AWH ENGINEERING COLLEGE

KOZHIKODE – 8

AWH ENGINEERING COLLEGE
CALICUT, KERALA, INDIA

CSL333 DATABASE MANAGEMENT SYSTEMS LAB
LABORATORY RECORD

Name ---

Roll No --

Branch -- Semester ----------------------------------

LIST OF EXPERIMENTS CONDUCTED

SI. No	EXPERIMENTS	Date of Performance	Page Number	Marks
1	Exercise - 1		1	
2	Exercise - 2		4	
3	Exercise - 3		7	
4	Exercise - 4		11	
5	Exercise - 5		12	
6	Exercise - 6		19	
7	Exercise - 7		20	
8	Exercise - 8		25	
9	Exercise - 9		26	
10	Exercise - 10		29	
11	Exercise - 11		34	
12	Exercise - 12		35	
13	Exercise - 13		36	
14	Exercise - 14		38	
15	Exercise - 15		39	
16	Exercise - 16		40	
17	Exercise - 17		41	
18	Exercise - 18		42	
19	Exercise - 19		46	
20	Exercise - 20		47	
21	Exercise - 21		48	

Part A

Exercise - 1

Design a normalized database schema for the following requirement.

The requirement: A library wants to maintain the record of books, members, book issue, book return, and fines collected for late returns, in a database. The database can be loaded with book information. Students can register with the library to be a member. Books can be issued to students with a valid library membership. A student can keep an issued book with him/her for a maximum period of two weeks from the date of issue, beyond which a fine will be charged. Fine is calculated based on the delay in days of return. For 0-7 days: Rs 10, For 7 – 30 days: Rs 100, and for days above 30 days: Rs 10 will be charged per day.

Sample Database Design

BOOK (Book_Id, Title, Language_Id, MRP, Publisher_Id, Published_Date, Volume, Status) // Language_Id, Publisher_Id are FK (Foreign Key)

AUTHOR(Author_Id, Name, Email, Phone_Number, Status)

BOOK_AUTHOR(Book_Id, Author_Id) // many-to-many relationship, both columns are PKFK (Primary Key and Foreign Key)

PUBLISHER(Publisher_id, Name, Address)

MEMBER(Member_Id, Name, Branch_Code, Roll_Number, Phone_Number, Email_Id, Date_of_Join, Status)

BOOK_ISSUE(Issue_Id, Date_Of_Issue, Book_Id, Member_Id, Expected_Date_Of_Return, Status) // Book+Id and Member_Id are FKs

BOOK_RETURN(Issue_Id, Actual_Date_Of_Return, LateDays, LateFee) // Issue_Id is PK and FK

LANGUAGE(Language_id, Name) //Static Table for storing permanent data

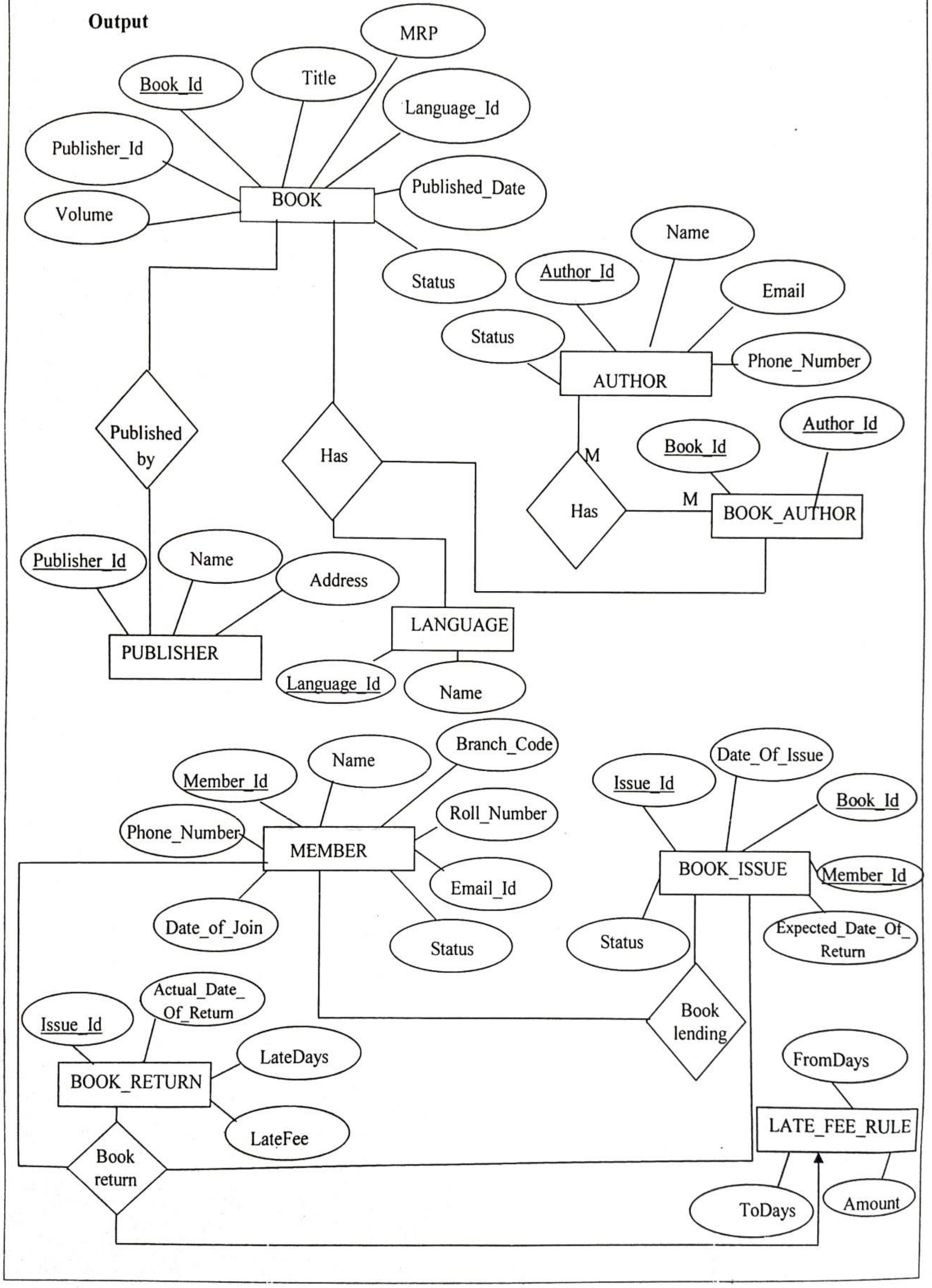
LATE_FEE_RULE(FromDays, ToDays, Amount) // Composite Key

EXERCISES

1. Create a normalized database design with proper tables, columns, column types, and constraints

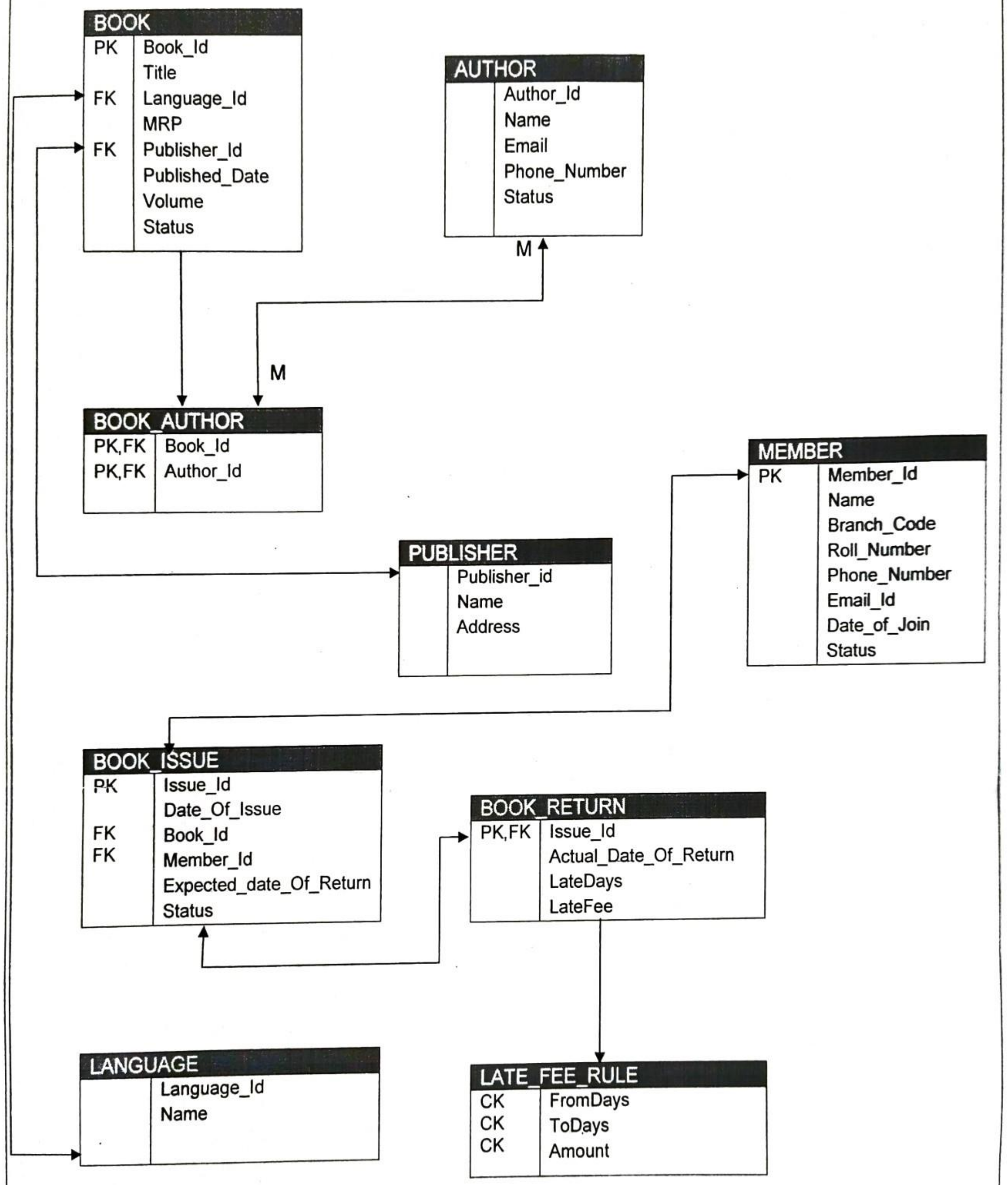
2. Create an ER diagram for the above database design

Output



MRP
Book_Id
Title
Language_Id
Publisher_Id
Published_Date
Volume
BOOK
Status
Name
Author_Id
Email
Status
Phone_Number
AUTHOR
Published by
Has
Author_Id
Book_Id
M
Has
M
BOOK_AUTHOR
Publisher_Id
Name
Address
LANGUAGE
PUBLISHER
Language_Id
Name
Branch_Code
Name
Member_Id
Date_Of_Issue
Issue_Id
Book_Id
Roll_Number
Phone_Number
MEMBER
BOOK_ISSUE
Member_Id
Email_Id
Date_of_Join
Status
Status
Expected_Date_Of_Return
Actual_Date_Of_Return
Issue_Id
Book lending
LateDays
FromDays
BOOK_RETURN
LATE_FEE_RULE
LateFee
Book return
ToDays
Amount

Output


BOOK
PK Book_Id
Title
FK Language_Id
MRP
FK Publisher_Id
Published_Date
Volume
Status
AUTHOR
Author_Id
Name
Email
Phone_Number
Status
M
M
BOOK_AUTHOR
PK,FK Book_Id
PK,FK Author_Id
MEMBER
PK Member_Id
Name
Branch_Code
Roll_Number
Phone_Number
Email_Id
Date_of_Join
Status
PUBLISHER
Publisher_id
Name
Address
BOOK_ISSUE
PK Issue_Id
Date_Of_Issue
FK Book_Id
FK Member_Id
Expected_date_Of_Return
Status
BOOK_RETURN
PK,FK Issue_Id
Actual_Date_Of_Return
LateDays
LateFee
LANGUAGE
Language_Id
Name
LATE_FEE_RULE
CK FromDays
CK ToDays
CK Amount

Exercise - 2

Create the following table.

Table name: Customers

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds	Maria	Obere	Berlin	12209	Germany
2	Ana	Ana	Avda	México	05021	Mexico
3	Antonio M	Antonio	Mataderos	México	05023	Mexico
4	Around	Thomas	Hanover Sq.	London	WA1 1DP	UK
5	Berglunds	Christina	Berguvsvägen	Lulea	S-958 22	Sweden
6	Blauer	Hanna	Forsterstr	Mannheim	68306	Germany
7	Blondel	Frédérique	Pplace Kléber	Strasbourg	67000	France

Some of the most important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

1. SELECT * FROM Customers;
2. SELECT DISTINCT Country FROM Customers;
3. SELECT COUNT(DISTINCT Country) FROM Customers;
4. SELECT * FROM Customers
   WHERE Country='Mexico';

The following SQL statement selects all the records in the "Customers" table:

- SELECT * FROM Customers;

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

- SELECT DISTINCT Country FROM Customers;

The following SQL statement lists the number of different (distinct) customer countries:

- SELECT COUNT(DISTINCT Country) FROM Customers;

The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Clause Example

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

SELECT * FROM Customers

WHERE CustomerID=1

Operators in the WHERE Clause

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Exercise - 3

Create the following table.

Table name: Products

ProductID	**ProductName**	**Price**
1	Alfreds	18
2	Ana	35
3	Antonio M	30
4	Around	45
5	Berglunds	55
6	Blauer	20
7	Blondel	10

Operator = example:

SELECT * FROM Products

WHERE Price = 18;

Operator > example:

SELECT * FROM Products

WHERE Price > 30;

Operator < example:

SELECT * FROM Products

WHERE Price < 30;

Operator >= example:

SELECT * FROM Products

WHERE Price >= 30;

Operator <= example:

ELECT * FROM Products

WHERE Price <= 30;

Operator <> example:

SELECT * FROM Products

WHERE Price <> 18;

Operator BETWEEN example:

SELECT * FROM Products

WHERE Price BETWEEN 50 AND 60;

Operator LIKE example:

SELECT * FROM Customers

WHERE City LIKE 's%';

Select * from Customers

Where CustomerName Like ‘%a’

Select * from Customers

Where CustomerName Like ‘%ro%’ → Find any value that have ‘or’ in any position.

Select * from Customers

Where CustomerName Like ‘_r%’ → Find any value that have ‘r’ in second position.

Select * from Customers

Where CustomerName Like ‘a_%’ → Find any value that have starts with a and are at least 2 characters in length.

Select * from Customers

Where CustomerName Like ‘a_ _%’ → Find any value that have starts with a and are at least 3 characters in length.

Where CustomerName Like ‘a%o’ → Find any value that have starts with a and ends with ‘o’.

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

The AND operator displays a record if all the conditions separated by AND are TRUE.

The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition3 ...;

OR Syntax

SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;

NOT Syntax

SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;

AND Example

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Example

```
SELECT * FROM Customers
WHERE City='Berlin' OR City=' Lulea ';
```

NOT Example

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC;

ORDER BY Example

```
SELECT * FROM Customers
ORDER BY Country;
```

ORDER BY DESC Example

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

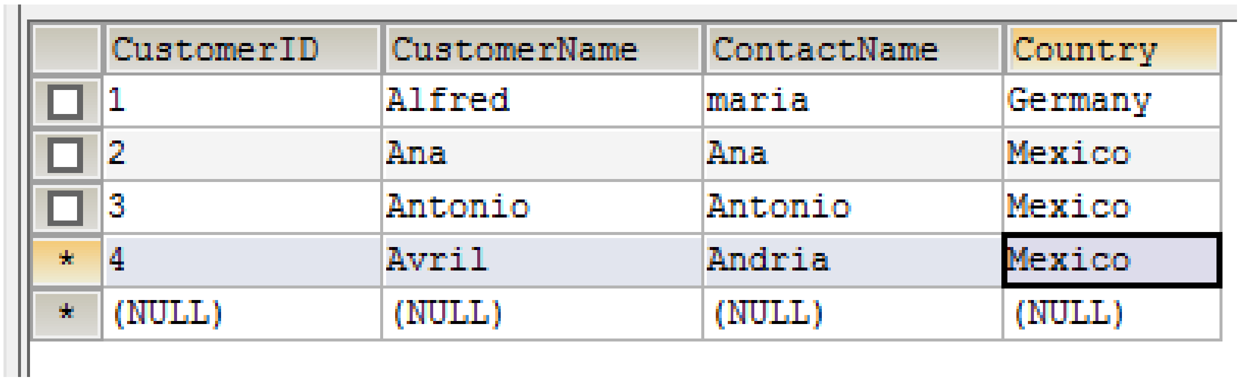
Exercise - 4

Rollno	**Name**	**Age**	**Dept**	**Marks**
1	Ram	18	CS	700
2	Sita	35	EC	300
3	Asha	30	CS	400

1. Create the above table students
2. Select * from students order by Marks
3. Select Name from students where Dept=’CS’ and marks>500
4. Select Name from students where Marks>(select(avg(marks))from students)
5. Select * from students where Rollno between 1 and 3
6. Select Name, Rollno from students where dept like ‘s%’
7. Select Name, Rollno from students where Rollno>2 AND Dept=’CS’
8. Select Name, Rollno from students where Rollno>2 OR Dept=’CS’
9. Alter table students add Subject Varchar(10)
10. Update students set Subject=’DBMS’ where Rollno=1;
11. Update students set Subject=’STLD’ where Rollno=2
12. Select * from students.
13. Delete from students where Rollno=3
14. Select * from students

Exercise – 5

Create the following table:



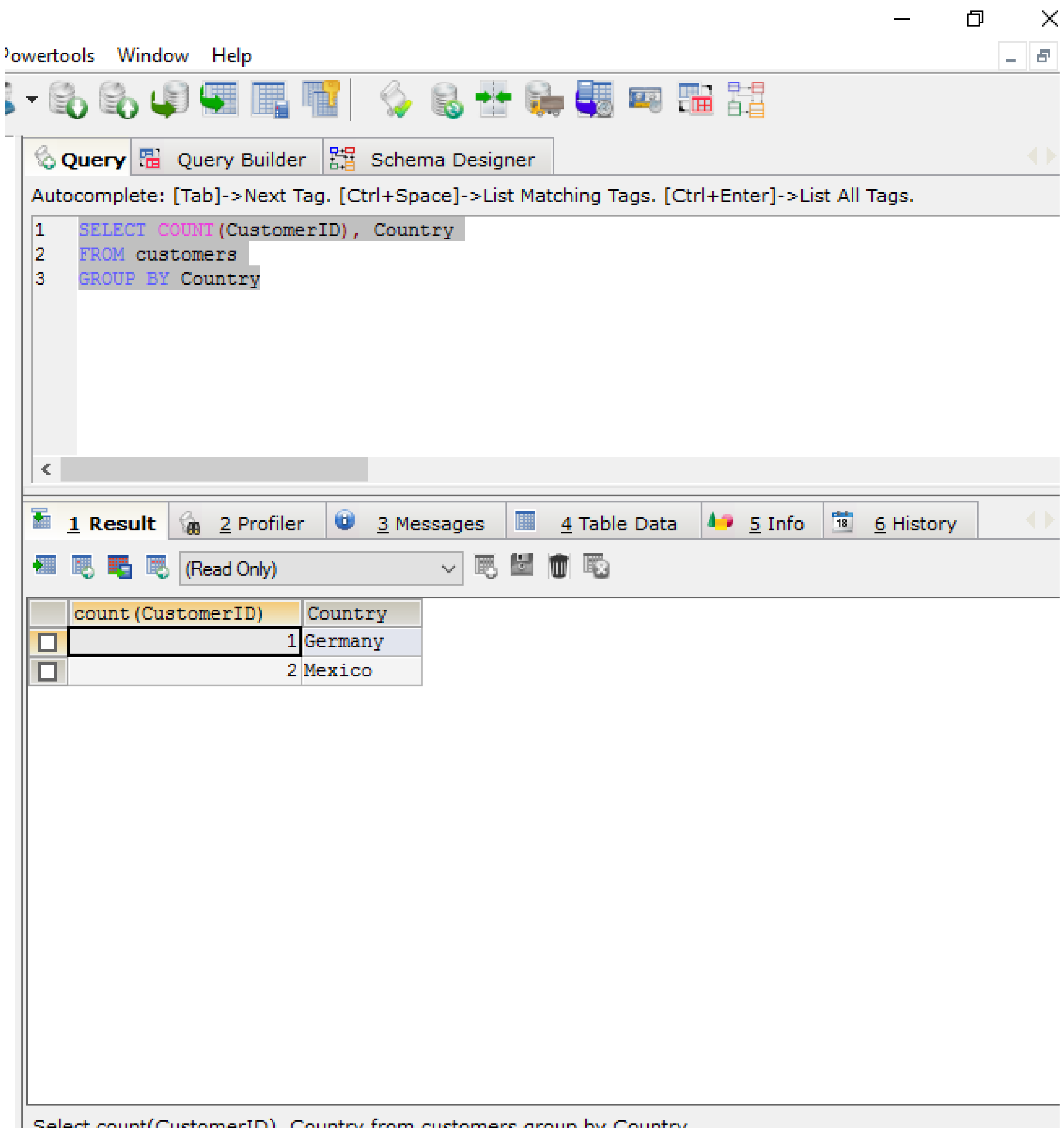
	CustomerID	CustomerName	ContactName	Country
☐	1	Alfred	maria	Germany
☐	2	Ana	Ana	Mexico
☐	3	Antonio	Antonio	Mexico
*	4	Avril	Andria	Mexico
*	(NULL)	(NULL)	(NULL)	(NULL)

Display the output of the following queries:

1. SELECT COUNT(CustomerID), Country

   FROM customers

   GROUP BY Country



```
SELECT COUNT(CustomerID), Country
FROM customers
GROUP BY Country
```

count(CustomerID)	Country
1	Germany
2	Mexico

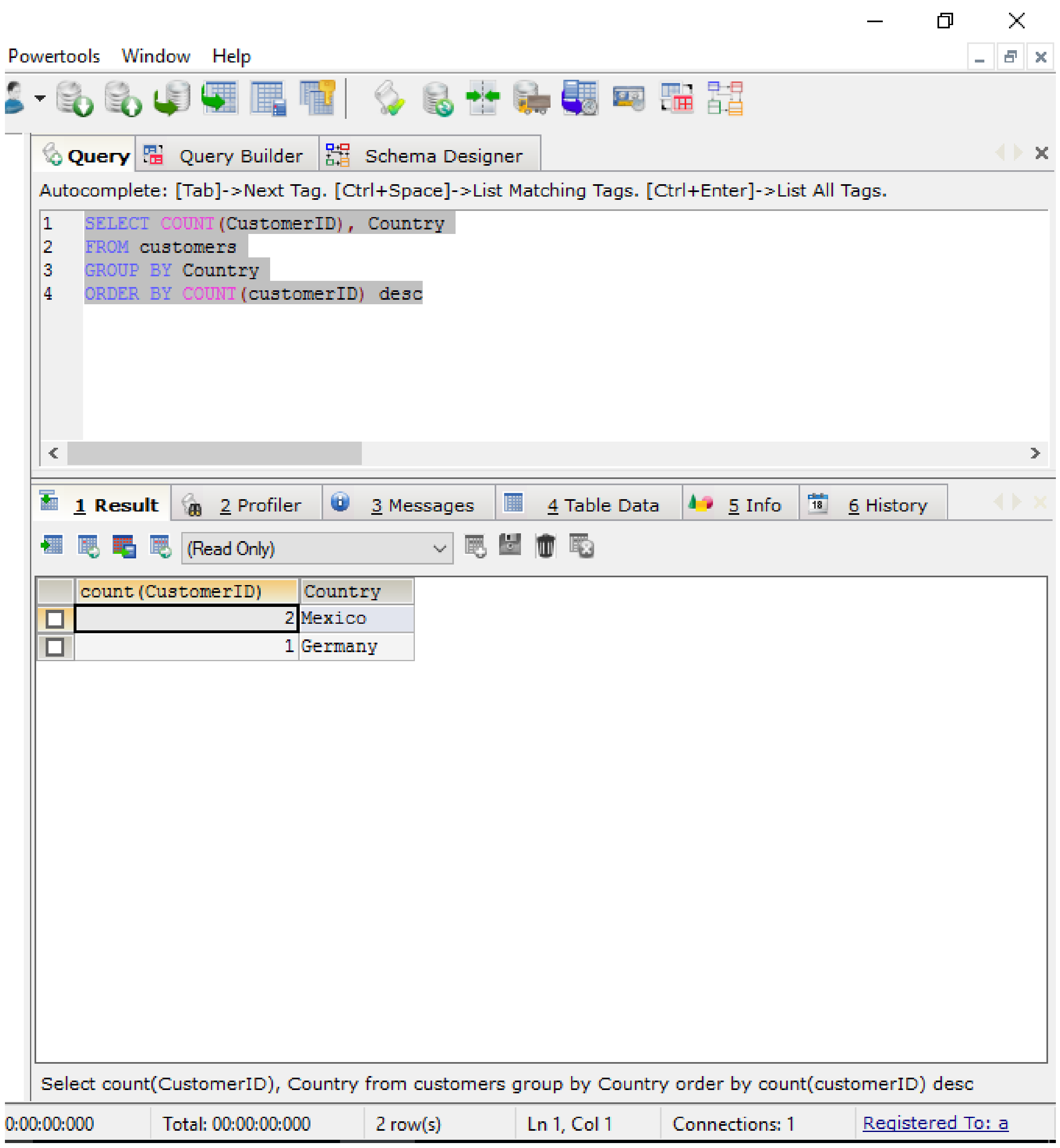
Select count(CustomerID), Country from customers group by Country

2. SELECT COUNT(CustomerID), Country

FROM customers

GROUP BY Country

ORDER BY COUNT(customerID) desc



Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

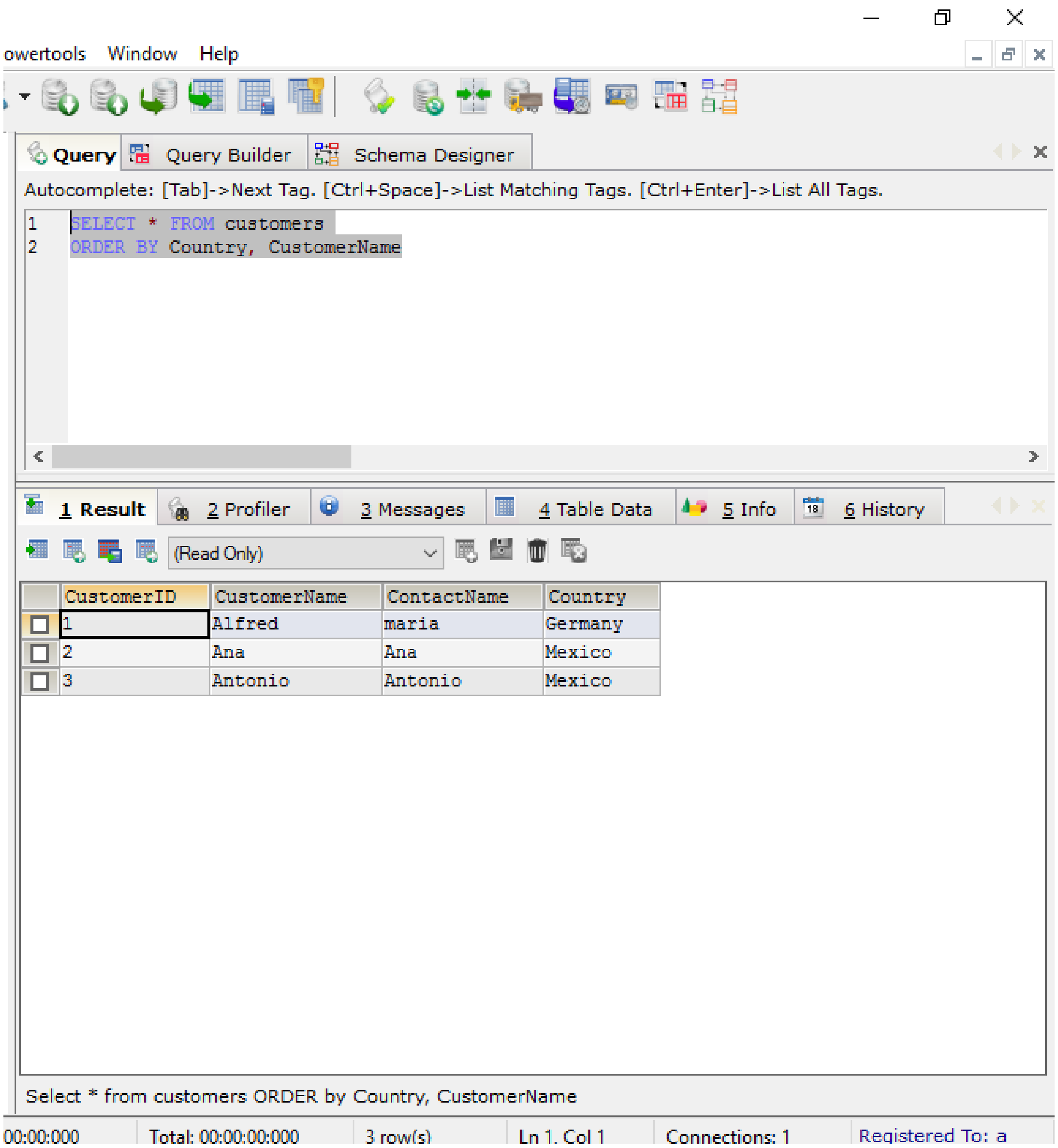
```
SELECT COUNT(CustomerID), Country
FROM customers
GROUP BY Country
ORDER BY COUNT(customerID) desc
```

count(CustomerID)	Country
2	Mexico
1	Germany

Select count(CustomerID), Country from customers group by Country order by count(customerID) desc

0:00:00:000 Total: 00:00:00:000 2 row(s) Ln 1, Col 1 Connections: 1 Registered To: a

3. SELECT * FROM customers
   ORDER BY Country, CustomerName



Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

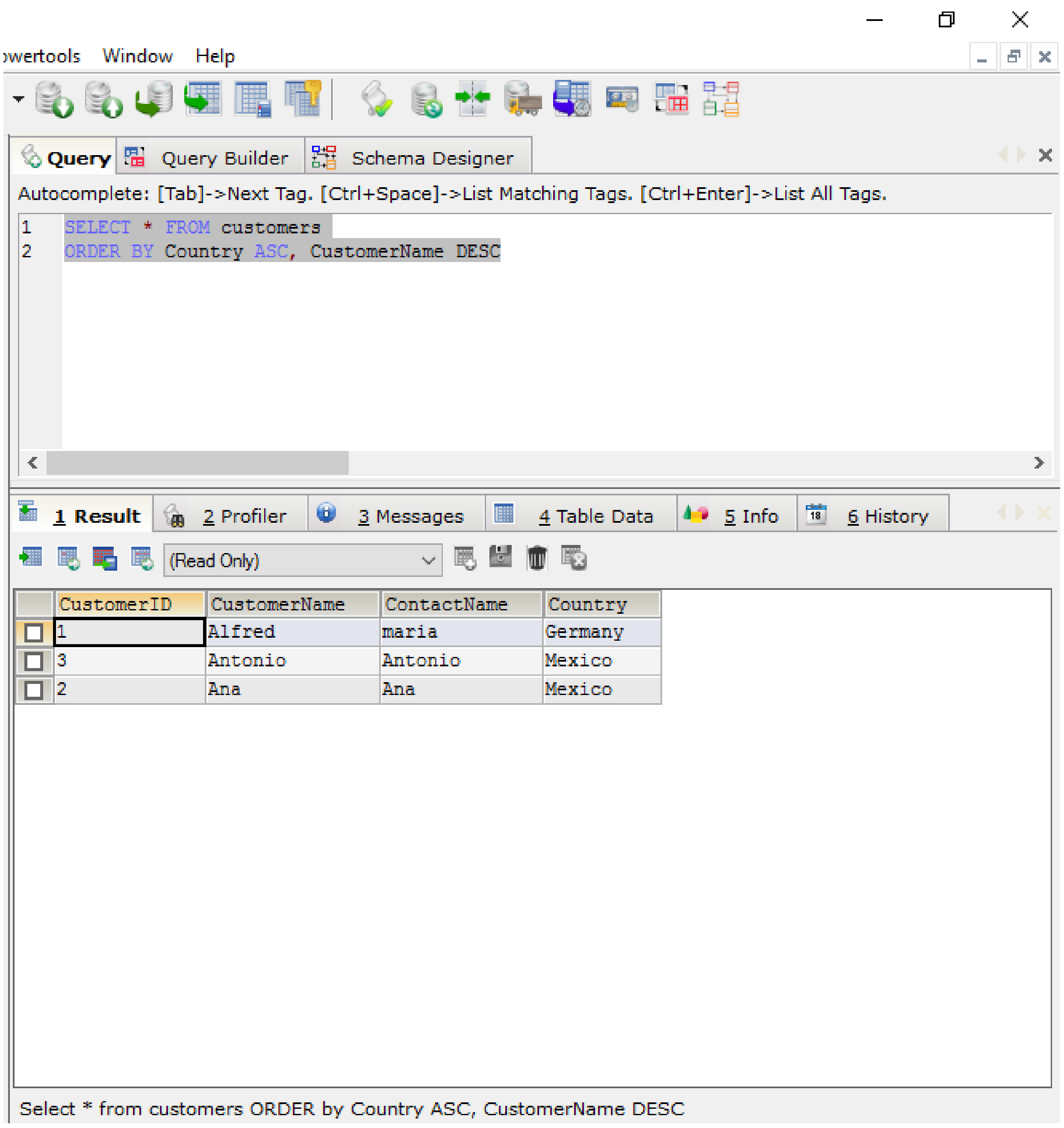
```
SELECT * FROM customers
ORDER BY Country, CustomerName
```

CustomerID	CustomerName	ContactName	Country
1	Alfred	maria	Germany
2	Ana	Ana	Mexico
3	Antonio	Antonio	Mexico

Select * from customers ORDER by Country, CustomerName

00:00:000 Total: 00:00:00:000 3 row(s) Ln 1, Col 1 Connections: 1 Registered To: a

4. SELECT * FROM customers
   ORDER BY Country ASC, CustomerName DESC



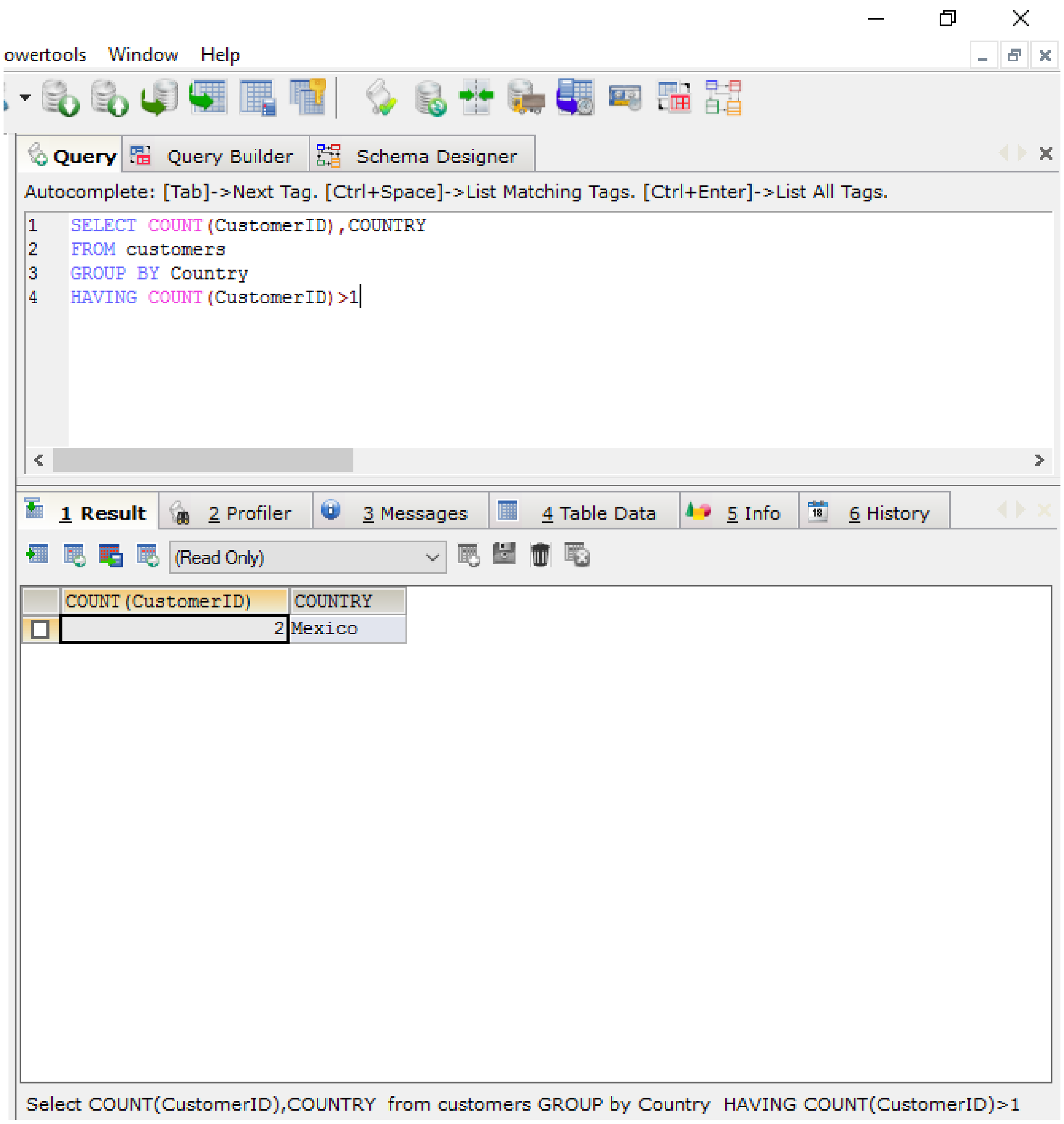
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

```
SELECT * FROM customers
ORDER BY Country ASC, CustomerName DESC
```

CustomerID	CustomerName	ContactName	Country
1	Alfred	maria	Germany
3	Antonio	Antonio	Mexico
2	Ana	Ana	Mexico

Select * from customers ORDER by Country ASC, CustomerName DESC

5. SELECT COUNT(CustomerID),COUNTRY
   FROM customers
   GROUP BY Country
   HAVING COUNT(CustomerID)>1



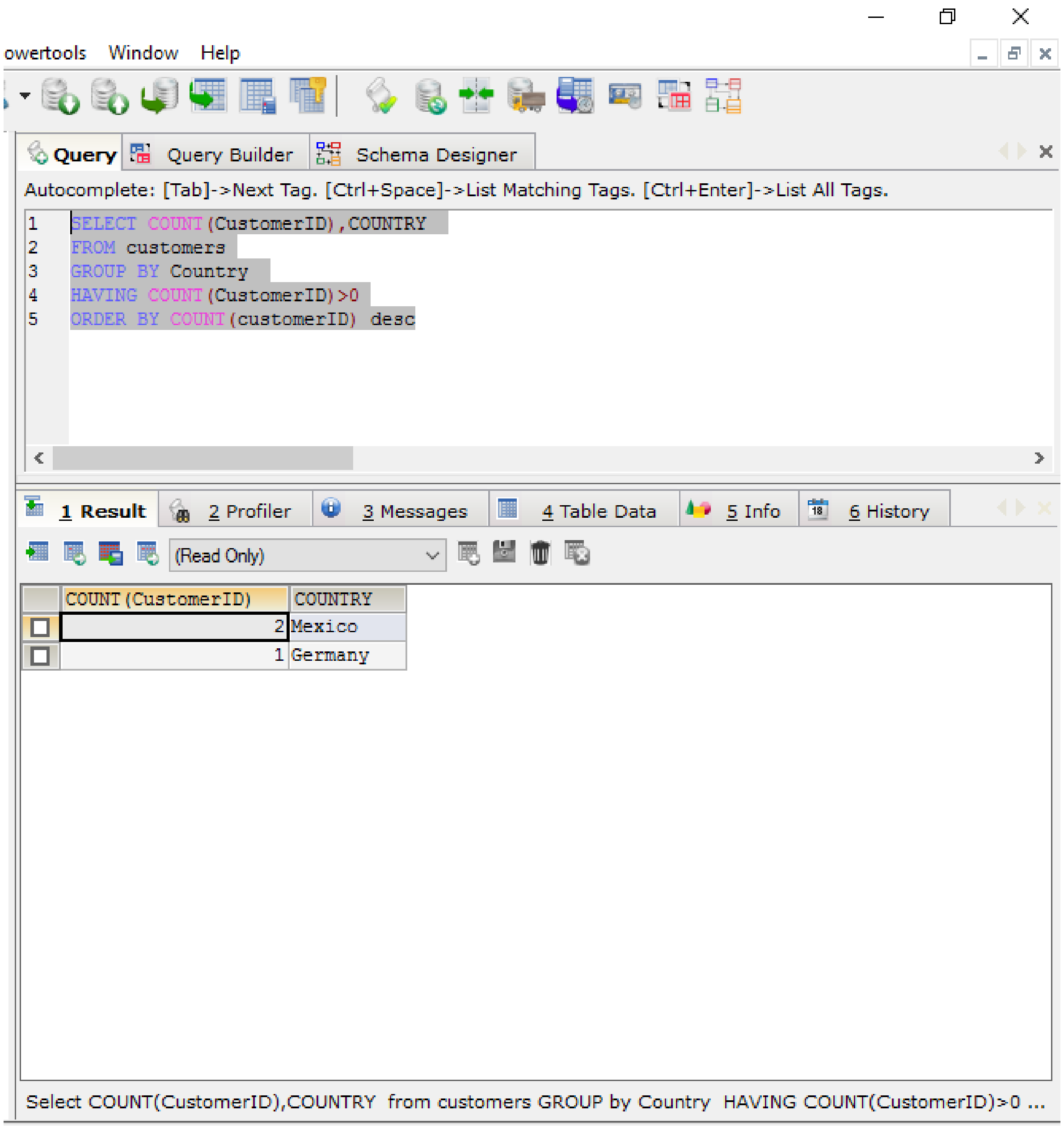
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

```
SELECT COUNT(CustomerID),COUNTRY
FROM customers
GROUP BY Country
HAVING COUNT(CustomerID)>1
```

COUNT(CustomerID)	COUNTRY
2	Mexico

Select COUNT(CustomerID),COUNTRY from customers GROUP by Country HAVING COUNT(CustomerID)>1

6. SELECT COUNT(CustomerID),COUNTRY
   FROM customers
   GROUP BY Country
   HAVING COUNT(CustomerID)>0
   ORDER BY COUNT(customerID) desc



Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

```
SELECT COUNT(CustomerID),COUNTRY
FROM customers
GROUP BY Country
HAVING COUNT(CustomerID)>0
ORDER BY COUNT(customerID) desc
```

COUNT(CustomerID)	COUNTRY
2	Mexico
1	Germany

Select COUNT(CustomerID),COUNTRY from customers GROUP by Country HAVING COUNT(CustomerID)>0 ...

Exercise - 6

SQL Aggregate functions:

AVG

COUNT

MIN

MAX

SUM

Products

ProductId	ProductName	Price
1	Book	18
2	Ink	19
3	Duster	10
4	Textbook	22
5	Bag	21.35

1. Select Min(Price) As SmallestPrice from Products
2. Select Max(Price) As LargestPrice from Products
3. Select count(ProductID) from Products
4. Select Avg(Price) from Products
5. Select Sum(Price) from Products

Exercise - 7

SETTING PRIMARY KEY AND FOREIGN KEY IN SQLYOG

Create the Following tables

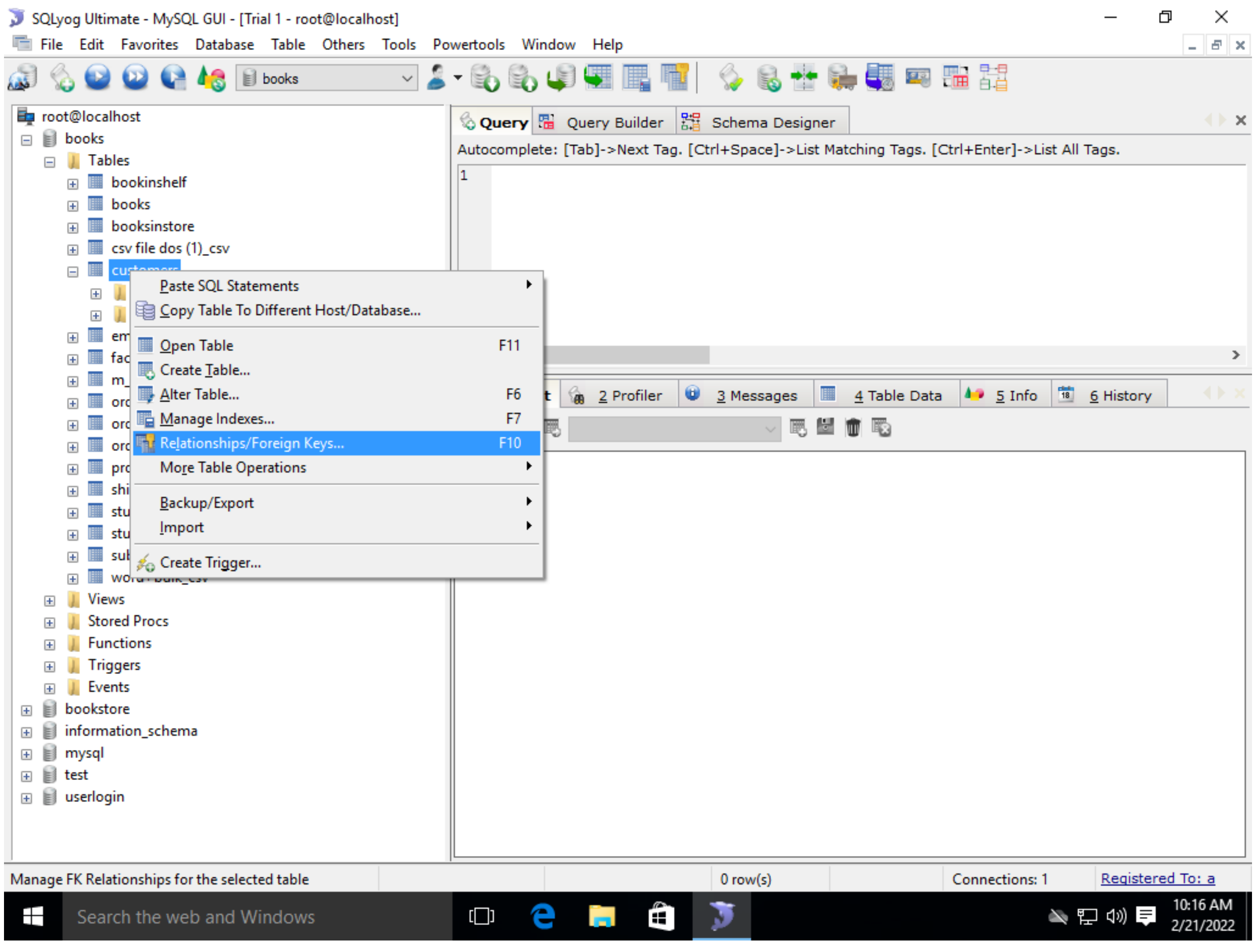
Orders

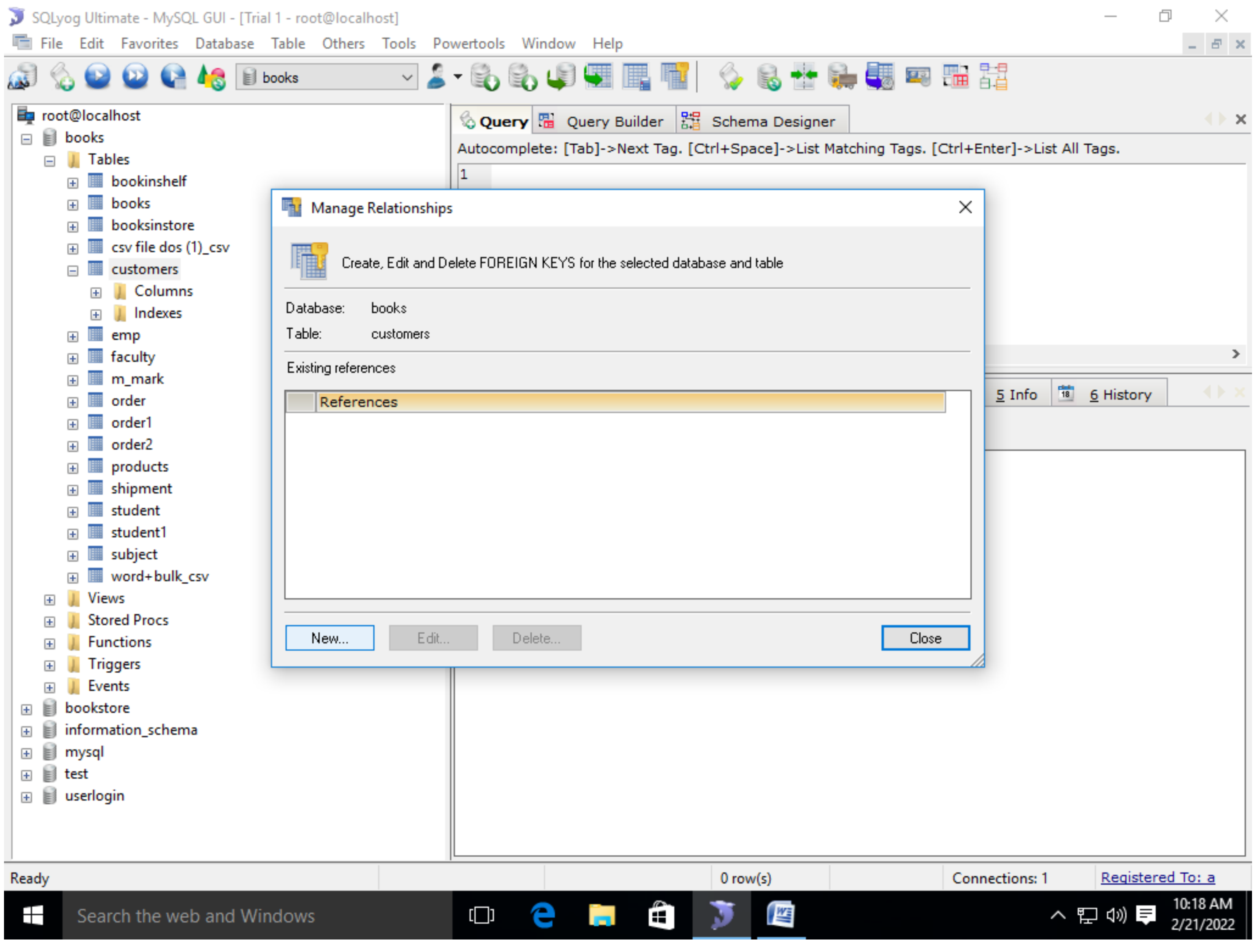
Id	CustomerID
10308	2
10309	37
10310	77

Customers

CustomerId	CustomerName	ContactName	Country
1	Alfred	Maria	Germany
2	Ana	Ana	Mexico
3	Antonio	Antonio	Mexico
4	Avril	Andria	Mexico

Set Id in Orders as Foreign Key and CustomerId in Customers as Primary Key.


SQLyog Ultimate - MySQL GUI - [Trial 1 - root@localhost]
File Edit Favorites Database Table Others Tools Powertools Window Help
books
root@localhost
books
Tables
bookinshelf
books
booksinstore
csv file dos (1)_csv
customers
Views
Stored Procs
Functions
Triggers
Events
bookstore
information_schema
mysql
test
userlogin
Paste SQL Statements
Copy Table To Different Host/Database...
Open Table F11
Create Table...
Alter Table... F6
Manage Indexes... F7
Relationships/Foreign Keys... F10
More Table Operations
Backup/Export
Import
Create Trigger...
Query
Query Builder
Schema Designer
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.
2 Profiler
3 Messages
4 Table Data
5 Info
6 History
Manage FK Relationships for the selected table
0 row(s)
Connections: 1
Registered To: a
Search the web and Windows
10:16 AM
2/21/2022



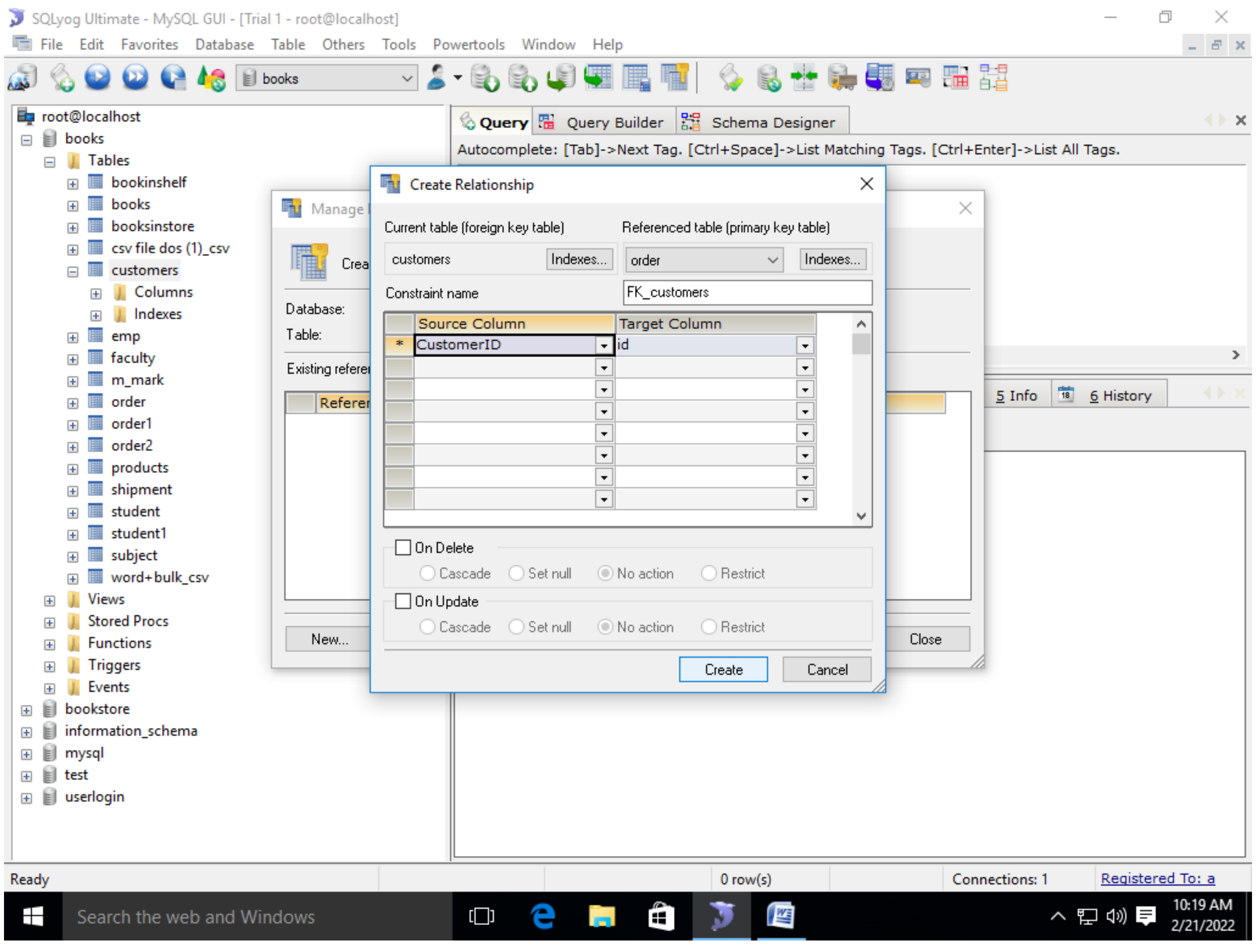
Autocomplete: [Tab]->Next Tag. [Ctrl+Space]->List Matching Tags. [Ctrl+Enter]->List All Tags.

Create, Edit and Delete FOREIGN KEYS for the selected database and table

Database: books
Table: customers

Existing references

References



Current table (foreign key table): customers
Referenced table (primary key table): order
Constraint name: FK_customers

Source Column	Target Column
CustomerID	id

On Delete: Cascade Set null [✓ No action] Restrict
On Update: Cascade Set null [✓ No action] Restrict

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

SELECT Order.OrderId, Customers.CustomerName
FROM Order
INNER JOIN Customers ON Order.CustomerID = Customers.CustomerID;

Output:

CustomerID	CustomerName
10308	Ana

Explanation: Inner Join returns records that have matching values on both sides

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Order.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Output:

```
CustomerID          CustomerName

10308                       Ana

10309                       Null

10310                       Null
```

Explanation: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

Exercise - 8

Order By, Group By & Having Clause

Table name: Customers

CustomerID	**CustomerName**	**ContactName**	**Address**	**City**	**PostalCode**	**Country**
1	Alfreds	Maria	Obere	Berlin	12209	Germany
2	Ana	Ana	Avda	México	05021	Mexico
3	Antonio M	Antonio	Mataderos	México	05023	Mexico
4	Around	Thomas	Hanover Sq.	London	WA1 1DP	UK
5	Berglunds	Christina	Berguvsvägen	Lulea	S-958 22	Sweden
6	Blauer	Hanna	Forsterstr	Mannheim	68306	Germany
7	Blondel	Frédérique	Pplace Kléber	Strasbourg	67000	France

1. List the number of customers in each country.
2. Select all customers from the 'customers' table sorted descending by the Country column.
3. Select all customers from the Customers table, sorted by the country and the Customer Name.
4. Select all customers from the Customers table, sorted ascending by the Country and descending by the CustomerName column.
5. List the number of customers in each country, only include countries with more than one customers

<u>Queries:</u>

1. Select * from Customers Order By Country desc;
2. Select * from Customers Order by Country, CustomerName;
3. Select * from Customers Order by Country ASC,CustomerName Desc;
4. Select count(CustomerID),Country from Customers Group by Country Having count(CustomerID)>1;

Exercise - 9

Execute the following query.

```
CREATE TABLE student(student_code VARCHAR(5)PRIMARY KEY,student_name
VARCHAR(15),dob DATE,student_branch VARCHAR(3),adate DATE,CHECK(student_branch
IN('cs','ec','ee','me')));

INSERT INTO student VALUES('1','Amitha','1987-01-12','cs','2000-06-01');

INSERT INTO student VALUES('2','Vaidehi','1988-12-25','me','2000-06-01');

INSERT INTO student VALUES('3','varun','1988-09-02','me','2000-06-02');

INSERT INTO student VALUES('4','turner','1988-08-05','ec','2000-06-01');

INSERT INTO student VALUES('5','vani','1988-07-20','ee','2000-06-05');

INSERT INTO student VALUES('6','binu','1988-08-13','me','2000-06-10');

INSERT INTO student VALUES('7','chitra','1986-11-14','me','1999-06-10');

INSERT INTO student VALUES('8','dona','1991-12-02','cs','2000-06-02');

INSERT INTO student VALUES('9','elena','1990-02-05','cs','2000-06-01');

INSERT INTO student VALUES('10','fahan','1988-03-20','ec','2000-06-05');

INSERT INTO student VALUES('11','ginu','1988-04-13','ec','2000-06-10');

INSERT INTO student VALUES('12','hamna','1985-05-14','ee','1999-06-19');


SELECT * FROM student


INSERT INTO m_mark VALUES(1,501,40);

INSERT INTO m_mark VALUES(1,502,70);

INSERT INTO m_mark VALUES(1,503,50);

INSERT INTO m_mark VALUES(1,504,80);

INSERT INTO m_mark VALUES(1,505,40);

INSERT INTO m_mark VALUES(1,508,70);

INSERT INTO m_mark VALUES(12,501,90);

INSERT INTO m_mark VALUES(2,502,89)

INSERT INTO m_mark VALUES(2,503,77)

INSERT INTO m_mark VALUES(2,504,95)

INSERT INTO m_mark VALUES(2,505,74)
INSERT INTO m_mark VALUES(2,508,98)
INSERT INTO m_mark VALUES(3,501,40)
INSERT INTO m_mark VALUES(3,502,43)
INSERT INTO m_mark VALUES(3,503,40)
INSERT INTO m_mark VALUES(3,504,40)
INSERT INTO m_mark VALUES(3,505,40)
INSERT INTO m_mark VALUES(3,508,35)
INSERT INTO m_mark VALUES(4,501,50)
INSERT INTO m_mark VALUES(5,501,60)
INSERT INTO m_mark VALUES(6,501,67)
INSERT INTO m_mark VALUES(7,501,23)
INSERT INTO m_mark VALUES(8,501,43)
INSERT INTO m_mark VALUES(9,501,42)
INSERT INTO m_mark VALUES(10,505,74)
INSERT INTO m_mark VALUES(11,508,98)
INSERT INTO m_mark VALUES(12,501,40)
INSERT INTO m_mark VALUES(5,502,43)
INSERT INTO m_mark VALUES(6,503,40)
INSERT INTO m_mark VALUES(7,504,40)
INSERT INTO m_mark VALUES(8,505,40)
INSERT INTO m_mark VALUES(9,508,35)
INSERT INTO m_mark VALUES(10,501,50)
INSERT INTO m_mark VALUES(11,501,60)
INSERT INTO m_mark VALUES(12,503,67)
INSERT INTO m_mark VALUES(5,504,23)
INSERT INTO m_mark VALUES(6,504,23)
INSERT INTO m_mark VALUES(9,504,1)
INSERT INTO m_mark VALUES(10,504,1)
INSERT INTO m_mark VALUES(6,502,43)
INSERT INTO m_mark VALUES(7,505,42)
```

Execute the following query.

1. SELECT * FROM m_mark
2. SELECT * FROM subjects
3. SELECT COUNT(*)"No: of Faculties" FROM faculty;
4. SELECT student_name,SUM(mark)"Total Mark" FROM m_mark,student WHERE student.student_code=m_mark.student_code GROUP BY student_name;
5. SELECT subject_name,ROUND(AVG(mark),2)"Average Mark"FROM subjects,m_mark WHERE subjects.subject_code=m_mark.subject_code GROUP BY subject_name;
6. SELECT subjects.subject_name,COUNT(student.student_name)"No: of Students"FROM subjects,m_mark,student WHERE student.student_code=m_mark.student_code AND m_mark.mark<(40*max_mark)/100 AND subjects.subject_code=m_mark.subject_code GROUP BY subjects.subject_name HAVING COUNT(DISTINCT(m_mark.subject_code))>=1;
7. SELECT student_name,subject_name,mark,max_mark,ROUND((m_mark.mark/max_mark)*100,2)"Percentage"FROM subjects,student,m_mark WHERE mark<(40*max_mark/100)AND subjects.subject_code=m_mark.subject_code AND student.student_code=m_mark.student_code;
8. SELECT faculty.f_name,subjects.subject_name FROM faculty,subjects WHERE faculty.f_code=subjects.faculty_code;
9. SELECT f_name FROM faculty WHERE(SELECT COUNT(subject_code)FROM subjects WHERE subjects.faculty_code=faculty.f_code)>1 GROUP BY faculty.f_name;
10. SELECT faculty.f_name,COUNT(subjects.subject_code)"No_of_subjects"FROM faculty,subjects WHERE(SELECT COUNT(*)FROM subjects WHERE subjects.faculty_code=faculty.f_code)>1 AND subjects.faculty_code=faculty.f_code GROUP BY faculty.f_name;
11. SELECT student_name,subject_name,mark FROM student,subjects,m_mark WHERE student.student_code=m_mark.student_code AND subjects.subject_code=m_mark.subject_code ORDER BY mark;

Exercise - 10

Consider the employee database given below

emp (<u>emp_id</u>,emp_name, Street_No, city)
works (<u>emp_id, company name</u>, salary)
company (<u>company name</u>, city)
manages (emp_id, manager_id)

Create table emp

Import the CSV file with the following information:

```
INSERT INTO emp VALUES('E-101','Adarsh',101,'MG Road');
INSERT INTO emp VALUES('E-102','Bonny',101, 'MG Road');
INSERT INTO emp VALUES('E-103','Catherine', 102, 'Cochin');
INSERT INTO emp VALUES('E-104','Glenn', 104, 'Ernakulam');
INSERT INTO emp VALUES('E-105','George',  201,'MG Road');
INSERT INTO emp VALUES('E-106','Hayes', 101, 'MG Road');
INSERT INTO emp VALUES('E-107','Johnson',102, 'Cochin');
INSERT INTO emp VALUES('E-108','Jones',  101, 'Cochin');
INSERT INTO emp VALUES('E-109','Karthik',  101, 'Ernakulam');
INSERT INTO emp VALUES('E-110','Lavanya',  101, 'Palace Road');
INSERT INTO emp VALUES('E-111','Niharika',  102, 'Ernakulam');
SELECT * FROM emp
```

Create table company

Import the CSV file with the following information:

```
insert into company values('SBI','MG Road');
insert into company values('SBT','MG Road' );
insert into company values('Federal',’Broadway’);
insert into company values('Indian Bank', 'Cochin');
insert into company values('SIB', 'Ernakulam');
insert into company values('HDFC', 'Palace Road');
insert into company values('Axis', 'Cochin');
insert into company values('City bank', 'Ernakulam');
```

Import the CSV file with the following information:

Create table works

```
insert into works values('E-101','SBI',71000);
insert into works values('E-102','SBI',90000);
insert into works values('E-103','SBT',40000);
insert into works values('E-104','Federal',37000);
insert into works values('E-105', 'SBT',17000);
insert into works values('E-106','Indian Bank',30000);
insert into works values('E-107', 'SIB',21000);
insert into works values('E-108','SIB',18000);
insert into works values('E-109', 'Indian Bank',28000);
insert into works values('E-110','SBT',250000);
insert into works values('E-111', 'Federal',40000);
```

Import the CSV file with the following information:

Create table works manages

```
insert into manages values('E-101', 'E-102');
insert into manages values('E-102', Null);
insert into manages values('E-103', 'E-110');
insert into manages values('E-104', 'E-111');
insert into manages values('E-105', 'E-110');
insert into manages values('E-106', 'E-109');
insert into manages values('E-107', Null);
insert into manages values('E-108', Null);
insert into manages values('E-109',Null);
insert into manages values('E-110', Null);
insert into manages values('E-111', null);
```

AIM

Consider the employee database created in Find results for the following questions

a. Find the names of all employees who work for SBI.

b. Find all employees in the database who live in the same cities as the companies for which they work.

c. Find all employees and their managers in the database who live in the same cities and on the same street number as do their managers.

d. Find all employees who earn more than the average salary of all employees of their company.

e. Find the company that pay least total salary along with the salary paid.

f. Give all managers of SBI a 10 percent raise.

g. Find the company that has the most employees

h. Find those companies whose employees earn a higher salary, on average than the average salary at Indian Bank.

i. Query to find name and salary of all employees who earn more than each employee of ‘Indian Bank’

COMMANDS

a) Find the names of all employees who work for SBI.

```
SELECT emp_name FROM works,emp WHERE company_name='SBI'
And emp.emp_id=works.emp_id;

EMP_NAME
------------------
Adarsh
```

b)Find all employees in the database who live in the same cities as the companies for which they work.

```
SELECT emp.emp_name FROM emp, works,company WHERE
emp.emp_id = works. emp_id AND works. company_name=
company.company_name AND emp.city = company.city

EMP_NAME
------------------
Adarsh
George
```

c)Find all employees and their managers in the database who live in the same cities and on the same street number as do their managers.

```
 SELECT emp.emp_name,e2.emp_name "manager name" FROM emp, emp
e2,manages WHERE emp.emp_id = manages.emp_id AND e2.emp_id =
manages.manager_id AND emp.street_no = e2.street_no AND emp.city = e2.city
        EMP_NAME            manager name
        ------------------  ------------------
        Adarsh                          Bonny
```

d) Find all employees who earn more than the average salary of all employees of their company.

```
SELECT emp_name,emp.emp_id,salary FROM works ,emp WHERE salary >
(SELECT AVG (salary) FROM works S WHERE works.company_name
 =S.company_name) and emp.emp_id=works.emp_id

        EMP_NAME            EMP_ID       SALARY
        ------------------  --------      -----------
        Bonny                        E-102          90000
        Hayes                       E-106          30000
        Johnson             E-107           21000
        Lavanya             E-110           250000
        Niharika            E-111           40000
```

e). Find the company that pay least total salary along with the salary paid.

```
    SELECT company_name , SUM(salary) "SALARY PAID" FROM WORK
    GROUP BY company_name HAVING SUM(salary) <= ALL (SELECT
    SUM(salary) FROM WORK GROUP BY company_name)

                 COMPANY_NAME     SALARY PAID
                ------------------  ------------------
                        SIB                          39000
```

f.) Give all managers of SBI a 10 percent raise.

```
UPDATE WORK SET salary = salary * 1.1 WHERE emp_id IN (SELECT
manager_id FROM manages ) AND company_name = 'SBT'
SELECT * FROM WORK
```

g). Find the company that has the most employees

```
SELECT company_name FROM WORK GROUP BY company_name HAVING
COUNT(DISTINCT emp_id) >= ALL (SELECT COUNT(DISTINCT emp_id)
FROM WORK GROUP BY company_name)
COMPANY_NAME
```

```
------------------
SBT
```

h) Find those companies whose employees earn a higher salary, on average than the average salary at Indian Bank.

```
SELECT company_name FROM WORK GROUP BY company_name HAVING
AVG(salary) > (SELECT AVG(salary) FROM WORK WHERE company_name =
'indian bank' GROUP BY company_name)COMPANY_NAME
------------------
SBI
Federal
SBT
```

i).Query to find name and salary of all employees who earn more than each employee of 'Indian Bank'

```
SELECT emp_name , salary FROM WORK , emp WHERE salary > (SELECT
MAX(salary) FROM WORK WHERE company_name = 'indian bank' GROUP BY
company_name) AND emp.emp_id = work.emp_id

EMP_NAME            SALARY
------------------  ----------
Adarsh               71000
Bonny                99000
Catherine            40000
Glenn                37000
Lavanya             250000
Niharika             40000
```

Part - C

PL/SQL

Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN ELSIF, CASE, WHILE using PL/SQL

Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL.

Basic Syntax of PL/SQL

```
DECLARE
        /* Variables can be declared here */
BEGIN
        /* Executable statements can be written here */
EXCEPTION
        /* Error handlers can be written here. */
END;
```

As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

```
        SET SERVEROUTPUT ON
```

CONDITIONAL CONTROL IN PL/SQL:

In PL/SQL, the if statement allows you to control the execution of a block of code. In PL/SQL you can use the IF – THEN – ELSIF – ELSE – END IF statements in code blocks that will allow you to write specific conditions under which a specific block of code will be executed

Exercise - 11

PL/SQL to find addition of two numbers

```
DECLARE
A INTEGER := 20;
B INTEGER := 40;
C INTEGER;
BEGIN
C := A + B;
DBMS_OUTPUT.PUT_LINE ('THE SUM IS '||C);
END;
```

Exercise - 12

Write a Pl/SQL program to add two numbers and to divide two numbers.

```
DECLARE
A INTEGER := 20;
B INTEGER := 40;
C INTEGER;
D INTEGER;
BEGIN
C := A + B;
DBMS_OUTPUT.PUT_LINE ('THE SUM IS '||C);
D:=50.0/3.0;
dbms_output.put_line('RESULT AFTER DIVISION'||D);
END;
```

Decision making with IF statement :-

The general syntax for the using IF--ELSE statement is

```
IF (TEST_CONDITION) THEN
        SET OF STATEMENTS
ELSE
        SET OF STATEMENTS
END IF;
```

For Nested IF—ELSE Statement we can use IF--ELSIF—ELSE as follows

```
IF (TEST_CONDITION) THEN
        SET OF STATEMENTS
ELSIF (CONDITION)
        SET OF STATEMENTS
END IF;
```

Exercise - 13

Write a Pl/SQL program to find largest of three numbers.

This program can be written in number of ways, here are two ways to write the program.

METHOD 1:

```
DECLARE

A NUMBER := 10;

B NUMBER := 30;

C NUMBER := 20;

BIG NUMBER;

BEGIN

IF (A > B) THEN

BIG := A;

ELSE

BIG := B;

END IF;

IF(BIG < C ) THEN

DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' || C);

ELSE

DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' || BIG);

END IF;

END; /
```

```
BIGGEST OF A, B AND C IS 30
```

METHOD 2:

```
DECLARE
A NUMBER := 10;
B NUMBER := 30;
C NUMBER := 20;
BEGIN
IF (A > B AND A > C) THEN
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || A);
ELSIF (B > C) THEN
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || B);
ELSE
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || C);
END IF;
END;
```

```
BIGGEST OF A, B AND C IS 30
```

Exercise - 14

Write a PL/SQL program to check for the number, say 50.

If the number is equal to 10, then print the output as 10

If the number is equal to 20, then print the output as 20

If the number is equal to 30, then print the output as 30

IF none of the above condition is true, then print ‘none of the above condition is true’.

Also, print the exact value of the number

```
DECLARE

A NUMBER :=50;

BEGIN

IF (A =10) THEN

DBMS_OUTPUT.PUT_LINE('value of A is 10');

ELSIF(A=20) THEN

DBMS_OUTPUT.PUT_LINE('value of A is 20');

ELSIF(A=30) THEN

DBMS_OUTPUT.PUT_LINE('value of A is 30');

ELSE

DBMS_OUTPUT.PUT_LINE('NONE OF THE ABOVE CONDITION IS TRUE');

END IF;

DBMS_OUTPUT.PUT_LINE('value of A is'||A);

END; /
```

```
OUTPUT: NONE OF THE ABOVE CONDITION IS TRUE
value of A is 50
```

Exercise - 15

Write a PL/SQL program to check for the mark, say 50.

If the mark is greater than or equal to 90, then print the output as ‘Excellent’

If the mark is greater than or equal to 80, then print the output as ‘Very good’

If the mark is greater than or equal to 70, then print the output as ‘Good’

If the mark is greater than or equal to 60, then print the output as ‘Pass’

If the mark is less than or equal to 59, then print the output as ‘Fail’

If none of the above conditions are met, print ‘None of the above’.

Also, print the exact mark in all the above conditions.

```
DECLARE
A NUMBER :=50;
BEGIN
IF (A >=90) THEN
DBMS_OUTPUT.PUT_LINE('Excellent');
ELSIF(A>=80) THEN
DBMS_OUTPUT.PUT_LINE('Very good');
ELSIF(A>=70) THEN
DBMS_OUTPUT.PUT_LINE('Good');
ELSIF(A>=60) THEN
DBMS_OUTPUT.PUT_LINE('Pass');
ELSIF(A<=59) THEN
DBMS_OUTPUT.PUT_LINE('Fail');
ELSE
DBMS_OUTPUT.PUT_LINE('NONE OF THE ABOVE CONDITION IS TRUE');
END IF;
DBMS_OUTPUT.PUT_LINE('Exact Marks is'||A);
END;
```

```
OUTPUT: Fail
Exact Marks is 50
```

CASE statement

```
CASE selector
  WHEN 'value1' THEN S1;
  WHEN 'value2' THEN S2;
  WHEN 'value3' THEN S3;
  ...
  ELSE Sn;  -- default case
END CASE;
```

Exercise - 16

```
DECLARE
        grade char(1) := 'A';
BEGIN
        CASE grade
                when 'A' then dbms_output.put_line('Excellent');
                when 'B' then dbms_output.put_line('Very good');
                when 'C' then dbms_output.put_line('Well done');
                when 'D' then dbms_output.put_line('You passed');
                when 'F' then dbms_output.put_line('Better try again');
        else dbms_output.put_line('No such grade');
        END CASE;
END; /
```

```
OUTPUT: Excellent
```

Exercise - 17

```
DECLARE

grade integer := 1;

BEGIN

CASE grade

when 1 then dbms_output.put_line('Monday');

when 2 then dbms_output.put_line('Tuesday');

when 3 then dbms_output.put_line('Wednesday');

when 4 then dbms_output.put_line('Thursday');

when 5 then dbms_output.put_line('Friday');

when 6 then dbms_output.put_line('Saturday');

when 7 then dbms_output.put_line('Sunday');

else dbms_output.put_line('No such day');

        END CASE;

END; /
```

```
OUTPUT: Monday
```

Iterative Control

This is the ability to repeat or skip sections of a code block. A loop repeats a sequence of statements. You have to place the keyword loop before the first statement in the sequence of statements that you want repeated and the keywords end loop immediately after the last statement in the sequence. Once a loop begins to run, it will go on forever. Hence loops are always accompanied by a conditional statement that keeps control on the number of times the loop is executed.

You can build user defined exits from a loop, where required.

THE WHILE LOOP:

```
WHILE<condition>

        LOOP <action>

END LOOP;
```

Exercise - 18

Write a PL/SQL program to print first ten numbers.

```
DECLARE
num number:=1;
BEGIN
while num<=10
LOOP
DBMS_OUTPUT.PUT_LINE(num);
num:=num+1;
END LOOP;
END
```

Write a PL/SQL program to find sum of number of digits.

```
DECLARE
        n           INTEGER;
        temp_sum INTEGER;
        r           INTEGER;
BEGIN
        n := 123456;
        temp_sum := 0;
        WHILE n <> 0 LOOP
                r := MOD(n, 10);
                temp_sum := temp_sum + r;
                n := Trunc(n / 10);
        END LOOP;
        dbms_output.Put_line('sum of digits = '|| temp_sum);
END;
```

```
OUTPUT: sum of digits = 21
```

PROCEDURES

A subprogram is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

S.No	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result −

```
Procedure created.
```

The procedure can also be called from another PL/SQL block −

```
BEGIN
   greetings;
END;
/
```

The above call will display −

```
Hello World

PL/SQL procedure successfully completed.
```

Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms −

S.No	Parameter Mode & Description
1	**IN** An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference**.
2	**OUT** An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**.
3	**IN OUT** An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read. The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value.**

IN & OUT Mode Example 1

This program finds the minimum of two values. Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.

Exercise - 19

Write a PL/SQL program using procedure to find the minimum of two numbers.

```
DECLARE

a number;

b number;

c number;

d number;

PROCEDURE findmin(x IN number, y IN number, d IN number, z OUT number) IS

BEGIN

If x<y and x<d then

z:=x;

Elsif

y<d then

z:=y;

ELSE

z:=d;

END IF;

END;

BEGIN

a:=21;

b:=45;

c:=36;

Findmin(a,b,c,d);

dbms_output.put_line('minimum is '||d);

END;
```

```
OUTPUT: minimum is 21
```

Exercise - 20

IN & OUT Mode Example

This procedure computes the square of value of a passed value. This example shows how we can use the same parameter to accept a value and then return another result.

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   squareNum(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Output: Square of (23):
```

Exercise - 21

Write a PL/SQL program using procedure to find the determinant.

```
DECLARE
a number;
b number;
c number;
d number;
PROCEDURE det(x IN number, y IN number, z IN number, d OUT number) IS
BEGIN
d:=y*y-4*x*z;
END;
BEGIN
a:=5;
b:=10;
c:=2;
det(a,b,c,d);
dbms_output.put_line('Determinant is'||d);
END;
```

```
Output: Determinant is 60
```

Exercise - 22

Write a PL/SQL program using procedure to find the area of a triangle.

```
DECLARE
a number;
b number;
c number;

PROCEDURE area(x IN number, y IN number, z OUT number) IS
BEGIN
Z:=0.5*x*y;
END;
BEGIN
a:=2;
b:=5;
area(a,b,c);
dbms_output.put_line('Area of triangle is'||c);
END;
```

```
OUTPUT: Area of triangle is 5
```

Exercise - 23

Write a PL/SQL program using procedure to find the area of a rectangle.

```
DECLARE
a number;
b number;
c number;

PROCEDURE area(x IN number, y IN number, z OUT number) IS
BEGIN
Z:=x*y;
END;
BEGIN
a:=2;
b:=5;
area(a,b,c);
dbms_output.put_line('Area of rectangle is'||c);
END;
```

```
OUTPUT: Area of rectangle is 10
```

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

CREATE TABLE CUSTOMERS(ID VARCHAR(10),NAME VARCHAR(10),AGE INT,ADDRESS VARCHAR(10), SALARY INT);

INSERT INTO CUSTOMERS VALUES('210001','Ramesh',32,'Ahmedabad', 2000.00)

INSERT INTO CUSTOMERS VALUES('2','Khilan',25,'Delhi',1500.00)

INSERT INTO CUSTOMERS VALUES('3','kaushik',23,'Kota',2000.00)

INSERT INTO CUSTOMERS VALUES('4','Chaitali',25,'Mumbai',6500.00)

INSERT INTO CUSTOMERS VALUES('5','Hardik',27,'Bhopal',8500.00)

INSERT INTO CUSTOMERS VALUES('6','Komal',22,'MP',4500.00)

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

```
CREATE OR REPLACE TRIGGER display_salary_changes1
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes1** will be fired and it will display the following result −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

```
Old salary:
New salary: 7500
Salary difference:

1 row(s) inserted.
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary: 1500
New salary: 2000
Salary difference: 500

1 row(s) updated.
```

Exercise - 24

FUNCTIONS

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;

   RETURN total;
END;
```

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
```

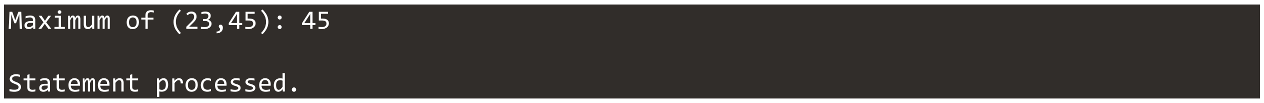
```
Total no. of Customers: 8

Statement processed.
```

Exercise - 25

Example

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
   a number;
   b number;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      Z:= y;
   END IF;
   RETURN z;
END;
BEGIN
   a:= 23;
   b:= 45;
   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

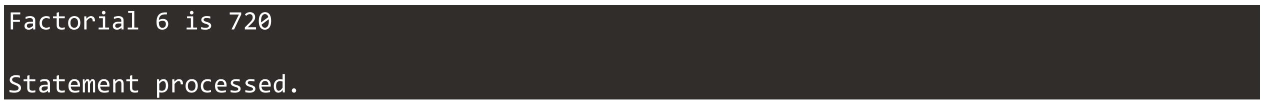


```
Maximum of (23,45): 45

Statement processed.
```

Exercise - 26

PL/SQL Recursive Functions

```
DECLARE
   num number;
   factorial number;

FUNCTION fact(x number)
RETURN number
IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;

BEGIN
   num:= 6;
   factorial := fact(num);
   dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
```



```
Factorial 6 is 720

Statement processed.
```

Exercise - 27

CURSORS

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

Example

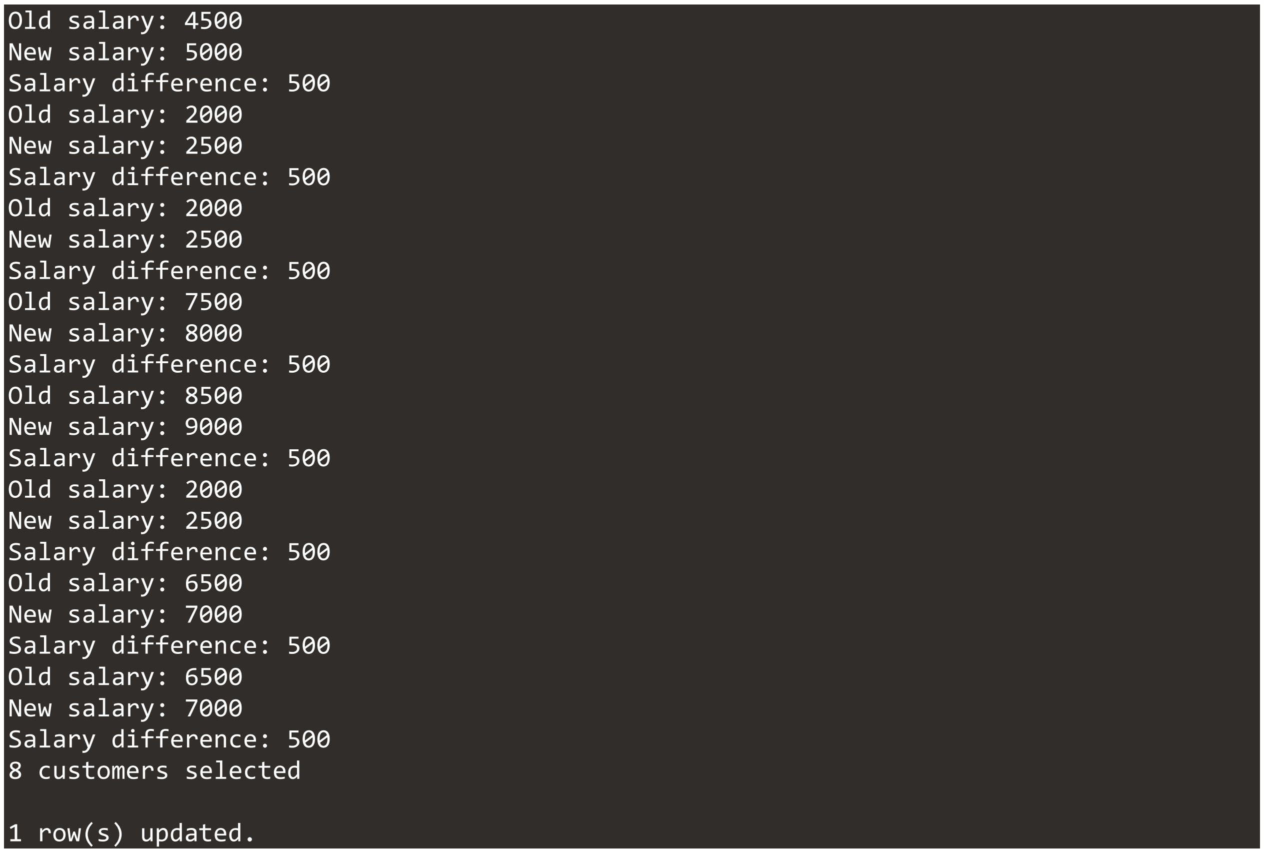
We will be using the CUSTOMERS table we had created and used in the previous chapters.

Select * from customers;

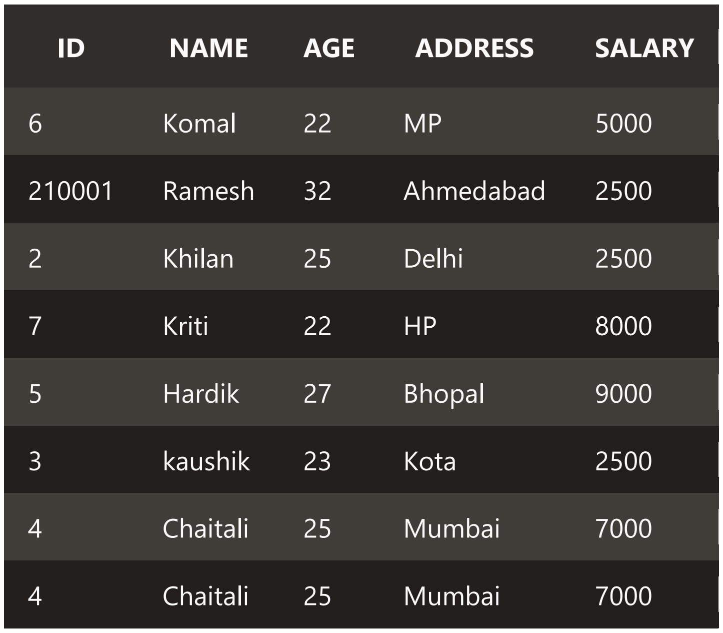
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected −

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
```



```
Old salary: 4500
New salary: 5000
Salary difference: 500
Old salary: 2000
New salary: 2500
Salary difference: 500
Old salary: 2000
New salary: 2500
Salary difference: 500
Old salary: 7500
New salary: 8000
Salary difference: 500
Old salary: 8500
New salary: 9000
Salary difference: 500
Old salary: 2000
New salary: 2500
Salary difference: 500
Old salary: 6500
New salary: 7000
Salary difference: 500
Old salary: 6500
New salary: 7000
Salary difference: 500
8 customers selected

1 row(s) updated.
```



ID	NAME	AGE	ADDRESS	SALARY
6	Komal	22	MP	5000
210001	Ramesh	32	Ahmedabad	2500
2	Khilan	25	Delhi	2500
7	Kriti	22	HP	8000
5	Hardik	27	Bhopal	9000
3	kaushik	23	Kota	2500
4	Chaitali	25	Mumbai	7000
4	Chaitali	25	Mumbai	7000

Exercise - 28

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
  SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

Exercise - 29

PACKAGES

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.

A package will have two mandatory parts −

- Package specification
- Package body or definition

The Package Specification

```
CREATE OR REPLACE PACKAGE c_package AS
  -- Adds a customer
  PROCEDURE addCustomer(c_id   customers.id%type,
  c_name customers.Name%type,
  c_age  customers.age%type,
  c_addr customers.address%type,
  c_sal  customers.salary%type);

  -- Removes a customer
  PROCEDURE delCustomer(c_id  customers.id%TYPE);
  --Lists all customers
  PROCEDURE listCustomer;

END c_package;
```

Creating the Package Body

```
CREATE OR REPLACE PACKAGE BODY c_package AS
  PROCEDURE addCustomer(c_id  customers.id%type,
    c_name customers.Name%type,
    c_age  customers.age%type,
    c_addr  customers.address%type,
    c_sal   customers.salary%type)
  IS
  BEGIN
    INSERT INTO customers (id,name,age,address,salary)
      VALUES(c_id, c_name, c_age, c_addr, c_sal);
  END addCustomer;

  PROCEDURE delCustomer(c_id   customers.id%type) IS
  BEGIN
    DELETE FROM customers
```

```
      WHERE id = c_id;
   END delCustomer;

   PROCEDURE listCustomer IS
   CURSOR c_customers is
      SELECT  name FROM customers;
   TYPE c_list is TABLE OF customers.Name%type;
   name_list c_list := c_list();
   counter integer :=0;
   BEGIN
      FOR n IN c_customers LOOP
      counter := counter +1;
      name_list.extend;
      name_list(counter) := n.name;
      dbms_output.put_line('Customer(' ||counter|| ')'||name_list(counter));
      END LOOP;
   END listCustomer;

END c_package;
```

```
Package Body created.
```

Using The Package

The following program uses the methods declared and defined in the package *c_package*.

```
DECLARE
   code customers.id%type:= 8;
BEGIN
   c_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);
   c_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);
   c_package.listcustomer;
   c_package.delcustomer(code);
   c_package.listcustomer;
END;
/
```

```
Old salary:
New salary: 3500
Salary difference:
Old salary:
New salary: 7500
Salary difference:
Customer(1)Komal
Customer(2)Ramesh
Customer(3)Khilan
Customer(4)Kriti
Customer(5)Hardik
Customer(6)kaushik
Customer(7)Chaitali
Customer(8)Chaitali
Customer(9)Rajnish
Customer(10)Subham
Customer(1)Komal
Customer(2)Ramesh
Customer(3)Khilan
Customer(4)Kriti
Customer(5)Hardik
Customer(6)kaushik
Customer(7)Chaitali
Customer(8)Chaitali
Customer(9)Rajnish

Statement processed.
```

Exercise - 30

EXCEPTIONS

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

```
DECLARE
  c_id customers.id%type := 8;
  c_name customerS.Name%type;
  c_addr customers.address%type;
BEGIN
  SELECT  name, address INTO  c_name, c_addr
  FROM customers
  WHERE id = c_id;
  DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
  DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
  WHEN others THEN
    dbms_output.put_line('Error!');
END;
```