

Softwarica

College of IT & E-commerce

Module Code & Module Title

STW7086CEM Data Management Systems

Assessment Weightage & Type

30% Group Coursework

Year and Semester

2023-24 Spring

Students Name:

Ratish Ranabhat (240127) PART B,

Meghant Dangol (240119) PART C,

Kaushal Rijal (240234) PART D,

Ayush Rayamajhi (240178) PART A

Assignment Due Date: May 17, 2024

Assignment Submission Date: May 17, 2024

Acknowledgment

We would like to express our gratitude to Mr. Ritesh Singh and Mr. Siddhartha Neupane, as well as to Coventry University, for providing us with an opportunity to undertake this coursework and for their unwavering support and guidance, which enabled us to achieve the project successfully. We also extend our thanks to our college for providing us with a conducive workspace and module teachers for their invaluable assistance throughout this module. Comprehending and completing the coursework duly presented tremendous challenges for us, and we wouldn't have succeeded in it without the help of our friends and tutors. So, I would like to thank our teachers for providing us with all the guidelines, and also our friends who were by our side till the end of the coursework warned us about our mistakes and helped us fix those errors. Therefore, we are grateful to our teachers for providing us with precise guidelines, and to our friends who stood by us throughout the coursework completion, alerting us to our errors and offering assistance in rectifying them.

Abstract

The coursework was based on ER Modeling, Oracle SQL, sequential and distributed processing, Hadoop, Research-based reports, and MongoDB. The coursework was completed successfully with the help of Oracle SQL, Lucid Chart, SQL developer, Hadoop, and MongoDB. The coursework mainly deals with generating ER diagram from the given scenario, converting it into relational tables using Oracle, applying NoSQL solutions to the relational tables and applying parallel processing to the queries. The coursework mostly gives us a clear knowledge of the data modeling, relational tables, non-relational documents, SQL and NoSQL programming and parallel and sequential processing. The main aim of the coursework was to implement the ER diagram of the given problem, convert it into schema diagram and relational tables, generating SQL queries from those diagrams, implementing NoSQL solutions to the relational tables and applying parallel processing to those sequential queries. The coursework was finally completed and sums up whether the program is running smoothly without any errors according to the question.

Table of Contents

Chapter 1.	Part A: Entity Relationship Modeling.....	1
1.1	Overview of the Topic.....	1
1.1.1	Entities	1
1.1.2	Attributes.....	2
1.1.3	Relationships of the Entities	3
1.1.4	Entity Relationship Diagram.....	4
1.2	Relational Table	5
1.2.1	Departments Table	6
1.2.2	Faculty Table	6
1.2.3	Courses Table.....	7
1.2.4	Students Table.....	7
1.2.5	Campuses Table	7
1.2.6	Administrative_Staff Table.....	7
1.2.7	Faculty_Departments Table	8
1.2.8	Department_Administrative_Staff Table	8
1.2.9	Campus_Administrative_Staff Table.....	8
1.2.10	Faculty_Courses Table.....	9
1.2.11	Students_Courses Table.....	9
1.3	Schema Diagram	10
Chapter 2.	Part B: SQL Programming.....	12
2.1	SQL Statements to Create Tables.....	12
2.1.1	User Table	12
2.1.2	Category Table.....	13
2.1.3	Music Table	14
2.1.4	Download Music Table.....	15
2.2	SQL Statements for insertion of data in tables.....	16

2.2.1	Insertion of given data in the User table	16
2.2.2	Insertion of given data in the Category table	17
2.2.3	Insertion of given data in the Music table.....	18
2.2.4	Insertion of given data in the DownloadMusic table	19
2.3	SQL Select Statements	20
2.3.1	Select Statement for User Table	20
2.3.2	Select Statement for Music Table	20
2.3.3	Select Statement for Category Table	21
2.3.4	Select Statement for DownloadMusic Table	21
2.4	SQL Statements to Return the Data	22
2.4.1	SQL statement to return the musicId, the title, and the categoryCode of all the music in the database, ordered by title.....	22
2.4.2	SQL statement to return the number of users who downloaded ‘Pop-Rock’ category of music.....	23
2.4.3	SQL statement to return the number of music downloads for each of the categories. The result listing should include the titles of the categories and the number of music downloads for each category title.....	24
2.4.4	SQL statement to return the titles of the categories for which music was downloaded more than once	25
2.5	Addition of additional records and output of all the above SQL statements after additional records are inserted	26
Chapter 3.	Part C: Sequential and distributed processing	33
3.1	Introduction	33
3.2	SQL Solution.....	33
3.2.1	Table Creation.....	33
3.2.2	Data Insertion.....	34
3.2.3	Data Selection	35
3.2.4	Group By Query.....	36

3.3	Map Reduce Solution	37
3.3.1	Step 1: Map	37
3.3.2	Step 2: Shuffle and Sort	38
3.3.3	Step 3: Reduce	38
3.4	Map Reduce Diagram.....	40
3.5	Map Reduce Implementation in Oracle SQL.....	41
Chapter 4.	Part D: Research Report.....	43
4.1	Research Report for the Given Scenario	43
4.1.1	Introduction.....	43
4.1.2	Challenges with SQL(MySQL)	43
4.1.3	Advantages of SQL(MySQL)	44
4.1.4	Challenges with MongoDB (NoSQL)	44
4.1.5	Advantages of MongoDB (NoSQL)	45
4.1.6	Conclusion	45
4.2	MongoDB Queries	46
4.2.1	Database Creation	46
4.2.2	Data Insertion.....	46
4.2.3	Query to retrieve product where price is less than \$50.....	48
4.2.4	Query to update the price of the product with productid “P003” to \$75	48
4.2.5	Insert a new product.....	49
4.2.6	Delete all products from the “Fashion” category.....	50
Chapter 5.	Conclusion	51
Chapter 6.	References.....	52
Chapter 7.	Appendix.....	53
7.1	Appendix For Part A	53
7.1.1	All the SQL Queries for Part A.....	53
7.1.2	Schema diagrams created on SQL Developer	57

7.2	Appendix For Part B	58
7.2.1	All the SQL Queries for Part B.....	58
7.3	Appendix For Part C	65
7.3.1	All the SQL Queries for Part C.....	65
7.4	Appendix For Part D	69
7.4.1	All the MongoDB Queries for Part D	69

Table of Figures

Figure 1: Er Diagram	4
Figure 2: Schema Diagram	10
Figure 3: Creating User Table.....	12
Figure 4: Creating Category Table	13
Figure 5: Creating Music Table	14
Figure 6: Creating Download Music Table	15
Figure 7: Insertion of given data in the User table.....	16
Figure 8: Insertion of given data in the Category table	17
Figure 9: Insertion of given data in the Music table	18
Figure 10: Insertion of given data in the DownloadMusic table	19
Figure 11: Select Statement for User Table	20
Figure 12: Select Statement for Music Table	20
Figure 13: Select Statement for Category Table.....	21
Figure 14: Select Statement for DownloadMusic Table.....	21
Figure 15: SQL statement to return the musicId, the title, and the categoryCode of all the music in the database, ordered by title	22
Figure 16: SQL statement to return the number of users who downloaded 'Pop-Rock' category of music.....	23
Figure 17: SQL statement to return the number of music downloads for each of the categories. The result listing should include the titles of the categories and the number of music downloads for each category title.....	24
Figure 18: SQL statement to return the titles of the categories for which music was downloaded more than once.....	25
Figure 19: Addition of additional records in the User Table	26
Figure 20: Addition of additional records in the Category Table	26
Figure 21: Addition of additional records in the Music Table.....	27
Figure 22: Addition of additional records in the Download Music Table	28
Figure 23: Select Query after the addition of additional data for User Table.....	29
Figure 24: Select Query after addition of additional data for Category Table	29
Figure 25: Select Query after addition of additional data for Music Table	30
Figure 26: Select Query after the addition of additional data for DownloadMusic Table	30

Figure 27: Query to return the music ID, the title, and the categoryCode of all the music in the database, ordered by title after additional data is added	31
Figure 28: Query to return the number of users who downloaded the ‘Pop-Rock’ category of music after additional data is added.....	31
Figure 29: Query to return the number of music downloads for each of the categories after additional data is added.....	32
Figure 30: Query to return the titles of the categories for which music was downloaded more than once after additional data is added	32
Figure 31: SalesDataStore Table Creation.....	33
Figure 32: Data Insertion in SalesDataStore Table.....	34
Figure 33: Select Query for SalesDataStore Table	35
Figure 34: Group By Query	36
Figure 35: Map.....	37
Figure 36: Shuffle and Sort.....	38
Figure 37: Reduce	39
Figure 38: Map Reduce.....	40
Figure 39: Map Reduce Implementation in Oracle SQL	41
Figure 40: ShopNow Database Creation.....	46
Figure 41: Data Insertion in users’ Collection	46
Figure 42: Data Insertion in Products Collection	47
Figure 43: Query to retrieve product where price is less than \$50	48
Figure 44: Query to update the price of the product with productid “P003” to \$75.....	48
Figure 45: Query to find the product with productId “P003.....	49
Figure 46: Inserting New Product.....	49
Figure 47: Finding the Inserted Product	50
Figure 48: Deleting all products from the “Fashion” category.....	50
Figure 49: Finding the product from the "Fashion" category	50

Table of Tables

Table 1: Departments Table.....	6
Table 2: Faculty Table	6
Table 3: Courses Table	7
Table 4: Students Table	7
Table 5: Campuses Table.....	7
Table 6: Administrative_Staff Table	7
Table 7: Faculty_Departments Table.....	8
Table 8: Department_Administrative_Staff Table.....	8
Table 9: Campus_Administrative_Staff Table	8
Table 10: Faculty_Courses Table	9
Table 11: Students_Courses Table.....	9

Table of Acronyms

SN	ACRONYMS	FULLFORMS
1	ER	Entity Relationship
2	ERD	Entity Relationship Diagram
3	SQL	Structured Query Language
4	JSON	JavaScript Object Notation
5	RDBMS	Relational Database Management System
6	ACID	Atomicity Consistency Integrity Durability
7	BASE	Basically available Soft state Eventually consistent
8	BSON	Binary JavaScript Object Notation

Chapter 1. Part A: Entity Relationship Modeling

1.1 Overview of the Topic

The university is implementing an integrated database management system to enhance efficiency in its academic and administrative activities. The system is expected to consist of multiple essential elements, embarking with the departments, each determined by a unique ID, location, and name. several courses, each identified by a course ID, credit hours, and title. Faculty members will be vital, each identified by their unique ID, name, and academic rank. They will be associated with one or more departments according to their areas of specialization and assigned teaching responsibilities. These faculty staff will be teaching a variety of courses in several semesters. Likewise, students will be enrolled in classes each semester and obtain grades ranging from A to F, each student will be identified by their student ID, name, and major. The university offers courses that students can enroll in interchangeably throughout its several campuses, each identified by their ID, name, and address. Every campus will have its administrative staff, identified by their ID, name, and job titles, accountable for the operations on campuses. Depending on their roles, these staff may work in particular departments or campuses, making sure that administrative duties are completed efficiently. Nonetheless, the database management system will efficiently manage an intricate hierarchy of administrative and academic functions to guarantee smooth operation across university departments and campuses.

1.1.1 Entities

All of the required Entities were abstracted from the given scenario. Entities are listed below.

1. Departments
2. Courses
3. Faculty
4. Students
5. Campuses
6. Administrative_Staff

1.1.2 Attributes

All of the required Attributes of corresponding Entities were extracted from the given scenario. Attributes of corresponding Entities are listed below.

1. Departments:

a) Attributes:

- Department_ID (Primary Key)
- Name
- Location

2. Courses:

a) Attributes:

- Course_ID (Primary Key)
- Title
- Credit_Hours

3. Faculty:

a) Attributes:

- Faculty_ID (Primary Key)
- Name
- Academic_Rank

4. Students:

a) Attributes:

- Student_ID (Primary Key)
- Name
- Major

5. Campuses:

a) Attributes:

- Campus_ID (Primary Key)
- Name
- Address

6. Administrative_Staff:

a) Attributes:

- Staff_ID (Primary Key)
- Name
- Job_Title

1.1.3 Relationships of the Entities

All the relationships, cardinalities, and natures are extracted from the given scenarios.

All the relationships with their cardinalities are as follows:

1. Faculty_Departments:

- Nature: Many to Many
- Cardinality: Many departments can have many faculty members, and many faculty members can be associated with many departments.
- Mandatory or Optional: Mandatory (Faculty members should be affiliated with at least one department).

2. Faculty_Courses:

- Nature: Many to Many
- Cardinality: Many faculty members can teach many courses, and many courses can be taught by many faculty members.
- Mandatory or Optional: Mandatory (Faculty Members should teach at least one course)

3. Students_Courses:

- Nature: Many to Many
- Cardinality: Many courses can have many students enrolled and many students can enroll in many courses.
- Mandatory or Optional: Mandatory (For students to progress in their academic programs, it is necessary to enroll in courses).

4. Campus_Administrative_Staff:

- Nature: Many to Many
- Cardinality: Many staff members can work on many campuses, and many campuses can have many administrative staff members.
- Mandatory or Optional: Optional (Depending on the administrative needs, the placement of administrative staff to the campuses may vary).

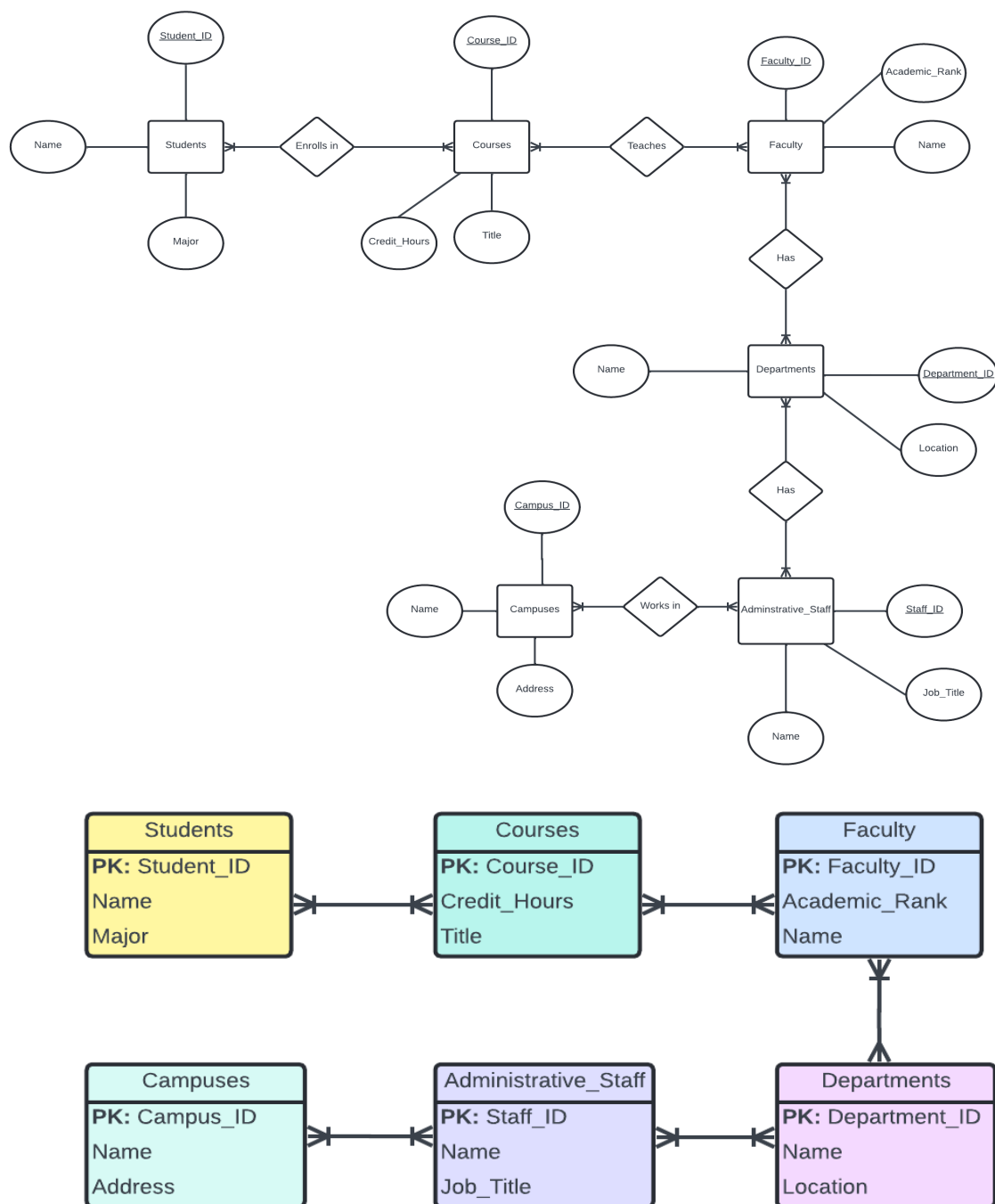
5. Department_Administrative_Staff:

- Nature: Many to Many
- Cardinality: Many staff members can work in many departments, and many departments can have many administrative staff members.
- Mandatory or Optional: Optional (Depending on the needs of the administration, staff members may be assigned to each department.).

1.1.4 Entity Relationship Diagram

All the entities and relationships previously were used to build the ER diagrams below:

Figure 1: Er Diagram



The ER diagram above represents the university's requirements. In this diagram, at least one student enrolls in one or many courses. At least one or more faculty members teach one or many courses a semester. A faculty member belongs to at least one or more departments and a department has at least one or more faculty members. A university has multiple campuses that offer one or many courses. Every campus has at least one or many administrative staff and those administrative staff can also be assigned to many departments. (Ralph Kimball, 2002)

1.2 Relational Table

These tables have been developed based on the entities and relationships in a given scenario's conceptual model. How each table was made and the transformation rules applied are explained as follows. Entity to Table transformation rule is used, where each entity in a conceptual model is transformed into a table in a relational model. The 'Departments' table represents the "Departments" entity, which includes attributes such as Department_ID, Name, and Location. Similarly, the 'Courses' table represents the "Courses" entity and contains attributes such as Course_ID, Title, and Credit_Hours. The "Faculty" entity is transformed into a 'Faculty' table with the attributes Faculty_ID, Name, and Academic_Rank. The "Students" entity is transformed into a 'Students' table with attributes Student_ID, Name, and Major. The "Campuses" entity is transformed into a 'Campuses' table, with the attributes Campus_ID, Name, and Address. The "Administrative_Staff" entity is transformed into an 'Administrative_Staff' table, with the attributes Staff_ID, Name, and Job_Title.

In addition, for the tables that portray relationships. The Faculty_Departments table conveys the many-to-many relationships between Faculty and Departments. A Many to Many Relationships to Associative Table transformation rule is used in this instance. It consists of foreign keys referencing the Faculty and Departments tables. The Faculty_Courses table conveys the many-to-many relationships between Faculty and Courses. A Many to Many Relationships to Associative Table transformation rule is used in this instance. It consists of foreign keys referencing the Faculty and Courses tables. It also consists of one additional attribute Semester specifying when the course is taught. Students_Courses table conveys the many-to-many relationships between Students and Courses. A Many to Many Relationships to Associative Table transformation rule is used in this instance. It consists of foreign keys referencing the Students and Courses tables.

It also consists of one additional attribute grade specifying the obtained grade by the students for a particular semester.

Campus_Administrative_Staff table conveys the many-to-many relationships between Campuses and Administrative_Staffs. A Many to Many Relationships to Associative Table transformation rule is used in this instance. It consists of foreign keys referencing the Campuses and Administrative_Staffs tables.

Department_Administrative_Staff table conveys the many-to-many relationships between Departments and Administrative_Staffs. A Many to Many Relationships to Associative Table transformation rule is used in this instance. It consists of foreign keys referencing the Departments and Administrative_Staffs tables.

The following tables shall indicate all the tables with the appropriate attributes, data types, and identifiers such as Primary Key and Foreign Key used for creating ER and Schema Diagrams.

1.2.1 Departments Table

Table 1: Departments Table

Attributes	Data Type	Constraints	Nullable
Department_ID	Number	Primary Key	False
Name	Varchar2(50)		False
Location	Varchar2(100)		False

1.2.2 Faculty Table

Table 2: Faculty Table

Attributes	Data Type	Constraints	Nullable
Faculty_ID	Number	Primary Key	False
Name	Varchar2(100)		False
Academic_Rank	Varchar2(50)		False

1.2.3 Courses Table

Table 3: Courses Table

Attributes	Data Type	Constraints	Nullable
Course_ID	Number	Primary Key	False
Credit_Hour	Number		False
Title	Varchar2(100)		False

1.2.4 Students Table

Table 4: Students Table

Attributes	Data Type	Constraints	Nullable
Student_ID	Number	Primary Key	False
Name	Varchar2(100)		False
Major	Varchar2(100)		True

1.2.5 Campuses Table

Table 5: Campuses Table

Attributes	Data Type	Constraints	Nullable
Campus_ID	Number	Primary Key	False
Name	Varchar2(100)		False
Address	Varchar2(255)		False

1.2.6 Administrative_Staff Table

Table 6: Administrative_Staff Table

Attributes	Data Type	Constraints	Nullable
Staff_ID	Number	Primary Key	False
Name	Varchar2(100)		False
Job_Title	Varchar2(100)		False

1.2.7 Faculty_Departments Table

Table 7: Faculty_Departments Table

Attributes	Data Type	Constraints	Nullable
Faculty_ID	Number	Primary Key Foreign Key	False
Department_ID	Number	Primary Key Foreign Key	False

1.2.8 Department_Administrative_Staff Table

Table 8: Department_Administrative_Staff Table

Attributes	Data Type	Constraints	Nullable
Department_ID	Number	Primary Key Foreign Key	False
Staff_ID	Number	Primary Key Foreign Key	False

1.2.9 Campus_Administrative_Staff Table

Table 9: Campus_Administrative_Staff Table

Attributes	Data Type	Constraints	Nullable
Campus_ID	Number	Primary Key Foreign Key	False
Staff_ID	Number	Primary Key Foreign Key	False

1.2.10 Faculty_Courses Table

Table 10: Faculty_Courses Table

Attributes	Data Type	Constraints	Nullable
Faculty_ID	Number	Primary Key Foreign Key	False
Course_ID	Number	Primary Key Foreign Key	False
Semester	Number	Primary Key	False

1.2.11 Students_Courses Table

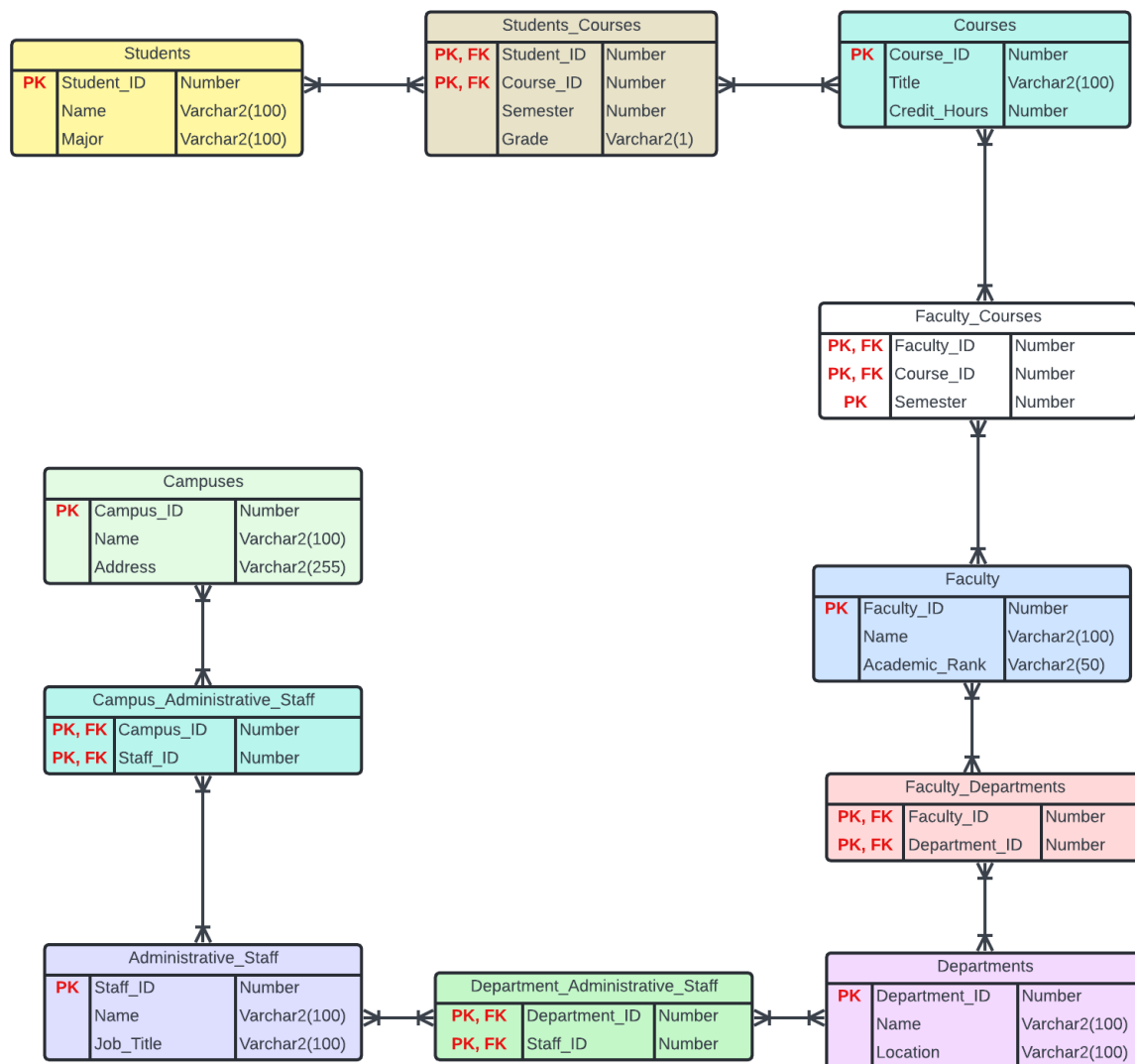
Table 11: Students_Courses Table

Attributes	Data Type	Constraints	Nullable
Student_ID	Number	Primary Key Foreign Key	False
Course_ID	Number	Primary Key Foreign Key	False
Semester	Number	Primary Key	False
Grade	Varchar2(2)		False

1.3 Schema Diagram

The schema diagram provides a detailed and normalized view of ERD. In the schema diagram, we show tables that would normally be hidden in ERD and tables created after normalizing the data. Below is the schema diagram of this University Management System:

Figure 2: Schema Diagram



In this schema diagram we can see that the first we can see Campus_Administrative_Staff table which connects the administrative staff table with the campus table, the primary keys of administrative staff and campus are stored in this table. Another table is Department_Administrative_Staff which connects administrative staff with the department and stores the primary key of the department and administrative staff.

We have also added a Faculty_Departments table to store which faculty member belongs in which departments. The faculty member stores the primary keys of the faculty and department table. We have also linked the faculty table to the course table using a faculty course table where the faculty ID of a faculty member, the course ID of the course and the semester in which the faculty member is teaching the course are stored.

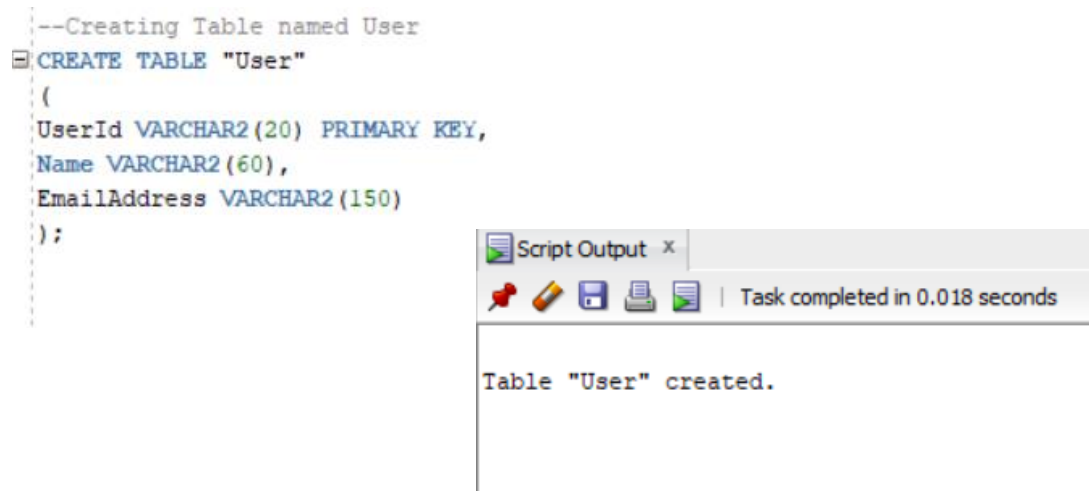
The Students_Courses table links the student table and course table, it holds the primary key of both the tables as foreign keys and stores the semester the student is enrolled in. This table also stores the grade of the student which the student received in a certain course in a semester. The grade ranges from A to F.

Chapter 2. Part B: SQL Programming

2.1 SQL Statements to Create Tables

2.1.1 User Table

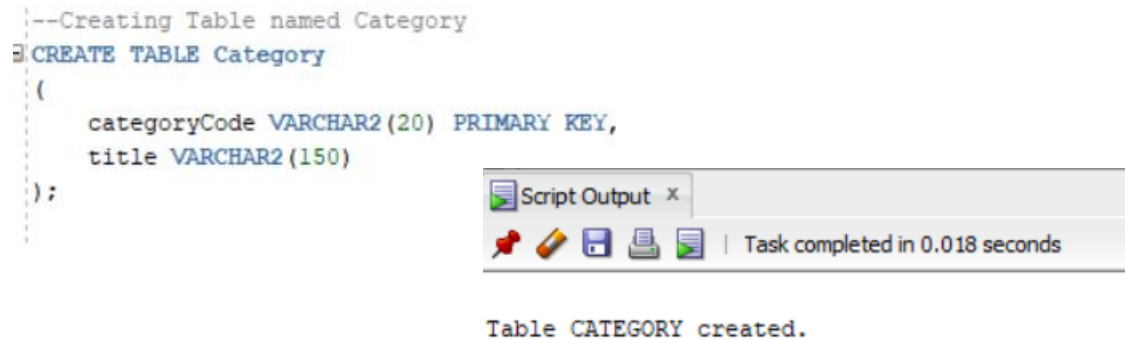
Figure 3: Creating User Table



To create a user table, we have used a create table statement, in this table UserId is a primary key which is of varchar data type, since the maximum length of UserId provided in the data was less than 20 we have used a length of 20 for UserId. The name attribute is used to store the name of the user is of varchar type and has a length of 60, the name attribute stores both the first name and last name of users, so we have given it a length of 60 characters. The EmailAddress attribute stores the email of the user we have used Varchar to store the email as email is always a string, and have given it a length of 150 characters as the email of some users can be long, the email also has unique constraint as we only store unique emails in this database.

2.1.2 Category Table

Figure 4: Creating Category Table

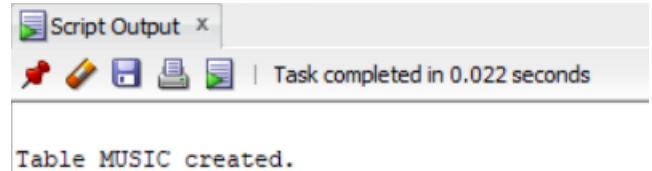


To create a Category table, we have used the create table statement, this table has only two attributes which are the categoryCode which is the primary key of this table and stores the code of a category. This table also has a title attribute to store the title of a category, it is of varchar type and can store up to 150 characters.

2.1.3 Music Table

Figure 5: Creating Music Table

```
--Creating Table named Music  
CREATE TABLE Music (  
    musicId VARCHAR2(20) PRIMARY KEY,  
    title VARCHAR2(150),  
    categoryCode VARCHAR2(20),  
    costPerDownload NUMBER(20, 2),  
    FOREIGN KEY (categoryCode) REFERENCES Category (categoryCode)  
);
```

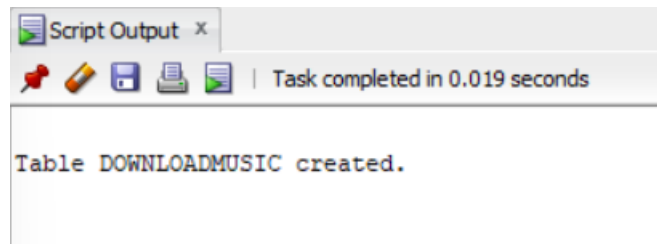


A Music table is a table to stores data related to music, this table consists of four attributes which are musicId which is the primary key for this table, title to store the title of the music, categoryCode to store the category of the table which is also the foreign key of category table and it links category table with music table and costPerDownload attribute which is a number which shows the amount required to download any music, we have used two decimal places to store costPerDownload of any music.

2.1.4 Download Music Table

Figure 6: Creating Download Music Table

```
--Creating Table named DownloadMusic  
CREATE TABLE DownloadMusic  
(  
    UserID VARCHAR2(10),  
    musicId VARCHAR2(10),  
    downloadDate DATE,  
    FOREIGN KEY (UserID) REFERENCES "User"(UserId),  
    FOREIGN KEY (musicId) REFERENCES Music(musicId),  
    PRIMARY KEY (UserId, musicId)  
);
```



Download music table is used to store data of users who have downloaded music. This table has three attributes which are uesrId, musicId, and downloadDate. UserId and musicId are the foreign keys that references this table to the user and music table respectively. The primary key of this table is made up of two foreign keys which are UserId and musicId. The downloadDate is of date type which stores the date at which a music was downloaded by a user.

2.2 SQL Statements for insertion of data in tables

In the query below we have inserted data in all the tables created above, without violating any constraints. The screenshots below show the execution and result of the insert statements.

2.2.1 Insertion of given data in the User table

Figure 7: Insertion of given data in the User table

```
-- Inserting given data into User table
INSERT INTO "User" (userId, name, emailAddress)
VALUES ('wayne10', 'Wayne, R', 'wayne@hotmail.co.uk');

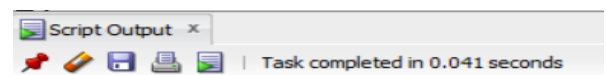
INSERT INTO "User" (userId, name, emailAddress)
VALUES ('santos17', 'Santos, C', 'santos@ntl.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('hary05', 'Hary, M', 'hary@freeserve.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('margot9', 'Margot, C', 'margot9@msn.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('mount77', 'Mount, M', 'mount@hotmail.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('nancy91', 'Nancy, L', 'nancy91@tesco.co.uk');
```



1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

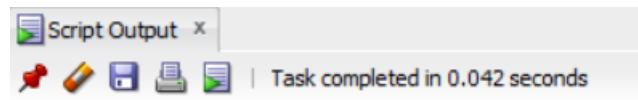
2.2.2 Insertion of given data in the Category table

Figure 8: Insertion of given data in the Category table

```
-- Inserting data into Category table
INSERT INTO Category (categoryCode, title)
VALUES ('M011', 'Love');

INSERT INTO Category (categoryCode, title)
VALUES ('M012', 'Pop');

INSERT INTO Category (categoryCode, title)
VALUES ('M013', 'Rock');
```



1 row inserted.

1 row inserted.

1 row inserted.

2.2.3 Insertion of given data in the Music table

Figure 9: Insertion of given data in the Music table

```
-- Inserting data into Music table
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M001', 'Born to run', 'M013', 0.99);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M002', 'Crawling', 'M013', 1.99);

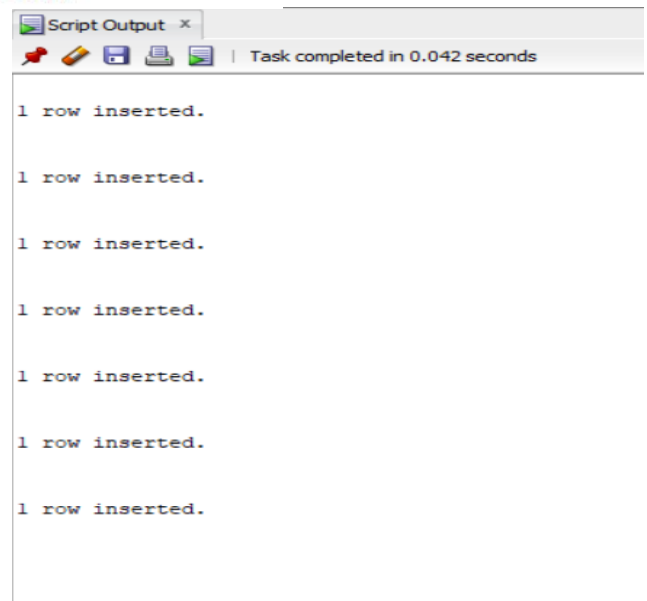
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M003', 'You're Beautiful', 'M011', 1.49);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M004', 'Summer of 69', 'M011', 1.79);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M005', 'Crazy Train', 'M013', 1.50);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M006', 'Yellow Submarine', 'M012', 1.10);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M007', 'Got to be there', 'M012', 0.89);
```



2.2.4 Insertion of given data in the DownloadMusic table

Figure 10: Insertion of given data in the DownloadMusic table

```
-- Inserting data into DownloadMusic table
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('wayne10', 'M002', TO_DATE('03-May-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('margot9', 'M005', TO_DATE('01-May-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('santos17', 'M002', TO_DATE('06-May-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('margot9', 'M001', TO_DATE('06-May-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('wayne10', 'M003', TO_DATE('01-Aug-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('mount77', 'M004', TO_DATE('02-Aug-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('nancy91', 'M007', TO_DATE('05-Sep-2021', 'DD-MON-YYYY'));
```

Script Output - x | Task completed in 0.044 seconds

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

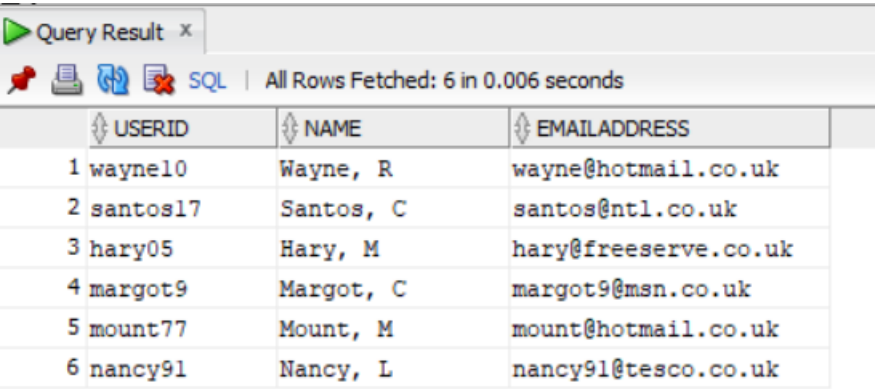
2.3 SQL Select Statements

In the queries below we have selected data from all the tables to validate if all the data are inserted correctly.

2.3.1 Select Statement for User Table

Figure 11: Select Statement for User Table

```
SELECT * FROM "User";
```

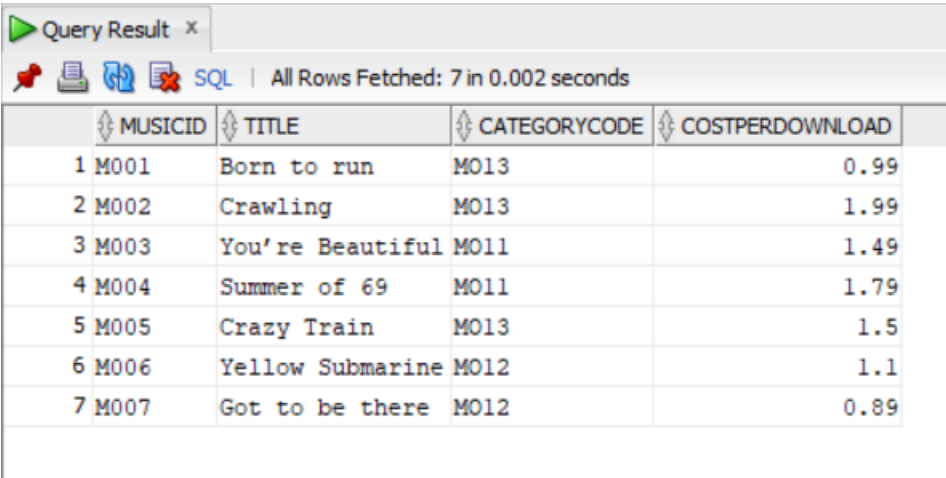


	USERID	NAME	EMAILADDRESS
1	wayne10	Wayne, R	wayne@hotmail.co.uk
2	santos17	Santos, C	santos@ntl.co.uk
3	hary05	Hary, M	hary@freeserve.co.uk
4	margot9	Margot, C	margot9@msn.co.uk
5	mount77	Mount, M	mount@hotmail.co.uk
6	nancy91	Nancy, L	nancy91@tesco.co.uk

2.3.2 Select Statement for Music Table

Figure 12: Select Statement for Music Table

```
SELECT * FROM Music;
```

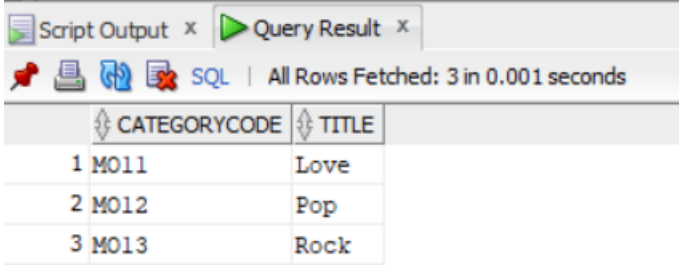


	MUSICID	TITLE	CATEGORYCODE	COSTPERDOWNLOAD
1	M001	Born to run	MO13	0.99
2	M002	Crawling	MO13	1.99
3	M003	You're Beautiful	MO11	1.49
4	M004	Summer of 69	MO11	1.79
5	M005	Crazy Train	MO13	1.5
6	M006	Yellow Submarine	MO12	1.1
7	M007	Got to be there	MO12	0.89

2.3.3 Select Statement for Category Table

Figure 13: Select Statement for Category Table

```
SELECT * FROM category;
```

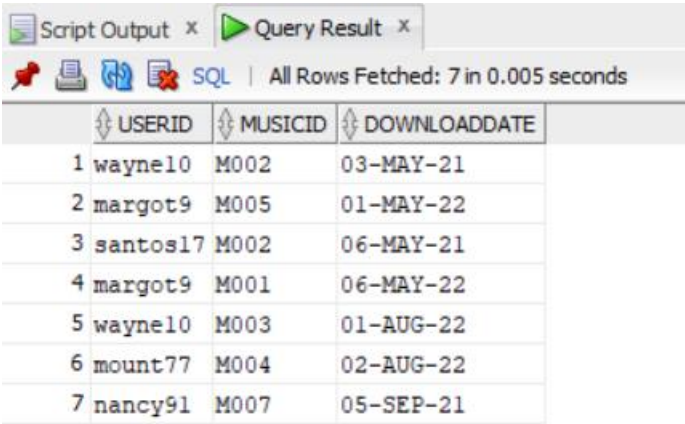


	CATEGORYCODE	TITLE
1	M011	Love
2	M012	Pop
3	M013	Rock

2.3.4 Select Statement for DownloadMusic Table

Figure 14: Select Statement for DownloadMusic Table

```
SELECT * FROM downloadmusic;
```



	USERID	MUSICID	DOWNLOADDATE
1	wayne10	M002	03-MAY-21
2	margot9	M005	01-MAY-22
3	santos17	M002	06-MAY-21
4	margot9	M001	06-MAY-22
5	wayne10	M003	01-AUG-22
6	mount77	M004	02-AUG-22
7	nancy91	M007	05-SEP-21

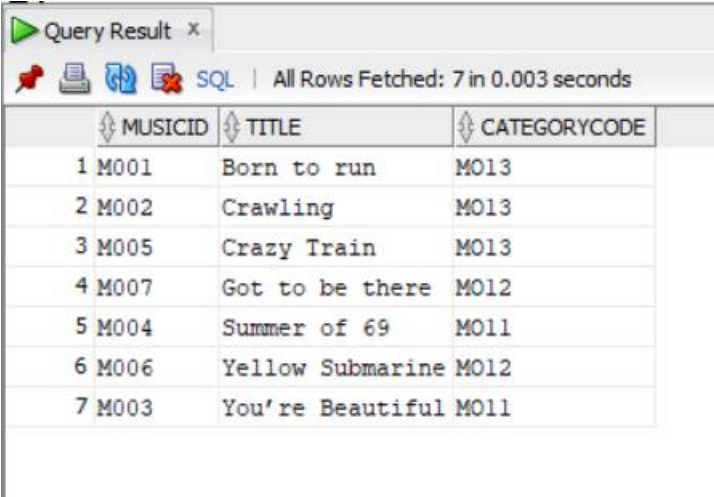
2.4 SQL Statements to Return the Data

2.4.1 SQL statement to return the musicId, the title, and the categoryCode of all the music in the database, ordered by title

The query below shows an SQL statement to return the musicId, title, and categoryCode of music stored in this database. In this query, firstly we have used a SELECT statement to select from the Music table and then we have projected musicId, title, and categoryCode attributes from the Music table. After projecting the required attributes, we used the ORDER BY keyword to order the results by their title in ascending order.

Figure 15: SQL statement to return the musicId, the title, and the categoryCode of all the music in the database, ordered by title

```
SELECT musicId, title, categoryCode  
FROM Music  
ORDER BY title;
```



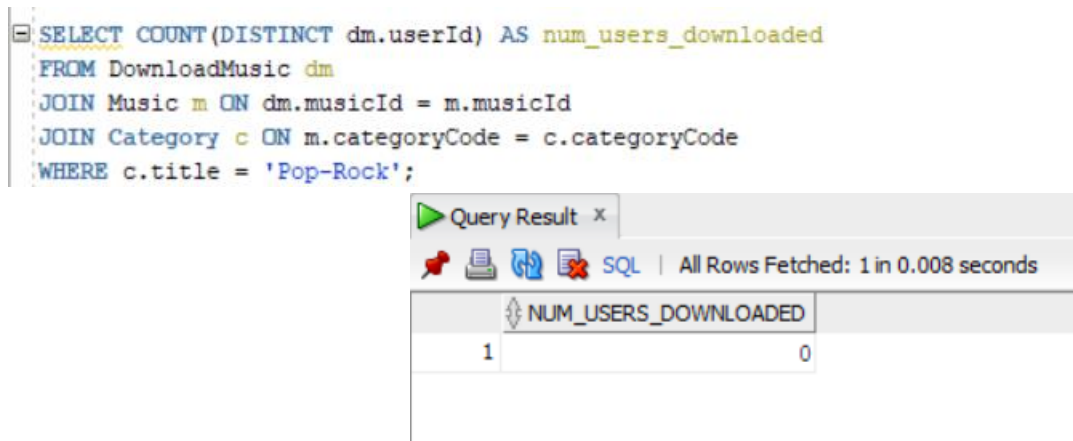
	MUSICID	TITLE	CATEGORYCODE
1	M001	Born to run	M013
2	M002	Crawling	M013
3	M005	Crazy Train	M013
4	M007	Got to be there	M012
5	M004	Summer of 69	M011
6	M006	Yellow Submarine	M012
7	M003	You're Beautiful	M011

In the output above we can see that we have all the data in the music table as a result and the results are also ordered by title in ascending order.

2.4.2 SQL statement to return the number of users who downloaded 'Pop-Rock' category of music

In the query below we have used a SELECT statement to show the number of users who have downloaded the 'Pop-Rock' category of music. In this query, we have used the SELECT statement to select data from the DownloadMusic table and inner joined that result to the Music table using musicId which is the primary key of the Music table and is being used as the foreign key in DownloadMusic table. We have then joined the Music table with the Category table using categoryCode which is the primary key of the Category table and is being used by a foreign key in the Music table. After selecting this data, we used the COUNT and DISTINCT keywords to count the unique userId of the DownloadMusic table who have downloaded any music. At the end of this query, we have used the WHERE keyword to filter the records to records that only include the 'Pop-Rock' category.

Figure 16: SQL statement to return the number of users who downloaded 'Pop-Rock' category of music

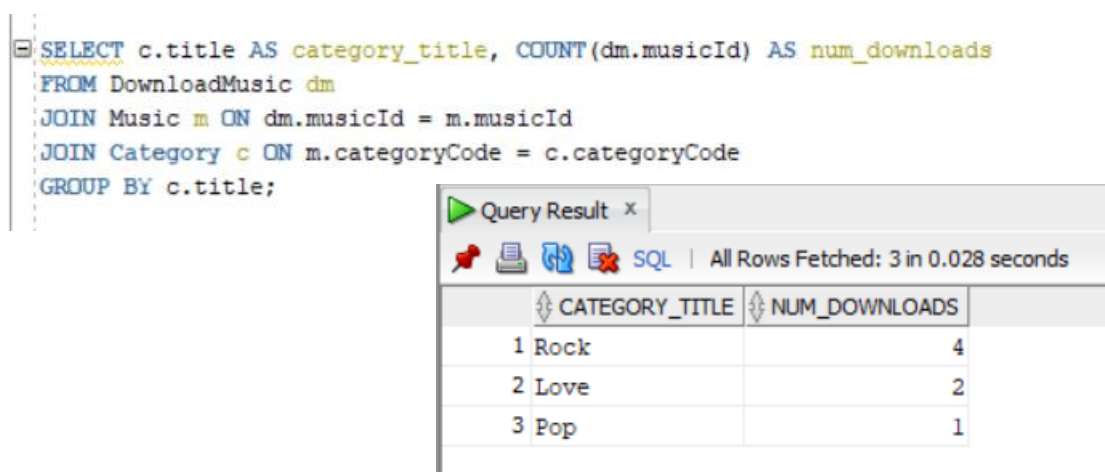


In the output above we can see the total number of users who have downloaded the 'Pop-Rock' category.

2.4.3 SQL statement to return the number of music downloads for each of the categories. The result listing should include the titles of the categories and the number of music downloads for each category title

In the query below we have used a SELECT statement to count the number of music downloads for each category. I have used the SELECT statement to select records from the DownloadMusic table and then used inner join to join that data with the Music table based on musicId which is the primary key of the Music table and is being used as a foreign key in DownloadMusic table. We used inner join again to join that data with the Category table based on categoryCode as categoryCode is the primary key of the Category table and is being used as a foreign key in the Music table. After joining these three tables we grouped the records on the title of the Category table to aggregate based on the title of the category. Finally, we have projected the title of the category table as category_title and have counted the music download of each category by using the COUNT keyword on the musicId attribute of the DownloadMusic table.

Figure 17: SQL statement to return the number of music downloads for each of the categories. The result listing should include the titles of the categories and the number of music downloads for each category title



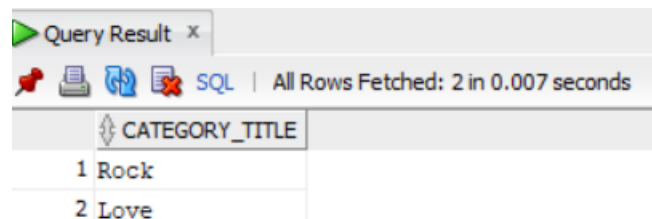
The output above shows the category_title and the total number of downloads for each category which is the required output.

2.4.4 SQL statement to return the titles of the categories for which music was downloaded more than once

In the query below we have used a select statement to return the titles of categories which was downloaded more than once. We first used SELECT statement on Category to get the title of the category. We have then used inner join to join the Category table with the Music table based on categoryCode as categoryCode is being used as a primary key in the Category table and a foreign key in the Music table. We have then used inner join again to join the DownloadMusic table with the Music table based on the musicId attribute as this attribute is being used as a primary key in the Music table and a foreign key in the DownloadMusic table. We have then grouped the results by title of Category table, to aggregate the results on category_title. Finally, we have used the HAVING keyword to filter aggregated results that were downloaded more than once. We have also projected the title of the Category table in this query as category_title.

Figure 18: SQL statement to return the titles of the categories for which music was downloaded more than once

```
SELECT c.title AS category_title
FROM Category c
JOIN Music m ON c.categoryCode = m.categoryCode
JOIN DownloadMusic dm ON m.musicId = dm.musicId
GROUP BY c.title
HAVING COUNT(dm.musicId) > 1;
```



CATEGORY_TITLE
1 Rock
2 Love

2.5 Addition of additional records and output of all the above SQL statements after additional records are inserted

Figure 19: Addition of additional records in the User Table

```
-- Inserting more data into User table
INSERT INTO "User" (userId, name, emailAddress)
VALUES ('Kaushal100', 'Kaushal, RI', 'Kaushal100@hotmail.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('Ratish3254', 'Ratish, RB', 'Ratish3254@gmail.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('Meghant200', 'Meghant, D', 'Meghant200@freemove.co.uk');

INSERT INTO "User" (userId, name, emailAddress)
VALUES ('Aayush200', 'Aayush, M', 'Aayush200@msn.co.uk');
```



Script Output x
Task completed in 0.038 seconds

```
1 row inserted.

1 row inserted.

1 row inserted.

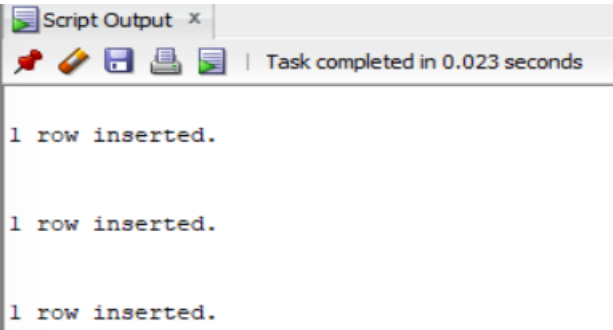
1 row inserted.
```

Figure 20: Addition of additional records in the Category Table

```
-- Inserting more data into Category table
INSERT INTO Category (categoryCode, title)
VALUES ('MO14', 'Reggae');

INSERT INTO Category (categoryCode, title)
VALUES ('MO15', 'Jazz');

INSERT INTO Category (categoryCode, title)
VALUES ('MO16', 'Metal');
```



Script Output x
Task completed in 0.023 seconds

```
1 row inserted.

1 row inserted.

1 row inserted.
```

Figure 21: Addition of additional records in the Music Table

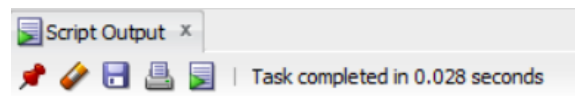
```
-- Inserting more into Music table
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M008', 'Three Little Birds', 'M014', 1.98);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M009', 'The Lazy Song', 'M014', 0.99);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M010', 'So Beautiful', 'M015', 1.49);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M011', 'Master of Puppets', 'M016', 1.50);

INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M012', 'Nothing Else Matters', 'M016', 1.45);
```



1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Figure 22: Addition of additional records in the Download Music Table

```
-- Inserting more data into DownloadMusic table
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Kaushal100', 'M008', TO_DATE('03-Jul-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Ratish3254', 'M012', TO_DATE('01-May-2023', 'DD-MON-YYYY'));

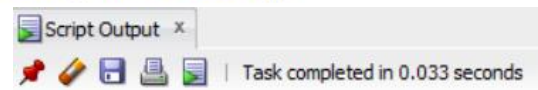
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Meghant200', 'M010', TO_DATE('06-Aug-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Aayush200', 'M011', TO_DATE('06-Sep-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Ratish3254', 'M009', TO_DATE('01-Aug-2023', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Meghant200', 'M012', TO_DATE('02-Aug-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Aayush200', 'M008', TO_DATE('05-Sep-2020', 'DD-MON-YYYY'));
```



1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

Figure 23: Select Query after the addition of additional data for User Table

```
select * From "User";
```

Query Result x

All Rows Fetched: 10 in 0.008 seconds

	USERID	NAME	EMAILADDRESS
1	wayne10	Wayne, R	wayne@hotmail.co.uk
2	santos17	Santos, C	santos@ntl.co.uk
3	hary05	Hary, M	hary@freeserve.co.uk
4	margot9	Margot, C	margot9@msn.co.uk
5	mount77	Mount, M	mount@hotmail.co.uk
6	nancy91	Nancy, L	nancy91@tesco.co.uk
7	Kaushal100	Kaushal, RI	Kaushal100@hotmail.co.uk
8	Ratish3254	Ratish, RB	Ratish3254@gmail.co.uk
9	Meghant200	Meghant, D	Meghant200@freeserve.co.uk
10	Aayush200	Aayush, M	Aayush200@msn.co.uk

Figure 24: Select Query after addition of additional data for Category Table

```
SELECT * From category;
```

Query Result x

All Rows Fetched: 6 in 0.039 seconds

	CATEGORYCODE	TITLE
1	M011	Love
2	M012	Pop
3	M013	Rock
4	M014	Reggae
5	M015	Jazz
6	M016	Metal

Figure 25: Select Query after addition of additional data for Music Table

```
select * From Music;
```

Query Result x

SQL | All Rows Fetched: 12 in 0.007 seconds

	MUSICID	TITLE	CATEGORYCODE	COSTPERDOWNLOAD
1	M001	Born to run	M013	0.99
2	M002	Crawling	M013	1.99
3	M003	You're Beautiful	M011	1.49
4	M004	Summer of 69	M011	1.79
5	M005	Crazy Train	M013	1.5
6	M006	Yellow Submarine	M012	1.1
7	M007	Got to be there	M012	0.89
8	M008	Three Little Birds	M014	1.98
9	M009	The Lazy Song	M014	0.99
10	M010	So Beautiful	M015	1.49
11	M011	Master of Puppets	M016	1.5
12	M012	Nothing Else Matters	M016	1.45

Figure 26: Select Query after the addition of additional data for DownloadMusic Table

```
select * From downloadmusic;
```


Query Result x

SQL | All Rows Fetched: 14 in 0.004 seconds

	USERID	MUSICID	DOWNLOADDATE
1	wayne10	M002	03-MAY-21
2	margot9	M005	01-MAY-22
3	santos17	M002	06-MAY-21
4	margot9	M001	06-MAY-22
5	wayne10	M003	01-AUG-22
6	mount77	M004	02-AUG-22
7	nancy91	M007	05-SEP-21
8	Kaushal100	M008	03-JUL-22
9	Ratish3254	M012	01-MAY-23
10	Meghant200	M010	06-AUG-21
11	Aayush200	M011	06-SEP-22
12	Ratish3254	M009	01-AUG-23
13	Meghant200	M012	02-AUG-21
14	Aayush200	M008	05-SEP-20

Figure 27: Query to return the music ID, the title, and the categoryCode of all the music in the database, ordered by title after additional data is added

```
SELECT musicId, title, categoryCode
FROM Music
ORDER BY title;
```



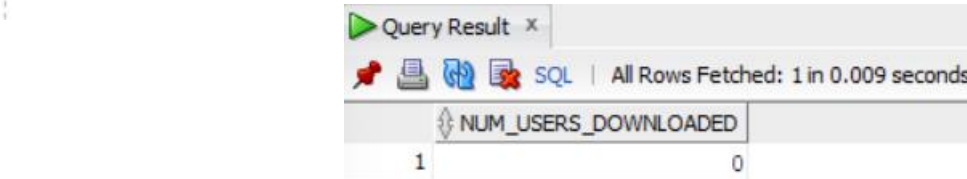
Query Result x

SQL | All Rows Fetched: 12 in 0.004 seconds

	MUSICID	TITLE	CATEGORYCODE
1	M001	Born to run	M013
2	M002	Crawling	M013
3	M005	Crazy Train	M013
4	M007	Got to be there	M012
5	M011	Master of Puppets	M016
6	M012	Nothing Else Matters	M016
7	M010	So Beautiful	M015
8	M004	Summer of 69	M011
9	M009	The Lazy Song	M014
10	M008	Three Little Birds	M014
11	M006	Yellow Submarine	M012
12	M003	You're Beautiful	M011

Figure 28: Query to return the number of users who downloaded the 'Pop-Rock' category of music after additional data is added

```
SELECT COUNT(DISTINCT dm.userId) AS num_users_downloaded
FROM DownloadMusic dm
JOIN Music m ON dm.musicId = m.musicId
JOIN Category c ON m.categoryCode = c.categoryCode
WHERE c.title = 'Pop-Rock';
```



Query Result x

SQL | All Rows Fetched: 1 in 0.009 seconds

NUM_USERS_DOWNLOADED
1

Figure 29: Query to return the number of music downloads for each of the categories after additional data is added

```

SELECT c.title AS category_title, COUNT(dm.musicId) AS num_downloads
FROM DownloadMusic dm
JOIN Music m ON dm.musicId = m.musicId
JOIN Category c ON m.categoryCode = c.categoryCode
GROUP BY c.title;

```

Query Result x

SQL | All Rows Fetched: 6 in 0.018 seconds

	CATEGORY_TITLE	NUM_DOWNLOADS
1	Reggae	3
2	Metal	3
3	Jazz	1
4	Rock	4
5	Love	2
6	Pop	1

Figure 30: Query to return the titles of the categories for which music was downloaded more than once after additional data is added

```

SELECT c.title AS category_title
FROM Category c
JOIN Music m ON c.categoryCode = m.categoryCode
JOIN DownloadMusic dm ON m.musicId = dm.musicId
GROUP BY c.title
HAVING COUNT(dm.musicId) > 1;

```

Query Result x

SQL | All Rows Fetched: 4 in 0.007 seconds

	CATEGORY_TITLE
1	Reggae
2	Metal
3	Rock
4	Love

Chapter 3. Part C: Sequential and distributed processing

3.1 Introduction

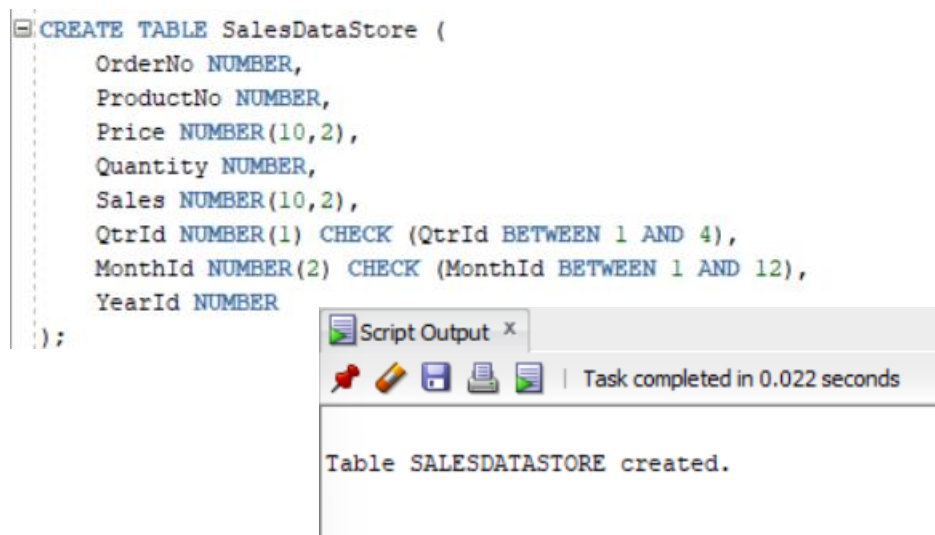
The sales data of a company contains eight attributes: orderId, productNo, price, quantity, sales, qtrId, monthId, and yearId. Together, these attributes indicate which items were sold in an order at a given time, aiding in the management of company transactions. Using this data, the company's managers seek to determine the number of products sold in each month of each year.

3.2 SQL Solution

To solve the problem given by the managers, we first need to create a table which stores the data based on those eight attributes.

3.2.1 Table Creation

Figure 31: SalesDataStore Table Creation



Above is the query to create a table SalesDataStore, the table has eight attributes as described in the scenario. The OrderNo attribute stores the order number of any product, ProductNo stores and identifies a product, and Price stores the price of the product we have assigned a decimal value of two places in the Price attribute because prices of any product in this company could have at a most decimal value up to two places, Quantity stores the amount of product sold, Sales stores the total price of the product sold we have assigned a decimal value of two places in Sales attribute because prices of any product in this company could have at most decimal value up to two places, QtrId stores the quarter in which product was sold, there is also a CHECK constraint in QtrId which checks if the value of quarter is between 1 and 4 as there are only four quarters in any

year. MonthId stores the month in which the product was sold, there is another CHECK constraint in MonthId which checks if the month is between 1 and 12 as there are only twelve months in any year. YearId stores the year in which the product was sold.

3.2.2 Data Insertion

Figure 32: Data Insertion in SalesDataStore Table

```
-- Inserting given sample data into the SalesData table
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10107, 2, 85.7, 30, 2587, 1, 2, 2003);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10107, 5, 95.8, 39, 3879.49, 1, 2, 2003);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10121, 5, 71.5, 34, 2700, 1, 2, 2003);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10134, 2, 94.74, 41, 3884.34, 3, 7, 2004);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10134, 5, 100, 27, 3307.77, 3, 7, 2004);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10159, 14, 100, 49, 5205.27, 4, 10, 2005);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10168, 1, 96.66, 36, 3479.66, 4, 10, 2006);

INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId, MonthId, YearId)
VALUES (10180, 12, 100, 42, 4695.6, 4, 11, 2006);
```

Script Output x
Task completed in 0.054 seconds

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

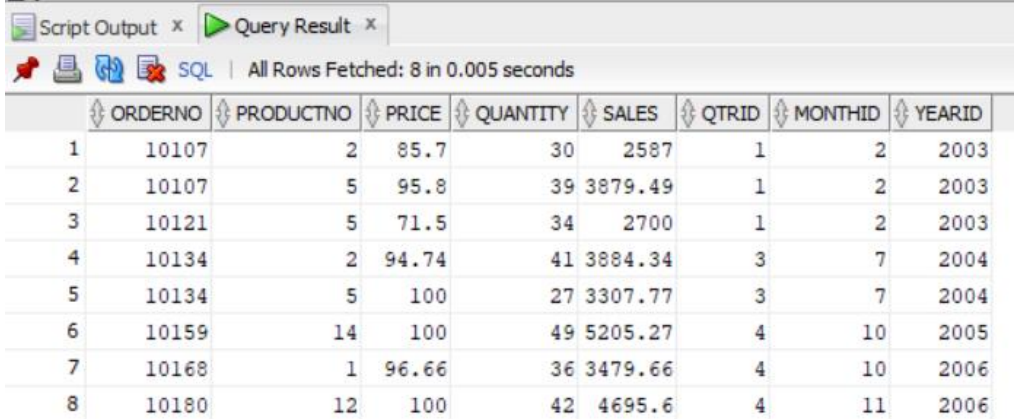
1 row inserted.

To generate a report which shows the number of products sold in each month of each year we need to insert data in the SalesDataStore table. Above is the query and its result to insert data in SalesDataStore table.

3.2.3 Data Selection

Figure 33: Select Query for SalesDataStore Table

```
select * from SalesDataStore;
```

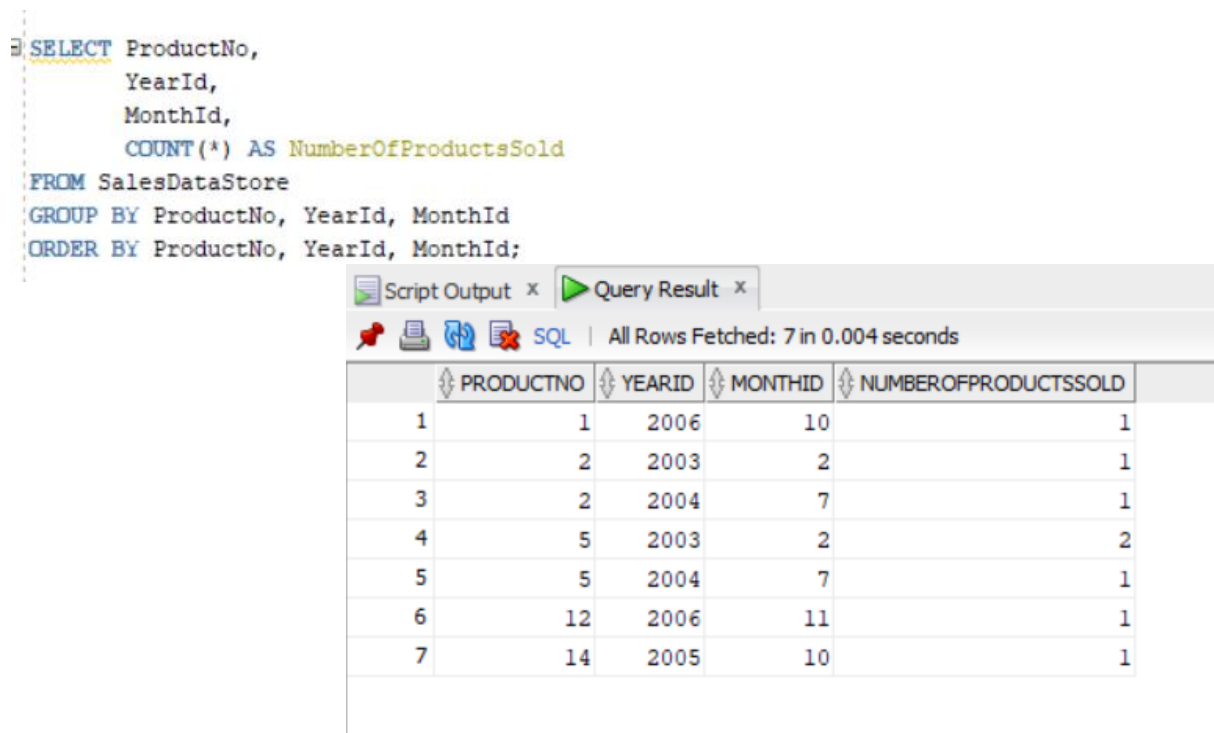


	ORDERNO	PRODUCTNO	PRICE	QUANTITY	SALES	QTRID	MONTHID	YEARID
1	10107	2	85.7	30	2587	1	2	2003
2	10107	5	95.8	39	3879.49	1	2	2003
3	10121	5	71.5	34	2700	1	2	2003
4	10134	2	94.74	41	3884.34	3	7	2004
5	10134	5	100	27	3307.77	3	7	2004
6	10159	14	100	49	5205.27	4	10	2005
7	10168	1	96.66	36	3479.66	4	10	2006
8	10180	12	100	42	4695.6	4	11	2006

To check if all the data is inserted correctly in the SalesDataStore table we have written a SELECT statement, this query selects everything from the SalesDataStore table. Above is the query and its output.

3.2.4 Group By Query

Figure 34: Group By Query



```

SELECT ProductNo,
       YearId,
       MonthId,
       COUNT(*) AS NumberOfProductsSold
FROM SalesDataStore
GROUP BY ProductNo, YearId, MonthId
ORDER BY ProductNo, YearId, MonthId;

```

Script Output x Query Result x

SQL | All Rows Fetched: 7 in 0.004 seconds

	PRODUCTNO	YEARID	MONTHID	NUMBEROFPRODUCTSSOLD
1	1	2006	10	1
2	2	2003	2	1
3	2	2004	7	1
4	5	2003	2	2
5	5	2004	7	1
6	12	2006	11	1
7	14	2005	10	1

To find the number of products sold in each month of each year, we have written a query which utilizes the GROUP BY feature of SQL. In the query above, we have selected ProductNo, YearId, and MonthId of every order from the SalesDataStore table, we have then grouped the orders by ProductNo, YearId and MonthId. This should group the products based on our requirements which is products sold in each month of each year. We have then counted the number of products sold in each month of each year and have given it an alias of NumberOfProductsSold, which shows the total number of products sold. After aggregating the required data, we used the ORDER BY statement to order the product by ProductNo, YearId, and MonthId. This should first order the product by ProductNo, then by YearId and finally by MonthId in ascending order.

3.3 Map Reduce Solution

Considering that the sales data is too large, using a relational database to determine, quantity and number of products sold in each month of each year would take a lot of time and processing and a relational database runs queries in a single thread. To calculate the result in parallel for a large dataset we use the map reduce method in a decentralized manner (Evermann, 2016). Below are the steps used by the map-reduce solution for this problem:

3.3.1 Step 1: Map

Considering that the data is stored in an array of JSONs, we skip an important process which is input splitting, and move directly to input mapping, because a lot of programming languages support JSON we do not need to read and split the file. While mapping the data we separate the data into key value pairs (Samaddar, Sinha, & De, 2019). For this sales data key would be productNo, monthId and yearId as it uniquely identifies the number and quantity of product sold in each month of each year. We also remove and filter unnecessary data, which in this case would be the sales, qtrId, and orderNo. The value for this sales data would be the quantity and count of the number of products sold. After separating the data into key value pairs, we send it as input for the next phase which is shuffling and sorting.

Figure 35: Map

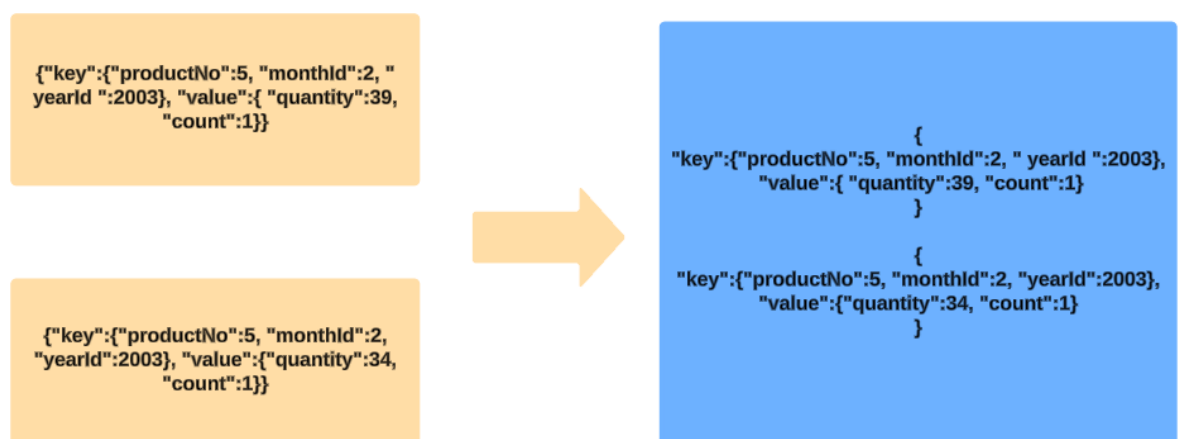


The figure above shows how input is mapped into key-value pairs.

3.3.2 Step 2: Shuffle and Sort

The shuffle and sort phase takes mapped data as an input and then returns grouped array or list of those inputs. In this step we take mapped data as an input and shuffle and sort them based on various parameters. For this sales data first, the products would be grouped based on their keys which are productNo, monthId and yearId. The grouped products would then be stored in an array or list. After that the array would be sorted based on yearId first and then it would be sorted again based on monthId, to properly arrange data to send on reduce phase.

Figure 36: Shuffle and Sort

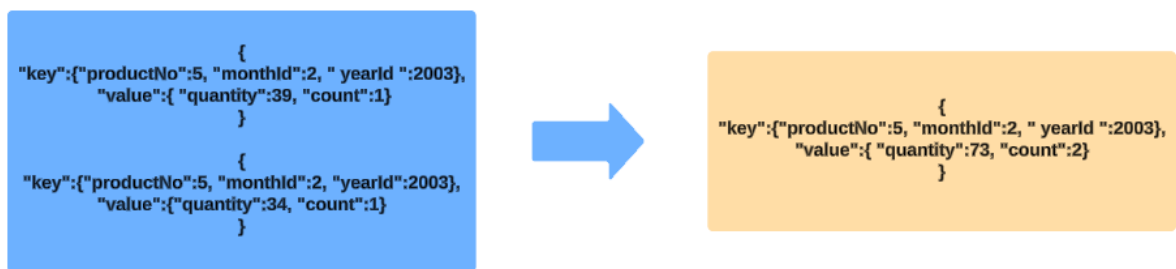


The figure above shows how the mapped data is shuffled and sorted in a group.

3.3.3 Step 3: Reduce

In this step, we take the grouped key, value pairs from shuffle, and sort phase as input and run a reduce function to generate the required output (Cho, 2019). For this sales data, we add up the quantity and count of the input and display them as output. After calculating the sum of all the quantities and counts we create a new array by merging all the data and sending them as the final output of the map reduce function.

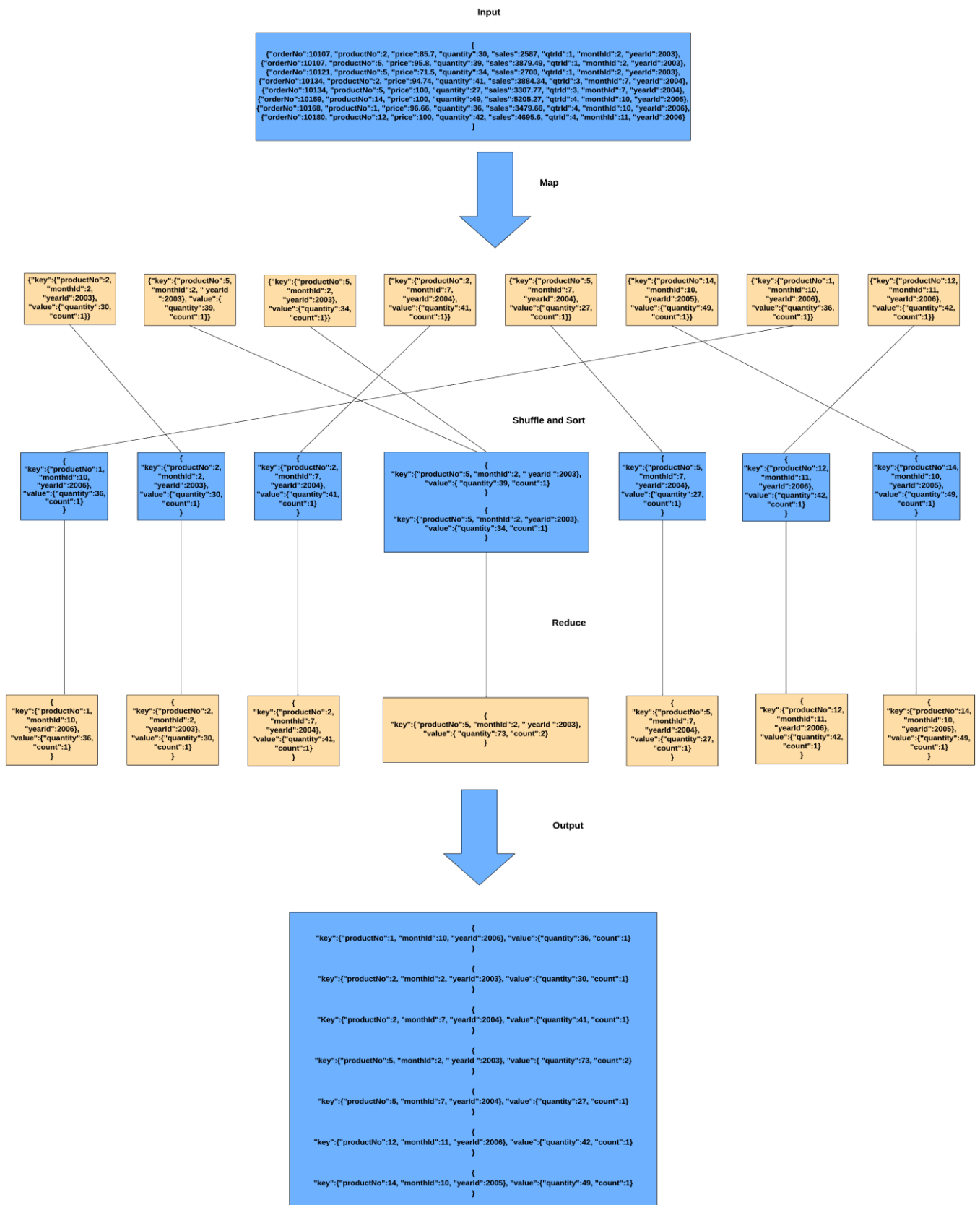
Figure 37: Reduce



The figure above shows how the shuffled and sorted data is reduced to get the required result for any problem. In the case of this sales data, the values which are quantity and count are added to get the number of products sold in a particular month of a particular year.

3.4 Map Reduce Diagram

Figure 38: Map Reduce



3.5 Map Reduce Implementation in Oracle SQL

Figure 39: Map Reduce Implementation in Oracle SQL

```
--Mapping Phase
WITH MappedData AS (
  SELECT ProductNo,
         YearId,
         MonthId,
         COUNT(productno) AS TotalProductsSold,
         SUM(quantity) AS TotalQuantityOfProductSold
  FROM SalesDataStore
  GROUP BY ProductNo, YearId, MonthId
),
--Reducing Phase
ReducedData AS (
  SELECT ProductNo,
         YearId,
         SUM(TotalProductsSold) AS TotalProductsSold,
         SUM(TotalQuantityOfProductSold) AS TotalQuantityOfProductSold
  FROM MappedData
  GROUP BY ProductNo, YearId
)
--Final output Phase
SELECT ProductNo,
       YearId,
       TotalProductsSold,
       TotalQuantityOfProductSold
FROM ReducedData
ORDER BY ProductNo, YearId;
```

Query Result x				
All Rows Fetched: 7 in 0.003 seconds				
	PRODUCTNO	YEARID	TOTALPRODUCTSSOLD	TOTALQUANTITYOFPRODUCTSOLD
1	1	2006	1	36
2	2	2003	1	30
3	2	2004	1	41
4	5	2003	2	73
5	5	2004	1	27
6	12	2006	1	42
7	14	2005	1	49

In Oracle, there is a way to demonstrate the map reduce function by using WITH keyword, in the query above we have first selected the ProductNo, YearId, MonthId, count of ProductNo as TotalProductsSold, the sum of quantity as TotalQuantityOfProductSold from SalesDataStore and then grouped it into ProductNo, YearId, and MonthId. We have then stored the result of this select statement in Mapped data, this is the implementation of the map method in figure no 38 where we have separated the data into key, and value pairs, in this select statement the keys are ProductNo, YearId and MonthId and the values are TotalProductsSold and TotalQuantityOfProductSold.

The second step demonstrated in this query is the reduce step, in this step we have reduced the mapped data into the sum of TotalProductsSold and the sum of TotalQuantityOfProductsSold, to get a number of products sold in each month of each year. In the reduced query we have selected the keys which are ProductNo and YearId from the mapped data and added the values which are TotalProductsSold and TotalQuantityOfProductSold.

After mapping and reducing the data, we used a final SELECT statement in the ReducedData to show the merged data of reduced data. In the final query, we have selected ProductNo, YearId, TotalProductsSold, TotalQuantityOfProductsSold from reduced data and ordered it according to ProductNo and YearId in ascending order. If we compare the outputs of figure 34, the final output of figure 38, and the output of figure 39 we can see that all the outputs are the same which justifies that this solution is correct and can work both on relational databases and while using parallel processing.

Chapter 4. Part D: Research Report

4.1 Research Report for the Given Scenario

4.1.1 Introduction

Businesses such as ShopNow frequently face problems in the efficient management of their increasing databases, due to the dynamism of the eCommerce landscape. ShopNow a growing e-commerce platform initially depended on conventional relational database management systems (RDBMS) like MYSQL. As the platform burgeoned and diversified, encountering a multitude of user interactions and product catalog expansions, the limitations of MySQL became apparent and faced several challenges from the relational system so switched to the NOSQL database system called MongoDB to fulfill all the demands of ShopNow expanding ecosystem. Comprehending the obstacles, benefits, and drawbacks of both SQL and MongoDB might help clarify ShopNow's reasoning for making this switch.

4.1.2 Challenges with SQL(MySQL)

As time passes, unlimited users and product catalogs generate diverse types of data, due to their varied formats, structures, and characteristics, the data become unstructured or semi-structured so that rigid schema of MYSQL databases cannot accommodate frequent schema modifications required for evolving data needs, resulting in downtime and schema migration challenges. It relies on join operations and complex relationships between tables. So, the large amount of data needs to be processed in join operations. This can lead to performance degradation. However, MySQL also offers vertical scalability by upgrading hardware resources, such as CPU and memory, to handle increased workload, this approach has its limitations and drawbacks. As it involves investing in high-end hardware, which can be expensive and may not provide a sustainable solution for long-term growth. (Taylor, 2011)

4.1.3 Advantages of SQL(MySQL)

In limited numbers of users and a straightforward inventory management system there were well-defined data structures. So, it is wise to use MYSQL which is more cost-effective (open-source software and is available for free), easy to use, mature, and stable (a large community of users and developers, as well as robust documentation and support resources), and trusted can efficiently handle all needs of shopnow. MySQL stores structured data in a relational format, with well-defined schema and relationships between entities one can query and manipulate data stored in the database. As well as MySQL enforces constraints such as primary keys, foreign keys, and unique constraints that are essential in normalization as well as adherence to ACID (Atomicity, Consistency, Isolation, Durability) helps to maintain data integrity and reliability. Furthermore, MySQL supports transactions, allowing multiple database operations to be grouped and executed atomically, ensuring that either all operations are completed successfully or none at all also applied to transactions. MySQL includes features for optimizing database performance, such as indexing, query optimization, and caching mechanisms. These features help improve the speed and efficiency of data retrieval and processing, ensuring that ShopNow operates smoothly. (Taylor, 2011)

4.1.4 Challenges with MongoDB (NoSQL)

The first thing is that Shopnow already adopted MYSQL since query language and data manipulation capabilities differ from MYSQL, so requires new developers to learn and adapt to a new set of tools and practices. furthermore, managing distributed MongoDB clusters and ensuring high availability and time of data modeling also can still be complex. Therefore, must be more careful and be more expert. Secondly, due to not having a rigid schema each document (equivalent to a row in a relational database) within a collection (equivalent to a table) can have a different structure. So, there's a risk of inconsistent data structures across documents within a collection, which could potentially impact data integrity and application logic. Furthermore, even though MongoDB provides support for transactions in recent versions, it may require additional configuration and overhead compared to MYSQL databases. (Lourenço, 2015)

4.1.5 Advantages of MongoDB (NoSQL)

In such conditions, MongoDB's document-oriented data model is well-suited for handling such data efficiently. This flexibility enables ShopNow to accommodate diverse data types and evolving data schemas without the need for costly schema migrations or complex data transformations. Furthermore, MongoDB's support for sharding enables ShopNow to partition data across multiple servers, distributing the workload evenly and allowing for linear scalability as the platform grows. This Horizontal scalability ensures that ShopNow to handle increasing volumes of user-generated content and analytical queries without experiencing performance degradation. Also, by using the BSON (Binary JSON) format, MongoDB made it possible to store and retrieve a wide range of data types efficiently. (Lourenço, 2015)

4.1.6 Conclusion

From above, at the starting phase, MySQL served ShopNow adequately with its structured data storage and relational format, ensuring data integrity and efficient querying. However, as ShopNow expanded, the limitations of MySQL became apparent with unstructured data and evolving schemas, leading to downtime and performance issues. Despite of complexity and having a chance to be inconsistent MongoDB emerged as a viable solution, offering flexibility, scalability, and efficient handling of diverse data types. Its document-oriented model and support for sharding enabled ShopNow to adapt seamlessly to growth and diverse data needs without costly schema migrations. MongoDB's introduction marked a pivotal shift, ensuring ShopNow's ability to handle increasing volumes of data and analytical queries while maintaining performance.

4.2 MongoDB Queries

4.2.1 Database Creation

In MongoDB, we can use the 'use' keyword to create a database. The 'Use' keyword creates a new database if it does not exist or if the database already exists it uses that database.

Figure 40: ShopNow Database Creation

```
test> use ShopNow;
switched to db ShopNow
ShopNow>
```

4.2.2 Data Insertion

The query below uses insertMany function of MongoDB to insert multiple data. In this query, we have inserted multiple documents in the users' collection. All the inserted users include an _id key which is the unique identifier of the user, userId which is the id of the user, name of the user, and email of the user.

Figure 41: Data Insertion in users' Collection

```
ShopNow> db.users.insertMany([
...   {
...     "_id": ObjectId("6123456789abcdef01234567"),
...     "userId": "user001",
...     "name": "John Doe",
...     "email": "john@example.com"
...   },
...   {
...     "_id": ObjectId("abcdef012345678901234567"),
...     "userId": "user002",
...     "name": "Jane Smith",
...     "email": "jane@example.com"
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6123456789abcdef01234567'),
    '1': ObjectId('abcdef012345678901234567')
  }
}
ShopNow>
```

Similarly, we have also inserted multiple documents in the products collection by using the insertMany function. In the product document, we have an _id key which is the unique identifier of a product, productId which is the id of the product, title of the product, category of the product and price of the product.

Figure 42: Data Insertion in Products Collection

```
ShopNow> db.products.insertMany([
...   {
...     "_id": ObjectId("0123456789abcdef01234567"),
...     "productId": "P001",
...     "title": "Laptop",
...     "category": "Electronics",
...     "price": 999.99
...   },
...   {
...     "_id": ObjectId("123456789abcdef012345678"),
...     "productId": "P002",
...     "title": "Smartphone",
...     "category": "Electronics",
...     "price": 699.99
...   },
...   {
...     "_id": ObjectId("23456789abcdef0123456789"),
...     "productId": "P003",
...     "title": "Headphones",
...     "category": "Electronics",
...     "price": 99.99
...   },
...   {
...     "_id": ObjectId("3456789abcdef01234567890"),
...     "productId": "P004",
...     "title": "Backpack",
...     "category": "Fashion",
...     "price": 49.99
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('0123456789abcdef01234567'),
    '1': ObjectId('123456789abcdef012345678'),
    '2': ObjectId('23456789abcdef0123456789'),
    '3': ObjectId('3456789abcdef01234567890')
  }
}
ShopNow>
```

4.2.3 Query to retrieve product where price is less than \$50

In the query below we have used the find function of MongoDB to select data from product collection. We have then added a price filter in the find function to filter documents that have a price less than \$50 and we have finally projected the title, category, and price keys to retrieve product details.

Figure 43: Query to retrieve product where price is less than \$50

```
ShopNow> db.products.find(
...   { "price": { $lt: 50 } },
...   { "title": 1, "category": 1, "price": 1, "_id": 0 }
... );
[ { title: 'Backpack', category: 'Fashion', price: 49.99 } ]
ShopNow>
```

4.2.4 Query to update the price of the product with productid “P003” to \$75

The query below uses the updateOne function of MongoDB to update the price of “P003” to \$75. In this query, we have first selected a product with productId of “P003” and then set the price of that product to \$75.

Figure 44: Query to update the price of the product with productid “P003” to \$75

```
ShopNow> db.products.updateOne(
... { "productId": "P003" },
... { $set: { "price": 75 } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
ShopNow>
```

Figure 45: Query to find the product with productId "P003"

```
ShopNow> db.products.find({ "productId": "P003" });
[
  {
    _id: ObjectId('23456789abcdef0123456789'),
    productId: 'P003',
    title: 'Headphones',
    category: 'Electronics',
    price: 75
  }
]
ShopNow>
```

The query above finds the product with productId “P003”, in the output we can see that the price of the product has been set to \$75, so we can confirm that the update query is working.

4.2.5 Insert a new product

The query below inserts a new product in the product collection with productId “P005”, the title “Smartwatch”, the category “Electronics” and the price “\$149.99”. In this query we have used insertOne function of MongoDB to insert only one document in the collection. We then passed the key and value pairs in the insertOne function as a document and inserted that document in the product collection.

Figure 46: Inserting New Product

```
ShopNow> db.products.insertOne({
...   "productId": "P005",
...   "title": "Smartwatch",
...   "category": "Electronics",
...   "price": 149.99
... });
{
  acknowledged: true,
  insertedId: ObjectId('663a0f2b0fb35c6d5b117b7b')
}
ShopNow>
```

Figure 47: Finding the Inserted Product

```
ShopNow> db.products.find({ "productId": "P005" });
[
  {
    _id: ObjectId('663a0f2b0fb35c6d5b117b7b'),
    productId: 'P005',
    title: 'Smartwatch',
    category: 'Electronics',
    price: 149.99
  }
]
ShopNow>
```

The query above uses the find function of MongoDB to find a product with productId “P005” because it shows one document, we can confirm that a new product has been inserted.

4.2.6 Delete all products from the “Fashion” category

The query below used the deleteMany function of MongoDB to delete multiple documents of a collection. In this query, I have passed the category as the key and Fashion as the value of the key to delete all the documents from the products collection containing the “Fashion” category.

Figure 48: Deleting all products from the “Fashion” category

```
ShopNow> db.products.deleteMany({ "category": "Fashion" });
{ acknowledged: true, deletedCount: 1 }
```

Figure 49: Finding the product from the “Fashion” category

```
ShopNow> db.products.find({ "category": "Fashion" });
ShopNow>
```

The query above uses the find function to find all the products having the “Fashion” category, because it returns no results, we can confirm that all the products with the “Fashion” category have been deleted.

Chapter 5. Conclusion

Doing this coursework was not easy at all. The coursework was given out on the 5th May 2024 and the submission was on the 17th May 2024. Not only completing the coursework was my priority but completing it on time was also very important. The knowledge we got from the lecture session was not quite enough for us to do the coursework. So, we had to do a lot of research and study. we also took help from our lecturer and tutor. We would also like to show our gratitude to our module leader for providing such amazing coursework. we got a chance to enhance our data modeling, SQL programming, NoSQL programming and parallel processing skills. Without the constant guidance of the teachers the completion of the coursework was almost impossible. Making a ER diagram from the scenario, implementing it into SQL and NoSQL queries and applying parallel processing to the given problem was a tough task to do. While doing the Oracle SQL queries, we faced a lot of problems, hell lot of errors. But wherever we got the bugs we fixed it through the help of our teachers, internet and the books. Finally, the coursework of making ER diagram, schema diagram, relational tables, converting those relational tables to SQL queries, applying NoSQL solutions and parallel processing to the SQL solution is successfully completed. we faced many problems regarding the implementation of the ER diagram and parallel processing. There were various queries with its specific works which was very challenging for us to implement SQL, NoSQL queries and parallel processing. But as we got on the depth of Oracle Queries challenges reduced. The problems keep on coming and we solved them with the help of regular visit with the tutors and thorough researches. The tutor taught us the best way to rectify the errors in various ways.

Chapter 6. References

- Cho, Y.-S. (2019). A Parallel Community Detection in Multi-Modal Social Network With Apache Spark. *IEEE Access*, 27465-27478. Retrieved 05 15, 2024
- Evermann, J. (2016). Scalable Process Discovery Using Map-Reduce. *IEEE Transactions on Services Computing*, 469-481. Retrieved 05 13, 2024
- Lourenço, J. R. (2015). Choosing the right NoSQL database for the job: a quality attribute evaluation. *Choosing the right NoSQL database for the job: a quality attribute evaluation*. Retrieved 05 09, 2024
- Ralph Kimball, M. R. (2002). *The Data Warehouse Toolkit, Second Edition, The Complete Guide to Dimensional Modeling*. Robert Ipsen. Retrieved 05 07, 2024
- Samaddar, S., Sinha, R., & De, R. K. (2019). A Model for Distributed Processing and Analyses of NGS Data under Map-Reduce Paradigm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 827-840. Retrieved 05 10, 2024
- Taylor, A. G. (2011). *SQL All-in-One For Dummies*. Wiley Publishing, Inc. Retrieved 05 06, 2024

Appendix

All the queries and diagrams of this assignment are saved in a GitHub repository. The link of that GitHub repository is given below:

<https://github.com/Meghant00/Data-Management-Assignment>

6.1 Appendix For Part A

6.1.1 All the SQL Queries for Part A

```
CREATE TABLE Departments (  
    Department_ID INT PRIMARY KEY,  
    Name VARCHAR2(100),  
    Location VARCHAR2(100)  
);
```

```
CREATE TABLE Courses (  
    Course_ID INT PRIMARY KEY,  
    Title VARCHAR2(100),  
    Credit_Hours INT  
);
```

```
CREATE TABLE Faculty (  
    Faculty_ID INT PRIMARY KEY,  
    Name VARCHAR2(100),  
    Academic_Rank VARCHAR2(50)  
);
```



```
CREATE TABLE Students (  
    Student_ID INT PRIMARY KEY,  
    Name VARCHAR2(100),  
    Major VARCHAR2(100)  
);
```

```
CREATE TABLE Campuses (  
    Campus_ID INT PRIMARY KEY,  
    Name VARCHAR2(100),  
    Address VARCHAR2(255)  
);
```

```
CREATE TABLE Administrative_Staff (  
    Staff_ID INT PRIMARY KEY,  
    Name VARCHAR2(100),  
    Job_Title VARCHAR2(100)  
);
```

```
CREATE TABLE Faculty_Departments (  
    Faculty_ID INT,  
    Department_ID INT,  
    PRIMARY KEY (Faculty_ID, Department_ID),  
    FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID),  
    FOREIGN KEY (Department_ID) REFERENCES Departments(Department_ID)
```

);

```
CREATE TABLE Faculty_Courses (  
    Faculty_ID INT,  
    Course_ID INT,  
    Semester VARCHAR2(50),  
    PRIMARY KEY (Faculty_ID, Course_ID, Semester),  
    FOREIGN KEY (Faculty_ID) REFERENCES Faculty(Faculty_ID),  
    FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID)  
);
```

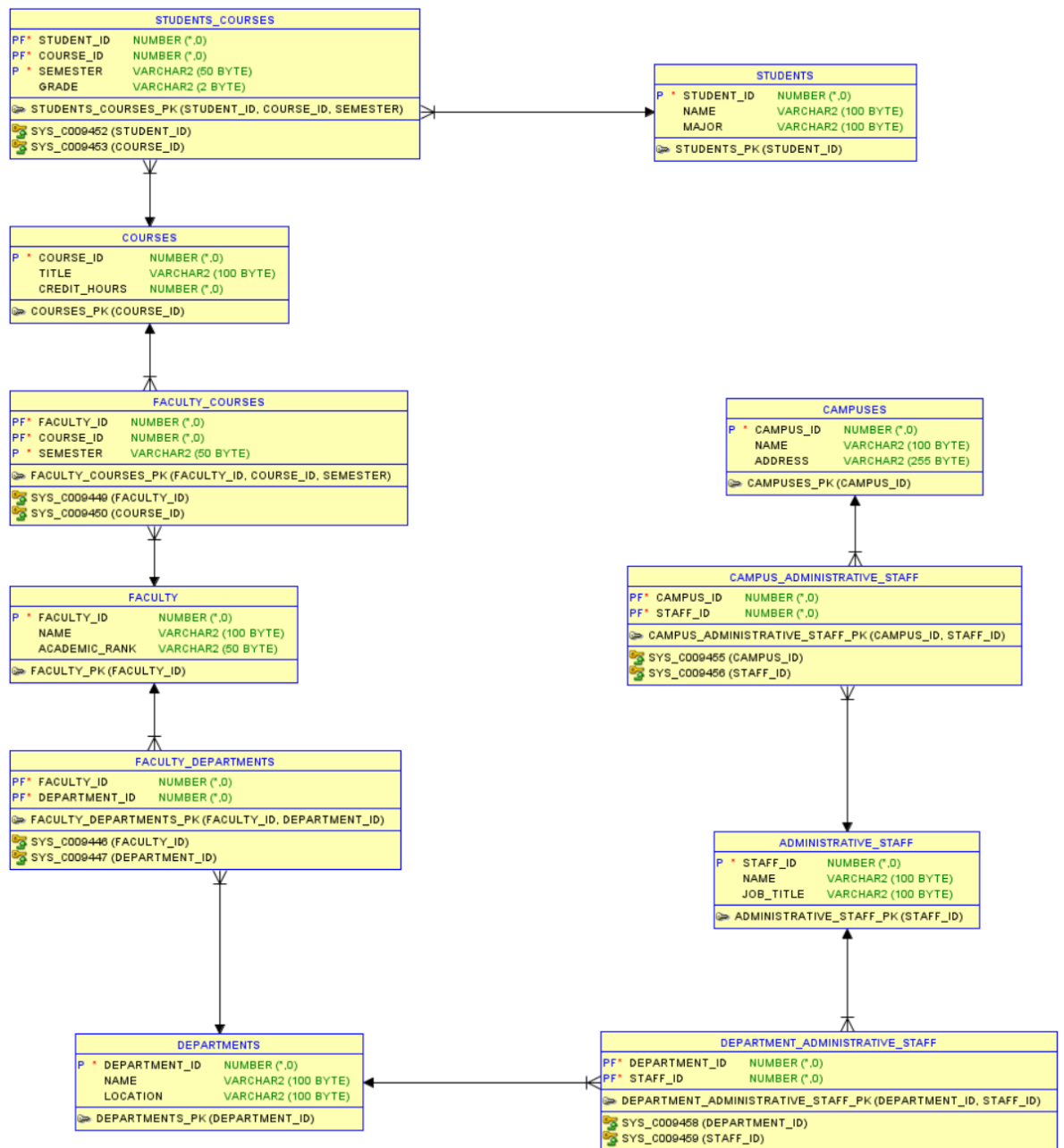
```
CREATE TABLE Students_Courses (  
    Student_ID INT,  
    Course_ID INT,  
    Semester VARCHAR2(50),  
    Grade VARCHAR2(2),  
    PRIMARY KEY (Student_ID, Course_ID, Semester),  
    FOREIGN KEY (Student_ID) REFERENCES Students(Student_ID),  
    FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID)  
);
```

```
CREATE TABLE Campus_Administrative_Staff (  
    Campus_ID INT,  
    Staff_ID INT,
```

```
PRIMARY KEY (Campus_ID, Staff_ID),  
FOREIGN KEY (Campus_ID) REFERENCES Campuses(Campus_ID),  
FOREIGN KEY (Staff_ID) REFERENCES Administrative_Staff(Staff_ID)  
);
```

```
CREATE TABLE Department_Administrative_Staff (  
    Department_ID INT,  
    Staff_ID INT,  
    PRIMARY KEY (Department_ID, Staff_ID),  
    FOREIGN KEY (Department_ID) REFERENCES Departments(Department_ID),  
    FOREIGN KEY (Staff_ID) REFERENCES Administrative_Staff(Staff_ID)  
);
```

6.1.2 Schema diagrams created on SQL Developer



6.2 Appendix For Part B

6.2.1 All the SQL Queries for Part B

--Creating Table named "User"

```
CREATE TABLE "User"

(

  UserId VARCHAR2(20) PRIMARY KEY,

  Name VARCHAR2(60),

  EmailAddress VARCHAR2(150)

);
```

--Creating Table named Category

```
CREATE TABLE Category

(

  categoryCode VARCHAR2(20) PRIMARY KEY,

  title VARCHAR2(150)

);
```

--Creating Table named Music

```
CREATE TABLE Music

(

  musicId VARCHAR2(20) PRIMARY KEY,

  title VARCHAR2(150),

  categoryCode VARCHAR2(20),

  costPerDownload NUMBER(20, 2),

  FOREIGN KEY (categoryCode) REFERENCES Category(categoryCode)
```

);

--Creating Table named DownloadMusic

CREATE TABLE DownloadMusic

(

UserID VARCHAR2(10),

musicId VARCHAR2(10),

downloadDate DATE,

FOREIGN KEY (UserID) REFERENCES "User"(UserId),

FOREIGN KEY (musicId) REFERENCES Music(musicId),

PRIMARY KEY (UserId, musicId)

);

-- Inserting given data into User table

INSERT INTO "User" (userId, name, emailAddress)

VALUES ('wayne10', 'Wayne, R', 'wayne@hotmail.co.uk');

INSERT INTO "User" (userId, name, emailAddress)

VALUES ('santos17', 'Santos, C', 'santos@ntl.co.uk');

INSERT INTO "User" (userId, name, emailAddress)

VALUES ('hary05', 'Hary, M', 'hary@freeserve.co.uk');

INSERT INTO "User" (userId, name, emailAddress)

```
VALUES ('margot9', 'Margot, C', 'margot9@msn.co.uk');
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('mount77', 'Mount, M', 'mount@hotmail.co.uk');
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('nancy91', 'Nancy, L', 'nancy91@tesco.co.uk');
```

```
-- Insert data into Category table
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO11', 'Love');
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO12', 'Pop');
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO13', 'Rock');
```

```
-- Inserting data into Music table
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M001', 'Born to run', 'MO13', 0.99);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M002', 'Crawling', 'MO13', 1.99);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M003', 'You're Beautiful', 'MO11', 1.49);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M004', 'Summer of 69', 'MO11', 1.79);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M005', 'Crazy Train', 'MO13', 1.50);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M006', 'Yellow Submarine', 'MO12', 1.10);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M007', 'Got to be there', 'MO12', 0.89);
```

```
-- Insert data into DownloadMusic table
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('wayne10', 'M002', TO_DATE('03-May-2021', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('margot9', 'M005', TO_DATE('01-May-2022', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
```



```
VALUES ('santos17', 'M002', TO_DATE('06-May-2021', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
```

```
VALUES ('margot9', 'M001', TO_DATE('06-May-2022', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
```

```
VALUES ('wayne10', 'M003', TO_DATE('01-Aug-2022', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
```

```
VALUES ('mount77', 'M004', TO_DATE('02-Aug-2022', 'DD-MON-YYYY'));
```

```
INSERT INTO DownloadMusic (userId, musicId, downloadDate)
```

```
VALUES ('nancy91', 'M007', TO_DATE('05-Sep-2021', 'DD-MON-YYYY'));
```

```
-- Inserting more data into User table
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('Kaushal100', 'Kaushal, RI', 'Kaushal100@hotmail.co.uk');
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('Ratish3254', 'Ratish, RB', 'Ratish3254@gmail.co.uk');
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('Meghant200', 'Meghant, D', 'Meghant200@freeserve.co.uk');
```

```
INSERT INTO "User" (userId, name, emailAddress)
```

```
VALUES ('Aayush200', 'Aayush, M', 'Aayush200@msn.co.uk');
```

```
-- Inserting more data into Category table
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO14', 'Reggae');
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO15', 'Jazz');
```

```
INSERT INTO Category (categoryCode, title)
```

```
VALUES ('MO16', 'Metal');
```

```
-- Inserting more into Music table
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M008', 'Three Little Birds', 'MO14', 1.98);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M009', 'The Lazy Song', 'MO14', 0.99);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M010', 'So Beautiful', 'MO15', 1.49);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
```

```
VALUES ('M011', 'Master of Puppets', 'MO16', 1.50);
```

```
INSERT INTO Music (musicId, title, categoryCode, costPerDownload)
VALUES ('M012', 'Nothing Else Matters', 'MO16', 1.45);

-- Inserting more data into DownloadMusic table

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Kaushal100', 'M008', TO_DATE('03-Jul-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Ratish3254', 'M012', TO_DATE('01-May-2023', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Meghant200', 'M010', TO_DATE('06-Aug-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Aayush200', 'M011', TO_DATE('06-Sep-2022', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Ratish3254', 'M009', TO_DATE('01-Aug-2023', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Meghant200', 'M012', TO_DATE('02-Aug-2021', 'DD-MON-YYYY'));

INSERT INTO DownloadMusic (userId, musicId, downloadDate)
VALUES ('Aayush200', 'M008', TO_DATE('05-Sep-2020', 'DD-MON-YYYY'));
```

6.3 Appendix For Part C

6.3.1 All the SQL Queries for Part C

```
CREATE TABLE SalesDataStore (  
  
    OrderNo NUMBER,  
  
    ProductNo NUMBER,  
  
    Price NUMBER(10,2),  
  
    Quantity NUMBER,  
  
    Sales NUMBER(10,2),  
  
    QtrId NUMBER(1) CHECK (QtrId BETWEEN 1 AND 4),  
  
    MonthId NUMBER(2) CHECK (MonthId BETWEEN 1 AND 12),  
  
    YearId NUMBER  
  
);  
  
-- Inserting given sample data into the SalesData table  
  
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,  
MonthId, YearId)  
  
VALUES (10107, 2, 85.7, 30, 2587, 1, 2, 2003);  
  
  
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,  
MonthId, YearId)  
  
VALUES (10107, 5, 95.8, 39, 3879.49, 1, 2, 2003);  
  
  
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,  
MonthId, YearId)  
  
VALUES (10121, 5, 71.5, 34, 2700, 1, 2, 2003);
```

```
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,
MonthId, YearId)
```

```
VALUES (10134, 2, 94.74, 41, 3884.34, 3, 7, 2004);
```

```
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,
MonthId, YearId)
```

```
VALUES (10134, 5, 100, 27, 3307.77, 3, 7, 2004);
```

```
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,
MonthId, YearId)
```

```
VALUES (10159, 14, 100, 49, 5205.27, 4, 10, 2005);
```

```
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,
MonthId, YearId)
```

```
VALUES (10168, 1, 96.66, 36, 3479.66, 4, 10, 2006);
```

```
INSERT INTO SalesDataStore (OrderNo, ProductNo, Price, Quantity, Sales, QtrId,
MonthId, YearId)
```

```
VALUES (10180, 12, 100, 42, 4695.6, 4, 11, 2006);
```

```
select * from SalesDataStore;
```

```
SELECT ProductNo,
```

```
    YearId,
```

```
    MonthId,
```

```
    COUNT(*) AS NumberOfProductsSold
```

```
FROM SalesDataStore
```

```
GROUP BY ProductNo, YearId, MonthId
```

```
ORDER BY ProductNo, YearId, MonthId;
```

```
--Mapping Phase
```

```
WITH MappedData AS (  
    SELECT ProductNo,  
           YearId,  
           MonthId,  
           COUNT(*) AS NumberOfProductsSold  
    FROM SalesDataStore  
    GROUP BY ProductNo, YearId, MonthId  
)  
  
--Reducing Phase  
ReducedData AS (  
    SELECT ProductNo,  
           YearId,  
           SUM(NumberOfProductsSold) AS TotalProductsSold  
    FROM MappedData  
    GROUP BY ProductNo, YearId  
)  
  
--Final output Phase  
SELECT ProductNo,  
       YearId,  
       TotalProductsSold  
FROM ReducedData  
ORDER BY ProductNo, YearId;  
  
select * From "User";  
  
select * From Music;
```

```
select * From category;  
  
select * From downloadmusic;  
  
SELECT musicId, title, categoryCode  
  
FROM Music  
  
ORDER BY title;
```

```
SELECT COUNT(DISTINCT dm.userId) AS num_users_downloaded  
  
FROM DownloadMusic dm  
  
JOIN Music m ON dm.musicId = m.musicId  
  
JOIN Category c ON m.categoryCode = c.categoryCode  
  
WHERE c.title = 'Pop-Rock';
```

```
SELECT c.title AS category_title, COUNT(dm.musicId) AS num_downloads  
  
FROM DownloadMusic dm  
  
JOIN Music m ON dm.musicId = m.musicId  
  
JOIN Category c ON m.categoryCode = c.categoryCode  
  
GROUP BY c.title;
```

```
SELECT c.title AS category_title  
  
FROM Category c  
  
JOIN Music m ON c.categoryCode = m.categoryCode  
  
JOIN DownloadMusic dm ON m.musicId = dm.musicId  
  
GROUP BY c.title  
  
HAVING COUNT(dm.musicId) > 1;
```

6.4 Appendix For Part D

6.4.1 All the MongoDB Queries for Part D

```
// Switch to the ShopNow database (if it doesn't exist, MongoDB will create it)
```

```
use ShopNow;
```

```
// Insert users into the users collection
```

```
db.users.insertMany([  
  
  {  
  
    "_id": ObjectId("6123456789abcdef01234567"),  
  
    "userId": "user001",  
  
    "name": "John Doe",  
  
    "email": "john@example.com"  
  
  },  
  
  {  
  
    "_id": ObjectId("abcdef012345678901234567"),  
  
    "userId": "user002",  
  
    "name": "Jane Smith",  
  
    "email": "jane@example.com"  
  
  }  
  
]);
```

```
// Insert products into the products collection
```

```
db.products.insertMany([  
  
  {
```



```
"_id": ObjectId("0123456789abcdef01234567"),  
"productId": "P001",  
"title": "Laptop",  
"category": "Electronics",  
"price": 999.99  
},  
  
{  
  "_id": ObjectId("123456789abcdef012345678"),  
  "productId": "P002",  
  "title": "Smartphone",  
  "category": "Electronics",  
  "price": 699.99  
},  
  
{  
  "_id": ObjectId("23456789abcdef0123456789"),  
  "productId": "P003",  
  "title": "Headphones",  
  "category": "Electronics",  
  "price": 99.99  
},  
  
{  
  "_id": ObjectId("3456789abcdef01234567890"),  
  "productId": "P004",  
  "title": "Backpack",
```

```
    "category": "Fashion",  
    "price": 49.99  
  }  
  );
```

//For Retrieve all product details (title, category, price) where the price is less than \$50.

```
db.products.find(  
  { "price": { $lt: 50 } },  
  { "title": 1, "category": 1, "price": 1, "_id": 0 }  
);
```

//For Update the product price of the product with productId "P003" to \$75

```
db.products.updateOne(  
  { "productId": "P003" }, // Filter to find the document with productId "P003"  
  { $set: { "price": 75 } } // Update operation to set the price to $75  
);
```

```
// For Inserting a new product with the following details: productId "P005", title  
"Smartwatch", category
```

```
"Electronics", and price $149.99
```

```
db.products.insertOne({  
  "productId": "P005",  
  "title": "Smartwatch",  
  "category": "Electronics",  
  "price": 149.99  
});
```

```
// For Deleting all products from the "Fashion" category.
```

```
db.products.deleteOne({ "category": "Fashion" });
```

```
// For checking if data are deleted
```

```
db.products.find({ "category": "Fashion" });
```