



**DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING**

AIL202 DATABASE MANAGEMENT SYSTEMS LAB

(AY:2021-22)

**TEACHERS LAB MANUAL
Version number: 1.0**

Prepared by:
Ms. DHANYA SUDARSAN
Ms. BISMIN V SHERIF
Assistant Professor
CSE Department

VISION

To be recognized as a socially accountable thought leader in applications of computational technologies and a centre of excellence in solving complex and cross-disciplinary problems.

MISSION

- Develop pedagogical practices that help equip students with the ability to self-learn and reason.
- Improve ability to communicate on complex engineering activities and interpersonal skills by developing proper training methods and professional networking.
- Improve problem solving ability by promoting computational thinking and working on cross-disciplinary projects.
- Create and sustain networks in professional, academic and startup environments to stay updated with changes in the field of Computer Science and Engineering.
- Instill social commitment in students and faculty by engaging in spreading computer literacy and other socially relevant services.
- Promote research towards developing insight into complex problems.

TABLE OF CONTENTS

Exp.No	Name of Experiment	Page Number
1	Familiarization of DDL commands	
2	Practice DML commands (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)	
3	Specifying Constraints	
4	Implementation of various aggregate functions in SQL	
5	Implementation of Order By, Group By& Having clause.	
6	Implementation of built-in functions in RDBMS	
7	Implementation of nested queries	
8	Implementation of set operators and join queries	
9	Creation of views	
10	Implementation of various control structures using PL/SQL	
11	Creation of Procedures and Functions	
12	Creation of Triggers	
13	Creation of Cursors	
14	Creation of Packages	
15	Creation of PL/SQL block for exception handling	
16	Practice of SQL TCL commands like Rollback, Commit, Savepoint	
17	Practice of SQL DCL commands for granting and revoking user privileges	
18	Familiarization of SQL Workbench.	
19	Familiarization of NoSQL Databases and CRUD operations	
20	Course project - Design a database application using any front end tool for any problem selected)	

Experiment 1

DDL COMMANDS

AIM

To familiarize with DDL Commands

DESCRIPTION

DDL (Data Definition Language)

It is a set of SQL commands used to create, modify and delete database structures and not data. They are normally used by DBA, database designer or application developer.

The commonly used DDL statements include:

- **CREATE** To create objects in the database.
- **ALTER** Alters the structure of the database.
- **DROP** Delete objects from the database.
- **TRUNCATE** Remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** Add comments to the data dictionary.

Table Fundamentals:

- A table is a database object that holds user data.
- Each column of the table will have a specific data type bound to it.

Basic Data Types:

<u>Data Type</u>	<u>Description</u>
CHAR (size)	This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of characters this data type can hold is 255 characters.
VARCHAR (size)/VARCHAR2 (size)	This data type is used to store variable length alphanumeric data. It is more flexible form of the CHAR data type. The maximum this data type can hold up to 4000 characters.
DATE	This data type is used to represent date and time. The standard format is DD-MON-YY as in 21-JUN-04.
NUMBER (P, S)	The NUMBER data type is used to store numbers. The precision P, determines the maximum length of the data, whereas the scale S, determines the number of places to the right of the decimal.

The CREATE TABLE Command

- Table creation is done using the Create Table syntax.
- The CREATE TABLE Command defines each column of the table uniquely.
- Each table column definition is separated from the other by a comma.
- The SQL statement is terminated with a semi colon.

Rules for Creating Tables

- 1) A name can have maximum up to 30 characters.
- 2) Alphabets from A-Z, a-z, and numbers from 0-9 are allowed.
- 3) A name should begin with an alphabet.
- 4) The use of only special character like _ is allowed and also recommended.
- 5) SQL reserved words not allowed. For example: create, select, and so on.

Syntax:

CREATE TABLE<TableName>(<ColumnName1><DataType>(<size>),
<ColumnName1><DataType>(<size>));

- Each column must have a data type. The column should either be defined as null or not null and if this value is left blank, the database assumes “null” as the default.

The ALTER TABLE Command:

- ALTER TABLE command allows changing the structure of an existing table.
- ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed.
- Adding new columns:

Syntax:

ALTER TABLE<TableName>
 ADD (<NewColumnName><DataType>(<Size>),
 <NewColumnName><DataType>(<Size>) . . .);

Dropping a column from a table:

Syntax:

ALTER TABLE<TableName>**DROP COLUMN**<ColumnName>;

Modifying existing columns:

Syntax:

```
ALTER TABLE<TableName>  
MODIFY (<ColumnName><NewDataType>(<NewSize>));
```

The RENAME Command:

- The rename operation is done atomically, which means that no other thread can access any of the tables while rename process is running.

Syntax:

```
RENAME <TableName> TO <NewTableName>;
```

The DROP TABLE Command:

- DROP TABLE statement with the table name can be used to destroy a specific table. Syntax is:

```
DROP TABLE<TableName>;
```

The TRUNCATE TABLE Command:

TRUNCATE TABLE command empties a table completely. But the structure will be retained

Syntax:

```
TRUNCATE TABLE<TableName>;
```

QUESTIONS

1. Create a table student (roll no,name,address,phone no.,gender,branch,mark1,mark2).
2. Modify the table structure to include new column total marks
3. Modify the size of the address attribute to 20.
4. Delete the column gender and branch from the table.
5. Rename the column mark1 to m1.
6. Rename the table student to student1
7. Remove the table structure.

ANSWERS

1. create table student(rollno number(2),name varchar(20),address varchar(50),phoneno number(10),gender varchar(10),branch varchar(10),mark1 number(20),mark2 number(20));

table created.

desc student;

Name	Null	Type
-----	----	-----
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
ADDRESS		VARCHAR2(50)
PHONENO		NUMBER(10)
GENDER		VARCHAR2(10)
BRANCH		VARCHAR2(10)
MARK1		NUMBER(20)
MARK2		NUMBER(20)

2. alter table student add totalmarks number(20);

Table altered

desc student;

Name	Null	Type
-----	----	-----
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
ADDRESS		VARCHAR2(50)
PHONENO		NUMBER(10)
GENDER		VARCHAR2(10)
BRANCH		VARCHAR2(10)
MARK1		NUMBER(20)
MARK2		NUMBER(20)
TOTALMARKS		NUMBER(20)

3. alter table student modify address varchar(20);

Table altered

desc student;

Name	Null	Type
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONENO		NUMBER(10)
GENDER		VARCHAR2(10)
BRANCH		VARCHAR2(10)
MARK1		NUMBER(20)
MARK2		NUMBER(20)
TOTALMARKS		NUMBER(20)

4. alter table student drop column gender;
alter table student drop column branch;

Table altered

Table altered

desc student;

Name	Null	Type
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONENO		NUMBER(10)
MARK1		NUMBER(20)
MARK2		NUMBER(20)
TOTALMARKS		NUMBER(20)

5. alter table student rename column mark1 to m1;

Table altered

desc student;

Name	Null	Type
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
ADDRESS		VARCHAR2(20)
PHONENO		NUMBER(10)
M1		NUMBER(20)
MARK2		NUMBER(20)
TOTALMARKS		NUMBER(20)

6. alter table student rename to student1;

Table altered

desc student;

```
desc student
ERROR:
-----
ERROR: object STUDENT does not exist
```

7. drop table student1;

table dropped

Experiment 2

DML COMMANDS

AIM

To familiarize with DML commands.

DESCRIPTION

DML: A data manipulation language (DML) is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data.

The commonly used DDL statements include

- **INSERT** -Insert data in to a table.
- **UPDATE**- Updates existing data within the table.
- **DELETE**- Deletes all records from a table, the space for records remain.
- **SELECT** Retrieve data from the database.

The INSERT INTO Command:

- Once a table is created, the next thing to do is load this table with data.
- When inserting a single row of data into the table, the insert operation:
 - Creates a new row (empty) in the database table.
 - Loads the values passed (by the SQL insert) into the columns specified.
- **Syntax:**

**INSERT INTO <tablename>(<columnname1>, < columnname2>)
VALUES (<expression1>, < expression2>);**

- Character expressions placed within the INSERT INTO statement must be enclosed in single quotes.
- In the INSERT INTO SQL sentence, table columns and values have a one to one relationship (i.e. the first value described is inserted into the first column, and the second value described is inserted into the second column and so on).

The UPDATE Command:

- The UPDATE command is used to change or modify data values in a table.
- Updating all rows:
 - The UPDATE statement updates columns in the existing table's rows with new values.

- ❑ The SET clause indicates which column data should be modified and the new values that they should hold.
- ❑ The WHERE clause, if given, specifies which rows should be updated. Otherwise, all table rows are updated.

Syntax:

UPDATE<TableName>

SET<ColumnName1> = <Expression1><ColumnName2>=<Expression2>;

- Updating records conditionally:

Syntax:

UPDATE<TableName>

SET<ColumnName1>=<Expression1><ColumnName2>=<Expression2>;

WHERE<Condition>;

The DELETE Command:

- The DELETE command delete rows from the table that satisfies the condition provided by its where clause, and returns the number of records deleted.
- If a DELETE statement without a WHERE clause is issued then, all rows are deleted.
- Removal of all Rows:

Syntax:

DELETE FROM <TableName>;

- Removal of specific rows:

Syntax:

DELETE FROM <TableName> WHERE <Condition>;

- Removal of specific rows based on the data held by the other table:
 - ❑ Sometimes it is desired to delete records in one table based on values in another table. Since it is not possible to list more than one table in the FROM clause while performing a delete, the EXISTS clause can be used.

The SELECT Command:

- Once the data has been inserted into a table, the next operation would be to view what has been inserted.
- The SELECT command is used to retrieve rows selected from one or more tables.
- All rows and columns

In order to view global table data the syntax is:

SELECT<ColumnName1> TO <ColumnName N> FROM TableName;

- Here ColumnName1 to ColumnName N represents table column names.
- When data from all rows and columns from the table are to be viewed the syntax of the SELECT statement will be like as shown below

Syntax:

SELECT * FROM <TableName>;

- SQL provides a method of filtering table data that is not required. The ways of filtering table data are
 - Selected Columns and All Rows
- The retrieval of specific columns from a table can be done as shown below:

Syntax:

SELECT<ColumnName1>,<ColumnName2>FROM<TableName>;

Selected Rows and all Columns

- Oracle provides the option of using a WHERE Clause in an SQL query to apply a filter on the rows retrieved.
- When a where clause is added to the SQL query, the Oracle engine compares each record in the table with the condition specified in the where clause. The Oracle engine displays only those records that satisfy the specified condition.

Syntax:

SELECT * FROM <TableName>WHERE<Condition>; Here,<Condition> is always quantified as <ColumnName=Value>

Selected Columns and Selected Rows

To view a specific set of rows and columns from a table the syntax will be

Syntax:

SELECT <ColumnName1>,<ColumnName2> FROM <TableName> WHERE<Condition>;

QUESTIONS

- 1.Create the table employees
Employee(empid,ename,eaddress,designation,department,salary,joindate)
2. Populate the employee table with 5 tuples.
3. Display the details of all employees.
4. Display name,designation and salary of all employees.
5. Find employee id of all employees whose salary is greater than 10,000.

6. Display the name and designation of all employees whose department is 'cse'
7. Display the details of all employees whose name ends with 'a'.
8. Display the average salary of employees of each department.
9. Find name and department of employees whose designation is 'manager'.
10. Remove the details of employees whose salary is between 5000 and 7500.
11. Change the designation of all employees to asst manager whose current designation is executive and who have joined before 1st Jan 2011
12. Display the name and increment of all employees .Assume 10% of increment.

QUERIES

1. create table employ(empid number(5),ename varchar(10),eaddr varchar(10),designation varchar(20),dept varchar(10),salary number(10),joindate date);
table EMPLOY created.

2. insert into employ values(101,'anu','xyz','clerk','cse',10000,'08-apr-2012');

insert into employ values(102,'bijoy','abc','hr','me',6000,'05-dec-2007');

insert into employ values(103,'maya','pqr','manager','civil',25000,'10-feb-2010');

insert into employ values(104,'anish','cde','executive','ec',15000,'05-nov-2003');

insert into employ values(105,'jain','efg','manager','eee',7000,'01-jan-2010');

5 rows inserted.

3. select * from employ;

	EMPID	ENAME	EADDR	DESIGNATION	DEPT	SALARY	JOINDATE
1	101	anu	xyz	clerk	cse	10000	08-04-12
2	102	bijoy	abc	hr	me	6000	05-12-07
3	103	maya	pqr	manager	civil	25000	10-02-10
4	104	anish	cde	executive	ec	15000	05-11-03
5	105	jain	efg	manager	eee	7000	01-01-10

4. select ename,designation,salary from employ;

	ENAME	DESIGNATION	SALARY
1	anu	clerk	10000
2	bijoy	hr	6000
3	maya	manager	25000
4	anish	executive	15000
5	jain	manager	7000

5. select empid from employ where salary>10000;

	EMPID
1	103
2	104

6. select ename,designation from employ where dept='cse';

	ENAME	DESIGNATION
1	anu	clerk

7. select * from employ where ename like '%a';

	EMPID	ENAME	EADDR	DESIGNATION	DEPT	SALARY	JOINDATE
1	103	maya	pqr	manager	civil	25000	10-02-10

8. select avg(salary),dept from employ group by dept;

	AVG(SALARY)	DEPT
1	10000	cse
2	25000	civil
3	7000	eee
4	15000	ec
5	6000	me

9. select ename,dept from employ where designation='manager';

	ENAME	DEPT
1	maya	civil
2	jain	eee

10. delete from employ where salary between 5000 and 7500;

2 rows deleted.

Select * from employ;

	EMPID	ENAME	EADDR	DESIGNATION	DEPT	SALARY	JOINDATE
1	101	anu	xyz	clerk	cse	10000	08-04-12
2	103	maya	pqr	manager	civil	25000	10-02-10
3	104	anish	cde	executive	ec	15000	05-11-03

11. update
employ set

designation='assistantmanager' where designation='executive' and joindate<'01-jan-2011';

1 row updated

12. select ename,salary*0.10 from employ;

select ename,salary*0.10 "increment" from employ;

	ENAME	increment
1	anu	1000
2	maya	2500
3	anish	1500

Experiment 3

CONSTRAINTS

AIM

To familiarize with various types of constraints that can be applied to data stored in database.

DESCRIPTION

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

- [NOT NULL Constraint](#): Ensures that a column cannot have NULL value.
- [DEFAULT Constraint](#): Provides a default value for a column when none is specified.
- [UNIQUE Constraint](#): Ensures that all values in a column are different.
- [PRIMARY Key](#): Uniquely identified each rows/records in a database table.
- [FOREIGN Key](#): Uniquely identified a rows/records in any another database table.
- [CHECK Constraint](#): The CHECK constraint ensures that all values in a column satisfy certain conditions.

The PRIMARY KEY Constraint:

- A primary key is one or more column(s) in a table used to uniquely identify each row in the table.
- A primary key column in a table has special attributes:
 - ⇒ It defines the column, as a mandatory column (i.e. the column cannot be left blank.). The NOT NULL attribute is active.
 - ⇒ The data held across the column MUST be UNIQUE.
- A primary key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.
- Primary key Constraint Defined at Column Level:

Syntax:

<ColumnName><Datatype>(<Size>) PRIMARY KEY

- Primary key Constraint Defined at Table Level:

Syntax:

PRIMARY KEY (<ColumnName>, <ColumnName>)

The FOREIGN KEY Constraint:

- Foreign keys represent relationships between tables.
- A foreign key is a column or a group of columns whose values are derived from the primary key or unique key of some other table.
- A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

- The master table can be referenced in the foreign key definition by using the clause REFERENCES TableName. ColumnName when defining the foreign key, column attributes, in the detail table.
This relationship ensures:
 - ⇒ Records cannot be inserted into a detail table if corresponding records in the master table do not exist.
 - ⇒ Records of the master table cannot be deleted if corresponding records in the detail table actually exist.
- Insert or Update Operation in the Foreign Key table:
 - ⇒ The existence of a foreign key implies that the table with the foreign key is related to the master table from which the foreign key is derived. A foreign key must have a corresponding primary key or unique key value in the master table.
- Delete Operation on the Primary key table:
 - ⇒ Oracle displays an error message when a record in the master table is deleted and corresponding records exists in a detail table and prevents the delete operation from going through.
 - ⇒ The default behavior of the foreign key can be changed by using the **ON DELETE CASCADE** option. When the **ON DELETE CASCADE** option is specified in the foreign key definition, if a record is deleted in the master table, all corresponding records in the detail table along with the record in the master table will be deleted.
 - ⇒ A foreign key with a **SET NULL ON DELETE** means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to null. The records in the child table will not be deleted.
- Principles of Foreign Key/References constraint:
 - ⇒ Rejects an INSERT or UPDATE of a value, if a corresponding value does not currently exist in the master key table.
 - ⇒ If the ON DELETE CASCADE option is set, a DELETE operation in the master table will trigger a DELETE operation for corresponding records in all detail tables.
 - ⇒ If the ON DELETE SET NULL option is set, a DELETE operation in the master table will set the value held by the foreign key of the detail tables to null.
 - ⇒ Rejects a DELETE from the master table if corresponding records in the DETAIL table exist.
 - ⇒ Must reference a PRIMARY KEY or UNIQUE column(s) in primary table.
 - ⇒ Requires that the FOREIGN KEY column(s) and the CONSTRAINT column(s) have matching data types.
 - ⇒ Can reference the same table named in the CREATE TABLE statement.
- Foreign Key Constraint Defined at the Column Level:

Syntax:

<ColumnName><Datatype>(<Size>)

**REFERENCES <TableName>[(<ColumnName>)]
[ON DELETE CASCADE]**

- Foreign Key Constraint Defined at the Table Level:

Syntax:

FOREIGN KEY (<ColumnName>, <ColumnName>)

REFERENCES<TableName>(<ColumnName>, <ColumnName>)

The Unique Key Constraint:

- The Unique column constraint permits multiple entries of NULL into the column.

UNIQUE Constraint Defined at the Column Level:

Syntax:

<ColumnName><Datatype>(<size>) UNIQUE

- UNIQUE Constraint Defined at the Table Level:

Syntax:

**CREATE TABLE TableName (<ColumnName1><Datatype>(<size>),
<ColumnName2><Datatype>(<size>),
UNIQUE (<ColumnName1><ColumnName2>));**

NOT NULL Constraint Defined at the Column Level:

- The NOT NULL constraint ensures that a table column cannot be left empty.
- When a column is defined as NOT NULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syntax:

<ColumnName><Datatype>(<Size>) NOT NULL

The CHECK Constraint:

CHECK constraints must be specified as a logical expression that evaluates either to TRUE or FALSE.

CHECK constraint defined at column level:

Syntax:

<ColumnName><Datatype>(<Size>) CHECK (<Logical Expression>)

CHECK constraint defined at table level:

Syntax:

CHECK (<Logical Expression>)

QUESTIONS

1. Create a table department(dept no,dept name,staffno). Set underlined attribute as primary key at column level.
2. Populate the table with following values

<u>DEPTNO</u>	<u>DEPTNAME</u>	<u>STAFFNO</u>
D1	S101	CS
D2	S110	EC
D1	S201	EEE

3. Create a table books (title,author). Set underlined attribute as primary key at table level.

4. Populate the table books with following values

Title	Author
DBMS	NAVATHE
C	Dennis Richie
DBMS	Korth

5. Create a table library(title,author,no of copies).set title as foreign key at column level with respect to table books.
6. Populate table library with following values

TITLE	AUTHOR	NO OF COPIES
C	Dennis Richie	5
DBMS	Navathe	10
OS	Korth	20

7. Set title,author as primary key for the table library
8. Create table s1(rollno,name,dnum).set underlined attribute as primary key. Also set the relation with table departmet(referencing clause should be included at table level)
9. List details of department
10. Populate the table s1 with following values

ROLLNO	NAME	PNO
10	Aswathy	D1
20	Akash	D3
30	Arun	D2

11. Try to populate book table with values null,navathe
12. Create a table account(accno,amount). Set accno as primary key. Use check option to ensure that amount does not fall below 250
13. Populate account table with following values

ACCNO	AMT
147	1000
210	5000
777	100
623	5001

QUERIES

1. create table dept1(deptno varchar(5) primary key,deptname varchar(5),staffno varchar(10));
table DEPT1 created.
2. insert into dept1 values('D1','CS','s101');

```
insert into dept1 values('D2','EC','S110');
insert into dept1 values('D1','EE','S201');
```

1 rows inserted.

1 rows inserted.

Error report -

SQL Error: ORA-00001: unique constraint violated
00001. 00000 - "unique constraint (%s.%s) violated"

*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.

For Trusted Oracle configured in DBMS MAC mode, you may see
this message if a duplicate entry exists at a different level.

*Action: Either remove the unique restriction or do not insert the key.

3. create table books(title varchar(20),author varchar(20),primary key(title));

table BOOKS created.

4 . insert into books values('dbms','navathe');

insert into books values('c','dennis ritche');

insert into books values('dbms','korth');

1 rows inserted.

1 rows inserted.

Error report -

SQL Error: ORA-00001: unique constraint violated
00001. 00000 - "unique constraint (%s.%s) violated"

*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.

For Trusted Oracle configured in DBMS MAC mode, you may see
this message if a duplicate entry exists at a different level.

*Action: Either remove the unique restriction or do not insert the key.

5. create table library(title varchar(20) references books(title),author varchar(20),noofcopies number(5));

table LIBRARY created.

6. insert into library values('c','dennis ritche',5);

insert into library values('dbms','navathe',10);

insert into library values('os','korth',20);

1 rows inserted.

1 rows inserted.

Error report -

SQL Error: ORA-02291: integrity constraint (MANEESHA.SYS_C008357) violated - parent key not
found

02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"

*Cause: A foreign key value has no matching primary key value.

*Action: Delete the foreign key or add a matching primary key.

7. alter table library modify (primary key(title,author));

table LIBRARY altered.

8. create table s1(rollno number(5) primary key,name varchar(10),dno varchar(5),foreign key(dno) references dept1(deptno));

table S1 created.

9. select * from dept1;

	DEPTNO	DEPTNAME	STAFFNO
1	D1	CS	s101
2	D2	EC	S110

10. insert into s1 values(10,'aswathy','D1');

insert into s1 values(20,'akash','D3');

insert into s1 values(30,'arun','D2');

1 rows inserted.

Error report -

SQL Error: ORA-02291: integrity constraint (MANEESHA.SYS_C008361) violated - parent key not found

02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"

*Cause: A foreign key value has no matching primary key value.

*Action: Delete the foreign key or add a matching primary key.

1 rows inserted.

11. insert into books values(' ','navathe');

Error report -

SQL Error: ORA-01400: cannot insert NULL into ("MANEESHA"."BOOKS"."TITLE")

01400. 00000 - "cannot insert NULL into (%s)"

*Cause: An attempt was made to insert NULL into previously listed objects.

*Action: These objects cannot accept NULL values.

12. create table account(accno number(5) primary key,amt number(20) check (amt>=250));

table ACCOUNT created.

```
13. insert into account values(147,1000);  
insert into account values(201,5000);  
insert into account values(777,100);  
insert into account values(623,5001);
```

1 rows inserted.

1 rows inserted.

Error starting at line : 40 in command -

```
insert into account values(777,100)
```

Error report -

SQL Error: ORA-02290: check constraint (MANEESHA.SYS_C008362) violated

02290. 00000 - "check constraint (%s.%s) violated"

*Cause: The values being inserted do not satisfy the named check

*Action: do not insert values that violate the constraint.

1 rows inserted.

```
select * from account;
```

	ACCNO	AMT
1	147	1000
2	201	5000
3	623	5001

EXPERIMENT 4

AGGREGATE FUNCTIONS

AIM

To familiarize various aggregate functions, group by, having & order by clause

DESCRIPTION

Aggregate functions return a single value, using values in a table column. Aggregate functions perform a variety of actions such as counting all the rows in a table, averaging a column's data, and summing numeric data. Aggregates can also search a table to find the highest "MAX" or lowest "MIN" values in a column.

Following is the list of all useful SQL aggregate functions –

- **SQL COUNT Function** - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- **SQL MAX Function** - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- **SQL MIN Function** - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- **SQL AVG Function** - The SQL AVG aggregate function selects the average value for certain table column.
- **SQL SUM Function** - The SQL SUM aggregate function allows selecting the total for a numeric column.

Use of DISTINCT, ALL keywords with Aggregate functions

By specifying DISTINCT keyword with the input parameter, group by function considers only the unique value of the column for aggregation. By specifying ALL keyword with the input parameter, group by function considers all the values of the column for aggregation, including nulls and duplicates. ALL is the default specification.

QUESTIONS

1. Create a table ACCOUNTS with attributes ACCNO, CUSTOMERNAME, BRANCH, TYPE, OPENINGDATE, CURRENTBALANCE
2. Retrieve average balance from the table
3. Find the total number of accounts
4. Find the name of the customer who have highest balance

QUERIES

1. create table accounts(accno varchar(5),name varchar(10),branchno varchar(5),type varchar(5),opendate date,currbal number(10));

insert into accounts values('CA1','John','B1','CA','4-Feb-2016',10000);

insert into accounts values('CA3','Greeshma','B2','CA','3-May-2014',20000);

insert into accounts values('SB4','Liya','B3','SB','07-Apr-2014',40000);

insert into accounts values('SB1','Yadhu','B1','SB','08-Dec-2016',20000);

insert into accounts values('SB2','Rohit','B2','SB','07-Nov-2016',10000);

insert into accounts values('CA2','Rohit','B2','CA','03-Oct-2013',20000);

insert into accounts values('SB3','Linju','B3','SB','03-Sep-2012',20000);

7 rows inserted

select * from accounts;

	ACCNO	NAME	BRANCHNO	TYPE	OPENDATE	CURRBAL
1	CA1	John	B1	CA	04-02-16	10000
2	CA3	Greeshma	B2	CA	03-05-14	20000
3	SB4	Liya	B3	SB	07-04-14	40000
4	SB1	Yadhu	B1	SB	08-12-16	20000
5	SB2	Rohit	B2	SB	07-11-16	10000
6	CA2	Rohit	B2	CA	03-10-13	20000
7	SB3	Linju	B3	SB	03-09-12	20000

2. select

avg(currbal)from accounts;

	AVG(CURRBAL)
1	20000

3. select count(accno) from accounts;

	COUNT(ACCNO)
1	7

4. select name from accounts where currbal=(select max(currbal) from accounts);

	NAME
1	Liya

EXPERIMENT 5

BUILT-IN FUNCTIONS

AIM :

To familiarize with numeric, date and string functions

DESCRIPTION

Date functions

To manipulate and extract values from the date column of a table

⇒ **ADD_MONTHS**: Returns date after adding the number of months specified in the function.

Syntax:

ADD_MONTHS (d, n)

⇒ **LAST_DAY**: Returns the last date of the month specified with the function

Syntax:

LAST_DAY (d)

⇒ **MONTHS_BETWEEN**: Returns number of months between d1 and d2

Syntax:

MONTHS_BETWEEN (d1, d2)

⇒ **NEXT_DAY**: Returns the date of the first weekday named by char that is after the date named by date.

Syntax:

NEXT_DAY (date, char)

⇒ **ROUND**: Returns a date rounded to a specific unit of measure.

Syntax:

ROUND (date, [format])

Numeric functions

⇒ **ABS**: Returns the absolute value of n.

Syntax:

ABS (n)

⇒ **POWER**: Returns the m raised to the nth power.

Syntax:

POWER (m, n)

⇒ **ROUND**: Returns n, rounded to m places to the right of a decimal point.

Syntax:

ROUND (n, m)

⇒ **SQRT**: Returns the square root of n.

Syntax:

SQRT (n)

⇒ **EXP**: Returns e raised to the nth power, where e=2.71828183.

Syntax:

EXP (n)

⇒ **GREATEST**: Returns the greatest value in a list of expressions.

Syntax:

GREATEST (expr1, expr2, ...expr_n)

Where expr1, expr2...expr_n are expressions that are evaluated by the greatest function.

⇒ **LEAST**: Returns the least value in a list of expressions.

Syntax:

LEAST (expr1, expr2...expr_n)

Where expr1, expr2...expr_n are expressions that are evaluated by the least function.

⇒ **MOD**: Returns the remainder of a first number divided by second number passed a parameter.

Syntax:

MOD (m, n)

⇒ **TRUNC**: Returns a number truncated to a certain number of decimal places.

Syntax:

TRUNC (number, decimal_places)

⇒ **FLOOR**: Returns the largest integer value that is equal to or less than a number.

Syntax:

FLOOR (n)

⇒ **CEIL**: Returns the smallest integer value that is equal to or greater than a number.

Syntax:

CEIL (n)

• **String Functions:**

⇒ **LOWER**: Returns char, with all letters in lower case.

Syntax:

LOWER (char)

Example:

SELECT LOWER ('ICET') "Lower" FROM DUAL;

⇒ **INITCAP**: Returns a string with the first letter of each word in upper case.

Syntax:

INITCAP (char)

⇒ **UPPER**: Returns char, with all letters forced to upper case.

Syntax:

UPPER (char)

⇒ **SUBSTR**: Returns a portion of characters, beginning at character m, and going up to character n.

Syntax:

SUBSTR (<string>, <start_position>, [<length>])

Where **string** is the source string

Start_position is the position for extraction

Length is the number of characters to extract.

⇒ **INSTR**: Returns the location of a sub string in a string.

Syntax:

SUBSTR (<string1>, <string2>,<start_position>, [<nth_appearance>])

Where **string1** is the string to search

String2 is the sub string to search for in string1.

Start_position is the position in string1 where the search will start

nth_appearance is the nth appearance of string2.

⇒ **TRANSLATE**: Replaces a sequence of characters in a string with another set of characters.

Syntax:

TRANSLATE (<string1>, <string_to_replace>, <replacement_string>)

Where **string1** is the string to replace a sequence of characters with another set of characters.

String_to_replace is the string that will be searched for in string1.

All characters in the **string_to_replace** will be replaced with the corresponding character in the

replacement string.

⇒ **LENGTH:** Returns the length of a word.

Syntax:

LENGTH (word)

Example:

SELECT LENGTH('ICET') "Length" FROM DUAL;

⇒ **LTRIM:** Removes characters from the left of char with initial characters removed up to the first character not in set.

Syntax:

LTRIM (char,set)

⇒ **RTRIM:** Returns char, with final characters removed after the last character not in the set

Syntax:

RTRIM (char,set)

⇒ **TRIM:** Removes all specified characters either from the beginning or the ending of a string.

Syntax:

LTRIM ([leading|trailing|both[<trim_character>FROM]]<string1>)

Where **leading**-remove **trim_string** from the front of **string1**.

trailing-remove **trim_string** from the end of **string1**.

both-remove **trim_string** from the front and end of **string1**.

Example:

SELECT LTRIM (' ICET ') "Trim Both sides" FROM DUAL;

SELECT TRIM (LEADING 'x' FROM 'xxxICETxxx') "Remove Prefixes" FROM DUAL;

⇒ **LPAD:** Returns char1, left padded to length n with sequence of characters specified in char2

Syntax:

LPAD (char1, n, char2)

⇒ **RPAD:** Returns char1, right padded to length n with sequence of characters specified in char2

Syntax:

RPAD (char1, n, char2)

⇒ **VSIZE:** Returns then number of bytes in the internal representation of an expression.

Syntax:

VSIZE (<expression>)

• **Conversion Functions:**

⇒ **TO_NUMBER:** Converts char, a CHARACTER value expressing a number, to a number data type

Syntax:

TO_NUMBER (char)

⇒ **TO_CHAR(number conversion):** Converts a value of a NUMBER data type to a character data type, using the optional format string.

Syntax:

TO_CHAR (n ,fmt)

⇒ **TO_CHAR (date conversion):** Converts a value of a DATE data type to a character data type, using the optional format string.

Syntax:

TO_CHAR (date, fmt)

⇒ **TO_DATE (date conversion):** Converts a character field to a date field.

Syntax:

TO_DATE (char, fmt)

QUESTIONS

1. Display today's date tomorrow's date.
2. Display the last date of this month.
3. Display the date of next monday.
4. Display the minimum length of employee name from employee table.
5. Display the details of employees whose name length is greater than 5.
6. Display the number of employees joined in each month.
7. Retrieve three characters from the word “encyclopedia” with respect to 4th position.
8. Replace all a's in the word “malayalam”.
9. Remove “OR” from the word “ORACLE”.
10. Write query to pad left of the word “DATABASE” with “.” to a column width of 12 and then pad right with “_” upto a length of 15.
11. Write a query to find daily salary of the employees in the table emp. Assume 30 days are in a month. Round daily salary into 2 positions. Then name the column as “daily pay”

ANSWERS

```
mysql> CREATE TABLE EMPLOYEE(EmpId int,Ename varchar(30),Eaddress  
varchar(30),Designation varchar(30),Dept varchar(30),Salary int,Join_date date);  
Query OK, 0 rows affected (0.43 sec)
```

```
mysql> DESC EMPLOYEE;
```

```
+-----+-----+-----+-----+-----+  
| Field      | Type      | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+  
| EmpId      | int       | YES  |     | NULL    |      |  
| Ename      | varchar(30) | YES  |     | NULL    |      |  
| Eaddress   | varchar(30) | YES  |     | NULL    |      |  
| Designation | varchar(30) | YES  |     | NULL    |      |  
| Dept       | varchar(30) | YES  |     | NULL    |      |  
| Salary     | int       | YES  |     | NULL    |      |  
| Join_date  | date      | YES  |     | NULL    |      |  
+-----+-----+-----+-----+-----+  
7 rows in set (0.06 sec)
```

```
mysql> INSERT INTO EMPLOYEE VALUES(1516,'Vishnu','Edappilly','HOD','Sales',45000,'2015-01-03');  
Query OK, 1 row affected (0.03 sec)
```

```
mysql> INSERT INTO EMPLOYEE VALUES(1517,'Vimal','Kakkanad','Manager','Food',22500,'2018-05-09');  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO EMPLOYEE VALUES(1518,'Vinu','Kolanchery','MD','Admin',75000,'2012-05-09');
```

Query OK, 1 row affected (0.01 sec)

```
mysql> INSERT INTO EMPLOYEE  
VALUES(1518,'Vrindha','Kolanchery','HOD','Market',35000,'2014-09-09');
```

Query OK, 1 row affected (0.01 sec)

```
mysql> SELECT * FROM EMPLOYEE;
```

EmpId	Ename	Eaddress	Designation	Dept	Salary	Join_date
1516	Vishnu	Edappilly	HOD	Sales	45000	2015-01-03
1517	Vimal	Kakkanad	Manager	Food	22500	2018-05-09
1518	Vinu	Kolanchery	MD	Admin	75000	2012-05-09
1518	Vrindha	Kolanchery	HOD	Market	35000	2014-09-09

4 rows in set (0.01 sec)

```
mysql> SELECT CURDATE() AS Today,DATE_ADD(CURDATE(),INTERVAL 1 DAY) AS  
Tomorrow;
```

Today	Tomorrow
2022-02-09	2022-02-10

1 row in set (0.01 sec)

```
mysql> SELECT LAST_DAY(NOW()) AS LAST_DAY_OF_MONTH;
```

LAST_DAY_OF_MONTH
2022-02-28

1 row in set (0.00 sec)

```
mysql> SELECT CURDATE()+INTERVAL 7-WEEKDAY(CURDATE())DAY AS NEXT_MONDAY;
```

NEXT_MONDAY
2022-02-14

```
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT LENGTH(Ename)FROM EMPLOYEE WHERE LENGTH(Ename)=(select
min(length(ename)) from EMPLOYEE);
```

```
+-----+
| LENGTH(Ename) |
+-----+
|          4 |
+-----+
```

```
1 row in set (0.01 sec)
```

```
mysql> SELECT * FROM EMPLOYEE WHERE LENGTH(Ename)>5;
```

```
+-----+-----+-----+-----+-----+-----+-----+
| EmpId | Ename  | Eaddress | Designation | Dept  | Salary | Join_date |
+-----+-----+-----+-----+-----+-----+-----+
| 1516 | Vishnu | Edappilly | HOD         | Sales | 45000 | 2015-01-03 |
| 1518 | Vrindha | Kolanchery | HOD         | Market | 35000 | 2014-09-09 |
+-----+-----+-----+-----+-----+-----+-----+
```

```
2 rows in set (0.00 sec)
```

```
mysql> SELECT monthname(Join_date) as Month,COUNT(*) AS COUNT FROM EMPLOYEE
GROUP BY monthname(Join_date);
```

```
+-----+-----+
| Month  | COUNT |
+-----+-----+
| January | 1 |
| May     | 2 |
| September | 1 |
+-----+-----+
```

```
3 rows in set (0.01 sec)
```

```
mysql> SELECT SUBSTR("ENCYCLOPEDIA",4,3);
```

```
+-----+
| SUBSTR("ENCYCLOPEDIA",4,3) |
+-----+
| YCL                        |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT REPLACE("MALAYALAM","A","B") AS RESULT;
```

```

+-----+
| RESULT |
+-----+
| MBLBYBLBM |
+-----+
1 row in set (0.00 sec)

```

```

mysql> SELECT TRIM(both 'OR' FROM "ORACLE") AS RESULT;
+-----+
| RESULT |
+-----+
| ACLE   |
+-----+
1 row in set (0.00 sec)

```

```

mysql> SELECT RPAD(LPAD('DATABASE',12,'.'),15,'_') AS PADDED;
+-----+
| PADDED      |
+-----+
| ....DATABASE____ |
+-----+
1 row in set (0.00 sec)

```

```

mysql> SELECT Ename,ROUND(Salary/30,2) AS daily_pay FROM EMPLOYEE;
+-----+-----+
| Ename  | daily_pay |
+-----+-----+
| Vishnu | 1500.00 |
| Vimal  | 750.00 |
| Vinu   | 2500.00 |
| Vrindha | 1166.67 |
+-----+-----+
4 rows in set (0.00 sec)

```

EXPERIMENT 6

GROUP BY, ORDER BY AND HAVING CLAUSES

AIM

To familiarize the SQL constructs GROUP BY, HAVING and ORDER BY clauses.

DESCRIPTION

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
SELECT column1, column2  
FROM table_name  
WHERE [ conditions ]  
GROUP BY column1, column2  
ORDER BY column1, column2
```

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

```
SELECT column-list  
FROM table_name  
[WHERE condition]  
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

```
SELECT  
FROM  
WHERE  
GROUP BY  
HAVING  
ORDER BY
```

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Questions

1. Find the total number of accounts segregated on the basis of account type per branch
2. Find out the customer having more than one account type in the bank
3. Find out the number of accounts opened in each branch after 3/01/2015 only if the number of accounts opened after 3/01/2015 exceed 1
4. Find out total number of accounts of each branch
5. Find out number of accounts at each branch in the order of branch number

Queries

1. select count(accno),branchno,type from accounts group by type,branchno;

	COUNT(ACCNO)	BRANCHNO	TYPE
1	1	B2	SB
2	1	B1	CA
3	2	B3	SB
4	2	B2	CA
5	1	B1	SB

2. select name from accounts group by branchno,name having (count(type)>1);

	NAME
1	Rohit

3. select count(accno) from accounts where opendate>'3-Jan-2015' group by branchno having (count(accno)>1);

	COUNT(AC...)
1	2

4. select branchno,count(accno) from accounts group by branchno;

	BRAN...	COUNT(ACCNO)
1	B1	2
2	B3	2
3	B2	3

5.select branchno,count(accno) from accounts group by branchno order by branchno;

	BRANCHNO	COUNT(ACCNO)
1	B1	2
2	B2	3
3	B3	2

EXPERIMENT 7

NESTED QUERIES

AIM

To manipulate data using sub queries.

DESCRIPTION

MULTIPLE SUBQUERIES

Purpose:

Nesting is the act of embedding sub query within another subquery. Sub queries can be nested as deeply as your implementation of SQL allows.

Syntax:

```
SELECT * FROMtablename  
WHERE subquery(subquery(subquery));
```

CORRELATED SUBQUERIES

A subquery is evaluated once for the entire parent statement whereas a correlated subquery is evaluated once for every row processed by the parent statement.

QUESTIONS

1. Implement a table employee(eno,ename,job,salary,dno,grade)
2. Populate table employee with following tuples

ENO	ENAME	JOB	SALARY	DNO	GRADE
1	ABC	Clerk	5000	10	A
2	BCD	manager	100000	30	C
3	CDE	it professor	200000	20	D
4	DEF	manager	1500000	20	F

3. Implement table dept(dno,dname,location)
4. Populate table dept with following tuples

DNO	DNAME	LOCATION
10	Banking	chennai

20	It	banglore
30	Finance	delhi
40	Hr	hydrabad

5. Implement another table salary_grade(grade,low_sal,high_sal)
6. Populate table salary_grade with following tuples

GRADE	LOW_SAL	HIGH_SAL
A	20000	50000
B	50001	99999
C	100000	149999
D	150000	199999
E	200000	250000

7. List all departments having atleast one employee
8. List all employees to get salary more than BCD
9. List all employees who do same job as that of DEF
10. List name and salary of all employees whose salary is greater than salary of all employees working in department 30
11. List low salary of all employees whose grade is same as that of ABC
12. List deptname of employees having highest salary
13. List average salary of employees in IT department
14. List details of all employees & IT professionals in IT dept whose salary is greater than or equal to 170000
15. Retrieve name & job of employees in IT dept which is sorted in the order of name
16. List the name of employees having salary greater than avg of low salary & high salary
17. List dept name having no. of employees greater than banking department.

QUERIES

1. create table employe(eno number(5),ename varchar(10),job varchar(20),salary number(10),dno number(5),grade varchar(5));
table EMPLOYE created.

2. insert into employe values(1,'abc','clerk',50000,10,'a');
insert into employe values(2,'bcd','manager',100000,30,'c');
insert into employe values(3,'cde','it professional',200000,20,'d');
insert into employe values(1,'def','manager',150000,20,'f');

4 rows inserted.

select * from employe;

	ENO	ENAME	JOB	SALARY	DNO	GRADE
1	1	abc	clerk	50000	10	a
2	2	bcd	manager	100000	30	c
3	3	cde	it professional	200000	20	d
4	1	def	manager	150000	20	f

3. create table dept2(dno number(5),dname varchar(20),location varchar(20));

table DEPT2 created.

4. insert into dept2 values(10,'banking','chennai');

insert into dept2 values(20,'it','banglore');

insert into dept2 values(30,'finance','delhi');

insert into dept2 values(40,'hr','hyderabad');

4 rows inserted.

select * from dept2;

	DNO	DNAME	LOCATION
1	10	banking	chennai
2	20	it	banglore
3	30	finance	delhi
4	40	hr	hyderabad

5. create table salary_grade(grade varchar(5),lowsal number(20),highsal number(20));

table SALARY_GRADE created.

6. insert into salary_grade values('a',20000,50000);

insert into salary_grade values('b',50001,99999);

insert into salary_grade values('c',100000,149999);

insert into salary_grade values('d',200000,250000);

insert into salary_grade values('e',150000,199999);

select * from salary_grade;

	GRADE	LOWSAL	HIGHSAL
1	a	20000	50000
2	b	50001	99999
3	c	100000	149999
4	d	200000	250000
5	e	150000	199999

7. select dname from dept2 where dno in (select dno from employe);

	DNAME
1	banking
2	finance
3	it

8. select ename from employe where salary > (select salary from employe where ename='bcd');

	E...
1	cde
2	def

9. select ename from employe where job in (select job from employe where ename='def') and ename != 'def';

	ENAME
1	bcd

10. select ename,salary from employe where salary > (select salary from employe where dno=30);

	ENAME	SALARY
1	cde	200000
2	def	150000

11. select lowsal from salary_grade where grade=(select grade from employee where ename='abc');

	LOWSAL
1	20000

12. select dname from dept2 where dno=(select dno from employee where salary=(select max(salary) from employee));

	DNAME
1	it

13. select avg(salary)from employee where dno=(select dno from dept2 where dname='it');

	AVG(SALARY)
1	175000

14. select ename from employee where job in('manager','it professional')and salary>=170000 and dno=(select dno from dept2 where dname='it');

	ENAME
1	cde

15. select ename,job from employe where dno=(select dno from dept2 where dname='it')order by ename;

	ENAME	JOB
1	cde	it professional
2	def	manager

16. select ename from employe where salary>(select avg(lowsal) from salary_grade) and salary>(select avg(highsal) from salary_grade);

	ENAME
1	cde
2	def

17. select dname from dept2 where dno=(select dno from employe group by dno having count(dno)>(select count(dno) from employe where dno=(select dno from dept2 where dname='banking')));

	DNAME
1	it

EXPERIMENT 8

FAMILIARIZATION OF SET OPERATORS AND JOINS

AIM

To familiarize with set operators and joins in sql

DESCRIPTION

Set operators

Set operators are used to join the results of two (or more) SELECT statements. The SET operators available in Oracle 11g are UNION, UNION ALL, INTERSECT, and MINUS.

To perform set operation participating select statement must satisfy the following condition

- Same number of columns must be selected by all participating SELECT statements. Column names used in the display are taken from the first query.
- Data types of the column list must be compatible.

UNION

When multiple SELECT queries are joined using UNION operator, Oracle displays the combined result from all the compounded SELECT queries, after removing all duplicates and in sorted order (ascending by default), without ignoring the NULL values.

Eg:

```
SELECT 1 NUM FROM DUAL
UNION
SELECT 5 FROM DUAL
UNION
SELECT 3 FROM DUAL
UNION
SELECT 6 FROM DUAL
UNION
SELECT 3 FROM DUAL;
```

NUM

1
3
5

UNION ALL

UNION and UNION ALL are similar in their functioning with a slight difference. But UNION ALL gives the result set without removing duplication and sorting the data.

Eg:

```
SELECT 1 NUM FROM DUAL
UNION ALL
SELECT 5 FROM DUAL
UNION ALL
SELECT 3 FROM DUAL
UNION ALL
SELECT 6 FROM DUAL
UNION ALL
SELECT 3 FROM DUAL;
```

NUM

1
5
3
6
3

INTERSECT

Using INTERSECT operator, Oracle displays the common rows from both the SELECT statements, with no duplicates and data arranged in sorted order (ascending by default).

Eg:

```
SELECT SALARY
FROM employees
WHERE DEPARTMENT_ID = 10
INTRESECT
SELECT SALARY
FROM employees
WHERE DEPARTMENT_ID = 20
```

SALARY

1500
1200
2000

MINUS

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data arranged in ascending order by default.

Eg:

```
SELECT JOB_ID
FROM employees
WHERE DEPARTMENT_ID = 10
MINUS
SELECT JOB_ID
FROM employees
WHERE DEPARTMENT_ID = 20;
```

JOB_ID

HR
FIN
ADMIN

Joins

There are different types of joins available in SQL:

- **INNER JOIN**: returns rows when there is a match in both tables.
- **LEFT OUTER JOIN**: returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT OUTER JOIN**: returns all rows from the right table, even if there are no matches in the left table.
- **FULL OUTER JOIN**: returns rows when there is a match in one of the tables.
- **SELF JOIN**: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- **CARTESIAN JOIN**: returns the Cartesian product of the sets of records from the two or more joined tables.

QUESTIONS

1. Create the following table
 - a) book_details(ISBN, title, MRP, publisher name, author)
 - b) publisher (publisher_id, publisher name, city, state, country)
2. Populate the tables.
3. Display the details of all books.
4. Retrieve the details of all publishers.
5. List the name of books, price and city of publisher of all books.
6. List the details of books and their corresponding publisher details.
7. List all publishers, details of books published by each publisher.
8. List the name of publishers which have got entry either in book_details or publisher table.
9. List the name of publisher which have got entry in both tables.
10. List the name of publisher that have got entry in book_details but not in publisher table.

QUERIES

1. create table book_detail(ISBN number(5),TITLE varchar(20),MRPnumber(5),PUBLISHER_NAME varchar(10),AUTHOR varchar(20));

table BOOK_DETAIL created.

create table publisher(publisher_id number(5),publisher_name varchar(20),city varchar(20),state varchar(20),country varchar(20));

table PUBLISHER created.

2. insert into book_detail values(978,'DB',650,'PEARSON','NAVATHE');

insert into book_detail values(980,'OS',625,'WILEY','SILBERSCHATZ');

insert into book_detail values(920,'JAVA',649,'MCGRAWHILL','SCHILDT');

insert into book_detail values(989,'MICROPROCESSOR',720,'MCGRAWHILL','GAONKER');

insert into book_detail values(970,'ALGORITHMS',995,'MIT','COREMAN');

insert into publisher values(101,'PEARSON','LA','CALIFORNIA','USA');

insert into publisher values(102,'WILEY','SHEFFIELD','LONDON','UK');

insert into publisher values(103,'MCGRAWHILL','NYCITY','NY','USA');

insert into publisher values(104,'BAVARIA','BAVARIA','BERLIN','GERMANY');

3. select * from book_detail;

	ISBN	TITLE	MRP	PUBLISHER_NAME	AUTHOR
1	978	DB	650	PEARSON	NAVATHE
2	980	OS	625	WILEY	SILBERSCHATZ
3	920	JAVA	649	MCGRAWHILL	SCHILDT
4	989	MICROPROCESSOR	720	MCGRAWHILL	GAONKER
5	970	ALGORITHMS	995	MIT	COREMAN

4. select * from publisher;

	PUBLISHER_ID	PUBLISHER_NAME	CITY	STATE	COUNTRY
1	101	PEARSON	LA	CALIFORNIA	USA
2	102	WILEY	SHEFFIELD	LONDON	UK
3	103	MCGRAWHILL	NYCITY	NY	USA
4	104	BAVARIA	BAVARIA	BERLIN	GERMANY

5. select b.title,b.mrp,p.city
from book_detail b left
outer join publisher p on

b.publisher_name=p.publisher_name;

	TITLE	MRP	CITY
1	DB	650	LA
2	OS	625	SHEFFIELD
3	MICROPROCESSOR	720	NYCITY
4	JAVA	649	NYCITY
5	ALGORITHMS	995	(null)

6. select * from book_detail b left outer join publisher p on b.publisher_name=p.publisher_name;

	ISBN	TITLE	MRP	PUBLISHER_NAME	AUTHOR
1	978	DB	650	PEARSON	NAVATHE
2	980	OS	625	WILEY	SILBERSCHATZ
3	989	MICROPROCESSOR	720	MCGRRAWHILL	GAONKER
4	920	JAVA	649	MCGRRAWHILL	SCHILDT
5	970	ALGORITHMS	995	MIT	COREMAN

PUBLISHER_ID	PUBLISHER_NAME_1	CITY	STATE	COUNTRY
101	PEARSON	LA	CALIFORNIA	USA
102	WILEY	SHEFFIELD	LONDON	UK
103	MCGRRAWHILL	NYCITY	NY	USA
103	MCGRRAWHILL	NYCITY	NY	USA
(null)	(null)	(null)	(null)	(null)

7. select * from publisher p right outer join

book_detail b on b.publisher_name=p.publisher_name;

	PUBLISHER_ID	PUBLISHER_NAME	CITY	STATE
1	101	PEARSON	LA	CALIFORNIA
2	102	WILEY	SHEFFIELD	LONDON
3	103	MCGRRAWHILL	NYCITY	NY
4	103	MCGRRAWHILL	NYCITY	NY
5	(null)	(null)	(null)	(null)

COUNTRY	ISBN	TITLE	MRP	PUBLISHER_NAME_1	AUTHOR
USA	978	DB	650	PEARSON	NAVATHE
UK	980	OS	625	WILEY	SILBERSCHATZ
USA	989	MICROPROCESSOR	720	MCGRRAWHILL	GAONKER
USA	920	JAVA	649	MCGRRAWHILL	SCHILDT
(null)	970	ALGORITHMS	995	MIT	COREMAN

8. select

publisher_name from book_detail union select publisher_name from publisher;

PUBLISHER_NAME
1 BAVARIA
2 MCGRRAWHILL
3 MIT
4 PEARSON
5 WILEY

9. select publisher_name from book_detail intersect select publisher_name from publisher;

	PUBLISHER_NAME
1	MCGRAWHILL
2	PEARSON
3	WILEY

10. select publisher_name from book_detail minus select publisher_name from publisher;

	PUBLISHER_NAME
1	MIT

EXPERIMENT 9

VIEWS

AIM

To familiarize with views.

DESCRIPTION

Views

A view is a virtual table based on the result-set of an SQL statement.

Creating Views

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.

The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE [condition];
```

The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the conditions in the view definition.

Consider the following query,

```
CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS WHERE age IS NOT NULL WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Read-Only View

We can create a view with read-only option to restrict access to the view.

Syntax

```
create or replace view view_name as select column(s) from table_name where condition with read only;
```

Dropping Views:

The syntax for removing created view is

```
DROP VIEW view_name;
```


QUESTIONS

1. Create table product (pid,pname,unitprice,manufacturer,category,country).
2. Populate the table and display the details.
3. Create view v1 with pid, pname and category.
4. List the pname and country of all products whose category is 'home appliances' using views.
5. Populate v1 with values 501,XY505,mobilephone.
6. Modify pname as Xseries whose pid is 200.
7. Display the view to reflect the updation.
8. Modify the product table to set pid as primary key.
9. Create another view v2 with attributes pname,unit price and category.
10. Populate v2 with a row.
11. Delete the details of product whose pname is 'Lenovo'.
12. Define another view v3 that contains employee id,ename,salary,dept id,and dept name.(Create appropriate tables).
13. Modify the salary of all employees who belong to dept 10 by rupees 1000.
14. Modify the department table to set dept id as primary key.
15. Modify the table to set eid as primary and dept id as foreign key.
16. Insert a row to v3.

QUERIES

1. create table product(pid number(10),pname varchar(20),unitprice number(20),manufac varchar(20),categ varchar(20),country varchar(20));

table PRODUCT created.

2. 5 rows inserted.

select * from product;

	PID	PNAME	UNITPRICE	MANUFAC	CATEG	COUNTRY
1	101	Iphone8	80000	Apple	Mobile	USA
2	102	Mi6	32000	Xiaomi	Mobile	China
3	173	Ideapad	45000	Lenovo	Laptop	Korea
4	178	Jazz	2000	Zoook	Speaker	Germany
5	217	Cruzer	5000	Titan	Watch	India
6	200	Zappa	10000	Phillips	Home Ap...	Korea

3. create view v1 as select pid,pname,categ from product;

view V1 created.

select * from v1;

	PID	PNAME	CATEG
1	101	Iphone8	Mobile
2	102	Mi6	Mobile
3	173	Ideapad	Laptop
4	178	Jazz	Speaker
5	217	Cruzer	Watch
6	200	Zappa	Home Appliances

4. select pname,country from v1 where categ='Home Appliances';

ORA-00904: "COUNTRY": invalid identifier

5. insert into v1 values(501,'XY505','Mobile Phone');

1 rows inserted.

select * from v1;

	PID	PNAME	CATEG
1	101	Iphone8	Mobile
2	102	Mi6	Mobile
3	173	Ideapad	Laptop
4	178	Jazz	Speaker
5	217	Cruzer	Watch
6	200	Zappa	Home Appliances
7	501	XY505	Mobile Phone

6. update v1 set pname='X Series' where pid=200;

1 rows updated.

7. `select * from v1;`

	PID	PNAME	CATEG
1	101	Iphone8	Mobile
2	102	Mi6	Mobile
3	173	Ideapad	Laptop
4	178	Jazz	Speaker
5	217	Cruzer	Watch
6	200	X Series	Home Appliances
7	501	XY505	Mobile Phone

8. `alter table product modify(primary key(pid));`

table PRODUCT altered.

9. `create view v2 as select pname,unitprice,categ from product;`

view V2 created.

10. `insert into v2 values('3D700',90001,'TV');`

Error: These objects cannot accept NULL values

The previous row inserted to views made changes in parent table also. Therefore creating another view causes fields with null values.

`select * from v2;`

	PNAME	UNIT...	CATEG
1	Iphone8	80000	Mobile
2	Mi6	32000	Mobile
3	Ideapad	45000	Laptop
4	Jazz	2000	Speaker
5	Cruzer	5000	Watch
6	X Series	10000	Home Appliances
7	XY505	(null)	Mobile Phone

11. `delete from v2 where
pname='Lenovo';`

0 rows deleted

12. `create table depart(dno number(10),dname varchar(20),mgrid number(10));`

`insert into depart values(10,'cse',2);`

```
insert into depart values(20,'ece',11);
```

```
insert into depart values(30,'eee',3);
```

```
insert into depart values(40,'civil',7);
```

```
insert into depart values(50,'mech',6);
```

```
select * from depart;
```

```
select * from emplo;
```

create view v3 as select	DNO	D...	MGRID	EID	ENAME	SALARY	HIREDATE	DNO
	1	10 cse	2	1	1 aby	80000	01-01-01	10
	2	20 ece	11	2	2 denny	32000	06-06-06	20
	3	30 eee	3	3	3 hiran	45000	17-09-03	30
	4	40 civil	7	4	4 kamal	2000	20-08-04	40
	5	50 mech	6	5	5 noyal	5000	01-05-05	50

```
e.eid,e.ename,e.salary,d.dno,d.dname from  
emplo e left outer join depart d on d.dno=e.dno;
```

view V3 created.

```
select * from v3;
```

EID	ENAME	SALARY	DNO	DNAME
1	1 aby	80000	10 cse	
2	2 denny	32000	20 ece	
3	3 hiran	45000	30 eee	
4	4 kamal	2000	40 civil	
5	5 noyal	5000	50 mech	

```
13. update v3 set salary=salary+1000 where dno=10;
```

```
select * from v3;
```

Error: "cannot modify a column which maps to a non key-preserved table"

*Cause: An attempt was made to insert or update columns of a join view which map to a non-key-preserved table.

*Action: Modify the underlying base tables directly.

```
14. alter table depart modify(primary key(dno));
```

table DEPART altered.

15. alter table emplo modify(primary key(eid),foreign key(dno) references depart(dno));

table EMPLO altered.

16. insert into v3 values(6,'Sabu',200,60,'manager');

SQL Error: ORA-01776: cannot modify more than one base table through a join view

01776. 00000 - "cannot modify more than one base table through a join view"

*Cause: Columns belonging to more than one underlying table were either
inserted into or updated.

*Action: Phrase the statement as two or more separate statements.

EXPERIMENT 10

.BASICS OF PL/SQL

AIM

To familiarize with basics of PL/SQL

DESCRIPTION

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a **block**. All PL/SQL programs are made up of blocks, which can be nested within each other. Typically, each block performs a logical action in the program.

Benefits of using PL/SQL include:

- ◆ Modularity
- ◆ Variables
- ◆ Control structures
- ◆ Superior Performance
- ◆ Error Handling
- ◆ Support for SQL
- ◆ Portability
- ◆ Support for object orientation

Types of PL/SQL blocks – PL/SQL blocks come in two varieties:

- ◆ Anonymous blocks –These are pieces of PL/SQL code that have no header with a name. They are send to PL/SQL engine through SQL*Plus.
- ◆ Named blocks – A named block can be called one or more times by its name. There are four types of named sub programs: Procedures, Functions, packages, Triggers

Block Structure –The basic structure of an anonymous block is shown below

DECLARE

Declarations

BEGIN

Executable statements

EXCEPTION

Error handlers

END;

- ◆ DECLARE and EXCEPTION keywords are optional.
- ◆ Each statement can take up one or more lines and include tabs and spaces.
- ◆ Each statement must end with a semicolon (;)
- ◆ The PL/SQL code within any type of block is case insensitive.
- ◆ Single line comments: Denoted by two dashes (-) immediately before the comment.
- ◆ Multi line comment: This begins with /* and ends with */.

PL/SQL Data Types: The default data types that can be declared in PL/SQL are

- ◆ Number – for storing numeric data
- ◆ Char – for storing character data
- ◆ Date – for storing date and time data
- ◆ Boolean – for storing TRUE, FALSE or NULL
- ◆ %TYPE – declares a variable or constant to have same data type as that of a previously defined variable or of a column in a table or in a view.
- ◆ NOT NULL – causes creation of a variable or a constant that cannot be assigned a null value.

Displaying User messages on Display screen:

- ◆ DBMS_OUTPUT is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrieved later.
- ◆ PUT_LINE puts a piece of information in the package buffer followed by an end of line marker.
- ◆ Eg: dbms_output.put_line('Any message to display');
- ◆ To display messages, the SERVEROUTPUT should be set ON.
SET SERVEROUTPUT ON

Lab questions on PL/SQL

1. Sum of two numbers.

set serveroutput on;

PROGRAM

Declare

Var1 integer;

Var2 integer;

Var3 integer;

Begin

Var1:=&var1;

Var2:=&var2;

Var3:=var1+var2;

Dbms_output.put_line('SUM IS '||var3);

End;

OUTPUT

Enter the value of Var1 : 10

Enter the value of Var2 : 15

anonymous block completed

SUM IS 25

2. Largest of three numbers

PROGRAM

set serveroutput on;

Declare

a number;

b number;

c number;

Begin

a:=&a;

b:=&b;

c:=&c;

if (a>b) and (a>c)

then

dbms_output.put_line('A is GREATEST '||A);

elsif (b>a) and (b>c)


```

then

    dbms_output.put_line('B is GREATEST '||B);
else

    dbms_output.put_line('C is GREATEST '||C);
end if;

End;

```

OUTPUT

Enter the vlaue of a: 5

Enter the value of b :10

Enter the value of c : 8

anonymous block completed

B is GREATEST 10

3. Factorial of a number

PROGRAM

```

set serveroutput on;

declare

    i number(4):=1;

    n number(4):=&n;

    f number(4):=1;

begin

    for i in 1..n

    loop

        f:=f*i;

    end loop;

    Dbms_output.put_line('the factorial of '||n||' is:'||f);

end;

```

OUTPUT

anonymous block completed

Enter the value of n : 5

the factorial of 5 is:120

4. Fibonacci series

PROGRAM

```
set serveroutput on;
```

```
declare
```

```
    first number:=0;
```

```
    second number:=1;
```

```
    third number;
```

```
    n number:=&n;
```

```
    i number;
```

```
begin
```

```
    dbms_output.put_line('Fibonacci series is:');
```

```
    dbms_output.put_line(first);
```

```
    dbms_output.put_line(second);
```

```
    for i in 2..n
```

```
    loop
```

```
        third:=first+second;
```

```
        first:=second;
```

```
        second:=third;
```

```
        dbms_output.put_line(third);
```

```
end loop;
```

```
end;
```

OUTPUT

anonymous block completed

Enter the value of n : 7

```
Fibonacci series is:  
0  
1  
1  
2  
3  
5  
8  
13
```

5. Reverse of a number

PROGRAM

```
declare
```

```
    n number;
```

```
    i number;
```

```
    rev number:=0;
```

```
    r number;
```

```
begin
```

```
    n:=&n;
```

```
    while n>0
```

```
    loop
```

```
        r:=mod(n,10);
```

```
        rev:=(rev*10)+r;
```

```
n:=trunc(n/10);  
end loop;  
  
dbms_output.put_line('reverse is '||rev);  
end;
```

OUTPUT

anonymous block completed

Enter the value of n : 1245

reverse is 5421

6. String palindrome

PROGRAM

DECLARE

```
len number;  
palstr varchar2(20) := '&palstr';  
chkstr varchar2(20);
```

BEGIN

```
len := length(palstr);  
for i in REVERSE 1..len loop  
    chkstr := chkstr||substr(palstr,i,1);  
end loop;  
if chkstr = palstr then  
    dbms_output.put_line(palstr||' is a PALINDROME');  
else  
    dbms_output.put_line(palstr||' is not a PALINDROME');
```

end if;

END;

OUTPUT

anonymous block completed

Enter the vlue of palstr : hai hello

hai hello is not a PALINDROME

anonymous block completed

Enter the vlue of palstr : malayalam

malayalam is a PALINDROME

7. Write a program to store even and odd numbers from 1 to 20 to the tables EVEN and ODD respectively.

PROGRAM

create table even1(nos number(2));

create table odd1(nos number(2));

declare

n number:=&n;

begin

for i in 1..n

loop

if mod(i,2)=0

then

insert into even1 values(i);

else

insert into odd1 values(i);

end if;

end loop;

end;

select * from even1;

select * from odd1;

OUPUT

Enter the value of n : 20

	NOS	
1	2	
2	4	
3	6	
4	8	
5	10	
6	12	
7	14	
8	16	
9	18	
10	20	

	NOS	
1	1	
2	3	
3	5	
4	7	
5	9	
6	11	
7	13	
8	15	
9	17	
10	19	

EXPERIMENT 11

PROCEDURES AND FUNCTIONS

AIM

To implement programs using procedures and functions

DESCRIPTION

Just as you can in other languages, you can create your own procedures in Oracle.

Syntax

The syntax to create a procedure in Oracle is:

CREATE [OR REPLACE] PROCEDURE procedure_name

 [(parameter [,parameter])]

IS

 [declaration_section]

BEGIN

 executable_section

[EXCEPTION

 exception_section]

END [procedure_name];

There are three types of parameters that can be declared:

1. **IN** - The parameter can be referenced by the procedure or function. The value of the parameter can not be overwritten by the procedure or function.
2. **OUT** - The parameter can not be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **IN OUT** - The parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

Drop Procedure

Once you have created your procedure in Oracle, you might find that you need to remove it from the database.

Syntax

The syntax to drop a procedure in Oracle is:

DROP PROCEDURE procedure_name;

procedure_name -The name of the procedure that you wish to drop.

QUESTION

1) write a PL/SQL Procedure to find largest of two Numbers

2) create a table employee(empid,empname,salary,dept,wef)

write a procedure to accept two arguments empid and salary increment(in %).update the employee table with the salary increment also record the effective date.

PROGRAM

1. create procedure largest (a1 in number,b1 in number,c1 out number)

as

begin

 if (a1>b1) then

 c1:=a1;

 else

 c1:=b1;

 end if;

end largest;

set serveroutput on;

declare

 a2 number:=&a2;

 b2 number:=&b2;

 c2 number:=0;

begin

 largest(a2,b2,c2);

 dbms_output.put_line('Largest: '||c2);

end;

OUTPUT

anonymous block completed

Enter the value of a2 : 12

Enter the value of b2 : 25

Largest: 25

```
2. create table employ(eid number(5),nam varchar(6),sal number(6),dep varchar(8),wef date);
```

```
insert into employ values(1,'jose',1000,'finance','03-05-16');
```

```
insert into employ values(2,'vimal',2000,'hr','11-06-16');
```

```
insert into employ values(3,'yadhu',3000,'sales','17-08-16');
```

```
insert into employ values(4,'robin',4000,'finance','09-01-16');
```

```
insert into employ values(5,'roshin',5000,'hr','07-04-16');
```

```
create procedure upda(a1 in out number,b1 in out number)
```

```
as
```

```
d date;
```

```
begin
```

```
    select sysdate into d from DUAL;
```

```
    update employ set sal=sal+((sal*b1)/100) where eid=a1;
```

```
    update employ set wef=d where eid=a1;
```

```
end upda;
```

```
set serveroutput on;
```

```
declare
```

```
    a2 number:=&a2;
```

a3 number:=&a3;

begin

upda(a2,a3);

end;

OUTPUT

select * from employ;

	EID	NAM	SAL	DEP	WEF
1	1	jose	1000	finance	03-05-16
2	2	vimal	2000	hr	11-06-16
3	3	yadhu	3000	sales	17-08-16
4	4	robin	4000	finance	09-01-16
5	5	roshin	5000	hr	07-04-16

anonymous block completed

	EID	NAM	SAL	DEP	WEF
1	1	jose	1000	finance	03-05-16
2	2	vimal	2400	hr	29-11-17
3	3	yadhu	3000	sales	17-08-16
4	4	robin	4000	finance	09-01-16
5	5	roshin	5000	hr	07-04-16

FUNCTION

A standalone function is created using the **CREATE FUNCTION** statement. The syntax is given by

CREATE [OR REPLACE] FUNCTION function_name ((parameter_name {IN} type {, ...}))

RETURN return_datatype

{IS | AS}

<declaration section>

BEGIN

< function_body >

END;

Where,

- *function-name* specifies the name of the function.

- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *Function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value.

QUESTIONS

- 1) write a PL/SQL function to find factorial of a Number.
- 2) write a PL/SQL function to find sum of 1st N even Numbers (additional question)
- 3) sales of different products in one week is recorded

Product (productid, productname, grade)

Sales (prdcname, salesamount, salesday)

Do the following

- a) write a function that displays the product name and grade of the given product
- b) whenever the product sales is greater than the target value it is given a A grade, if there is no sale for a product an exception to be raised

PROGRAM

```

1) create function factorial (a1 in number)
return number
as
  f number:=1;
  i number:=1;
begin
  while(i<=a1)

```

```

loop
  f:=f*i;
  i:=i+1;
end loop;
return (f);
end;

```

```

set serveroutput on;
declare
  a2 number:=&a2;
  c2 number:=0;
begin
  c2:=factorial(a2);
  dbms_output.put_line('Factorial: '||c2);
end;

```

OUTPUT

anonymous block completed
Enter the value of a2: 6
Factorial: 720

2) create table product(pid number(5),pname varchar(10),pgrade varchar(3));

```

insert into product values(1,'HDD','a');
insert into product values(2,'GoPro','b');
insert into product values(3,'laptop','b');
insert into product values(4,'mobile','b');
insert into product values(5,'DVD','c');

```

```

create table sales(pid number(5),samount number(10),sdate date,sday varchar(15));
insert into sales values(1,1000,'2-10-2017','Monday');
insert into sales values(3,1500,'4-10-2017','Wednesday');
insert into sales values(3,2000,'5-10-2017','Thursday');
insert into sales values(1,3500,'7-10-2017','Saturday');
insert into sales values(3,4000,'3-10-2017','Tuesday');

```

```

create function funct(a1 in number,a3 in number)
return number
as

```

```

q number;
z number;
r number;
x varchar(10);
y varchar(10);
begin
select pname into x from product where pid=a1;
select pgrade into y from product where pid=a1;

dbms_output.put_line('name of product :'||x);
dbms_output.put_line('grade of product :'||y);

select sum(samount) into q from sales where pid=a1;
dbms_output.put_line('sum of sales of '||a1||' is'||q);
if(q>a3) then
    update product set pgrade='a' where pid=a1;
end if;
select count(pid) into z from sales where pid=a1;
if(z<1) then
    r:=0;
else
    r:=1;
end if;
return(r);
end;

```

```

set serveroutput on;
declare
d1 number;
e1 number;
f1 number;
pnull exception;
begin
d1:=&d1;
e1:=&e1;
f1:=funct(d1,e1);
if f1=0 then
    raise pnull;
end if;
exception
when pnull then

```

```

    dbms_output.put_line('no sales corresponding to this pid ');
when no_data_found then
    dbms_output.put_line('no data found:');
end;

```

OUTPUT

```
select * from sales;
```

	PID	SAMOUNT	SDATE	SDAY
1	1	1000	02-10-17	Monday
2	3	1500	04-10-17	Wednesday
3	3	2000	05-10-17	Thursday
4	1	3500	07-10-17	Saturday
5	3	4000	03-10-17	Tuesday

```
select * from product;
```

	PID	PNAME	PGRADE
1	1	HDD	a
2	2	GoPro	b
3	3	laptop	b
4	4	mobile	b
5	5	DVD	c

anonymous block completed

Enter the value of d1 : 3

Enter the value of e1 : 5000

name of product :laptop

grade of product :b

sum of sales of 3 is 7500

table updated.

select * from product;

	PID	PNAME	PGRADE
1	1	HDD	a
2	2	GoPro	b
3	3	laptop	a
4	4	mobile	b
5	5	DVD	c

anonymous block completed

Enter the value of d1 : 2

Enter the value of e1 : 1000

name of product :GoPro

grade of product :b

sum of sales of 2 is

no sales corresponding to this pid

EXPERIMENT 12

TRIGGER

AIM

To familiarize triggers.

DESCRIPTION

- The Oracle engine allows the definition of procedures that are implicitly executed when an insert, update or delete is issued against a table from SQL*Plus or through an application. These procedures are called database triggers.
- A trigger has 3 basic parts:
 - ◆ A triggering event or statement – It is a SQL statement that causes a trigger to be fired. It can be INSERT, UPDATE or DELETE statement for a specific table.
 - ◆ A trigger restriction – Its function is to conditionally control the execution of a trigger.
 - ◆ A trigger action – It is the PL/SQL code to be executed when a triggering statement is encountered and any trigger restriction evaluates to TRUE.

- Types of Triggers:

- ◆ Row Triggers – A row trigger is fired each time a row in the table is affected by the triggering statement.
- ◆ Statement Triggers – Statement level triggers fire once for each triggering event.
- ◆ Before Triggers – Before triggers execute the trigger action before the triggering statement.
- ◆ After Triggers – After trigger execute the trigger action after the triggering statement.

- Syntax:

CREATE OR REPLACE TRIGGER [Schema.]<TriggerName>

{ BEFORE, AFTER }

{ DELETE, INSERT, UPDATE [OF Column,...] }

ON [Schema.]<TableName>

[REFERENCING{ OLD AS old, NEW AS new }]

[FOR EACH ROW [WHEN Condition]]

DECLARE

<VariableDeclarations>;

<ConstantDeclarations>;

BEGIN

<PL/SQL subprogrambody>;

EXCEPTION

<Exception PL/SQL block>;

END;

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name – Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} – This specifies the DML operation.
- [OF col_name] – This specifies the column name that will be updated.
- [ON table_name] – This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] – This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] – This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

QUESTIONS

1)create a trigger which shows the salary difference of a particular employee whenever salary is getting updated

Worker(id,name,salary)

PROGRAM

create table worker(id number(5),name varchar(10),sal number(10));

insert into worker values(1,'riya',25000);

insert into worker values(2,'febi',30000);

insert into worker values(3,'varsha',20000);

```
insert into worker values(4,'anjali',50000);
insert into worker values(5,'athira',35000);
```

```
create trigger t after update of sal on worker for each row
```

```
declare
```

```
diff number(10);
```

```
begin
```

```
diff:=new.sal-old.sal;
```

```
dbms_output.put_line('Difference in salary is '||diff);
```

```
end;
```

```
set serveroutput on;
```

```
declare
```

```
n number(3):=&workerid;
```

```
nwsal number(5):=&nwsal;
```

```
begin
```

```
update worker set sal=nwsal where id=n;
```

```
end;
```

OUTPUT

```
select * from worker;
```

	ID	NAME	SAL
1	1	riya	25000
2	2	febi	30000
3	3	varsha	20000
4	4	anjali	50000
5	5	athira	35000

TRIGGER T compiled

anonymous block completed

Enter the worker id =1

Enter the new salary = 28000

anonymous block completed

Difference in salary is 3000

select * from worker;

		NAME	SAL
1		1 riya	28000
2		2 febi	30000
3		3 varsha	20000
4		4 anjali	50000
5		5 athira	35000

2) create a table theater (movie id, movie name,language,review_ratings).Whenever rating goes below 5 , the movies has to be removed from theater table and add to table outdated movies with attributes movie id and movie name

PROGRAM

```
create table theatre(mov_id number(10),mov_name varchar(20),lang varchar(20),revw number(10));
```

```
insert into theatre values(101,'Junglebook','English',6);
```

```
insert into theatre values(501,'Parava','Malayalam',8);
```

```
insert into theatre values(601,'OSO','Hindi',9);
```

```
insert into theatre values(701,'Avengers','English',9);
```

```
insert into theatre values(801,'Hobbit','English',7);
```

```
insert into theatre values(901,'Don','Hindi',6);
```

```
create table outdat(mov_id number(10),mov_name varchar(20));
```

```
create trigger trii
```

```
after delete on theatre
```

```
for each row
```

```
begin
```

```
insert into outdat values(:old.mov_id,:old.mov_name);
```

```
end;
```

```
set serveroutput on;
```

```
declare
```

```

a1 number:=&movie_id;
a2 number:=&new_rating;
begin
  if(a2<5) then
    delete from theatre where mov_id=a1;
  end if;
end;

```

OUTPUT

```
select * from theatre;
```

	MOV_ID	MOV_NAME	LANG	REVV
1	101	Junglebook	English	6
2	501	Parava	Malayalam	8
3	601	OSO	Hindi	9
4	701	Avengers	English	9
5	801	Hobbit	English	7
6	901	Don	Hindi	6

anonymous block completed

Enter the movie_id : 801

Enter the new_rating : 4

```
select * from theatre;
```

	MOV_ID	MOV_NAME	LANG	REVV
1	101	Junglebook	English	6
2	501	Parava	Malayalam	8
3	601	OSO	Hindi	9
4	701	Avengers	English	9
5	901	Don	Hindi	6

```
select * from outdat;
```

	MOV_ID	MOV_NAME
1	801	Hobbit

EXPERIMENT 13

CURSORS

AIM

To familiarize with Cursors

DESCRIPTION

- The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL's operations and is called a cursor.
- The data that is stored in the cursor is called the Active Data Set.
- Cursors are classified depending on the circumstances under which they are opened.
 - Implicit Cursor
 - Explicit Cursor
- Implicit Cursor - If the Oracle engine opened a cursor for its internal processing it is known as an Implicit Cursor.

Implicit Cursor Attributes:

- %ISOPEN
 - %FOUND
 - %NOTFOUND
 - %ROWCOUNT
 - Explicit Cursor – When individual records in a table have to be processed inside a PL/SQL code block a cursor is used. This cursor will be declared and mapped to an SQL query in the Declare Section of the PL/SQL block and used within its Executable Section. A cursor thus created and used is known as Explicit Cursor.
- Explicit Cursor Management: The steps involved in using an explicit cursor and manipulating data in its active data set are

- Declare a cursor mapped to a SQL select statement that retrieves data for processing.
- Open the cursor
- Fetch data from the cursor one row at a time into memory variables.
- Process the data held in the memory variables as required using a loop.
- Exit from the loop after processing is complete.
- Close the cursor.

Syntax:

CURSOR CursorName IS SELECT statement;

OPEN CursorName;

FETCH CursorName INTO Variable1, Variable2,...;

CLOSE CursorName;

Explicit Cursor Attributes

- %ISOPEN
- %FOUND
- %NOTFOUND
- %ROWCOUNT

➤ Cursor FOR Loops

Syntax:

FOR memory variable IN CursorName

A cursor for loop automatically does the following:

- Implicitly declares its loop index as a %rowtype record.
- Opens a cursor.
- Fetches a row from the cursor for each loop iteration.
- Closes the cursor when all rows have been processed.

Parameterized cursor

PL/SQL Parameterized cursor pass the parameters into a cursor and use them in to query. It define data type of parameter and not need to define its length.

Parameterized cursor passes values to the cursor; and cannot pass values out of the cursor through parameters. It is also known as static cursors that can passed parameter value when cursor are opened.

Syntax:

CURSOR CursorName(arg1,arg2,...argn) IS SELECT statement;

OPEN CursorName;

FETCH CursorName INTO Variable1, Variable2,...;

CLOSE CursorName;

QUESTIONS

1) Consider the table Customer (accout no, customer name,balance amount,date of join).

Implement a PL/SQL block to insert those customers who have current balance greater than 1 Lakh and date of join before 1 january 2010 into the table premium customer who doesnt meet above criteria are to be inserted into table nonpremium customer .

PROGRAM

```
create table customers(acc_no number(16),c_name varchar(15),bal_amt number(10),DOJ date);
create table premium_customers(acc_no number(16),c_name varchar(15),bal_amt number(10),DOF date);
create table nonpremium_customers(acc_no number(16),c_name varchar(15),bal_amt number(10),DOF
date);

insert into customers values(101,'anu',150000,'12-12-1998');
insert into customers values(102,'anjana',200000,'19-08-1997');
insert into customers values(208,'achu',50000,'8-09-2003');
insert into customers values(305,'gopika',95000,'10-10-2010');
insert into customers values(409,'irene',25000,'4-03-2011');
insert into customers values(111,'dany',100000,'11-05-2010');

set serveroutput on;
declare
    cursor s is select * from customers;
begin
    for r in s
    loop
        if r.bal_amt>100000 and r.doj<'01-01-2010' then
            insert into premium_customers values(r.acc_no,r.c_name,r.bal_amt,r.doj);
        else
            insert into nonpremium_customers values(r.acc_no,r.c_name,r.bal_amt,r.doj);
        end if;
    end loop;
end;
```

OUTPUT

select * from customers;

	ACC_NO	C_NAME	BAL_AMT	DOJ	
1	101	anu	150000	12-12-98	
2	102	anjana	200000	19-08-97	
3	208	achu	50000	08-09-03	
4	305	gopika	95000	10-10-10	
5	409	irene	25000	04-03-11	
6	111	dany	100000	11-05-10	

select * from premium_customers;

	ACC_NO	C_NAME	BAL_AMT	DOF	
1	101	anu	150000	12-12-98	
2	102	anjana	200000	19-08-97	

select * from nonpremium_customers;

	ACC_NO	C_NAME	BAL...	DOF	
1	208	achu	50000	08-09-03	
2	305	gopika	95000	10-10-10	
3	409	irene	25000	04-03-11	
4	111	dany	100000	11-05-10	

2) Consider the table Account(Customer name,account number,date_last transaction,amount).
Implement a PL/SQL block to perform the following action on the table .

Calculate the interest of each person if it satisfies the condition

a)if the last transaction is not on the current month insert the records into inactive customer

b)otherwise check the balance amount and display the interest amount

i)if the balance amount is less than 50000 interest rate is 5% of the amount

ii)if it is between 250000 and 5 Lakhs interest rate is 10%

iii)if the amount is greater than 5 lakh interest rate is 15%

PROGRAM

```
set serveroutput on;
```

```
create table accdetails(accno number(10),cname varchar(20),lastdate date,amount number(7));
```

```
insert into accdetails values(101,'anu','08-11-2017',50000);
```

```
insert into accdetails values(102,'anagha','10-10-2017',100000);
```

```
insert into accdetails values(103,'elizabeth','17-11-2017',25000);
```

```
insert into accdetails values(104,'george','06-10-2017',300000);
```

```
insert into accdetails values(105,'albyn','15-11-2017',650000);
```

```
create table inactive_customer(accno number(10),cname varchar(20));
```

```
declare
```

```
cursor c1 is select * from accdetails;
```

```
a accdetails %rowtype;
```

```
n number;
```

```
i number;
```

```
begin
```

```
open c1;
```

```
loop
```

```
    fetch c1 into a.accno,a.cname,a.lastdate,a.amount;
```

```
    exit when c1 %notfound;
```

```
    n:=months_between(sysdate,a.lastdate);
```

```
    if n>=1 then
```

```
        insert into inactive_customer values(a.accno,a.cname);
```

```
    else
```

```
        if a.amount<250000 then
```

```
            i:=a.amount*0.05;
```

```
        dbms_output.put_line('interest of ||a.cname|| ' is :'||i);
```

```
        elsif a.amount>250000 and a.amount<500000 then
```

```
            i:=a.amount*0.1;
```

```
        dbms_output.put_line('interest of ||a.cname|| ' is :'||i);
```

```
        elsif a.amount>500000 then
```

```
            i:=a.amount*0.15;
```

```
        dbms_output.put_line('interest of ||a.cname|| ' is :'||i);
```

```

else
dbms_output.put_line('error');
end if;
end if;
end loop;
end;

```

OUTPUT

```
select * from accdetails;
```

	ACCNO	CNAME	LASTDATE	AMOUNT
1	101	anu	08-11-17	50000
2	102	anagha	10-10-17	100000
3	103	elizabeth	17-11-17	25000
4	104	george	06-10-17	300000
5	105	albyn	15-11-17	650000

anonymous block completed

interest of anu is :2500

interest of elizabeth is :1250

interest of albyn is :97500

```
select * from inactive_customer;
```

	ACCNO	CNAME
1	102	anagha
2	104	george

EXPERIMENT 14

PACKAGES

AIM

To familiarize with packages.

DESCRIPTION

A package is a schema object that groups logically related PL/SQL types, variables, and subprograms. Packages usually have two parts, a specification (spec) and a body.

The specification is the interface to the package. It declares the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. The body defines the queries for the cursors and the code for the subprograms.

The package body contains the implementation of every cursor and subprogram declared in the package spec. Subprograms defined in a package body are accessible outside the package only if their specs also appear in the package spec. If a subprogram spec is not included in the package spec, that subprogram can only be called by other subprograms in the same package. A package body must be in the same schema as the package spec.

The following is contained in a PL/SQL package

1. Get and Set methods for the package variables
2. Cursor declarations with the text of SQL queries.
3. Declarations for exceptions.
4. Declarations for procedures and functions that call each other.
5. Declarations for overloaded procedures and functions
6. Variables that you want to remain available between procedure calls in the same session

Advantages in using packages

Modularity

Easier Application

Design Information

Hiding Added

Functionality Better

Performance

```
CREATE [ OR REPLACE ] PACKAGE [ schema. ]
```

```
package [ invoker_rights_clause ]
```

```
{ IS | AS } [ item_list_1 ] END [ package_name ] ;
```

Where

Schema- Specify the schema to contain the package. If you omit *schema*, then the database creates the package in your own schema.

Item list 1 : Declares package elements.

invoker_rights_clause : Specifies the AUTHID property of the member functions and procedures of the object type. The AUTHID clause determines whether all the packaged subprograms execute with the privileges of their definer (the default) or invoker, and whether their unqualified references to schema objects are resolved in the schema of the definer or invoker.

```
CREATE [ OR REPLACE ] PACKAGE BODY
```

```
[ schema. ] package
```

```
{ IS | AS } [ declare_section ] { body | END package_name } ;
```

Referencing Package Contents

To reference the types, items, subprograms, and call specs declared within a package spec, use dot notation:

```
package_name.type_name
```

```
package_name.item_name
```

```
package_name.subprogram_name
```

```
package_name.call_spec_name
```

QUESTION

1) create the table supplier(supplier id,suppliername,commission and city). create a package comprising of procedure and function .

1. The procedure will accept a supplier id and display the corresponding supplier name,

2. The function will accept the supplier id and calculate the annual commission and finally return the amount

Algorithm

1. Create a package with declaration for a procedure and a package.
2. Create the package body including the definitions for procedure and function

PROGRAM

```
create table supplier(suid number(3),suname varchar(10),sucom number(10),sucity varchar(10));
```

```
insert into supplier values(1,'adam',1000,'pune');
```

```
insert into supplier
```

```
values(2,'john',2000,'mumbai'); insert into
```

```
supplier values(3,'adhya',1500,'kochi'); insert
```

```
into supplier values(4,'farhaan',3000,'kochi');
```

```
insert into supplier
```

```
values(5,'merina',2500,'bangalore');
```

```
select * from supplier;
```

```
create package pack_age as
```

```
procedure alpha(a1 in number,c1 out
```

```
varchar); function beta(a1 in number)
```

```
return number;
```

```
end pack_age;
```

```
create package body pack_age as
```

```
procedure alpha(a1 in number,c1 out
```

```
varchar) is
```

begin

select suname into c1 from supplier where

suid=a1; end alpha;

function beta(a1 in

number) return number

is

s1

number:=0;

s2 number;

begin

select sucom into s2 from supplier where

suid=a1; s1:=s2*12;

return(s1)

; end;

end pack_age;

set serveroutput

on; declare

a1

number:=&a1;

c1 varchar(10);

bb1 number:=0;

begin

```

pack_age.alpha(a1,c1);

dbms_output.put_line('supplier correspodng to '||a1 ||' is '||c1);

end;

```

```

set serveroutput
on; declare

```

```

b1
number:=&b1; c1
varchar(10); bb1
number:=0;

begin

    bb1 :=pack_age.beta(b1);

    dbms_output.put_line('annual commission for '||b1||' is '||bb1);

end;

```

OUTPUT

```

create table supplier(suid number(3),suname varchar(10),sucom number(10),sacity
varchar(10)); insert into supplier values(1,'adam',1000,'pune');

insert into supplier
values(2,'john',2000,'mumbai'); insert into
supplier values(3,'adhya',1500,'kochi'); insert
into supplier values(4,'farhaan',3000,'kochi');

insert into supplier

```

```
values(5,'merina',2500,'banglore');
```

```
select * from supplier;
```

	SUID	SUNAME	SUCOM	SUCITY
1	1	adam	1000	pune
2	2	john	2000	mumbai
3	3	adhya	1500	kochi
4	4	farhaan	3000	kochi
5	5	merina	2500	banglore

on Executing package and package body(do it as

step 1) PACKAGE PACK_AGE compiled

PACKAGE BODY PACK_AGE

compiled On Executing plsql

containing procedure input: Enter the

value of a1 : 3

output: supplier correspodng to 3 is

adhya On Executing plsql containing

function

EXPERIMENT 15

EXCEPTIONS

AIM

To familiarize exceptions in DBMS

DESCRIPTION

An Exception is an error situation, which arises during program execution. When an error occurs exception is raised, normal execution is stopped and control transfers to exception handling part.

Exception handlers are routines written to handle the exception. The exceptions can be internally defined (system-defined or pre-defined) or User-defined exception.

Syntax:

```
EXCEPTION
    WHEN <ExceptionName> THEN
    <User Defined Action To Be Carried Out>
```

Predefined exception:

It is raised automatically whenever there is a violation of Oracle coding rules. Predefined exceptions are those like ZERO_DIVIDE, which is raised automatically when we try to divide a number by zero. Other built-in exceptions are given below. You can handle unexpected Oracle errors using OTHERS handler. It can handle all raised exceptions that are not handled by any other handler. It must always be written as the last handler in exception block.

Exception	Raised when....
DUP_VAL_ON_INDEX	When you try to insert a duplicate value into a unique column.
INVALID_CURSOR	It occurs when we try accessing an invalid cursor.
INVALID_NUMBER	On usage of something other than number in place of number value.
LOGIN_DENIED	At the time when user login is denied.
TOO_MANY_ROWS	When a select query returns more than one row and the destination variable can take only single value.
VALUE_ERROR	When an arithmetic, value conversion, truncation, or constraint error occurs.
CURSOR_ALREADY_OPEN	Raised when we try to open an already open cursor.

Predefined exception handlers are declared globally in package STANDARD. Hence we need not have to define them rather just use them. The biggest advantage of exception handling is it

improves readability and reliability of the code. Errors from many statements of code can be handles with a single handler. Instead of checking for an error at every point we can just add an exception handler and if any exception is raised it is handled by that. For checking errors at a specific spot it is always better to have those statements in a separate begin – end block.

User-defined Exceptions :

The technique that is used is to bind a numbered exception handler to a name using Pragma Exception_init (). This binding of a numbered exception handler, to a name (i.e. a String), is done in the Declare section of a PL/SQL block.

The Pragma action word is a call to a pre-compiler, which immediately binds the numbered exception handler to a name when encountered.

The function Exception_init() takes two parameters the first is the user defined exception name the second is the Oracle engine's exception number. These lines will be included in the Declare section of the PL/SQL block.

The user defined exception name must be the statement that immediately precedes the Pragma Exception_init() statement.

Syntax:

```
DECLARE
    < ExceptionName > EXCEPTION ;
    PRAGMA EXCEPTION_INIT (< ExceptionName >, <ErrorCodeNo>);
BEGIN
```

Using this technique it is possible to bind appropriate numbered exception handlers to names and use these names in the Exception section of a PL/SQL block. When this is done the default exception handling code of the exception handler is overridden and the user-defined exception handling code is executed .

Syntax:

```
DECLARE
    < ExceptionName > EXCEPTION ;
    PRAGMA EXCEPTION_INIT (< ExceptionName >, <ErrorCodeNo>);
BEGIN
    . . . .
EXCEPTION
    WHEN < ExceptionName > THEN
        < Action >
```

```
END;
```

User Defined Exception Handling :

To trap business rules being violated the technique of raising user-defined exceptions and then handling them, is used.

User-defined error conditions must be declared in the declarative part of any PL/SQL block. In the executable part, a check for the condition that needs special attention is made. If that condition exists, the call to the user-defined exception is made using a RAISE statement. The exception once raised is then handled in the Exception handling section of the PL/SQL code block.

Syntax:

```
DECLARE
    < ExceptionName > EXCEPTION ;
BEGIN
    <SQL Sentence >;
    IF < Condition > THEN
        RAISE <ExceptionName>;
    END IF;
EXCEPTION
    WHEN <ExceptionName> THEN {User Defined Action To Be Taken};
END;
```

Questions

1. Write a PL/SQL block to raise an exception when a division by zero occurs.
2. Create table named emp have column like id with notnull constraint,name etc. Raise an exception “Not null values not allowed “ when the user tries to enter a null value for id.
3. Write a PL/SQL block to raise an exception “No transactions today”,when the sysday is Sunday.

Answers

1.

```
DECLARE
    N number;
BEGIN
    N:=10/0;
```

```
EXCEPTION WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('divide by zero error occurs..');
END;
/
```

2.

```
DECLARE
    e_MissingNull EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_MissingNull, -1400);
BEGIN
    INSERT INTO emp(id) VALUES (NULL);
EXCEPTION
    WHEN e_MissingNull then
        DBMS_OUTPUT.put_line('ORA-1400 occurred');
END;
/
```

3.

```
DECLARE
    ex EXCEPTION;
BEGIN
    IF TO_CHAR(SYSDATE,'DY')=='SUN' THEN
        RAISE ex;
    END IF;
EXCEPTION
    WHEN ex then
        DBMS_OUTPUT.put_line('No Transcations Today');
END;
/
```

EXPERIMENT 16

TCL COMMANDS

AIM

To familiarize with TCL commands

DESCRIPTION

Syntax:

SAVEPOINT: SAVEPOINT <SAVE POINT NAME>;

ROLLBACK: ROLL BACK <SAVE POINT NAME>;

COMMIT: Commit;

These statements provide control over use of transactions:

START TRANSACTION or BEGIN start a new transaction.

COMMIT commits the current transaction, making its changes permanent.

ROLLBACK rolls back the current transaction, canceling its changes.

SET autocommit disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.

```
create database tcl_dcl;
use tcl_dcl;
create table students (id int primary key auto_increment,name char(50),dob date,age int);
start transaction;
insert into students (name,dob,age) values ('naveen',"1996-08-20",20);
insert into students (name,dob,age) values ('sathya',"1996-06-10",21);
SELECT * FROM students;
```

```
+---+-----+-----+-----+
| id | name | dob | age|
```

```
+---+-----+-----+-----+
| 1 | naveen | 1997-08-20 | 20 |
```

```
| 2 | sathya | 1996-09-10 | 21 |
+---+-----+-----+-----+
```

ROLLBACK;

SELECT * FROM students;

```
+---+-----+-----+-----+
| id | name | dob | age|
```

```
+---+-----+-----+-----+
```

1	naveen	1997-08-20	20
2	sathya	1996-09-10	21

-----+

*No change happened since **By default, MySQL runs with autocommit mode enabled.** This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.*

So to do rollback we have to disable autocommit.

```
SET autocommit=0 ;
savepoint s1;
update students SET dob="1997-09-10" where id=2;
SELECT * FROM students;
```

-----+

id	name	dob	age
----	------	-----	-----

-----+

1	naveen	1997-08-20	20
2	sathya	1997-09-10	21

-----+

```
ROLLBACK to s1;
SELECT * FROM students;
```

-----+

id	name	dob	age
----	------	-----	-----

-----+

1	naveen	1997-08-20	20
2	sathya	1996-09-10	21

-----+

```
update students SET dob="1997-09-10" and age=20 where id=2;
SELECT * FROM students;
```

-----+

id	name	dob	age
----	------	-----	-----

-----+

1	naveen	1997-08-20	20
2	sathya	1997-09-10	20

-----+

```
commit;
insert into students (name,dob,age) values ('kathir',"1995-06-15",22);
SELECT * FROM students;
```

-----+

id	name	dob	age
----	------	-----	-----

-----+

1	naveen	1997-08-20	20
2	sathya	1997-09-10	20

```
| 3 | kathir | 1995-06-15 | 22 |
+---+-----+-----+-----+
ROLLBACK;
+---+-----+-----+-----+
| id | name | dob | age|
+---+-----+-----+-----+
| 1 | naveen | 1997-08-20 | 20 |
| 2 | sathya | 1997-09-10 | 20 |
+---+-----+-----+-----+
```

EXPERIMENT 17

DCL COMMANDS

AIM

To familiarize with DCL commands

DESCRIPTION

The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated.

The privilege commands are namely Grant and Revoke. The various privileges that can be granted or revoked are: Select Insert Delete Update References ExecuteAll

1. GRANT COMMAND

It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

You can grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, GRANT OPTION or ALL.

Syntax

GRANT privileges ON object TO user;

Grant < database_priv [database_priv.....] > to <user_name> identified by <password> [,<password.....>];

Grant <object_priv> | All on <object> to <user | public> [With Grant Option];

2. REVOKE COMMAND

Using this command , the DBA can revoke the granted database privileges from the user.

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

Syntax

REVOKE privileges ON object FROM user;

Revoke <database_priv> from <user [, user] >;

Revoke <object_priv> on <object> from < user | public >;

<database_priv> -- Specifies the system level privileges to be granted to the users or

roles. This includes create / alter / delete any object of the system.

<object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.

<all> -- Indicates all the privileges.

[With Grant Option] – Allows the recipient user to give further grants on the objects.

The privileges can be granted to different users by specifying their names or to all users by using the “Public” option.

DCL Commamnds

1.Check the current user.

```
Select user();
```

```
+-----+
| user() |
+-----+
| root@localhost |
+-----+
```

2.create a new database and use the new DB

```
create database class;
```

```
use class;
```

3.Create a table in class DB.

```
create table stud (id int primary key auto_increment,name char(20),age int);
```

4.Create a new user.

```
create user 'user1'@'localhost' identified by 'user123';
```

5.Check what are the permissions allowed for new user.(by default new user don't have any privileges)

```
show grants for 'user1'@'localhost';
```

```
+-----+
| Grants for user1@localhost@% |
+-----+
| GRANT USAGE ON *.* TO `user1@localhost` @`%` |
+-----+
```

9.Grant select permissions on all tables in the database class to new user user1.

```
grant select on *.* to 'user1'@'localhost';
```

//(first * indicates select access to all databases in mysql and second * indicates select access to all tables in that databases)

10. Grant permission on a specific table. (insert)

grant insert on class.stud to 'user1'@'localhost';

11. Grant more specific permission to a particular column.

grant update(name) on class.stud to 'user1'@'localhost';

12. Show grants for new user user1.

show grants for 'user1'@'localhost';

```
+-----+
| Grants for user1 @localhost@%          |
+-----+
| GRANT SELECT ON *.* TO `user1`@`%`      |
| GRANT INSERT, UPDATE (`name`) ON `class`.`stud` TO `user1`@`%` |
+-----+
```

13. Now move to new user and check whether all granted permissions are obtained or not.

system mysql -u user1 -p;

Enter password: ***** (pw is user123)

13a)

select user();

```
+-----+
| user()          |
+-----+
| user1 @localhost |
+-----+
1 row in set (0.00 sec)
```

13b) shift to the database

Use class;

13c) insert into stud values(1,"anu",18);

13d) update stud set name="binu" where id=1;

13e) Try to do one operation for which access not granted.

update stud set age=80 where id=1;

ERROR 1143 (42000): UPDATE command denied to user 'user3'@'localhost' for column 'age' in table 'stud'

Revoke: For performing revoke operations, switch back to the super user.

system mysql -u root -p;

Enter password: ***** (Enter mysql password)

```
mysql> select user();
```

```
+-----+
| user() |
+-----+
| root@localhost |
+-----+
```

14.Revoke Update ,insert permission.

revoke update(name) on class.stud from 'user1'@'localhost';

revoke insert on class.stud from 'user1'@'localhost';

15.Show the permissions on user 1.

```
mysql> show grants for 'user1'@'localhost';
```

```
+-----+
| Grants for user1 @localhost |
+-----+
| GRANT SELECT ON *.* TO `user1`@`localhost` |
+-----+
```

15.show that all the permissions are revoked back.

```
mysql> system mysql -u 'user1' -p;
```

Enter password: ***** (password is user123)

15a)select database

Use class;

15b) insert denied

insert into stud values(2,"cinu",19);

ERROR 1142 (42000): INSERT command denied to user 'user3'@'localhost' for table 'stud'

16.CREATING SUPER USER

(Give all the permission that are there for root to the new user user1)

16a)move to root user

```
system mysql -u root -p;
```

Enter password: ***** (Enter mysql password)

16b) Display all the privileges of superuser.

```
show grants for 'root'@'localhost';
```

16c) Grant all the privileges of super user to user1('with grant option' gives

new user also the permission to grant privileges to new users)
grant all privileges on *.* to 'user1'@'localhost' with grant option;

16 d)view all the privileges of user 1
show grants for 'user1'@'localhost';

17.Change the password of localhost.
alter user 'user1'@'localhost' identified by '1234';

EXPERIMENT 18

MYSQL WORKBENCH

AIM

To Familiarize MYSQL work bench for automating the ER diagram creation and performing bulk import

DESCRIPTION

Database initialization - Data insert, Data import to a database

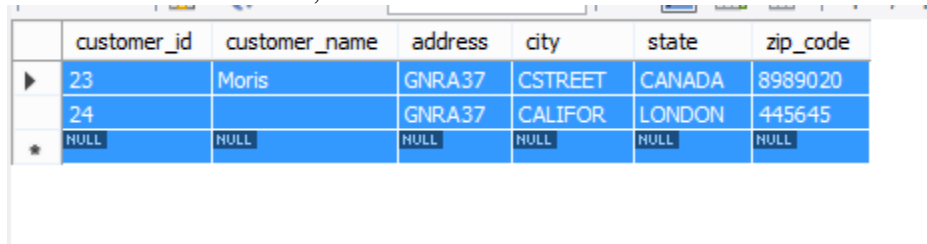
Steps

Create a SQL table called customers with columns customer_id, customer_name, address, city, state, zip_code

Code:

```
CREATE TABLE customers
( customer_id int NOT NULL,
  customer_name char(50) NOT NULL,
  address char(50),
  city char(50),
  state char(25),
  zip_code char(10),
  CONSTRAINT customers_pk PRIMARY KEY (customer_id)
);
```

```
insert into customers values (23,'Moris','GNRA37','CSTREET','CANADA','8989020');
insert into customers values (24,',','GNRA37','CALIFOR','LONDON','445645');
select * from customers;
```



	customer_id	customer_name	address	city	state	zip_code
▶	23	Moris	GNRA37	CSTREET	CANADA	8989020
	24	,	GNRA37	CALIFOR	LONDON	445645
★	NULL	NULL	NULL	NULL	NULL	NULL

Step2

CREATE AN EXCEL FILE AND SAVE AS CSV FILE IN THE SAME FORMAT AS TABLE EMP

Open a excel file --□ save as filename .csv(comma delimited)

ADD VALUES INTO THE customers.CSV FILE.

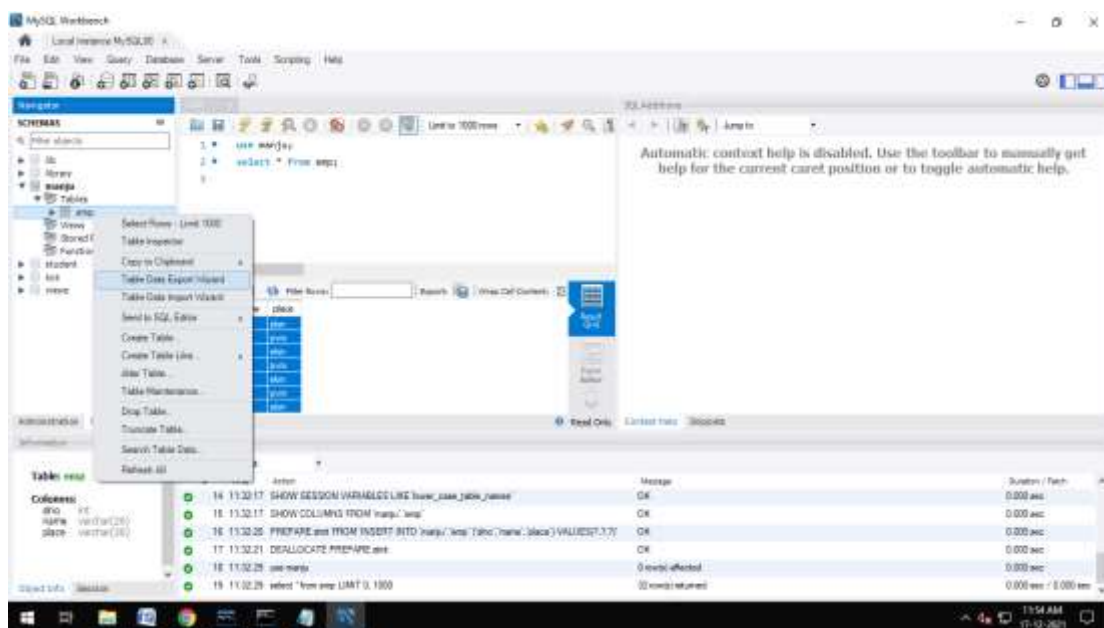
A	B	C	D	E	F	G
customer_id	customer_name	address	city	state	zip_code	
100	SHRON	DOORNO-21	HAUSTON	CANADA	384848584	
101	REEBA	GNARA37/21	CALIFORN	AUSTRIA	4565	
102	SAMUEL	PHIL32	BANGLOR	INDIA	667	
103	JOHAN	ABS231	MUMBAI	INDIA	34344	
104	GEN	SDMF	ASKDK	CANADA	4556	
105	GENIA	CDERG	CHICAGO	AMERICA	7788	
106	KEITH	RRT	HYDRABA	INDIA	877	
107	JONATHAN	AWE43	KDKF	LONDON	453	
108	JIM	RGTY	FG	CANADA	45456567	
109	JERRY	ERR45	TGGG	AUSTRIA	45456567	
110	AMEL	HUIO00	GGGG	INDIA	565464565	
111	ADHIL	SEDD	FFFF	LONDON	456646	
112	BIBIYA	ASW34	HYJ	LONDON	2334	
113	SCARIA	AWSDDE	GGGG	CANADA	444445	
114	SILVER	NANSJS	MELBOUR	AUSTRIA	32333	
115	WOODS	DERND	FGGFB	INDIA	2333	
116	LOPEZ	ERDNSIOS	BGG	LONDON	6777	
117	JENNIFER	DR	BN	LONDON	67877	
118	PRIYA	F	YHGB	CANADA	128977	
119	MEEHA	GNARA37	LJHG	AUSTRIA	190077	
120	KIRAN	3FGG	JKKKL	INDIA	251177	
121	MARIA	BH	WEED	LONDON	312277	

Insert values(all fields should match with table created)

Step3

BULK IMPORT STEPS IN MYSQL WORKBENCH

SELECT SCHEMA -> SELECT DATABASE --> SELECT TABLE customers



THEN- SELECT CSV FILE –NEXT-NEXT -----FINISH

2. ER Diagram

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

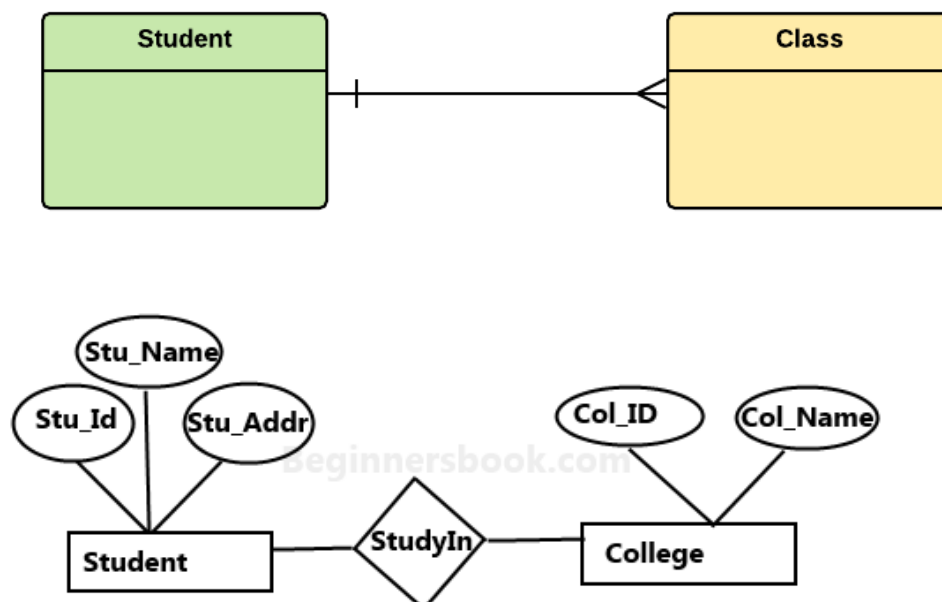
ENTITY

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. example

- **Person:** Employee, Student, Patient
- **Place:** Store, Building

Entity set:

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.



Sample E-R Diagram

Relationship

Relationship is nothing but an association among two or more entities. E.g., student studying in the College

Attributes

It is a single-valued property of either an entity-type or a relationship-type.

Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships
- Many-to-Many Relationships

shapes and their meaning in an E-R Diagram.

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

Question:

Design a database schema for an application with ER diagram from a problem description

To draw the ER diagram of the Application Development problem assigned to each group

Steps

Create all the tables and relations

Select database - ☐ select Reverse Engineering-- ☐ it opens connection option- ☐ next- ☐

Enter password - ☐ click next

Select schema you want to include - ☐ select the schema- ☐ click next- ☐ execute- ☐ next-

☐ finish

It will display the EER (enhanced ER) diagram

EXPERIMENT 19

NO SQL DATABASE

AIM

To familiarize any one NOSQL database.

DESCRIPTION

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

NoSQL database features

- Flexible schemas
- Horizontal scaling
- Fast queries due to the data model
- Ease of use for developers

MongoDB, CouchDB, CouchBase, Cassandra, HBase, Redis, Riak, Neo4J are the popular NoSQL databases examples.

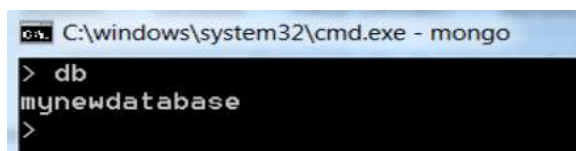
CRUD operations In MongoDB

Creating a Database: Syntax

use database_name

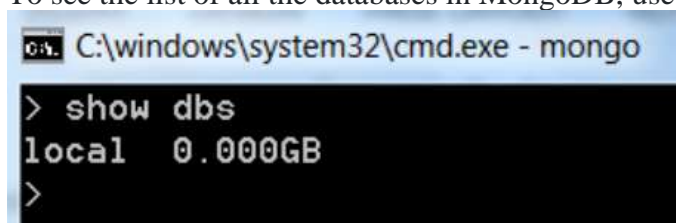
This will create a new database with the name **database_name** if there is no database already present with the same name. If a database already exists with the mentioned name, then it just connects to that database.

To check the current connected database –type db



```
C:\windows\system32\cmd.exe - mongo
> use mynewdatabase
>
```

To see the list of all the databases in MongoDB, use command show dbs

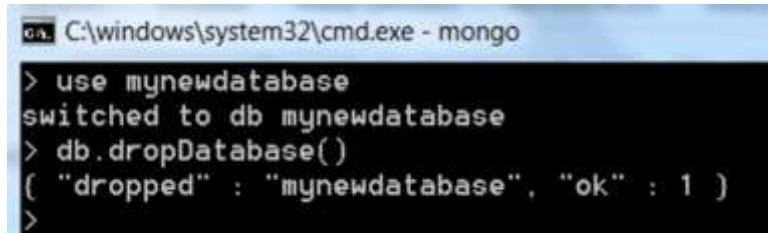


```
C:\windows\system32\cmd.exe - mongo
> show dbs
local  0.000GB
>
```

Please note that the newly created database **mynewdatabase** has not been listed after running the above command. This is because, no records have been inserted into that database yet.

Drop a Database

Before deleting the database, connect to the required database which is to be deleted. And type `db.dropDatabase()`



```
C:\windows\system32\cmd.exe - mongo
> use mynewdatabase
switched to db mynewdatabase
> db.dropDatabase()
{ "dropped" : "mynewdatabase", "ok" : 1 }
>
```

Creating a Collection

A collection is **the equivalent of an RDBMS table**.we can explicitly create a collection using the `createCollection()` command.The syntax of `createCollection` method is:

`db.createCollection(name, options)`

name will be the name of the collection and **options** is a document which can be used to specify configurations for the collection.

options parameter is optional, and you can simply create a collection by just providing a name for the collection. But if you want to configure your collection, you can use various options available to do so.

Field	Type	Description
Capped	boolean	(Optional) To create a capped collection, where in we specify the maximum size or document counts to prevent it from growing beyond a set maximum value.
Size	number	(Optional) To specify the maximum size in bytes for a capped collection. If a collection is capped and reaches its maximum size limit, MongoDB then removes older documents from it to make space for new.
Max	number	(Optional) This can be used to specify the maximum number of documents allowed in a capped collection.

validator	document	(Optional) Validates any document inserted or updated against provided validation rules.
validationLevel	string	<p>(Optional) This is used to define how strictly validation rules must be applied to existing documents during an update.</p> <p>Available values are :</p> <p>off No validation for inserts or updates.</p> <p>strict This is the default value for this option. This instructs MongoDB to apply validation rules to all inserts and updates.</p> <p>moderate This is used when we want to apply validation rules to inserts and updates on only the existing valid documents and not on the existing invalid documents.</p>
validationAction	string	(Optional) This can be used to set the action to be taken upon validation i.e. if any document fails the set validation then whether to show error or just warn about the violations. Default value is error.

Creating a Capped Collection

We can create a capped collection using the following command.

`db.createCollection("student", { capped : true, size : 5242880, max : 5000 })` This will create a collection named student, with maximum size of 5 megabytes and maximum of 5000 documents.

Drop a Collection

Any collection in a database in MongoDB can be dropped easily using the following command:

`db.collection_name.drop()`

`drop()` method will return true if the collection is dropped successfully, else it will return false.

Documents

MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents

Document Structure

MongoDB documents are composed of field-and-value pairs and have the following structure:

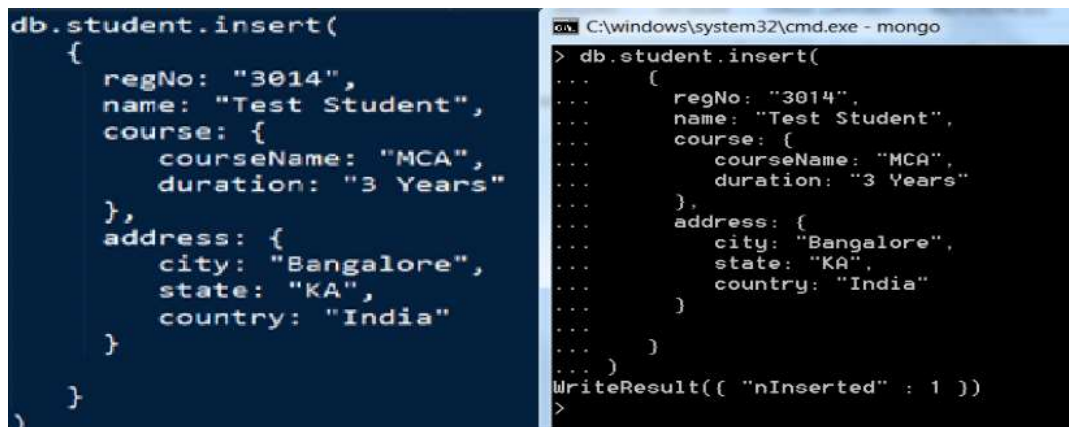
```
{
  field1: value1,
  field2: value2,
  field3: value3,
  ...
  fieldN: valueN
}
```

CRUD Operations in MongoDB

CRUD operations refer to the basic Insert, Read, Update and Delete operations.

Inserting a document into a collection

The Command `db.collection.insert()` will perform an insert operation into a collection of a document. Let us insert a document to a student collection. You must be connected to a database for doing any insert.



```
db.student.insert(
{
  regNo: "3014",
  name: "Test Student",
  course: {
    courseName: "MCA",
    duration: "3 Years"
  },
  address: {
    city: "Bangalore",
    state: "KA",
    country: "India"
  }
}
)
```

```
C:\windows\system32\cmd.exe - mongo
> db.student.insert(
...  {
...    regNo: "3014",
...    name: "Test Student",
...    course: {
...      courseName: "MCA",
...      duration: "3 Years"
...    },
...    address: {
...      city: "Bangalore",
...      state: "KA",
...      country: "India"
...    }
...  }
... )
WriteResult({ "nInserted" : 1 })
>
```

You can also pass an array of documents into the insert() method as shown below:

```
> db.createCollection("post")
> db.post.insert([
  {
    title: "MongoDB Overview",
    description: "MongoDB is no SQL database",
    by: "tutorials point",
    url: "http://www.tutorialspoint.com",
    tags: ["mongodb", "database", "NoSQL"],
    likes: 100
  },
  {
    title: "NoSQL Database",
    description: "NoSQL database doesn't have tables",
    by: "tutorials point",
    url: "http://www.tutorialspoint.com",
    tags: ["mongodb", "database", "NoSQL"],
    likes: 20,
    comments: [
      {
        user: "user1",
        message: "My first comment",
        dateCreated: new Date(2013,11,10,2,35),
        like: 0
      }
    ]
  }
])
```

```

}
])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

```
>db.COLLECTION_NAME.insertOne(document)
```

```

> db.createCollection("empDetails")
{ "ok" : 1 }

> db.empDetails.insertOne(
  {
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26",
    e_mail: "radhika_sharma.123@gmail.com",
    phone: "9848022338"
  })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>

```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

```

> db.empDetails.insertMany(
  [
    {
      First_Name: "Radhika",
      Last_Name: "Sharma",
      Date_Of_Birth: "1995-09-26",
      e_mail: "radhika_sharma.123@gmail.com",
      phone: "9000012345"
    },
    {
      First_Name: "Rachel",
      Last_Name: "Christopher",
      Date_Of_Birth: "1990-02-16",
      e_mail: "Rachel_Christopher.123@gmail.com",
      phone: "9000054321"
    },
    {
      First_Name: "Fathima",
      Last_Name: "Sheik",
      Date_Of_Birth: "1990-02-16",
      e_mail: "Fathima_Sheik.123@gmail.com",
      phone: "9000054321"
    }
  ]
)

```

Read/Query Document

The find() Method : **To retrieve (Select) the inserted document, use MongoDB's find() method.**

Syntax

Db.collectionname.find()

Eg: db.empDetails.find()

To display the results in a formatted way, you can use pretty() method.

syntax

>db.COLLECTION_NAME.find().pretty()

```
> db.customer.find().pretty()
{
  "_id" : ObjectId<"61fc1775acb9229c20e7fd46">,
  "name" : "maya",
  "age" : 34,
  "gen" : "f",
  "amount" : 3000
}
{
  "_id" : ObjectId<"61fc188cacb9229c20e7fd47">,
  "name" : "Divya",
  "age" : 34,
  "gen" : "f",
  "amount" : 6000
}
{
  "_id" : ObjectId<"61fc1902acb9229c20e7fd48">,
  "name" : "Divya",
  "age" : 34,
  "gen" : "f",
  "amount" : 6000
}
```

The findOne() method

Apart from the find() method, there is **findOne()** method, that returns only one document.

Syntax

>db.COLLECTIONNAME.findOne()

Example

Following example retrieves the document with title MongoDB Overview.

```
> db.customer.findOne({name: "Paru"})
```

```
null
> db.customer.findOne({name: "Paru"})
{
  "_id" : ObjectId<"61fc1902acb9229c20e7fd49">,
  "name" : "Paru",
  "age" : 31,
  "gen" : "f",
  "amount" : 10000
}
```

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{<\$eq:<value>}}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{<\$lt:<value>}}	db.mycol.find({"likes":{\$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{<\$lte:<value>}}	db.mycol.find({"likes":{\$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{<\$gt:<value>}}	db.mycol.find({"likes":{\$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{<\$gte:<value>}}	db.mycol.find({"likes":{\$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{<\$ne:<value>}}	db.mycol.find({"likes":{\$ne:50}}).pretty()	where likes != 50

Syntax

To query documents based on the AND condition, you need to use \$and keyword. Following is the basic syntax of AND –

```
>db.nameofthecollection.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
>
> db.customer.find(<<$and : [ <age : 34>, <name : "maya"> ] >>)
{ "_id" : ObjectId("61fc1775acb9229c20e7fd46"), "name" : "maya", "age" : 34, "gender" : "f", "amount" : 3000 }
>
```

Other conditions can be OR,NOR,NOT

Update Document

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows –

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
Property id C:\shell17-1-42
> db.customer.update(<'name': "maya">,<{$set:<'name': "meenu">}>)
WriteResult(<< "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 >>)
> _
```


findOneAndUpdate() method

The **findOneAndUpdate()** method updates the values in the existing document.

Syntax

The basic syntax of **findOneAndUpdate()** method is as follows –

```
>db.COLLECTION_NAME.findOneAndUpdate(SELECTION_CRITERIA,  
UPDATED_DATA)
```

Example

Assume we have created a collection named empDetails and inserted three documents in it , updates the age and email values of the document with name 'Radhika'.

```
> db.empDetails.findOneAndUpdate(  
  {First_Name: 'Radhika'},  
  { $set: { Age: '30', e_mail: 'radhika_newemail@gmail.com'}}  
)  
{  
  "_id" : ObjectId("5dd6636870fb13eec3963bf5"),  
  "First_Name" : "Radhika",  
  "Last_Name" : "Sharma",  
  "Age" : "30",  
  "e_mail" : "radhika_newemail@gmail.com",  
  "phone" : "9000012345"  
}
```

updateOne() method

This methods updates a single document which matches the given filter.

Syntax

The basic syntax of updateOne() method is as follows –

```
>db.COLLECTION_NAME.updateOne(<filter>, <update>)
```

MongoDB updateMany() method

The updateMany() method updates all the documents that matches the given filter.

Syntax

The basic syntax of updateMany() method is as follows –

```
>db.COLLECTION_NAME.update(<filter>, <update>)
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.

- **justOne** – (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of **remove()** method is as follows –

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA)
```

Eg:db.customer.remove(name:"meenu")

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
> db.mycol.remove({})
```