



slington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC5051NI- Database

Year and Semester

2020-21 Autumn

Student Name: Pratik Amatya

Group: C1

London Met ID: 19031389

College ID: NP01CP4A190024

Assignment Due Date: Sunday 31 January 2021

Assignment Submission Date: Monday 01 February 2021

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

1.	Introduction.....	1
1.1	Introduction of the Store	1
1.1.1	Aim	1
1.1.2	Objectives	1
1.2	Identification of Entities and Attributes	2
2.	Normalization	3
2.1	Assumptions	3
2.2	Normalization.....	3
2.2.1	UNF (Un-Normalized Form)	3
2.2.2	1NF	4
2.2.3	2NF	4
2.2.4	3NF	5
3.	Entity Relation Diagram	7
4.	Database Implementation.....	8
4.1	Tables Generation (DDL Scripts)	8
4.1.1	Customer Table.....	8
4.1.2	Product Table	8
4.1.3	Address Table	9
4.1.4	Orders Table.....	10
4.1.5	Orders_Details Table	10
4.2	Populate DB Tables.....	11
4.2.1	Inserting values into the Customer table.....	12
4.2.2	Inserting values into the Product table.....	13
4.2.3	Inserting values into Address Table.....	14

4.2.4	Inserting values into Orders Table	15
4.2.5	Inserting Values into the Order_Details table.....	16
5.	Database Querying for case study.....	18
6.	Database Dump file Creation.....	27
6.1	Screenshot of the Dump File created	27
6.2	Command for deleting all the files in the database with the sequential order of deletion 30	
7.	Bibliography	31

List of Figures

Figure 1: Final ER-Diagram After Normalization.....	7
Figure 2: Creating the Customer Table.....	8
Figure 3: Creating the Product Table.....	9
Figure 4: Creating the Address Table	9
Figure 5: Creating the Orders Table	10
Figure 6: Creating the Order_Details Table.....	11
Figure 7: Insertion of values into the Customer tables	12
Figure 8: Customer Selection Statement.....	12
Figure 9: Insertion of values into the Product table.....	13
Figure 10: Product Selection Statement.....	13
Figure 11: Insertion of values into the Address table	14
Figure 12: Address Selection Statement	14
Figure 13: Insertion of values into the Orders table	15
Figure 14: Orders Selection Statement	16
Figure 15: Insertion of values into the Order_Details table.....	17
Figure 16: Order_Details Selection Statement	17
Figure 17: Query 1	18
Figure 18: Query 2	19
Figure 19: Query 3	20
Figure 20: Query 4	21
Figure 21: Query 5	22
Figure 22: Query 6	23
Figure 23: Query 7	24
Figure 24: Query 8	25
Figure 25: Query 9	26
Figure 26: Creation of dump file - Part 1.....	27
Figure 27: Creation of dump file - Part 2.....	28
Figure 28: Creation of actual dump file.....	29
Figure 29: Dropping all the tables	30

List of Tables

Table 1: Table showing the initial entity and attributes.....	3
---	---

1. Introduction

1.1 Introduction of the Store

The database system is designed for a department store to store detailed records of the daily transactions made. The store stores all the products which are foods such as noodles, rice, coffee, oil, biscuits, or products that are required for households such as a dishwasher. The store delivers the products ordered by their customers to the specified address by the customers. This store delivers the products all around the country.

1.1.1 Aim

The main goal of this store is to deliver quality products to their customers at the cheapest rates.

1.1.2 Objectives

- To sell products at the most affordable prices
- To treat customers like family and deliver their orders as fast as possible
- To deliver the products anywhere in the country with eco-friendly packaging

1.2 Identification of Entities and Attributes

An entity can be a real-world object either animate or inanimate, that is easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes that give them their identity.

For example, a student entity may have a name, class, and age as attributes.

The different types of attributes are:

- Simple Attribute

The attributes that have atomic values and cannot be divided further are called simple attributes. For example, a student's gender is an atomic value of a single word.

- Composite Attribute

Composite attributes are those attributes made up of more than one simple attribute.

For example, a student's full name may have first_name and last_name.

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student uniquely identifies him/her among students.

The different types of keys are:

- Super Key – A set of attributes (single or multiple attributes) that collectively identifies an entity in an entity set.
- Candidate Key – A minimal key is called a candidate key. An entity set may have more than one candidate key.
- Primary key – A primary key is one of the candidate keys that uniquely identify the entity set. (Tutorialspoint, 2020)
- Foreign key – A foreign key is a column or a combination of columns whose values match a Primary key in a different table. It is used to link two tables together. (Tutorialspoint, 2020)

Entities	Attributes
Product	Product_ID, Product_Name, Unit_Price
Order_Details	Order_ID, Product_ID, Customer_ID, Customer_Name, Customer_Address, Order_Date, Quantity
Address	Address_ID, Country, Zone, City

Table 1: Table showing the initial entity and attributes

2. Normalization

2.1 Assumptions

- The addresses stored in the address table are the locations requested by the customer for the delivery of each product of the order. These are not the living addresses of the customer.
- The customers can add different products to the same order on different dates. Hence, there may be different order dates for a different product in the same order.

2.2 Normalization

“Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divide larger tables into smaller tables and link them using relationships.” (Guru99, 2020)

2.2.1 UNF (Un-Normalized Form)

The scenario for UNF:

- Each order should belong to only one customer.
- Each order has one or multiple products that may have different order dates.
- Each product in a single order has one address specified by the customer

Now, the next step is to identify and list all the attributes into a single entity.

Showing repeating groups:

Orders(Order_ID (PK) , Customer_ID , Customer_Name , {Product_ID, Product_Name, Unit_Price, Quantiy, Order_Date, Address_ID, Country, Zone, City})

Here, all the attributes have been listed and the repeating groups have been separated. The attributes which are considered as repeating groups are kept inside curly brackets.

2.2.2 1NF

For a table to be in First Normal form (1NF), each column should have a single-valued attribute. Hence, all the repeating groups form a separate entity. After the repeating groups and the non-repeating groups are formed as separated entities, the primary key as well the foreign keys in each entity are identified.

In the 1NF below, the primary keys are denoted by the PK symbol and the attributes that have the FK symbol are the foreign keys. This step is easier as the repeating groups were already identified in the UNF.

Separating the repeating groups into their separate tables:

Orders (Order_ID (PK) , Customer_ID , Customer_Name)

Order_Details(Product_ID (PK), Order_ID (PK, FK) , Product_Name, Unit_Price, Quantity, Order_Date, Address_ID, Country, Zone, City)

2.2.3 2NF

In the second normal form, the partial dependencies are removed and a new entity is formed. The removal of partial dependencies results in minimizing data redundancy.

In the Order_Details Table,

Product_ID \rightarrow Product_Name, Unit_Price

Order_ID, Product_ID \rightarrow Quantity, Order_Date, Address_ID, Country, Zone, City

Hence,

Product(Product_ID (PK), Product_Name, Unit_Price)

Order_Details (Order_ID(PK,FK), Product_ID (PK,FK) , Quantity, Order_Date, Address_ID, Country, Zone, City)

Therefore, Total tables formed:

Orders (Order_ID (PK) , Customer_ID , Customer_Name)

Product (Product_ID (PK), Product_Name, Unit_Price)

Order_Details (Order_ID(PK,FK), Product_ID (PK,FK) , Quantity, Order_Date, Address_ID, Country, Zone, City)

In the steps above, the attributes Product_Name and Unit_Price are dependent on Product_ID of the Order_Details table and not on the Order_ID. And, remaining attributes of the Order_Details table i.e. Quantity, Order_Date, Address_ID, Country, Zone, and City are dependent on both the Order_ID and Product_ID. Hence, the attributes Product_Name and Unit_Price are separated from the Order_Details Table where Product_ID is the primary key.

2.2.4 3NF

In the third normal form (3NF), the transitive dependency has to be eliminated.

“When an indirect relationship causes functional dependency, it is called Transitive Dependency.

If $P \rightarrow Q$ and $Q \rightarrow R$ are true, then $P \rightarrow R$ is a transitive dependency.” (Onsman, 2018)

In Orders Table,

$\text{Order_ID} \rightarrow \text{Customer_ID} \rightarrow \text{Customer_Name}$

Hence,

Orders (Order_ID (PK), Customer_ID (FK))

Customer (Customer_ID (PK), Customer_Name)

Here, the Order_ID determines the Customer_ID and the Customer_ID determines the Customer_Name. Therefore, to remove the transitive dependency, the Customer_Name is separated from the orders table and stored in a new table i.e., Customer where Customer_ID is the primary key and also the foreign key in the Orders tables.

In Order_Details Table,

$\text{Order_ID} \rightarrow$

$\text{Product_ID} \rightarrow$

$\text{Order_ID}, \text{Product_ID} \rightarrow \text{Quantity}, \text{Order_Date}, \text{Address_ID}$

$\text{Address_ID} \rightarrow \text{Country}, \text{Zone}, \text{City}$

Hence,

Order_Details (Order_ID (PK, FK), Product_ID (PK,FK), Order_Date, Address_ID (FK))

Address (Address_ID (PK), Country, Zone, City)

Here, the Address_ID determines the Country, Zone, City. And the Order_ID and Product_ID determine the values of Quantity, Order_Date, and Address_ID. Therefore, to remove the transitive dependencies, Country, Zone, and City attributes are separated from the Order_Details table and stored in a new table i.e., Address table where Address_ID is the primary key and also the foreign key in the Order_Details table.

Therefore, total tables formed:

Orders (Order_ID (PK), Customer_ID (FK))

Customer (Customer_ID (PK), Customer_Name)

Product(Product_ID (PK), Product_Name, Unit_Price)

Order_Details (Order_ID (PK, FK), Product_ID (PK,FK), Address_ID (FK) ,Quantity , Order_Date)

Address (Address_ID (PK), Country, Zone, City)

3. Entity Relation Diagram

ER Diagram also called Entity Relationship Diagram, is a diagram that shows the relationship between the entity sets stored in a database and helps in the clarification of the logical structure of databases. ER diagrams are created based on the three basic concepts: entities, attributes, and relationships.

The ER Diagram includes many specialized symbols that have their specialized meaning and differentiates itself from the flowchart. The purpose of ER Diagram is to represent the entity framework infrastructure. (Guru99, 2020)

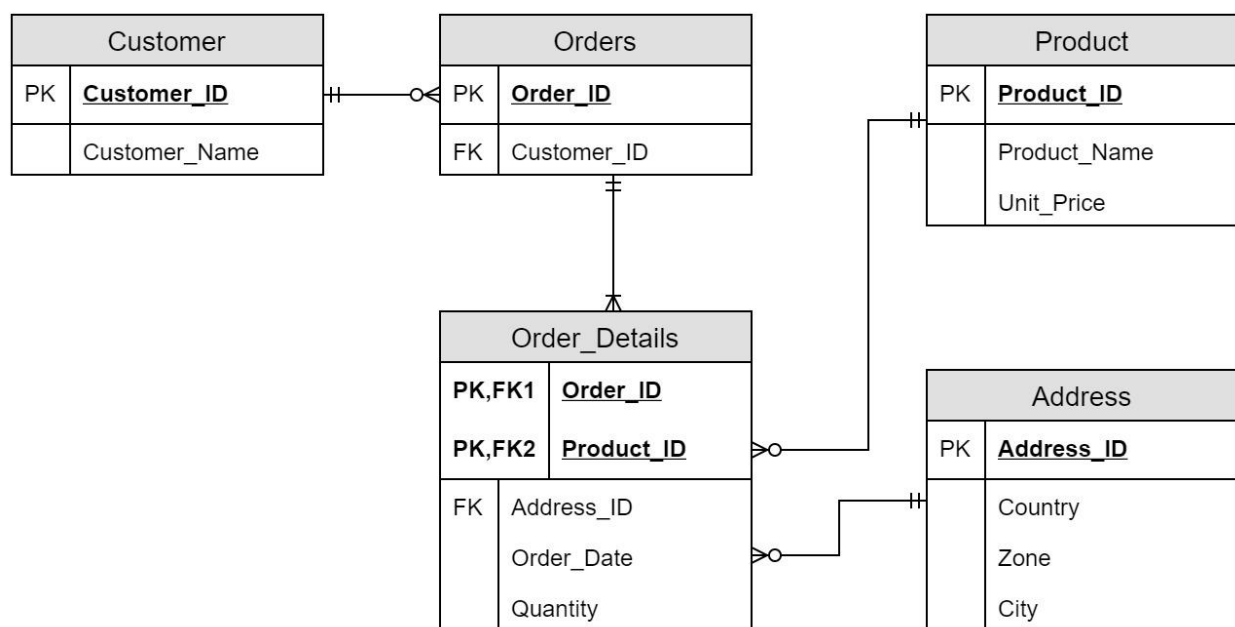


Figure 1: Final ER-Diagram After Normalization

The above ERD is obtained after normalization. Many problems such as data redundancy have been decreased and the data integrity has been improved. It is easier to update and retrieve records of the tables. The bridge entity Order_Details is formed after normalization which helps avoid many-to-many relationships between the tables such as Orders and Product or Orders and Address.

4. Database Implementation

4.1 Tables Generation (DDL Scripts)

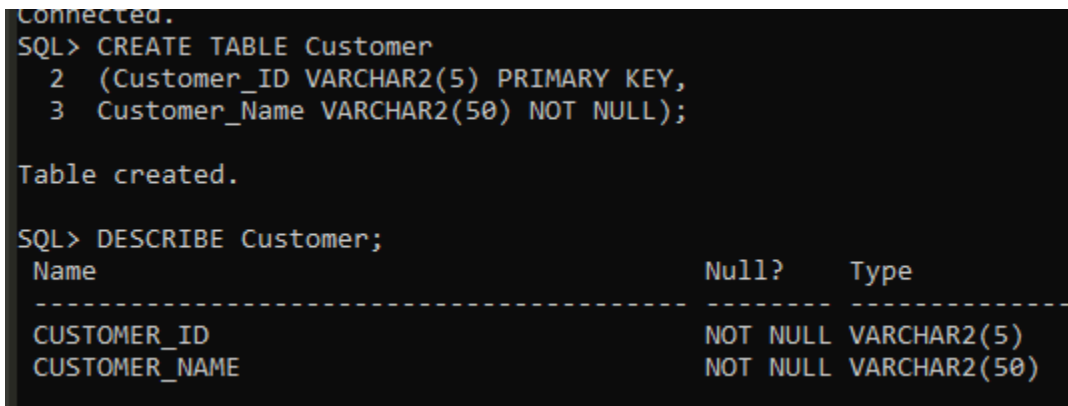
For the creation of tables, CREATE command is used. It falls under the Data Definition Language (DDL). The DDL consists of the SQL commands that are used to define the database schema.

4.1.1 Customer Table

CREATE TABLE Customer

(Customer_ID VARCHAR2(5) PRIMARY KEY,

Customer_Name VARCHAR2(50) NOT NULL);



```

Connected.
SQL> CREATE TABLE Customer
  2  (Customer_ID VARCHAR2(5) PRIMARY KEY,
  3  Customer_Name VARCHAR2(50) NOT NULL);

Table created.

SQL> DESCRIBE Customer;

```

Name	Null?	Type
CUSTOMER_ID	NOT NULL	VARCHAR2(5)
CUSTOMER_NAME	NOT NULL	VARCHAR2(50)

Figure 2: Creating the Customer Table

4.1.2 Product Table

CREATE TABLE Product

(Product_ID VARCHAR2(5) PRIMARY KEY,

Product_Name VARCHAR2(50) NOT NULL,

Unit_Price NUMBER(9,2) NOT NULL);

```

SQL> CREATE TABLE Product
2  (Product_ID VARCHAR2(5) PRIMARY KEY,
3   Product_Name VARCHAR2(50) NOT NULL,
4   Unit_Price NUMBER(9,2) NOT NULL);

Table created.

SQL> DESCRIBE Product;
Name                                Null?    Type
-----
PRODUCT_ID                         NOT NULL VARCHAR2(5)
PRODUCT_NAME                       NOT NULL VARCHAR2(50)
UNIT_PRICE                         NOT NULL NUMBER(9,2)

```

Figure 3: Creating the Product Table

4.1.3 Address Table

```

CREATE TABLE Address
(Address_ID VARCHAR2(10) PRIMARY KEY,
Country VARCHAR2(20) NOT NULL,
Zone VARCHAR2(50) NOT NULL,
City VARCHAR2(20) NOT NULL);

```

```

SQL> CREATE TABLE Address
2  (Address_ID VARCHAR2(10) PRIMARY KEY,
3   Country VARCHAR2(20) NOT NULL,
4   Zone VARCHAR2(50) NOT NULL,
5   City VARCHAR2(20) NOT NULL);

Table created.

SQL> DESCRIBE Address;
Name                                Null?    Type
-----
ADDRESS_ID                         NOT NULL VARCHAR2(10)
COUNTRY                           NOT NULL VARCHAR2(20)
ZONE                              NOT NULL VARCHAR2(50)
CITY                              NOT NULL VARCHAR2(20)

```

Figure 4: Creating the Address Table

4.1.4 Orders Table

CREATE TABLE Orders

(Order_ID VARCHAR2(5) PRIMARY KEY,

Customer_ID VARCHAR2(5) NOT NULL,

FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID));

```
SQL> CREATE TABLE Orders
  2  (Order_ID VARCHAR2(5) PRIMARY KEY,
  3  Customer_ID VARCHAR2(5) NOT NULL,
  4  FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID));

Table created.

SQL>
SQL> DESCRIBE Orders;
Name                                Null?    Type
-----
ORDER_ID                           NOT NULL VARCHAR2(5)
CUSTOMER_ID                         NOT NULL VARCHAR2(5)
```

Figure 5: Creating the Orders Table

4.1.5 Orders_Details Table

CREATE TABLE Order_Details

(Order_ID VARCHAR2(5),

Product_ID VARCHAR2(5),

Address_ID VARCHAR2(10) NOT NULL,

Order_Date DATE NOT NULL,

Quantity INT NOT NULL,

PRIMARY KEY (Order_ID, Product_ID),

FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),

FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),

FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID));

```

SQL> CREATE TABLE Order_Details
2  (Order_ID VARCHAR2(5),
3  Product_ID VARCHAR2(5),
4  Address_ID VARCHAR2(10) NOT NULL,
5  Order_Date DATE NOT NULL,
6  Quantity INT NOT NULL,
7  PRIMARY KEY (Order_ID, Product_ID),
8  FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
9  FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
10 FOREIGN KEY (Address_ID) REFERENCES Address(Address_ID));

Table created.

SQL> DESCRIBE Order_Details;

```

Name	Null?	Type
ORDER_ID	NOT NULL	VARCHAR2(5)
PRODUCT_ID	NOT NULL	VARCHAR2(5)
ADDRESS_ID	NOT NULL	VARCHAR2(10)
ORDER_DATE	NOT NULL	DATE
QUANTITY	NOT NULL	NUMBER(38)

Figure 6: Creating the Order_Details Table

4.2 Populate DB Tables

To insert data into the tables, the INSERT command is used which is a Data Manipulation Language (DML). DML consists of commands that deal with the manipulation of the data present in the database. (GeeksforGeeks, 2019)

“The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

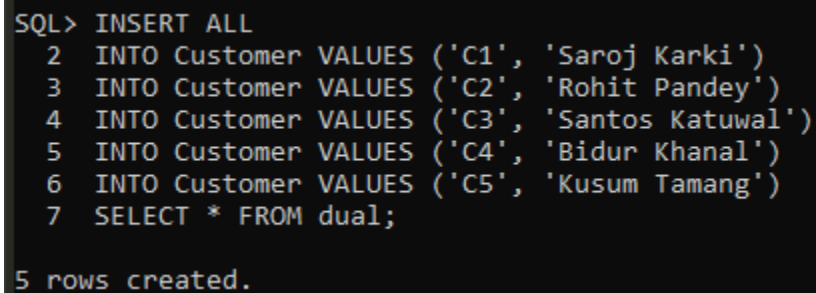
The syntax for the COMMIT command is as follows.

COMMIT;” (Tutorialspoint, 2020)

After the creation of the required tables of the database, the data is inserted into each table.

4.2.1 Inserting values into the Customer table

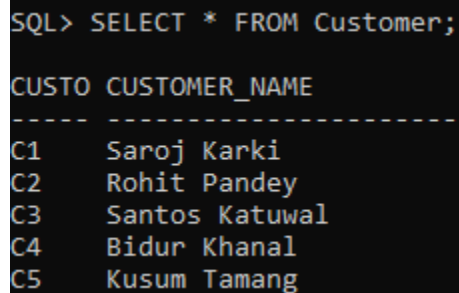
```
INSERT ALL  
  INTO Customer VALUES ('C1', 'Saroj Karki')  
  INTO Customer VALUES ('C2', 'Rohit Pandey')  
  INTO Customer VALUES ('C3', 'Santos Katuwal')  
  INTO Customer VALUES ('C4', 'Bidur Khanal')  
  INTO Customer VALUES ('C5', 'Kusum Tamang')  
SELECT * FROM dual;
```



```
SQL> INSERT ALL  
  2 INTO Customer VALUES ('C1', 'Saroj Karki')  
  3 INTO Customer VALUES ('C2', 'Rohit Pandey')  
  4 INTO Customer VALUES ('C3', 'Santos Katuwal')  
  5 INTO Customer VALUES ('C4', 'Bidur Khanal')  
  6 INTO Customer VALUES ('C5', 'Kusum Tamang')  
  7 SELECT * FROM dual;  
  
5 rows created.
```

Figure 7: Insertion of values into the Customer tables

```
SELECT * FROM Customer;
```



```
SQL> SELECT * FROM Customer;  
  
CUSTO  CUSTOMER_NAME  
-----  
C1      Saroj Karki  
C2      Rohit Pandey  
C3      Santos Katuwal  
C4      Bidur Khanal  
C5      Kusum Tamang
```

Figure 8: Customer Selection Statement

4.2.2 Inserting values into the Product table

INSERT ALL

INTO Product VALUES ('P1','Noodles',30)

INTO Product VALUES ('P2','Rice',900)

INTO Product VALUES ('P3','Coffee',400)

INTO Product VALUES ('P4','Oil',200)

INTO Product VALUES ('P5','Biscuits',60)

INTO Product VALUES ('P6','Dishwasher',90)

SELECT * FROM dual;

```
SQL> INSERT ALL
  2 INTO Product VALUES ('P1','Noodles',30)
  3 INTO Product VALUES ('P2','Rice',900)
  4 INTO Product VALUES ('P3','Coffee',400)
  5 INTO Product VALUES ('P4','Oil',200)
  6 INTO Product VALUES ('P5','Biscuits',60)
  7 INTO Product VALUES ('P6','Dishwasher',90)
  8 SELECT * FROM dual;

6 rows created.
```

Figure 9: Insertion of values into the Product table

SELECT * FROM Product;

```
SQL> SELECT * FROM Product;

PRODU PRODUCT_NAME                                UNIT_PRICE
-----
P1      Noodles                                30
P2      Rice                                900
P3      Coffee                               400
P4      Oil                                 200
P5      Biscuits                               60
P6      Dishwasher                             90

6 rows selected.
```

Figure 10: Product Selection Statement

4.2.3 Inserting values into Address Table

INSERT ALL

INTO Address VALUES ('Add1','Nepal','Koshi','Itahari')

INTO Address VALUES ('Add2','Nepal','Mechi','Jhapa')

INTO Address VALUES ('Add3','Nepal','Bagmati','Kathmandu')

INTO Address VALUES ('Add4','Nepal','Bagmati','Lalitpur')

INTO Address VALUES ('Add5','Nepal','Koshi','Dharan')

SELECT * FROM dual;

```
SQL> INSERT ALL
  2 INTO Address VALUES ('Add1','Nepal','Koshi','Itahari')
  3 INTO Address VALUES ('Add2','Nepal','Mechi','Jhapa')
  4 INTO Address VALUES ('Add3','Nepal','Bagmati','Kathmandu')
  5 INTO Address VALUES ('Add4','Nepal','Bagmati','Lalitpur')
  6 INTO Address VALUES ('Add5','Nepal','Koshi','Dharan')
  7 SELECT * FROM dual;

5 rows created.
```

Figure 11: Insertion of values into the Address table

SELECT * FROM Address;

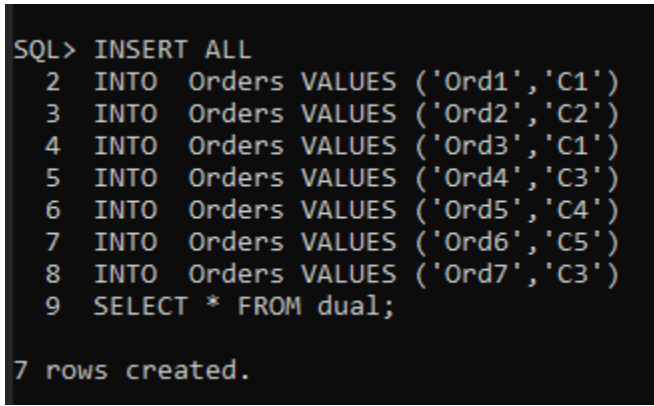
```
SQL> SELECT * FROM Address;
```

ADDRESS_ID	COUNTRY	ZONE	CITY
Add1	Nepal	Koshi	Itahari
Add2	Nepal	Mechi	Jhapa
Add3	Nepal	Bagmati	Kathmandu
Add4	Nepal	Bagmati	Lalitpur
Add5	Nepal	Koshi	Dharan

Figure 12: Address Selection Statement

4.2.4 Inserting values into Orders Table

```
INSERT ALL  
INTO Orders VALUES ('Ord1','C1')  
INTO Orders VALUES ('Ord2','C2')  
INTO Orders VALUES ('Ord3','C1')  
INTO Orders VALUES ('Ord4','C3')  
INTO Orders VALUES ('Ord5','C4')  
INTO Orders VALUES ('Ord6','C5')  
INTO Orders VALUES ('Ord7','C3')  
SELECT * FROM dual;
```

A screenshot of a SQL command prompt window with a black background and white text. The prompt shows a series of SQL commands being entered and executed. The commands are: SQL> INSERT ALL, followed by seven lines of INSERT INTO Orders VALUES statements with different order IDs and categories, and finally a SELECT * FROM dual; statement. The output shows '7 rows created.'

```
SQL> INSERT ALL  
2 INTO Orders VALUES ('Ord1','C1')  
3 INTO Orders VALUES ('Ord2','C2')  
4 INTO Orders VALUES ('Ord3','C1')  
5 INTO Orders VALUES ('Ord4','C3')  
6 INTO Orders VALUES ('Ord5','C4')  
7 INTO Orders VALUES ('Ord6','C5')  
8 INTO Orders VALUES ('Ord7','C3')  
9 SELECT * FROM dual;  
  
7 rows created.
```

Figure 13: Insertion of values into the Orders table

```
SELECT * FROM Orders;
```

```
SQL> SELECT * FROM Orders;

ORDER CUSTO
-----
Ord1   C1
Ord2   C2
Ord3   C1
Ord4   C3
Ord5   C4
Ord6   C5
Ord7   C3

7 rows selected.
```

Figure 14: Orders Selection Statement

4.2.5 Inserting Values into the Order_Details table

INSERT ALL

```
INTO Order_Details VALUES ('Ord1','P3','Add1','20-JAN-20',3)
INTO Order_Details VALUES ('Ord1','P4','Add1','20-JAN-2020',4)
INTO Order_Details VALUES ('Ord2','P5','Add2','05-APR-2020',2)
INTO Order_Details VALUES ('Ord3','P3','Add2','05-APR-2020',1)
INTO Order_Details VALUES ('Ord4','P2','Add3','25-JUN-2020',5)
INTO Order_Details VALUES ('Ord4','P4','Add4','25-JUN-2020',4)
INTO Order_Details VALUES ('Ord5','P1','Add2','25-JUN-2020',4)
INTO Order_Details VALUES ('Ord5','P6','Add1','21-JUL-2020',2)
INTO Order_Details VALUES ('Ord6','P6','Add5','08-JUL-2020',1)
INTO Order_Details VALUES ('Ord7','P3','Add5','03-APR-2020',6)
SELECT * FROM dual;
```

```

SQL> INSERT ALL
  2 INTO Order_Details VALUES ('Ord1','P3','Add1','20-JAN-20',3)
  3 INTO Order_Details VALUES ('Ord1','P4','Add1','20-JAN-2020',4)
  4 INTO Order_Details VALUES ('Ord2','P5','Add2','05-APR-2020',2)
  5 INTO Order_Details VALUES ('Ord3','P3','Add2','05-APR-2020',1)
  6 INTO Order_Details VALUES ('Ord4','P2','Add3','25-JUN-2020',5)
  7 INTO Order_Details VALUES ('Ord4','P4','Add4','25-JUN-2020',4)
  8 INTO Order_Details VALUES ('Ord5','P1','Add2','25-JUN-2020',4)
  9 INTO Order_Details VALUES ('Ord5','P6','Add1','21-JUL-2020',2)
 10 INTO Order_Details VALUES ('Ord6','P6','Add5','08-JUL-2020',1)
 11 INTO Order_Details VALUES ('Ord7','P3','Add5','03-APR-2020',6)
 12 SELECT * FROM dual;

10 rows created.

```

Figure 15: Insertion of values into the Order_Details table

SELECT * FROM Order_Details;

```

SQL>
SQL> SELECT * FROM Order_Details;

ORDER  PRODU  ADDRESS_ID  ORDER_DAT  QUANTITY
-----
Ord1   P3      Add1        20-JAN-20      3
Ord1   P4      Add1        20-JAN-20      4
Ord2   P5      Add2        05-APR-20      2
Ord3   P3      Add2        05-APR-20      1
Ord4   P2      Add3        25-JUN-20      5
Ord4   P4      Add4        25-JUN-20      4
Ord5   P1      Add2        25-JUN-20      4
Ord5   P6      Add1        21-JUL-20      2
Ord6   P6      Add5        08-JUL-20      1
Ord7   P3      Add5        03-APR-20      6

10 rows selected.

```

Figure 16: Order_Details Selection Statement

5. Database Querying for case study

1. Write a SQL statement to add new column in Address Table with column name as District with data type varchar and length 40 and insert data in that column.

```
SQL> ALTER TABLE Address ADD District VARCHAR2(40);

Table altered.

SQL> UPDATE Address
  2 SET District =
  3 CASE Address_ID
  4 WHEN 'Add1' THEN 'Sunsari'
  5 WHEN 'Add2' THEN 'Jhapa'
  6 WHEN 'Add3' THEN 'Kathmandu'
  7 WHEN 'Add4' THEN 'Lalitpur'
  8 WHEN 'Add5' THEN 'Sunsari'
  9 END
10 WHERE Address_ID in ('Add1','Add2','Add3', 'Add4', 'Add5');
```

5 rows updated.

```
SQL> SELECT * FROM Address;
```

ADDRESS_ID	COUNTRY	ZONE	CITY	DISTRICT
Add1	Nepal	Koshi	Itahari	Sunsari
Add2	Nepal	Mechi	Jhapa	Jhapa
Add3	Nepal	Bagmati	Kathmandu	Kathmandu
Add4	Nepal	Bagmati	Lalitpur	Lalitpur
Add5	Nepal	Koshi	Dharan	Sunsari

Figure 17: Query 1

```
ALTER TABLE Address ADD District VARCHAR2(40);
```

```
UPDATE Address
```

```
SET District =
```

```
    CASE Address_ID
```

```
        WHEN 'Add1' THEN 'Sunsari'
```

```
        WHEN 'Add2' THEN 'Jhapa'
```

```
        WHEN 'Add3' THEN 'Kathmandu'
```

```
        WHEN 'Add4' THEN 'Lalitpur'
```

```
        WHEN 'Add5' THEN 'Sunsari'
```

```
    END
```

```
WHERE Address_ID in ('Add1','Add2','Add3', 'Add4', 'Add5');
```

```
SELECT * FROM Address;
```

In the above query, the Address has been altered and a new column District which has data type VARCHAR2(40) is added. And, the empty column has been filled with data using the UPDATE query with the CASE command.

2. Write a SQL statement to find name of the product and total no of that product sales till date.

```
SQL> SELECT P.Product_Name, SUM(OD.Quantity) "Total Sales"
2  FROM Product P
3  JOIN Order_Details OD
4  ON P.Product_ID = OD.Product_ID
5  GROUP BY P.Product_name
6  ORDER BY SUM(OD.Quantity) DESC;

PRODUCT_NAME                                Total Sales
-----
Coffee                                         10
Oil                                             8
Rice                                           5
Noodles                                       4
Dishwasher                                    3
Biscuits                                      2

6 rows selected.
```

Figure 18: Query 2

```
SELECT P.Product_Name, SUM(OD.Quantity) "Total Sales"
FROM Product P
JOIN Order_Details OD
    ON P.Product_ID = OD.Product_ID
GROUP BY P.Product_name
ORDER BY SUM(OD.Quantity) DESC;
```

From the query above, the Product_Name and the total sales made so far is displayed. The SUM function has been used to calculate the total quantities of each product sold. The GROUP BY statement has also been used to group different rows with the same Product_Name into summary rows and ORDER BY statement arranges the values of SUM(OD.Quantity) in descending order as DESC keyword has been used.

3. Write a SQL statement to update any customer name.

```
SQL> SELECT * FROM Customer;

CUSTO CUSTOMER_NAME
-----
C1     Saroj Karki
C2     Rohit Pandey
C3     Santos Katuwal
C4     Bidur Khanal
C5     Kusum Tamang

SQL> UPDATE Customer SET Customer_Name = 'Ram Amatya' WHERE Customer_ID = 'C1';

1 row updated.

SQL> SELECT * FROM Customer;

CUSTO CUSTOMER_NAME
-----
C1     Ram Amatya
C2     Rohit Pandey
C3     Santos Katuwal
C4     Bidur Khanal
C5     Kusum Tamang
```

Figure 19: Query 3

```
SELECT * FROM Customer;
```

```
UPDATE Customer SET Customer_Name = 'Ram Amatya' WHERE Customer_ID = 'C1';
```

```
SELECT * FROM Customer;
```

The UPDATE statement has been used to modify the value of the Customer_Name of the tuples where the value of the Customer_ID is 'C1'. The SELECT statement is used before and after the use of the UPDATE statement to display the changes made by the UPDATE statement.

4. Write a SQL statement to find the order details including product name and customer name who purchase the order.

CUSTOMER_NAME	ORDER	QUANTITY	ORDER_DAT	PRODUCT_NAME

Bidur Khanal	Ord5	4	25-JUN-20	Noodles
Bidur Khanal	Ord5	2	21-JUL-20	Dishwasher
Kusum Tamang	Ord6	1	08-JUL-20	Dishwasher
Ram Amatya	Ord3	1	05-APR-20	Coffee
Ram Amatya	Ord1	3	20-JAN-20	Coffee
Ram Amatya	Ord1	4	20-JAN-20	Oil
Rohit Pandey	Ord2	2	05-APR-20	Biscuits
Santos Katuwal	Ord4	5	25-JUN-20	Rice
Santos Katuwal	Ord4	4	25-JUN-20	Oil
Santos Katuwal	Ord7	6	03-APR-20	Coffee
10 rows selected.				

Figure 20: Query 4

```

SELECT c.Customer_Name, od.Order_ID, od.Quantity, od.Order_Date, p.Product_Name
FROM Customer c
JOIN Orders O
    ON c.Customer_ID = o.Customer_ID
JOIN Order_Details od
    ON o.Order_ID = od.Order_ID
JOIN Product p
    ON p.Product_ID = od.Product_ID
ORDER BY c.Customer_Name;

```

The above query displays the Customer_Name, Product_Name, and details of the orders made such as the Order ID, Quantity and the Order_Date. The four tables Customer, Orders, Order_Details and Product are connected with the use of the JOIN statement. And the records are arranged alphabetically based on the value of the Customer_Name using ORDER BY statement.

5. Write a SQL statement to find most sold product.

```
SQL> SELECT * FROM
  2  (SELECT p.Product_name, SUM(od.Quantity) AS total
  3  FROM Product p
  4  JOIN Order_Details od
  5  on p.Product_id = od.Product_id
  6  GROUP BY p.Product_name
  7  ORDER BY total DESC)
  8  where ROWNUM <= 1;
```

PRODUCT_NAME	TOTAL
Coffee	10

Figure 21: Query 5

```
SELECT * FROM
  (SELECT p.Product_name, SUM(od.Quantity) AS total
  FROM Product p
  JOIN Order_Details od
  on p.Product_id = od.Product_id
  GROUP BY p.Product_name
  ORDER BY total DESC)
where ROWNUM <= 1;
```

In the table above, the subquery returns the Product_Name and the sum of the Quantity (SUM(od.Quantity)) which are grouped based on the product name. The ORDER BY statement has been used with the DESC keyword which displays the SUM(od.Quantity) in descending order. The ROWNUM clause has been used in the main query to display only one record.

6. List all the customer with product name that has been sold at '2020-06-25'

```
SQL> SELECT c.Customer_Name, p.Product_Name , od.Order_Date
2 FROM Customer c
3 JOIN Orders O
4 ON c.Customer_ID = o.Customer_ID
5 JOIN Order_Details od
6 ON o.Order_ID = od.Order_ID
7 JOIN Product p
8 ON p.Product_ID = od.Product_ID
9 WHERE od.Order_Date = TO_DATE('2020-06-25','yyyy-mm-dd');
```

CUSTOMER_NAME	PRODUCT_NAME	ORDER_DAT
Santos Katuwal	Rice	25-JUN-20
Santos Katuwal	Oil	25-JUN-20
Bidur Khanal	Noodles	25-JUN-20

Figure 22: Query 6

```
SELECT c.Customer_Name, p.Product_Name , od.Order_Date
FROM Customer c
JOIN Orders O
    ON c.Customer_ID = o.Customer_ID
JOIN Order_Details od
    ON o.Order_ID = od.Order_ID
JOIN Product p
    ON p.Product_ID = od.Product_ID
WHERE od.Order_Date = TO_DATE('2020-06-25','yyyy-mm-dd');
```

In the above query, the values of the Customer_Name, Product_Name, Order_Date are displayed of the records which have the value of Order_Date column, '2020-06-25'. The TO_DATE function has been used to convert the VARCHAR value to DATE datatype.

7. List down all the customer details with their address who has bought any product more than 2 times.

```
SQL> SELECT DISTINCT c.Customer_ID ,c.Customer_Name, a.Country , a.Zone , a.City FROM Customer c
2 JOIN Orders o
3 ON c.Customer_ID = o.Customer_ID
4 JOIN Order_Details od
5 ON o.Order_ID = od.Order_ID
6 JOIN Address a
7 ON od.Address_ID = a.Address_ID
8 WHERE c.Customer_ID = ANY (SELECT DISTINCT c.Customer_ID FROM Customer c
9 JOIN Orders o
10 ON c.Customer_ID = o.Customer_ID
11 JOIN Order_Details od
12 ON o.Order_ID = od.Order_ID
13 JOIN Address a
14 ON od.Address_ID = a.Address_ID
15 GROUP BY c.Customer_ID, od.Product_ID
16 HAVING SUM(od.Quantity)>2)
17 ORDER BY c.Customer_ID;
```

CUSTO	CUSTOMER_NAME	COUNTRY	ZONE	CITY
C1	Ram Amatya	Nepal	Koshi	Itahari
C1	Ram Amatya	Nepal	Mechi	Jhapa
C3	Santos Katuwal	Nepal	Bagmati	Kathmandu
C3	Santos Katuwal	Nepal	Bagmati	Lalitpur
C3	Santos Katuwal	Nepal	Koshi	Dharan
C4	Bidur Khanal	Nepal	Koshi	Itahari
C4	Bidur Khanal	Nepal	Mechi	Jhapa

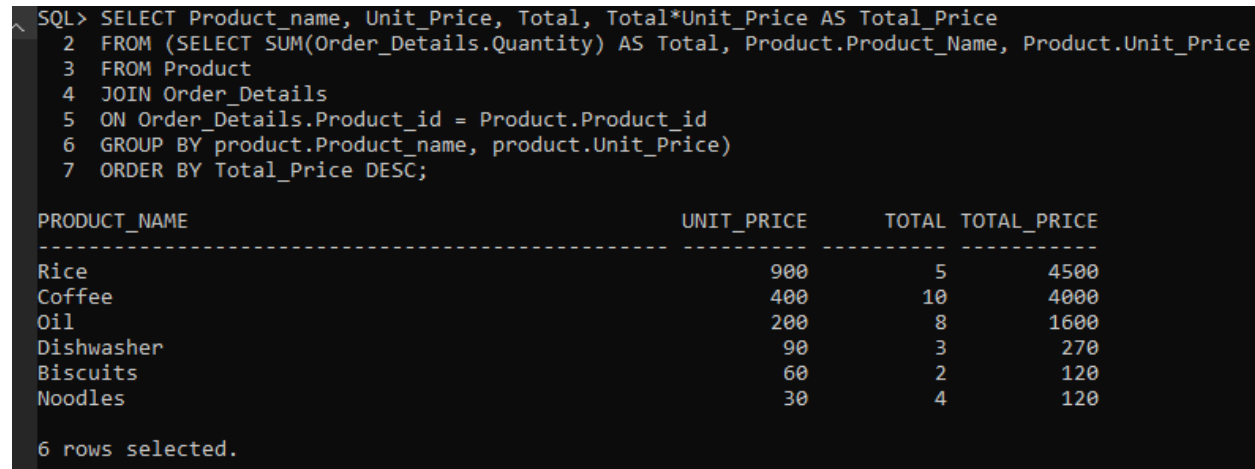
7 rows selected.

Figure 23: Query 7

```
SELECT DISTINCT c.Customer_ID ,c.Customer_Name, a.Country , a.Zone , a.City FROM
Customer c
JOIN Orders o
    ON c.Customer_ID = o.Customer_ID
JOIN Order_Details od
    ON o.Order_ID = od.Order_ID
JOIN Address a
    ON od.Address_ID = a.Address_ID
WHERE c.Customer_ID = ANY (SELECT DISTINCT c.Customer_ID FROM Customer c
    JOIN Orders o
        ON c.Customer_ID = o.Customer_ID
    JOIN Order_Details od
        ON o.Order_ID = od.Order_ID
    JOIN Address a
        ON od.Address_ID = a.Address_ID
GROUP BY c.Customer_ID, od.Product_ID
HAVING SUM(od.Quantity)>2)
ORDER BY c.Customer_ID;
```

The subquery returns distinct values of the Customer_ID of the Customers who have ordered any product more than twice. The main query displays the Customer_ID Customer_Name, Country , Zone , City of the customers whose Customer_ID matches with the values of the Customer_ID column returned from the subquery.

8. List all the product name with total price earned from selling those products.



```
SQL> SELECT Product_name, Unit_Price, Total, Total*Unit_Price AS Total_Price
2  FROM (SELECT SUM(Order_Details.Quantity) AS Total, Product.Product_Name, Product.Unit_Price
3  FROM Product
4  JOIN Order_Details
5  ON Order_Details.Product_id = Product.Product_id
6  GROUP BY product.Product_name, product.Unit_Price)
7  ORDER BY Total_Price DESC;
```

PRODUCT_NAME	UNIT_PRICE	TOTAL	TOTAL_PRICE
Rice	900	5	4500
Coffee	400	10	4000
Oil	200	8	1600
Dishwasher	90	3	270
Biscuits	60	2	120
Noodles	30	4	120

6 rows selected.

Figure 24: Query 8

```
SELECT Product_name, Unit_Price, Total, Total*Unit_Price AS Total_Price
FROM (SELECT SUM(Order_Details.Quantity) AS Total, Product.Product_Name,
Product.Unit_Price
FROM Product
JOIN Order_Details
ON Order_Details.Product_id = Product.Product_id
GROUP BY product.Product_name, product.Unit_Price)
ORDER BY Total_Price DESC;
```

The Product_Name, Unit_Price , Total column which displays the total quantity of the product ordered sold so far and the Total_Price is displayed from the above query. The arithmetic operator ‘*’ is used which multiplies the value of the Total and Unit_Price columns which is the total price earned by selling the product.

9. List the customer with the product name that the customer has bought maximum time.

```
SQL> SELECT customerid "Customer ID", MAX(Productname) "Product Name",MAX (total) "Max Orders"
2 FROM
3 (SELECT SUM(od.Quantity) AS Total, MAX(p.Product_name) AS Productname, c.Customer_ID AS customerid FROM Customer c
4 JOIN Orders o
5 ON c.Customer_ID = o.Customer_ID
6 JOIN Order_Details od
7 ON o.Order_ID = od.Order_ID
8 JOIN Product p
9 ON p.Product_id = od.Product_id
10 JOIN Address a
11 ON od.Address_ID = a.Address_ID
12 GROUP BY c.Customer_ID, od.Product_ID)
13 GROUP BY customerid
14 ORDER BY customerid;
```

Custo	Product Name	Max Orders
C1	Oil	4
C2	Biscuits	2
C3	Rice	6
C4	Noodles	4
C5	Dishwasher	1

Figure 25: Query 9

SELECT customerid "Customer ID", MAX(Productname) "Product Name",MAX (total) "Max Orders"

FROM

(SELECT SUM(od.Quantity) AS Total, MAX(p.Product_name) AS Productname, c.Customer_ID AS customerid FROM Customer c

JOIN Orders o

ON c.Customer_ID = o.Customer_ID

JOIN Order_Details od

ON o.Order_ID = od.Order_ID

JOIN Product p

ON p.Product_id = od.Product_id

JOIN Address a

ON od.Address_ID = a.Address_ID

GROUP BY c.Customer_ID, od.Product_ID)

GROUP BY customerid

ORDER BY customerid;

The above query displays the customer id, the name, and the quantity of the product that the customer has ordered the maximum amount of time.

6. Database Dump file Creation

6.1 Screenshot of the Dump File created

EXP DatabaseExamination/Islington file = coursework.dmp

```

C:\Users\Pratik>D:
D:\>CD D:\Downloads\Islington\Database\PratikCW
D:\Downloads\Islington\Database\PratikCW>EXP DatabaseExamination/islington file = coursework.dmp
Export: Release 11.2.0.2.0 - Production on Sun Jan 31 23:21:13 2021
Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
server uses AL32UTF8 character set (possible charset conversion)

About to export specified users ...
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user DATABASEEXAMINATION
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user DATABASEEXAMINATION
About to export DATABASEEXAMINATION's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export DATABASEEXAMINATION's tables via Conventional Path ...
. . exporting table ADDRESS 5 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table CUSTOMER 5 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table ORDERS 7 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table ORDER_DETAILS 10 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table PRODUCT 6 rows exported
EXP-00091: Exporting questionable statistics.

```

Figure 26: Creation of dump file - Part 1


```
EXP-00091: Exporting questionable statistics.  
. exporting synonyms  
. exporting views  
. exporting stored procedures  
. exporting operators  
. exporting referential integrity constraints  
. exporting triggers  
. exporting indextypes  
. exporting bitmap, functional and extensible indexes  
. exporting posttables actions  
. exporting materialized views  
. exporting snapshot logs  
. exporting job queues  
. exporting refresh groups and children  
. exporting dimensions  
. exporting post-schema procedural objects and actions  
. exporting statistics  
Export terminated successfully with warnings.  
  
D:\Downloads\Islington\Database\PratikCW>_
```

Figure 27: Creation of dump file - Part 2

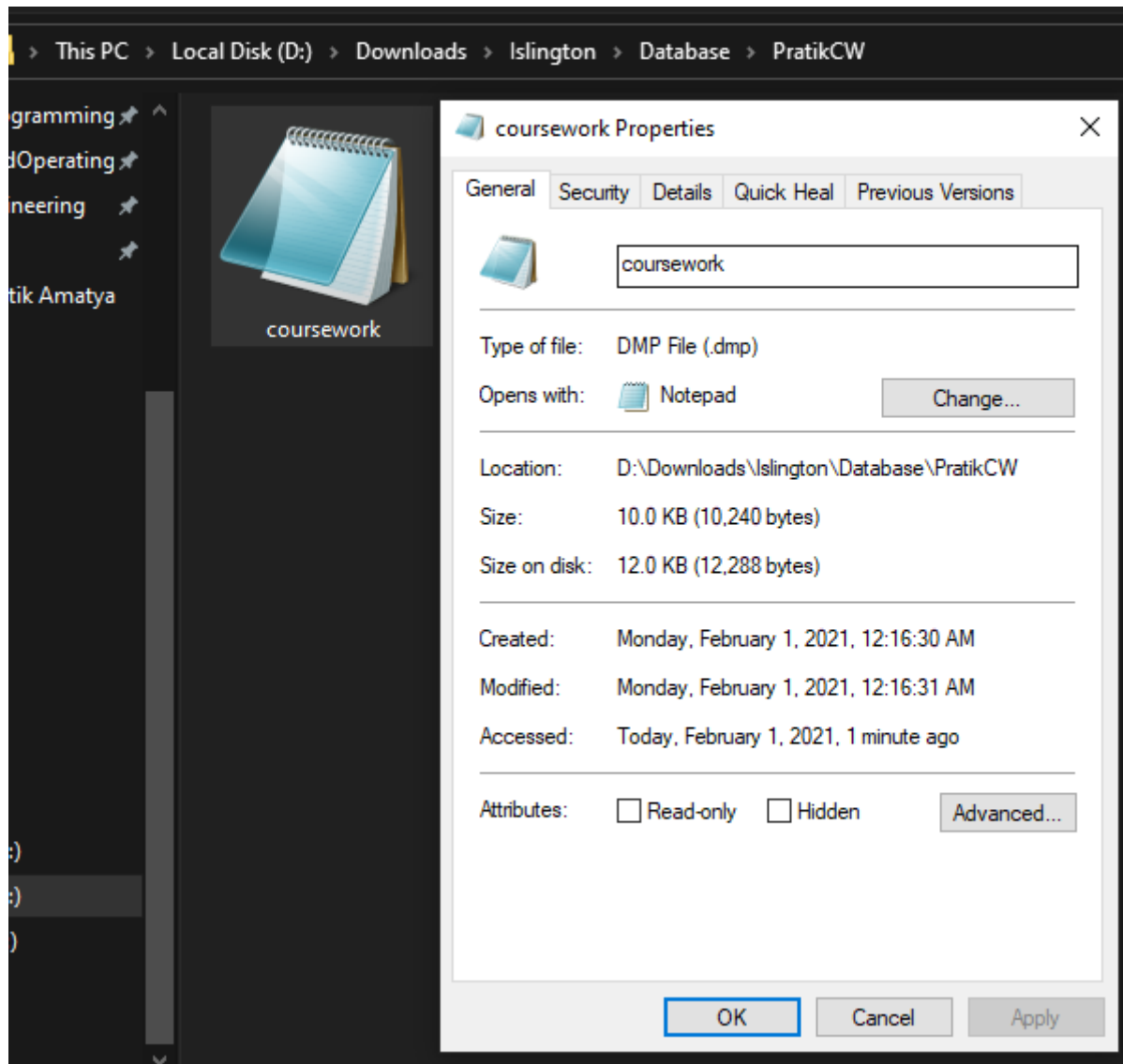


Figure 28: Creation of actual dump file

The dump file 'coursework.dmp' is created after the above process which stores the structure and data of the database.

6.2 Command for deleting all the files in the database with the sequential order of deletion

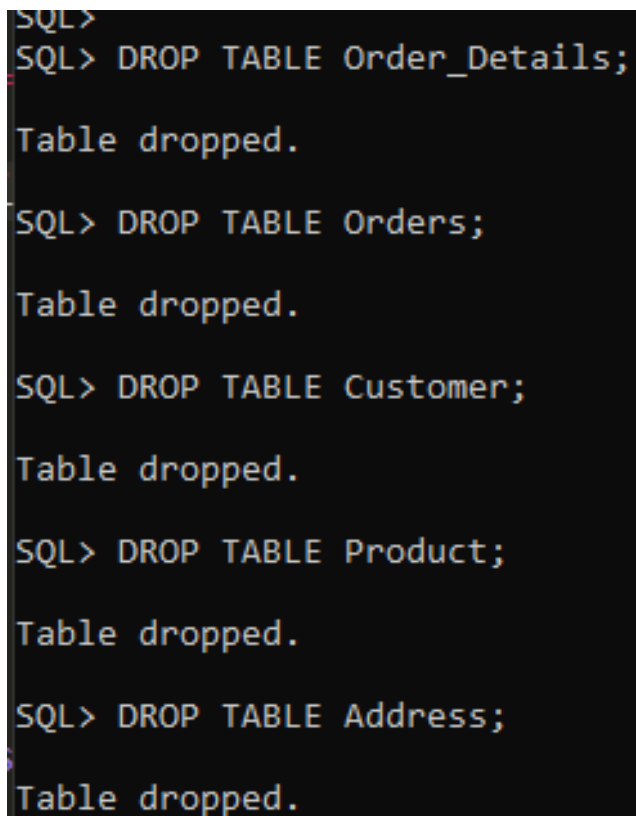
DROP TABLE Order_Details;

DROP TABLE Orders;

DROP TABLE Customer;

DROP TABLE Product;

DROP TABLE Address;



```
SQL>  
SQL> DROP TABLE Order_Details;  
Table dropped.  
SQL> DROP TABLE Orders;  
Table dropped.  
SQL> DROP TABLE Customer;  
Table dropped.  
SQL> DROP TABLE Product;  
Table dropped.  
SQL> DROP TABLE Address;  
Table dropped.
```

Figure 29: Dropping all the tables

7. Bibliography

- Brumm, B. (2017) *How to Handle a Many-to-Many Relationship in Database Design - DZone Database* [Online]. Available from: <https://dzone.com/articles/how-to-handle-a-many-to-many-relationship-in-datab> [Accessed 1 December 2020].
- Castro, K. (2018) *Many-to-Many Relationship in DBMS* [Online]. Available from: <https://www.tutorialspoint.com/Many-to-Many-Relationship-in-DBMS> [Accessed 1 December 2020].
- FreeTutes. (n.d.) *E-R Model concept* [Online]. Available from: <https://www.freetutes.com/systemanalysis/sa7-e-r-model-concept.html> [Accessed 1 December 2020].
- GeeksforGeeks. (2019) *SQL _ DDL, DQL, DML, DCL and TCL Commands - GeeksforGeeks* [Online]. Available from: <https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/> [Accessed 17 December 2020].
- Guru99. (2020) *ER Diagram_ Entity Relationship Diagram Model _ DBMS Example* [Online]. Available from: <https://www.guru99.com/er-diagram-tutorial-dbms.html> [Accessed 1 December 2020].
- Guru99. (2020) *What is Normalization 1NF, 2NF, 3NF, BCNF Database Example* [Online]. Available from: <https://www.guru99.com/database-normalization.html> [Accessed 3 December 2020].
- Islington. (2018) *Vision, Mission, Values - Islington College* [Online]. Available from: <https://islington.edu.np/about/vision-mission-values/> [Accessed 7 December 2020].
- My Reading Room. (2016) *Completeness Constraint* [Online]. Available from: [http://www.myreadingroom.co.in/notes-and-studymaterial/65-dbms/493-completeness-constraint.html#:~:text=Partial%20completeness%20\(symbolized%20by%20a,not%20members%20of%20any%20subtype.&text=A%20double%20horizontal%20line%20under%20the%20circle%20represen](http://www.myreadingroom.co.in/notes-and-studymaterial/65-dbms/493-completeness-constraint.html#:~:text=Partial%20completeness%20(symbolized%20by%20a,not%20members%20of%20any%20subtype.&text=A%20double%20horizontal%20line%20under%20the%20circle%20represen) [Accessed 12 December 2020].
- Onsman, A. (2018) *Transitive dependency in DBMS* [Online]. Available from: <https://www.tutorialspoint.com/Transitive-dependency-in-DBMS> [Accessed 13 December 2020].
- Oracle. (2020) *Tables for Supertype and Subtype Entities in Oracle Communications Data Model* [Online]. Available from: https://docs.oracle.com/cd/E29633_01/CDMOG/GUID-80902404-F909-4884-B5AE-

[B0BE6EEE0781.htm#:~:text=A%20subtype%20discriminator%20is%20a%20one%20of%20the%20subtypes](https://www.cs.dartmouth.edu/~cs61/Lectures/05%20-%20Advanced%20Data%20Modeling/05%20-%20Advanced%20Data%20Modeling.html#:~:text=A%20subtype%20discriminator%20is%20a%20one%20of%20the%20subtypes). [Accessed 11 December 2020].

Palmer, C.C. (2020) *Lecture 05* [Online]. Available from: <https://www.cs.dartmouth.edu/~cs61/Lectures/05%20-%20Advanced%20Data%20Modeling/05%20-%20Advanced%20Data%20Modeling.html#:~:text=A%20subtype%20discriminator%20is%20a%20one%20of%20the%20subtypes>. [Accessed 11 December 2020].

Tutorialspoint. (2020) *ER Model - Basic Concepts - Tutorialspoint* [Online]. Available from: https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm [Accessed 1 December 2020].

Tutorialspoint. (2020) *SQL - Foreign Key - Tutorialspoint* [Online]. Available from: <https://www.tutorialspoint.com/sql/sql-foreign-key.htm> [Accessed 1 December 2020].

Tutorialspoint. (2020) *SQL - Transactions - Tutorialspoint* [Online]. Available from: <https://www.tutorialspoint.com/sql/sql-transactions.htm#:~:text=to%20the%20database.-,The%20COMMIT%20command%20is%20the%20transactional%20command%20used%20to%20save,last%20COMMIT%20or%20ROLLBACK%20command>. [Accessed 17 December 2020].