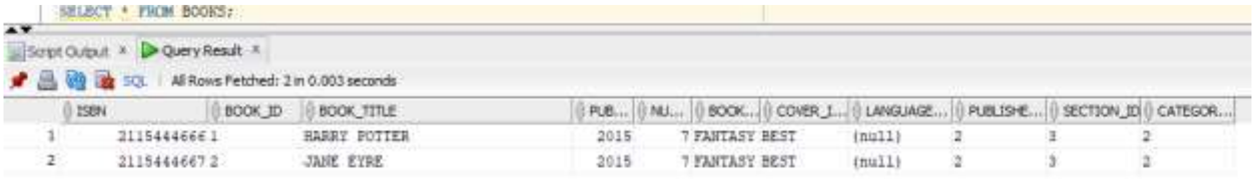
LANGUAGE_ID	LANGUAGE_NAME

```
// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';
```

Task completed in 0.042 seconds

```
1 row deleted.
```

Now the BOOKS will have NULL for Language ID columns which had 1.



```
SELECT * FROM BOOKS;
```

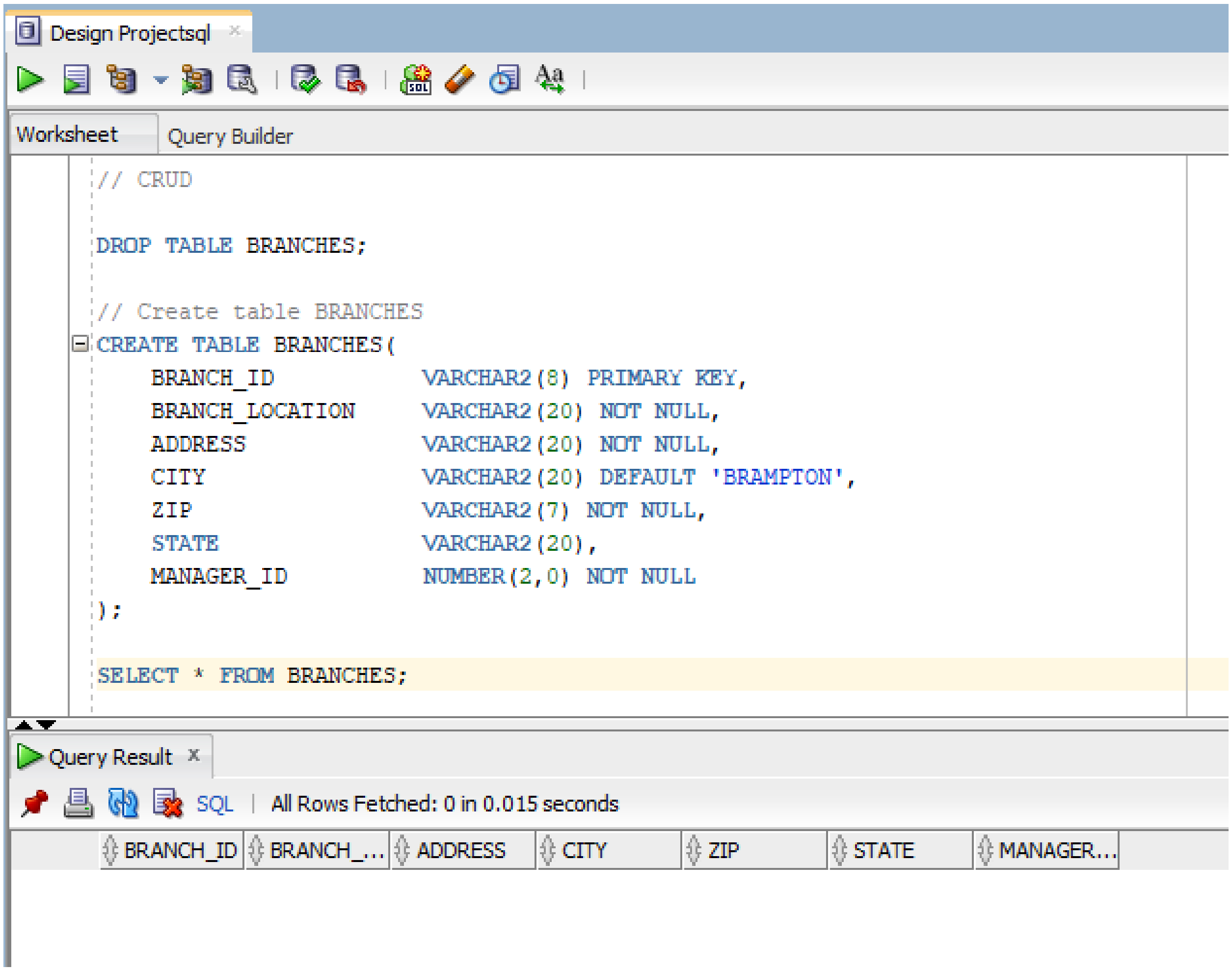
All Rows Fetched: 2 in 0.003 seconds

	ISBN	BOOK_ID	BOOK_TITLE	PUB...	NU...	BOOK...	COVER_I...	LANGUAGE...	PUBLISHE...	SECTION_ID	CATEGOR...
1	2115444666	1	HARRY POTTER	2015	7	FANTASY	BEST	(null)	2	3	2
2	2115444667	2	JANE EYRE	2015	7	FANTASY	BEST	(null)	2	3	2

6. PHYSICAL DATABASE DESIGN

Here we would consider a table which includes the branch details of BMO bank branches in the Peel Region, Ontario, Canada.

6.1 CREATE TABLE with name BRANCHES



```
// CRUD

DROP TABLE BRANCHES;

// Create table BRANCHES
CREATE TABLE BRANCHES(
    BRANCH_ID           VARCHAR2(8) PRIMARY KEY,
    BRANCH_LOCATION     VARCHAR2(20) NOT NULL,
    ADDRESS             VARCHAR2(20) NOT NULL,
    CITY                VARCHAR2(20) DEFAULT 'BRAMPTON',
    ZIP                 VARCHAR2(7) NOT NULL,
    STATE               VARCHAR2(20),
    MANAGER_ID          NUMBER(2,0) NOT NULL
);

SELECT * FROM BRANCHES;
```

All Rows Fetched: 0 in 0.015 seconds

BRANCH_ID	BRANCH_...	ADDRESS	CITY	ZIP	STATE	MANAGER...

The CREATE TABLE statement is used to create a new table named "BRANCHES."

branch_id is a unique identifier for each record in the table. It is of type VARCHAR and,is the primary key of the table, meaning it uniquely identifies each branch.

Branch_location, address, zip and manager_id are having NOT NULL constraints which means those fields cannot be NULL.

City has DEFAULT constraint ‘BRAMPTON’ which means if the user does not input a city for a record, its default city will be stored as BRAMPTON.

State can be NULL.

6.2 CRUD OPERATIONS ON THIS TABLE

Create: Insert new rows into the table with branch information.

Read: Fetch branch information using SELECT queries.

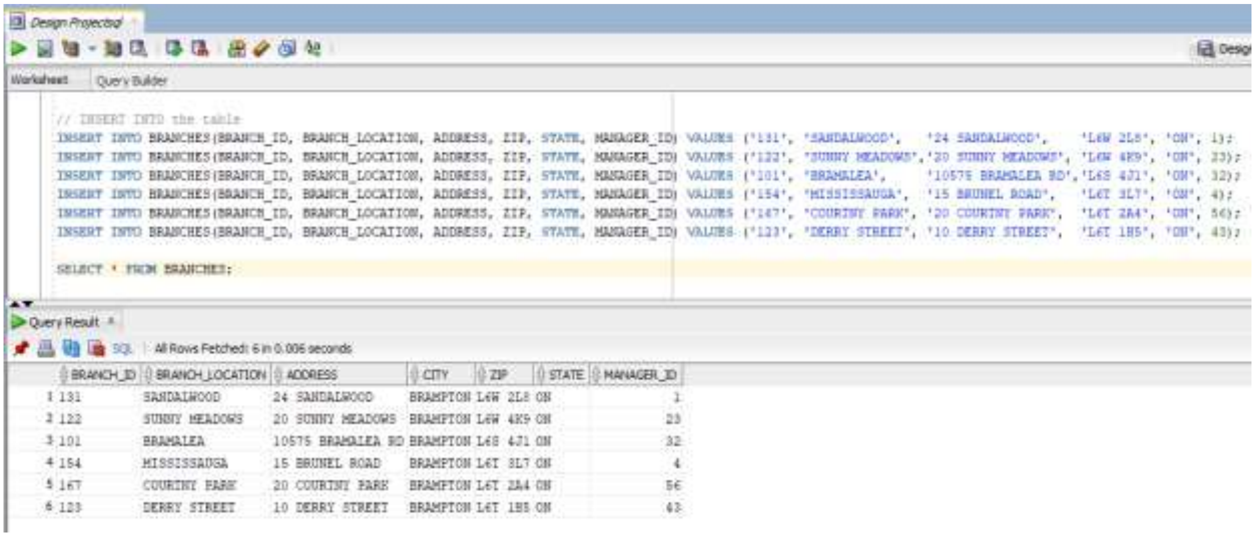
Update: Modify existing branch information using UPDATE queries.

Delete: Remove branch records from the table using DELETE queries.

6.2.1 CREATE - INSERT INTO

Now, let's insert 5 data rows into the "BRANCHES" table.

We have not entered city detail in the INSERT queries. So, the default city BRAMPTON will be added to all records.



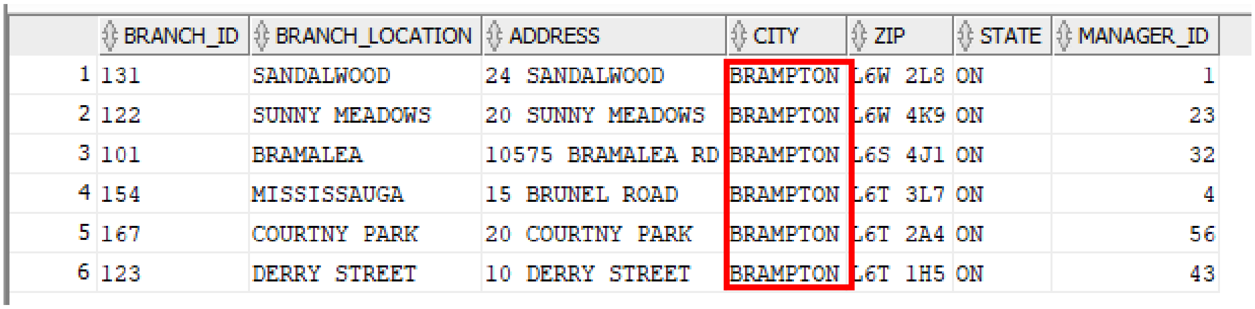
```
// INSERT INTO the table
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('131', 'SANDALWOOD',   '24 SANDALWOOD',    'L6W 2L8', 'ON', 1);
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('122', 'SUNNY MEADOWS','20 SUNNY MEADOWS', 'L6W 4K9', 'ON', 23);
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('101', 'BRAMALEA',     '10575 BRAMALEA RD','L6S 4J1', 'ON', 32);
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('154', 'MISSISSAUGA',  '15 BRUNEL ROAD',   'L6T 3L7', 'ON', 4);
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('167', 'COURTNY PARK', '20 COURTNY PARK',  'L6T 2A4', 'ON', 56);
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE, MANAGER_ID) VALUES ('123', 'DERRY STREET', '10 DERRY STREET',  'L6T 1H5', 'ON', 43);

SELECT * FROM BRANCHES;
```

All Rows Fetched: 6 in 0.006 seconds

	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	154	MISSISSAUGA	15 BRUNEL ROAD	BRAMPTON	L6T 3L7	ON	4
5	167	COURTNY PARK	20 COURTNY PARK	BRAMPTON	L6T 2A4	ON	56
6	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

6.2.2 READ - SELECT

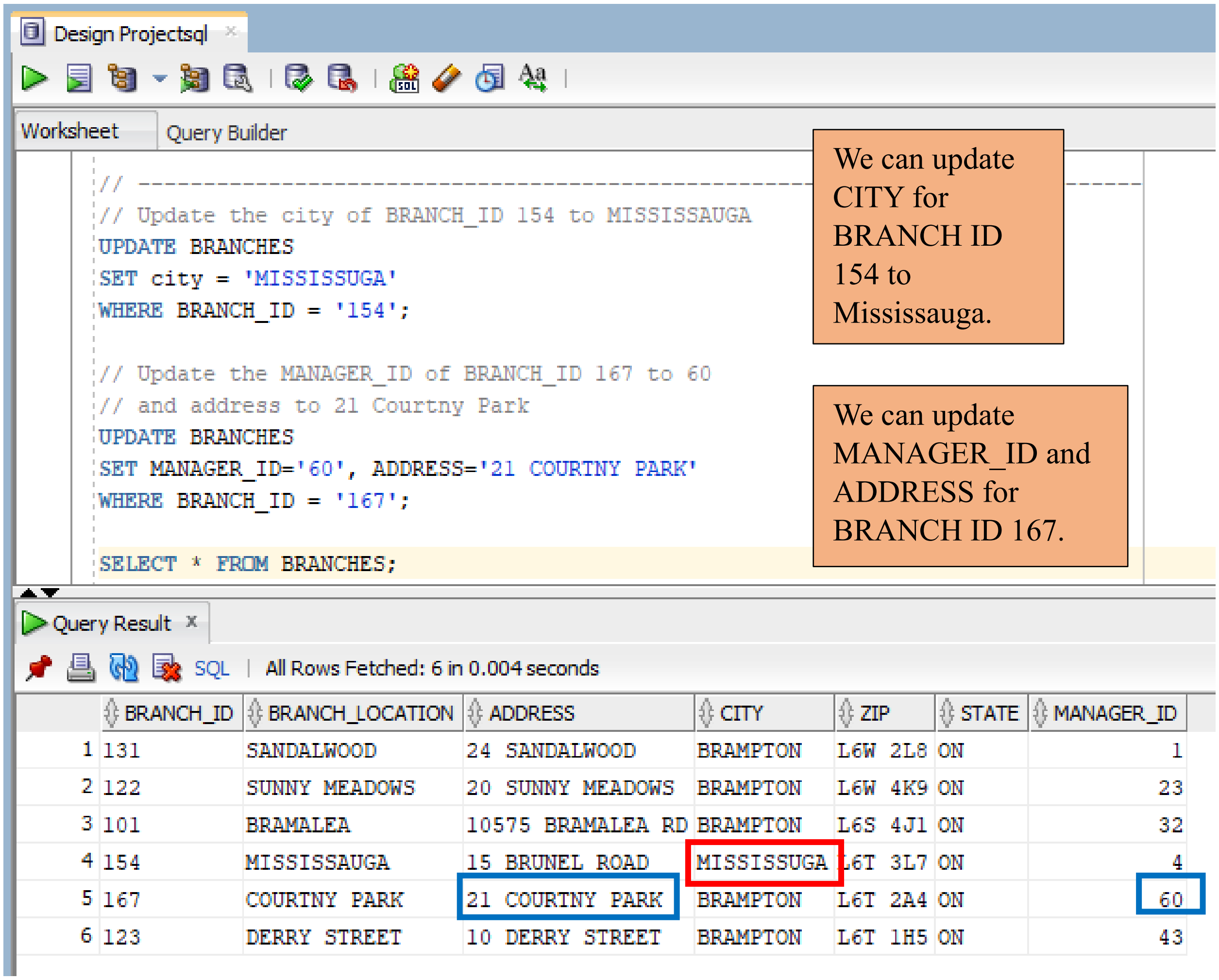
To retrieve author information from the "Branches" table, you can use the SELECT statement. For example, to get all the branches using ‘SELECT * FROM BRNACHES’ query.



	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	154	MISSISSAUGA	15 BRUNEL ROAD	BRAMPTON	L6T 3L7	ON	4
5	167	COURTNY PARK	20 COURTNY PARK	BRAMPTON	L6T 2A4	ON	56
6	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

6.2.3 UPDATE

To modify an existing branch information, we can use the UPDATE statement.



```
// -------------------------------------------------------------------------------------
// Update the city of BRANCH_ID 154 to MISSISSAUGA
UPDATE BRANCHES
SET city = 'MISSISSUGA'
WHERE BRANCH_ID = '154';

// Update the MANAGER_ID of BRANCH_ID 167 to 60
// and address to 21 Courtny Park
UPDATE BRANCHES
SET MANAGER_ID='60', ADDRESS='21 COURTNY PARK'
WHERE BRANCH_ID = '167';

SELECT * FROM BRANCHES;
```

We can update CITY for BRANCH ID 154 to Mississauga.

We can update MANAGER_ID and ADDRESS for BRANCH ID 167.

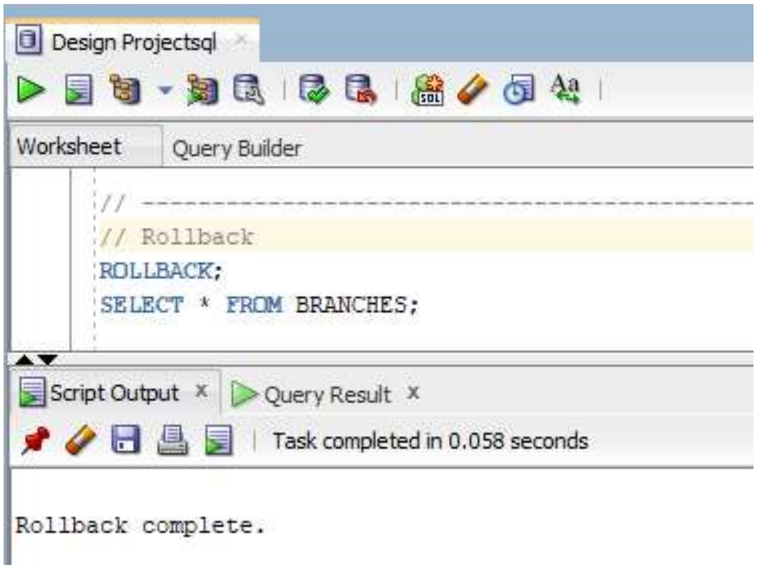
All Rows Fetched: 6 in 0.004 seconds

	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	154	MISSISSAUGA	15 BRUNEL ROAD	MISSISSUGA	L6T 3L7	ON	4
5	167	COURTNY PARK	21 COURTNY PARK	BRAMPTON	L6T 2A4	ON	60
6	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

This query updates the biography of J.K. Rowling in the "Author" table to “harry potter movies”.

6.2.4 ROLLBACK

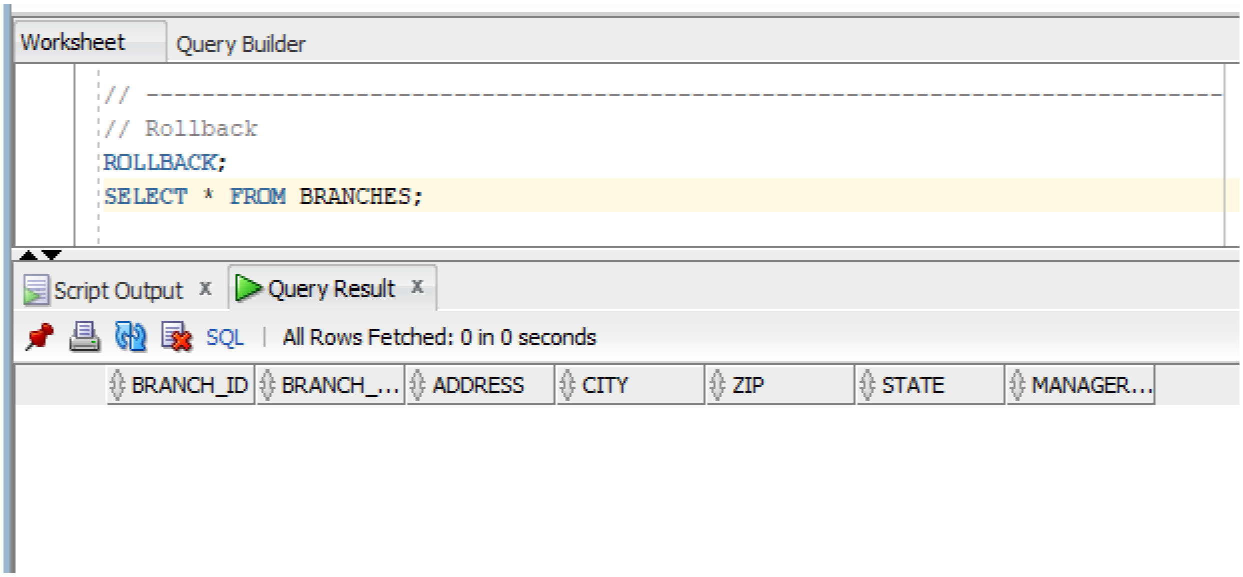
Rollback – It will rollback to empty table.



```
// -----------------------------------------
// Rollback
ROLLBACK;
SELECT * FROM BRANCHES;
```

Task completed in 0.058 seconds

```
Rollback complete.
```



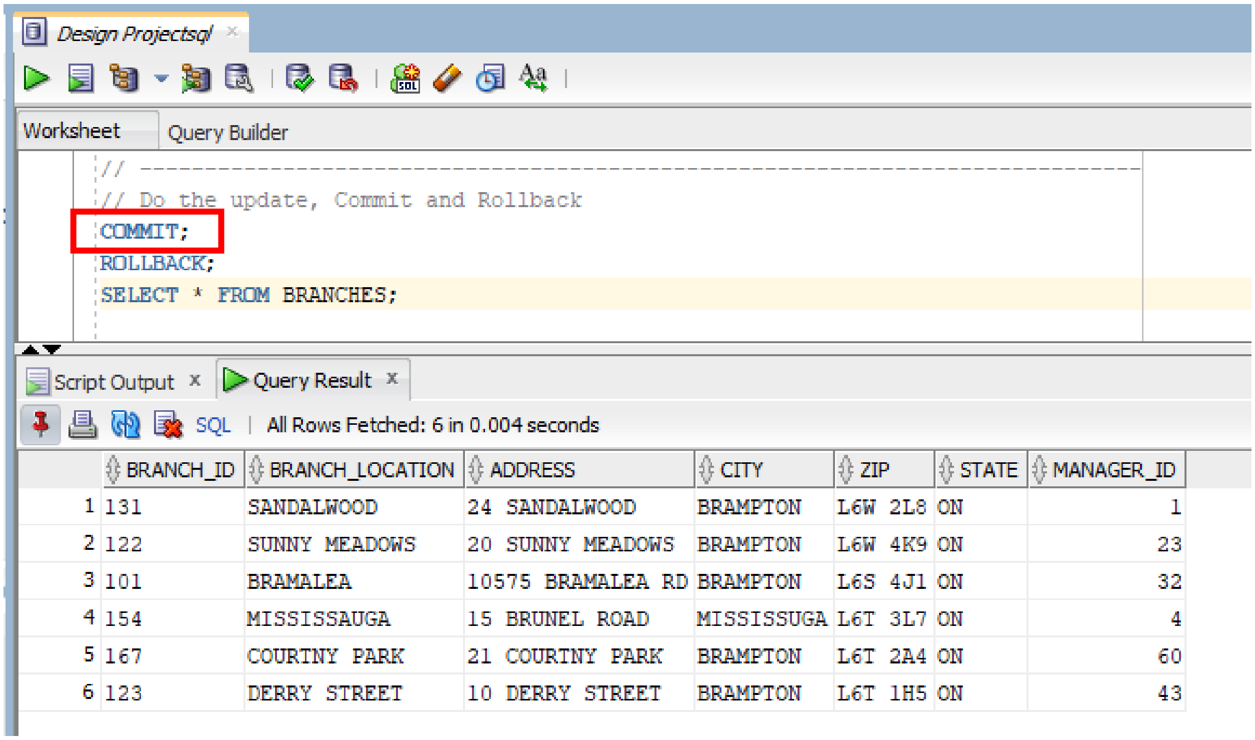
```
// ------------------------------------------------------------------
// Rollback
ROLLBACK;
SELECT * FROM BRANCHES;
```

All Rows Fetched: 0 in 0 seconds

BRANCH_ID	BRANCH_...	ADDRESS	CITY	ZIP	STATE	MANAGER...

6.2.5 COMMIT and ROLLBACK

Now it will not rollback. It will retain all the records and updates.

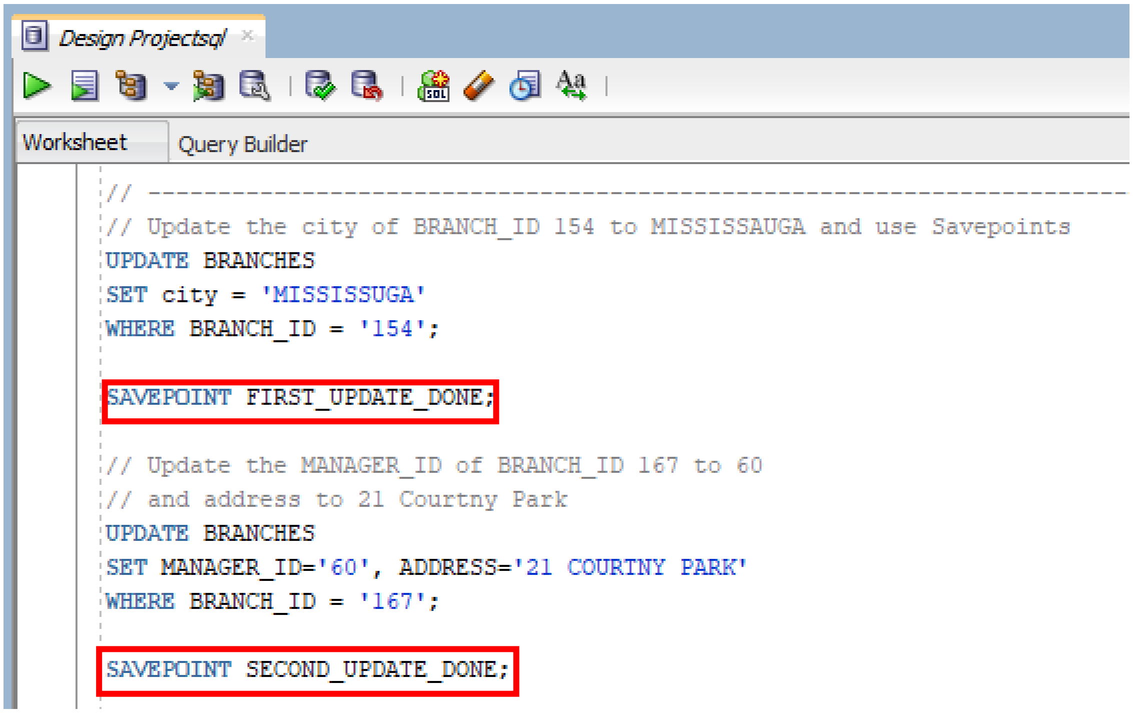


```
// -------------------------------------------------------------------------
// Do the update, Commit and Rollback
COMMIT;
ROLLBACK;
SELECT * FROM BRANCHES;
```

All Rows Fetched: 6 in 0.004 seconds

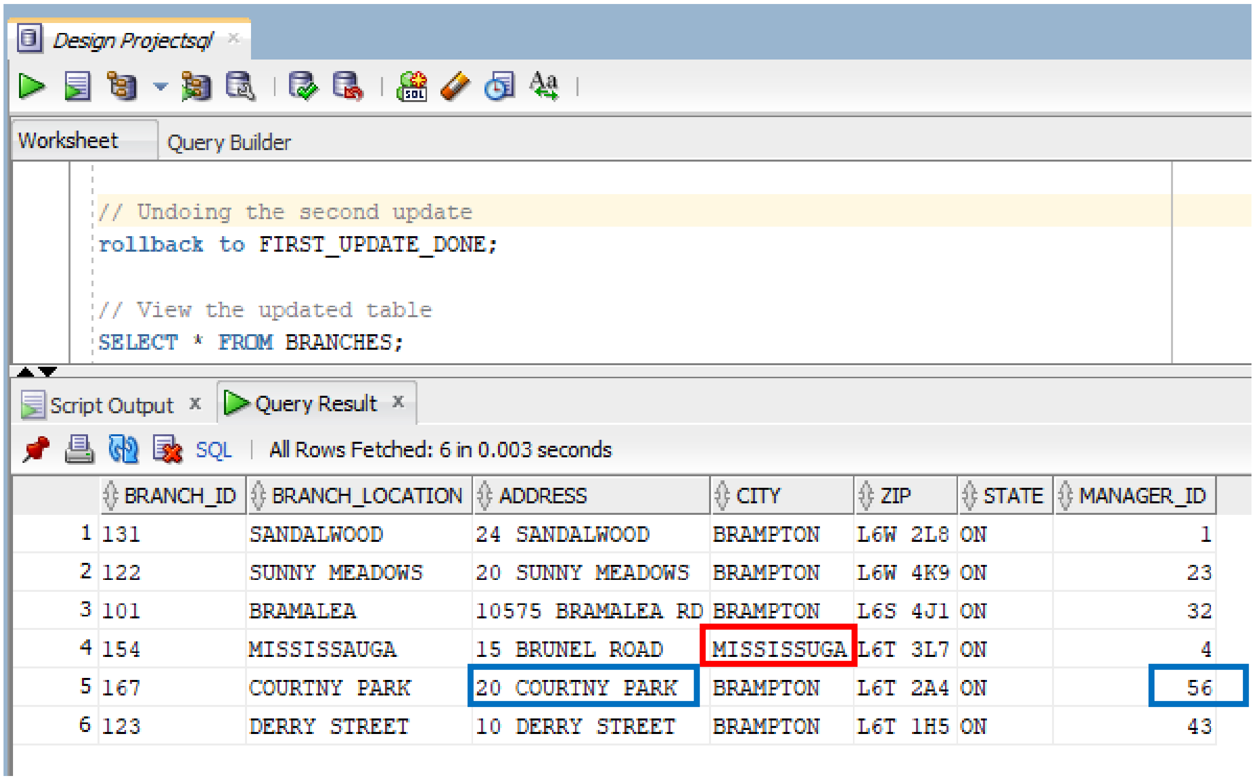
	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	154	MISSISSAUGA	15 BRUNEL ROAD	MISSISSUGA	L6T 3L7	ON	4
5	167	COURTNY PARK	21 COURTNY PARK	BRAMPTON	L6T 2A4	ON	60
6	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

6.2.6 SAVEPOINT



```
// -----------------------------------------------------------------
// Update the city of BRANCH_ID 154 to MISSISSAUGA and use Savepoints
UPDATE BRANCHES
SET city = 'MISSISSUGA'
WHERE BRANCH_ID = '154';

SAVEPOINT FIRST_UPDATE_DONE;

// Update the MANAGER_ID of BRANCH_ID 167 to 60
// and address to 21 Courtny Park
UPDATE BRANCHES
SET MANAGER_ID='60', ADDRESS='21 COURTNY PARK'
WHERE BRANCH_ID = '167';

SAVEPOINT SECOND_UPDATE_DONE;
```

Let's rollback to FIRST_UPDATE_DONE stage. This will un-do the second update.



```
// Undoing the second update
rollback to FIRST_UPDATE_DONE;

// View the updated table
SELECT * FROM BRANCHES;
```

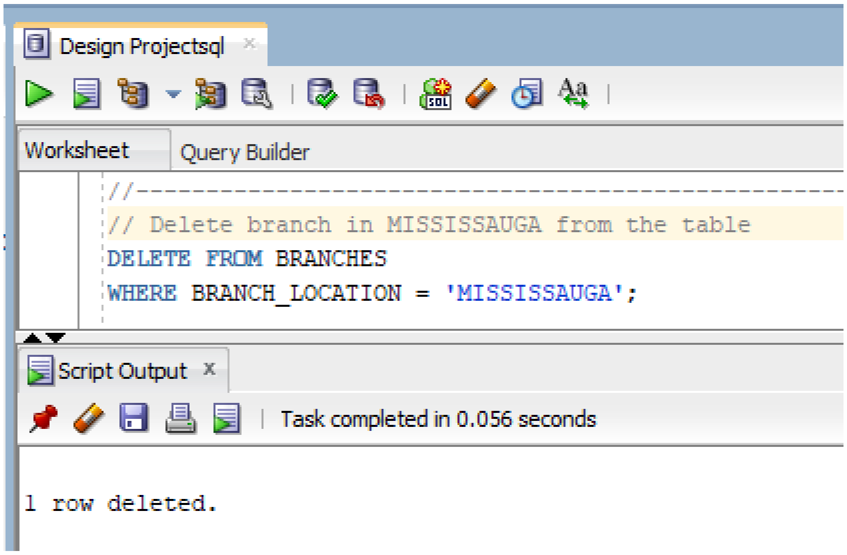
All Rows Fetched: 6 in 0.003 seconds

	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	154	MISSISSAUGA	15 BRUNEL ROAD	MISSISSUGA	L6T 3L7	ON	4
5	167	COURTNY PARK	20 COURTNY PARK	BRAMPTON	L6T 2A4	ON	56
6	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

It has rolled back successfully to the place where only the CITY of BRANCH ID 154 had been changed.

6.2.8 DELETE

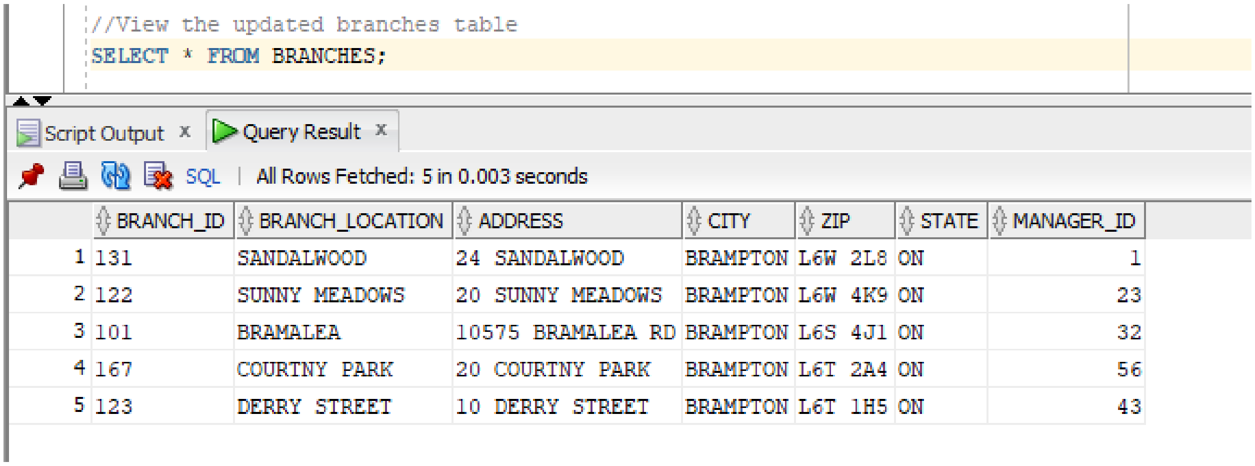
To remove an author from the "Author" table, you can use the DELETE statement. For example, let's delete the author with id 4 (Charles Dickens):



```
//------------------------------------------------
// Delete branch in MISSISSAUGA from the table
DELETE FROM BRANCHES
WHERE BRANCH_LOCATION = 'MISSISSAUGA';
```

Task completed in 0.056 seconds

```
1 row deleted.
```

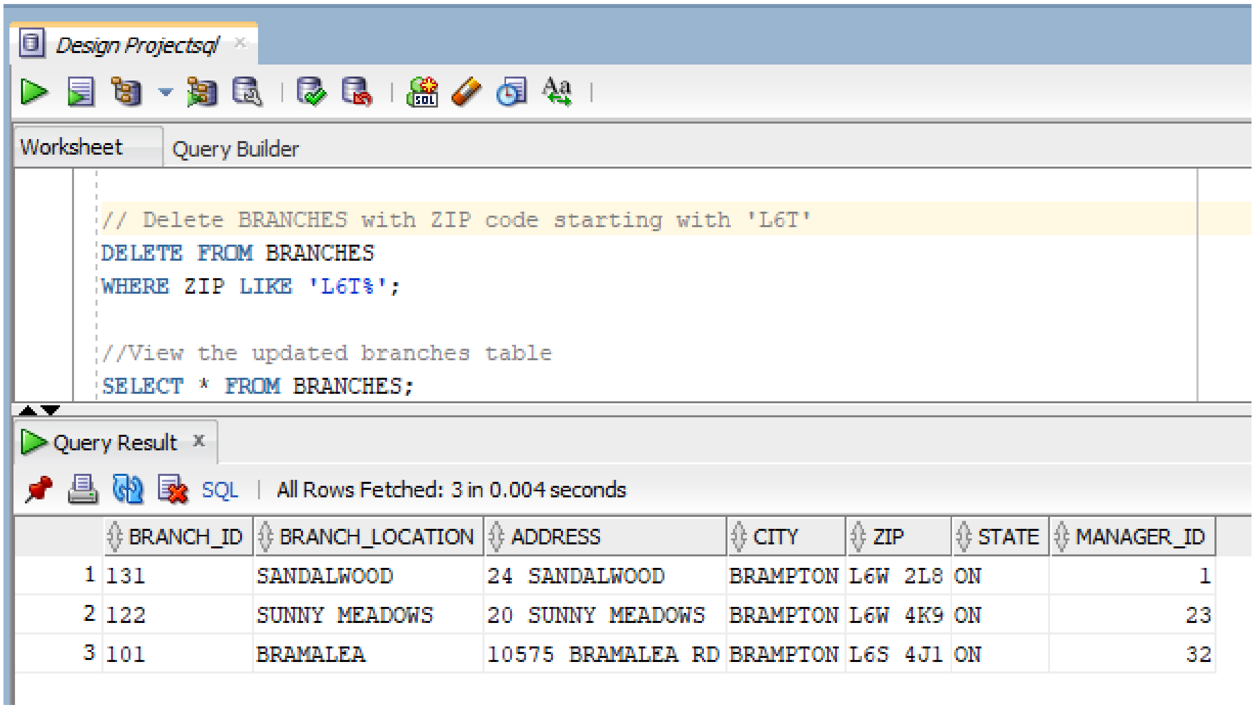


```
//View the updated branches table
SELECT * FROM BRANCHES;
```

All Rows Fetched: 5 in 0.003 seconds

	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32
4	167	COURTNY PARK	20 COURTNY PARK	BRAMPTON	L6T 2A4	ON	56
5	123	DERRY STREET	10 DERRY STREET	BRAMPTON	L6T 1H5	ON	43

Let’s DELETE all the branches having zip code starting with ‘L6T’.



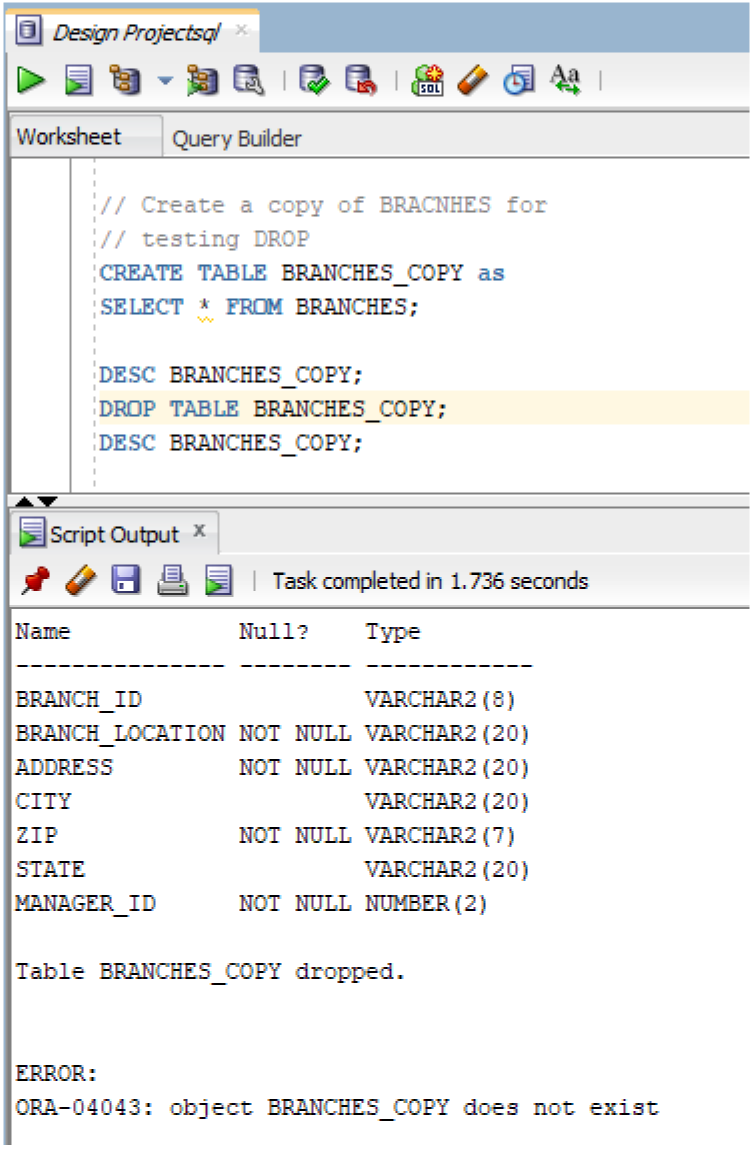
```
// Delete BRANCHES with ZIP code starting with 'L6T'
DELETE FROM BRANCHES
WHERE ZIP LIKE 'L6T%';

//View the updated branches table
SELECT * FROM BRANCHES;
```

All Rows Fetched: 3 in 0.004 seconds

	BRANCH_ID	BRANCH_LOCATION	ADDRESS	CITY	ZIP	STATE	MANAGER_ID
1	131	SANDALWOOD	24 SANDALWOOD	BRAMPTON	L6W 2L8	ON	1
2	122	SUNNY MEADOWS	20 SUNNY MEADOWS	BRAMPTON	L6W 4K9	ON	23
3	101	BRAMALEA	10575 BRAMALEA RD	BRAMPTON	L6S 4J1	ON	32

6.2.9 DROP

Let’s create a copy of the BRANCHES table for testing the DROP feature.



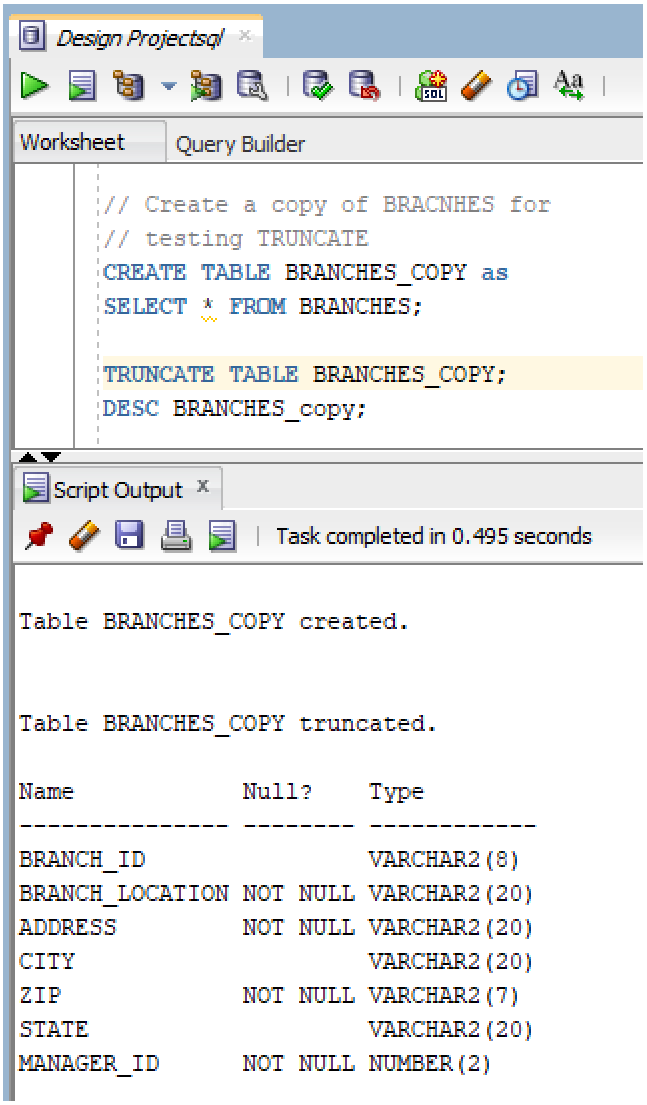
```
// Create a copy of BRACNHES for
// testing DROP
CREATE TABLE BRANCHES_COPY as
SELECT * FROM BRANCHES;

DESC BRANCHES_COPY;
DROP TABLE BRANCHES_COPY;
DESC BRANCHES_COPY;
```

```
Name            Null?    Type
--------------- -------- ------------
BRANCH_ID                VARCHAR2(8)
BRANCH_LOCATION NOT NULL VARCHAR2(20)
ADDRESS         NOT NULL VARCHAR2(20)
CITY                     VARCHAR2(20)
ZIP             NOT NULL VARCHAR2(7)
STATE                    VARCHAR2(20)
MANAGER_ID      NOT NULL NUMBER(2)

Table BRANCHES_COPY dropped.


ERROR:
ORA-04043: object BRANCHES_COPY does not exist
```

After dropping such table does not exist. It removes the object from the database.

6.2.10 TRUNCATE

Let’s create a copy of the BRANCHES table for testing the TRUNCATE feature.

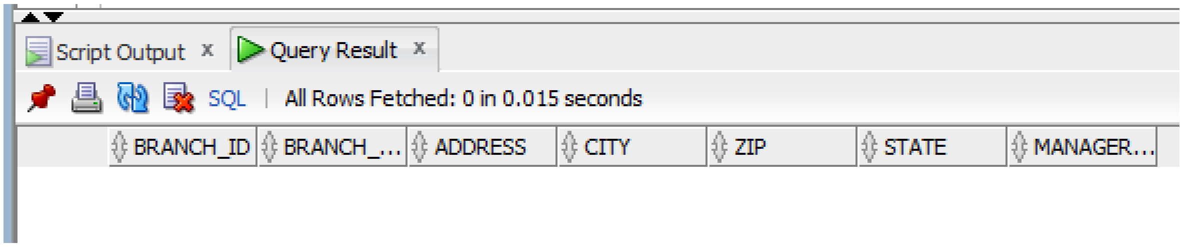


```
// Create a copy of BRACNHES for
// testing TRUNCATE
CREATE TABLE BRANCHES_COPY as
SELECT * FROM BRANCHES;

TRUNCATE TABLE BRANCHES_COPY;
DESC BRANCHES_copy;
```

Task completed in 0.495 seconds

```
Table BRANCHES_COPY created.


Table BRANCHES_COPY truncated.

Name            Null?    Type
--------------- -------- ------------
BRANCH_ID                VARCHAR2(8)
BRANCH_LOCATION NOT NULL VARCHAR2(20)
ADDRESS         NOT NULL VARCHAR2(20)
CITY                     VARCHAR2(20)
ZIP             NOT NULL VARCHAR2(7)
STATE                    VARCHAR2(20)
MANAGER_ID      NOT NULL NUMBER(2)
```

It removes all the records from the table. But the table still exists.



All Rows Fetched: 0 in 0.015 seconds

BRANCH_ID	BRANCH_...	ADDRESS	CITY	ZIP	STATE	MANAGER...

These are the basic CRUD operations that can be performed on the "Branches" table. We can combine them to manage and manipulate the data as needed. Always exercise caution when performing DELETE operations, as they permanently remove data from the table.

APPENDIX 1

```
// Create table BOOKS and add table level constraint parimary key.
CREATE TABLE BOOKS(
  BOOK_ID     VARCHAR(6),
  BOOK_TITLE          VARCHAR(30),
  ISBN                INTEGER,
  PUBLICATION_YEAR  INTEGER,
  NUMBER_OF_COPIES  INTEGER,
  BOOK_DESCRIPTION  VARCHAR(50),
  COVER_IMAGE_LINK  VARCHAR(30),
  LANGUAGE_ID    VARCHAR(6),
  PUBLISHER_ID   VARCHAR(6),
  SECTION_ID   VARCHAR(6),
  CATEGORY_ID    VARCHAR(6),
  PRIMARY KEY(BOOK_ID)
  );

// See Table Desicription in BOOKS table.
DESC BOOKS;

// Insert example records into BOOKS table.
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
  COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(1,'GAME OF THRONES',1115244665,2015,5,'FANTASY','GOOD',1,2,3,2);

// Inserting a 2nd record with same BOOK_ID gives primary key violation error.
```

```
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(1,'HARRY POTTER',2115444666,2015,7,'FANTASY','BEST',1,2,3,2);

// Inserting a record with BOOK_ID value set to NULL also gives an error
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(NULL,'CHRONICLES',1115244666,2015,5,'FANTASY','GOOD',1,2,3,2);

// Insertion adhered to Primary key constraint
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BOOK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(2,'GAME OF THRONES',1115244666,2015,5,'FANTASY','GOOD',1,2,3,2);

// Create SECTION Table to add NOT NULL Column Level Constraint.
CREATE TABLE SECTION(
   SECTION_ID INTEGER PRIMARY KEY,
   SECTION_NAME VARCHAR(25) NOT NULL);

INSERT INTO SECTION(SECTION_ID,SECTION_NAME) VALUES (1,'');

// Alter BOOKS table to add table level constraint Unique Key.
```

```
ALTER TABLE BOOKS
ADD CONSTRAINT ISBN_UK
UNIQUE (ISBN);

DELETE (BOOKS);

// Trying to add same ISBN no for two rows - Unique key violation testing.
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(3,'CHRONICLES',1115244667,2015,5,'FANTASY','GOOD',1,2,3,2);

INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(4,'CHRONICLES',1115244667,2015,5,'FANTASY','GOOD',1,2,3,2);

// Alter BOOKS table to add Check Constraint PUBLICATION_YEAR > 1900.
ALTER TABLE BOOKS
         ADD CONSTRAINT PUBLICATION_YEAR_CK CHECK (PUBLICATION_YEAR >
1900);

// Try to inserta record with Publication year 1880 which is < 1900.
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(3,'LORD OF RINGS',15515244665,1880,5,'FANTASY','GOOD',1,2,3,2);

// Try to update a record with Publication year = 1800
UPDATE BOOKS
SET PUBLICATION_YEAR = 1890 WHERE BOOK_ID = 1;

// Create LANGUAGE Table to implement Foreign Key Constraint.
CREATE TABLE LANGUAGE(
LANGUAGE_ID     VARCHAR(6)     PRIMARY KEY,
LANGUAGE_NAME   VARCHAR(25));

// DROP TABLE BOOKS and re-create it with above constraints.

// Now both tables are empty.
SELECT * FROM BOOKS;
SELECT * FROM LANGUAGE;

// Alter BOOKS table to add Foreign Key constraint on LANGUAGE_ID.
ALTER TABLE BOOKS
        ADD CONSTRAINT BOOK_LANGUAGE_FK
        FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID);

// Try to drop parent table.
DROP TABLE LANGUAGE;

// Insert into the parent table some language records.
INSERT INTO LANGUAGE(LANGUAGE_ID,LANGUAGE_NAME) VALUES
(1,'ENGLISH');
INSERT INTO LANGUAGE(LANGUAGE_ID,LANGUAGE_NAME) VALUES (2,'HINDI');
```

```
// INSERT INTO BOOKS (child) table some record with LANGUAGE_ID in above
parent records.
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(1,'HARRY POTTER',2115444666,2015,7,'FANTASY','BEST',1,2,3,2);

// INSERT INTO the child table a new row with a new LANGUAGE_ID which is not in
parent table.
INSERT INTO
BOOKS(BOOK_ID,BOOK_TITLE,ISBN,PUBLICATION_YEAR,NUMBER_OF_COPIES,BO
OK_DESCRIPTION,
   COVER_IMAGE_LINK,LANGUAGE_ID,PUBLISHER_ID,SECTION_ID,CATEGORY_ID)
VALUES
(2,'JANE EYRE',2115444667,2015,7,'FANTASY','BEST',3,2,3,2);

// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';

// Alter BOOKS table to add ON DELETE CASCADE Foreign Key constraint on
LANGUAGE_ID.
ALTER TABLE BOOKS
         ADD CONSTRAINT BOOK_LANGUAGE_FK
         FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID)
   ON DELETE CASCADE;

// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';
```

```
// Alter BOOKS table to add ON DELETE SET NULL Foreign Key constraint on
LANGUAGE_ID.
ALTER TABLE BOOKS
        ADD CONSTRAINT BOOK_LANGUAGE_FK
        FOREIGN KEY(LANGUAGE_ID) REFERENCES LANGUAGE(LANGUAGE_ID)
   ON DELETE SET NULL;

// DELETE parent record from LANGUAGE table.
DELETE FROM LANGUAGE WHERE LANGUAGE_NAME= 'ENGLISH';
```

APPENDIX 2

```
// CRUD


DROP TABLE BRANCHES;


// Create table BRANCHES
CREATE TABLE BRANCHES(
   BRANCH_ID          VARCHAR2(8) PRIMARY KEY,
   BRANCH_LOCATION    VARCHAR2(20) NOT NULL,
   ADDRESS            VARCHAR2(20) NOT NULL,
   CITY               VARCHAR2(20) DEFAULT 'BRAMPTON',
   ZIP                VARCHAR2(7) NOT NULL,
   STATE              VARCHAR2(20),
   MANAGER_ID         NUMBER(2,0) NOT NULL
);


SELECT * FROM BRANCHES;


// INSERT INTO the table
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('131', 'SANDALWOOD',   '24 SANDALWOOD',    'L6W 2L8', 'ON', 1);
```

```
INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('122', 'SUNNY MEADOWS','20 SUNNY MEADOWS', 'L6W 4K9',
'ON', 23);

INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('101', 'BRAMALEA',     '10575 BRAMALEA RD','L6S 4J1', 'ON', 32);

INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('154', 'MISSISSAUGA',  '15 BRUNEL ROAD',   'L6T 3L7', 'ON', 4);

INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('167', 'COURTNY PARK', '20 COURTNY PARK',  'L6T 2A4', 'ON', 56);

INSERT INTO BRANCHES(BRANCH_ID, BRANCH_LOCATION, ADDRESS, ZIP, STATE,
MANAGER_ID) VALUES ('123', 'DERRY STREET', '10 DERRY STREET',  'L6T 1H5', 'ON', 43);


SELECT * FROM BRANCHES;


// ------------------------------------------------------------------------------

// Update the city of BRANCH_ID 154 to MISSISSAUGA

UPDATE BRANCHES

SET city = 'MISSISSUGA'

WHERE BRANCH_ID = '154';


// Update the MANAGER_ID of BRANCH_ID 167 to 60

// and address to 21 Courtny Park

UPDATE BRANCHES

SET MANAGER_ID='60', ADDRESS='21 COURTNY PARK'
```

```
WHERE BRANCH_ID = '167';

SELECT * FROM BRANCHES;

// ------------------------------------------------------------------------
// Rollback
ROLLBACK;
SELECT * FROM BRANCHES;

// ------------------------------------------------------------------------
// Do the update, Commit and Rollback
COMMIT;
ROLLBACK;
SELECT * FROM BRANCHES;

// ------------------------------------------------------------------------
// Update the city of BRANCH_ID 154 to MISSISSAUGA and use Savepoints
UPDATE BRANCHES
SET city = 'MISSISSUGA'
WHERE BRANCH_ID = '154';

SAVEPOINT FIRST_UPDATE_DONE;
```

```
// Update the MANAGER_ID of BRANCH_ID 167 to 60
// and address to 21 Courtny Park
UPDATE BRANCHES
SET MANAGER_ID='60', ADDRESS='21 COURTNY PARK'
WHERE BRANCH_ID = '167';

SAVEPOINT SECOND_UPDATE_DONE;

// Undoing the second update
rollback to FIRST_UPDATE_DONE;

// View the updated table
SELECT * FROM BRANCHES;

//----------------------------------------------------------------------------
// Delete branch in MISSISSAUGA from the table
DELETE FROM BRANCHES
WHERE BRANCH_LOCATION = 'MISSISSAUGA';

//View the updated branches table
SELECT * FROM BRANCHES;

// Delete BRANCHES with ZIP code starting with 'L6T'
```

```
DELETE FROM BRANCHES
WHERE ZIP LIKE 'L6T%';

//View the updated branches table
SELECT * FROM BRANCHES;


rollback;

//View the updated BRANCHES table
SELECT * FROM BRANCHES;

// Create a copy of BRACNHES for
// testing DROP
CREATE TABLE BRANCHES_COPY as
SELECT * FROM BRANCHES;

DESC BRANCHES_COPY;
DROP TABLE BRANCHES_COPY;
DESC BRANCHES_COPY;

// Create a copy of BRACNHES for
// testing TRUNCATE
```

```
CREATE TABLE BRANCHES_COPY as

SELECT * FROM BRANCHES;


TRUNCATE TABLE BRANCHES_COPY;

DESC BRANCHES_copy;


SELECT * FROM BRANCHES_copy;
```