INDEX

Experiment No. 1

SQL INTRODUCTION

SQL (Structured Query Language):

Structured Query Language is a database computer language designed for managing data in relational database management systems(RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F. Codd's relational model in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems.
- SQL adopted as standard language for RDBS by ASNI in 1989.

DATA TYPES:

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 character is 255 characters.

1. VARCHAR (Size) / VERCHAR2 (Size): This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
2. NUMBER (P, S): The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as 9.99 * 10 ^{124}. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
3. DATE: This data type is used to represent date and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time

stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

4. LONG: This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.
5. RAW: The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

Syntax : VERB(Parameter_1,Parameter_2,Parameter_3,........Parameter_n);

SQL language is sub-divided into several language elements, including:

- Clauses, which are in some cases optional, constituent components of statements and queries.
- Expressions, which can produce either scalar values or tables consisting of columns and rows of data.
- Predicates which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- Queries which retrieve data based on specific criteria.
- Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant white space is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. DATA RETRIEVAL LANGUAGE (DRL)
4. TRANSACTIONAL CONTROL LANGUAGE (TCL)
5. DATA CONTROL LANGUAGE (DCL)

Viva Questions:

1. **What is SQL? When SQL appeared? What are the usages of SQL?**

Ans. SQL stands for structured query language. It is a database language used for database creation, deletion, fetching rows and modifying rows etc. sometimes it is pronounced as sequel. It appeared in 1974.

1. To execute queries against a database
2. To retrieve data from a database
3. To inserts records in a database
4. To updates records in a database
5. To delete records from a database
6. To create new databases
7. To create new tables in a database
8. To create views in a database

2. **Does SQL support programming? What are the subsets of SQL?**

Ans. No, SQL doesn't have loop or Conditional statement. It is used like commanding language to access databases. Subsets are:

1. Data definition language (DDL)
2. Data manipulation language (DML)
3. Data control language (DCL)

Experiment No. 2

DATA DEFINITION LANGUAGE (DDL) COMMANDS

DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of Five basic DDL commands:

1. CREATE
2. ALTER
3. DROP
4. RENAME
5. TRUNCATE

1. CREATE:

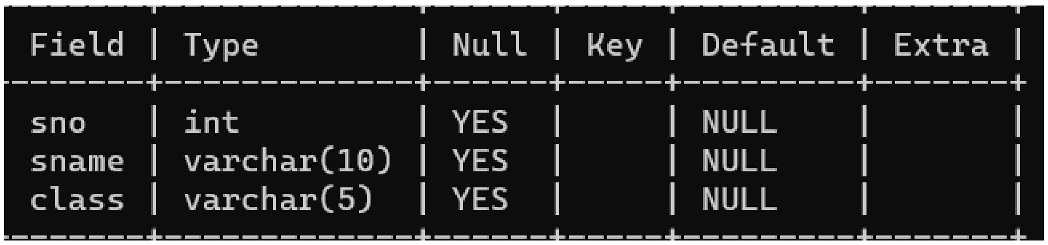
(a) CREATE TABLE: This is used to create a new relation and the corresponding

Syntax: CREATE TABLE relation_name

(field_1 data type(Size), field_2 data_type(Size) , .. ……………..);

Example:

SQL>CREATE TABLE Student (sno NUMBER(3), sname CHAR(10), class CHAR(5));

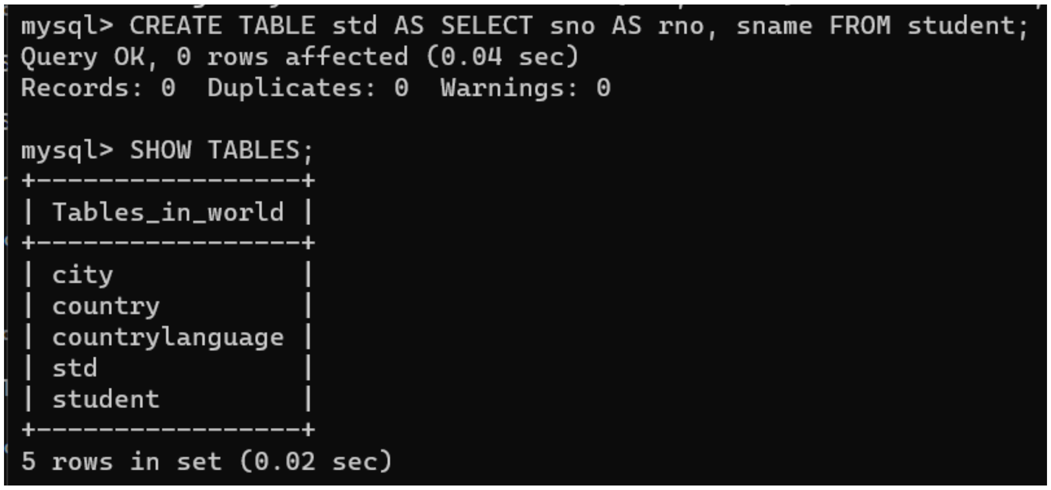


```
--------+-------------+------+-----+---------+-------+
 Field | Type        | Null | Key | Default | Extra |
--------+-------------+------+-----+---------+-------+
 sno   | int         | YES  |     | NULL    |       |
 sname | varchar(10) | YES  |     | NULL    |       |
 class | varchar(5)  | YES  |     | NULL    |       |
--------+-------------+------+-----+---------+-------+
```

(b) CREATE TABLE..AS SELECT....: This is used to create the structure of a new relation from the structure of an existing relation.

Syntax: CREATE TABLE (relation_name_1, field_1,field_2,.....field_n) AS SELECT field_1,field_2,...........field_n FROM relation_name_2;

Example: SQL>CREATE TABLE std(rno, sname) AS SELECT sno, sname FROM student;



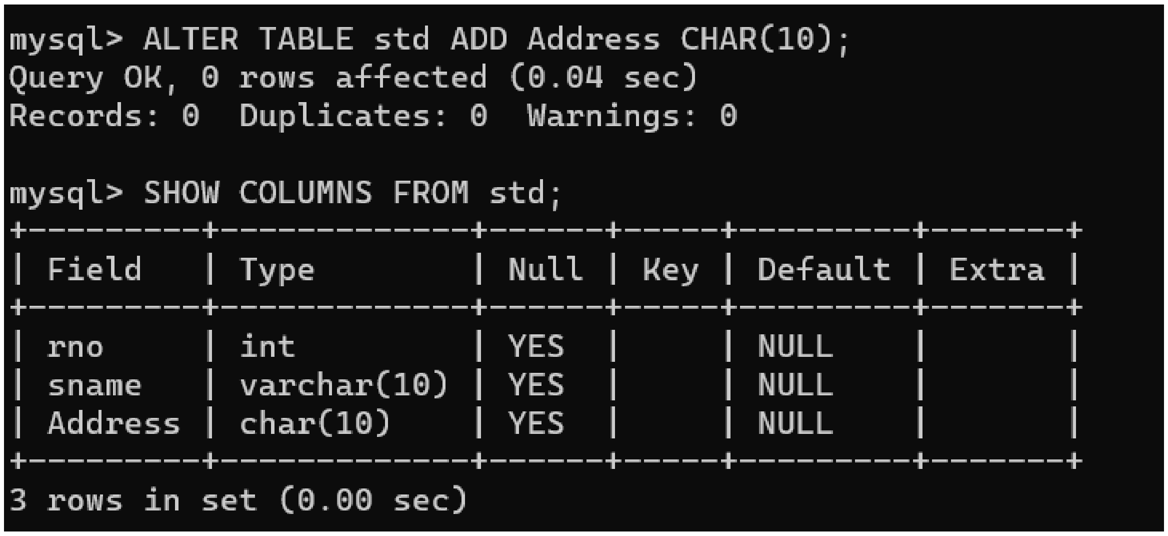
```
mysql> CREATE TABLE std AS SELECT sno AS rno, sname FROM student;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW TABLES;
+-----------------+
| Tables_in_world |
+-----------------+
| city            |
| country         |
| countrylanguage |
| std             |
| student         |
+-----------------+
5 rows in set (0.02 sec)
```

2. ALTER:

(a) ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD(new field_1 data_type(size), new field_2 data_type(size),…………);

Example: SQL>ALTER TABLE std ADD(Address CHAR(10));

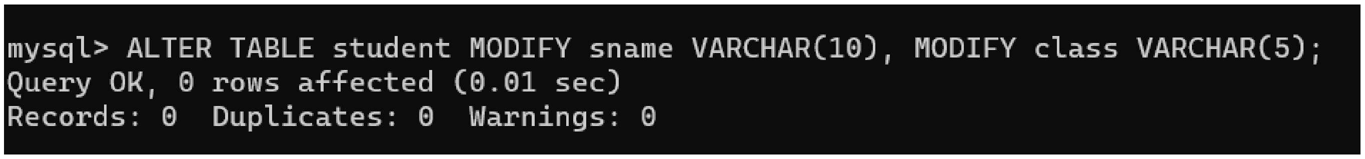


```
mysql> ALTER TABLE std ADD Address CHAR(10);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW COLUMNS FROM std;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| rno     | int         | YES  |     | NULL    |       |
| sname   | varchar(10) | YES  |     | NULL    |       |
| Address | char(10)    | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

(b) ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size),..............field_newdata_type(Size));

Example: SQL>ALTER TABLE student MODIFY(sname VARCHAR(10),class VARCHAR(5));



```
mysql> ALTER TABLE student MODIFY sname VARCHAR(10), MODIFY class VARCHAR(5);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax: DROP TABLE relation_name;

Example: SQL>DROP TABLE std;

4. RENAME: It is used to modify the name of the existing database object.

Syntax: RENAME TABLE old_relation_name TO new_relation_name;

Example: SQL>RENAME TABLE std TO std1;

5. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE student;

<u>Difference between Truncate & Delete:-</u>

- ✓ By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- ✓ By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- ✓ Truncate is a DDL command & delete is a DML command.

Viva Questions:

1. **What is the difference between DELETE and TRUNCATE statement in SQL?**

Ans. DELETE command is used to remove rows from the table, and WHERE clause can be used for conditional set of parameters. Commit and Rollback can be performed after delete statement.

TRUNCATE removes all rows from the table. Truncate operation cannot be rolled back.

2. **What is the difference between VARCHAR2 AND CHAR data types, VARCHAR AND VARCHAR (MAX) data types?**

Ans. **CHAR**

1. Used to store character string value of fixed length**.**
2. The maximum no. of characters the data type can hold is 255 characters.
3. It's 50% faster than VARCHAR.
4. Uses static memory allocation.

VARCHAR

1. Used to store variable length alphanumeric data.
2. The maximum this data type can hold is up to
3. Pre-MySQL 5.0.3: 255 characters**.**
4. In MySQL 5.0.3+: 65,535 characters shared for the row.
5. It's slower than CHAR.
6. Uses dynamic memory allocation*.

VARCHAR stores variable-length character data whose range varies up to 8000 bytes; varchar (MAX) stores variable-length character data whose range may vary beyond 8000 bytes and till 2 GB.

Experiment No. 3

DATA MANIPULATION LANGUAGE (DML) COMMANDS

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

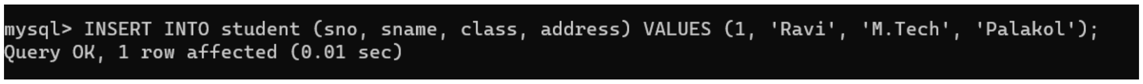
1. INSERT **2. UPDATE** **3. DELETE**

1. INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries which are as

a) Inserting a single record

Syntax: INSERT INTO relationname(field_1,field_2,………field_n)
VALUES (data_1,data_2,........data_n);

Example: SQL>INSERT INTO student (sno,sname,class,address)VALUES
(1,'Ravi','M.Tech','Palakol');

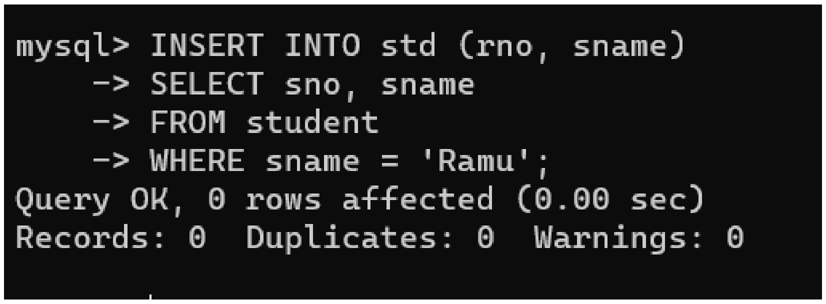


```
mysql> INSERT INTO student (sno, sname, class, address) VALUES (1, 'Ravi', 'M.Tech', 'Palakol');
Query OK, 1 row affected (0.01 sec)
```

b) Inserting all records from another relation

Syntax: INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n
FROM relation_name_2 WHERE field_x=data;

Example: SQL>INSERT INTO std SELECT sno,sname FROM student
WHERE name = 'Ramu';



```
mysql> INSERT INTO std (rno, sname)
    -> SELECT sno, sname
    -> FROM student
    -> WHERE sname = 'Ramu';
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

c) Inserting multiple records

Syntax: INSERT INTO relation_name field_1,field_2,.....field_n) VALUES

(&data_1,&data_2,........&data_n);

Example: SQL>INSERT INTO student(sno,sname,class,address)

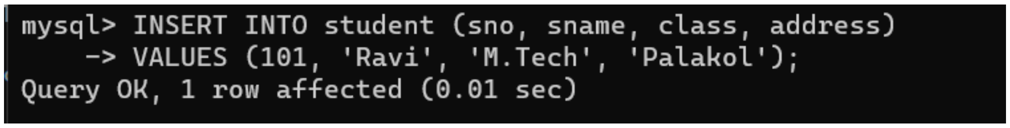
VALUES(&sno,'&sname','&class','&address');

Enter value for sno: 101

Enter value for name: Ravi

Enter value for class: M.Tech

Enter value for name: Palakol



```
mysql> INSERT INTO student (sno, sname, class, address)
    -> VALUES (101, 'Ravi', 'M.Tech', 'Palakol');
Query OK, 1 row affected (0.01 sec)
```

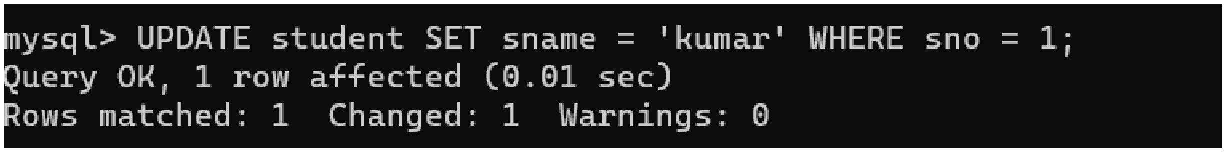
2. UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name

SET Field_name1=data,field_name2=data,……..

WHERE field_name=data;

Example: SQL>UPDATE student SET sname = 'kumar' WHERE sno=1;



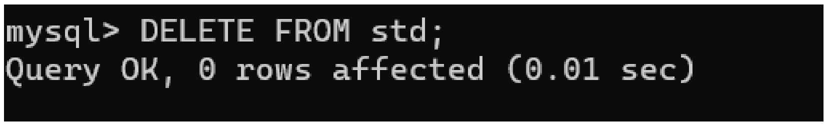
```
mysql> UPDATE student SET sname = 'kumar' WHERE sno = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

3. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax: SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

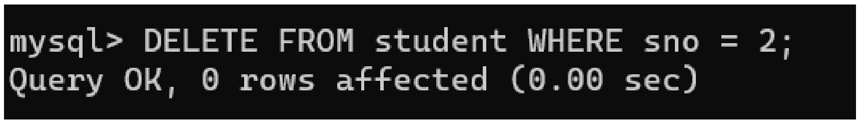


```
mysql> DELETE FROM std;
Query OK, 0 rows affected (0.01 sec)
```

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation.

Syntax: SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;



```
mysql> DELETE FROM student WHERE sno = 2;
Query OK, 0 rows affected (0.00 sec)
```

Viva Questions:

1. **What is the syntax to add record to a table? How do you add a column to a table?**

Ans. 1. INSERT INTO tblCustomers (CustomerID, [Last Name], [First Name])
VALUES (1, 'Kelly', 'Jill');
2. ALTER TABLE table_name ADD column_name datatype;

2. **How to delete duplicate records in a table?**

Ans. (DELETE FROM <table name>
WHERE <primary key columns> NOT IN
(SELECT MAX (rowid) FROM <table name>
Group by <primary key columns> ;)

Experiment No. 4

SELECT QUERY

Retrieve data from one or more tables.

1. SELECT FROM: To display all fields for all records.

Syntax: SELECT * FROM relation_name;

Example: SQL> select * from dept;

```
DEPTNO      DNAME
  --------      -----------
  10          ACCOUNTING
  20          RESEARCH
  30          SALES
 40          OPERATIONS
```

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: SELECT a set of fields FROM relation_name;

Example: SQL> select deptno, dname from dept;

```
 DEPTNO          DNAME
  -------           ----------
   10             ACCOUNTING
   20             RESEARCH
   30             SALES
```

3. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation.

Syntax: SELECT a set of fields FROM relation_name WHERE condition;

Example: SQL> select * FROM dept WHERE deptno<=20;

```
DEPTNO      DNAME
 ------          -----------
10            ACCOUNTING
20            RESEARCH
```

4. SELECT - FROM -GROUP BY: This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

Syntax: SELECT a set of fields FROM relation_name GROUP BY field_name;

Example: SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

```
EMPNO        SUM (SALARY)
------          ----------
 1                 3000
 2                 4000
 3                 5000
 4                 6000
```

4 rows selected.

5. SELECT - FROM -ORDER BY: This query is used to display a selected set of fields from a relation in an ordered manner base on some field (ascending and descending order).

Syntax: SELECT a set of fields FROM relation_name
ORDER BY field_name;

Example: SQL> SELECT empno,ename,job FROM emp ORDER BY salary;

```
EMPNO        ENAME        SALARY
------        ---------       --------
 4           RAVI          3000
 2           aravind       4000
 1           sagar         5000
 3           Laki          6000
```

4rows selected.

Example: SQL> SELECT empno, ename, job FROM emp ORDER BY salary desc;

```
EMPNO        ENAME        SALARY
------        ---------       --------
 4           Laki          6000
 2           sagar         5000
 1           aravind       4000
 3           RAVI          3000
```

6. **HAVING clause**: HAVING clause is used to specify which groups are to be displayed, and thus, you further restrict the groups on the basis of aggregate information. In the syntax:

group_condition restricts the groups of rows returned to those groups for which the specified condition is true

Example: SQL> SELECT department_id, MAX (salary) FROMemployees
GROUP BY department_id **HAVING MAX (salary)>10000;**

```
        Department_id                 MAX (SALARY)
             20                          5000
             30                          8000
             50                          7500
             60                          5900
```

7. JOIN using SELECT - FROM - ORDER BY: This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

Syntax: SELECT a set of fields from both relations FROM relation_1, relation_2 WHERE relation_1.field_x = relation_2.field_y ORDER BY field_z;

Example: SQL>SELECT empno,ename,job,dname FROM emp,dept
WHERE emp.deptno = 20 ORDER BY job;

```
EMPNO      ENAME       JOB         DNAME
 ------     ------     -------      ----------
7788       SCOTT      ANALYST     ACCOUNTING
7902       FORD       ANALYST     ACCOUNTING
7566       JONES      MANAGER     OPERATIONS
7566       JONES      MANAGER     SALES
20 rows selected.
```

8. JOIN using SELECT - FROM - GROUP BY: This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

Syntax: SELECT a set of fields from both relations FROM relation_1,relation_2 WHERE relation_1.field-x=relation_2.field-y GROUP BY field-z;

Example: SQL> SELECT empno,SUM(SALARY) FROM emp,dept
WHERE emp.deptno =20 GROUP BY JOB;

```
EMPNO         SUM (SALARY)
  -------            --------
7369               3200
7566               11900
7788               12000
7876               4400
```

9. UNION: This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows.

Syntax: SELECT field_1, field_2,....... FROM relation_1 WHERE (Condition) UNION SELECT field_1, field_2,....... FROM relation_2 WHERE (Condition);

Example:

```
        SQL> SELECT * FROM STUDENT;            SQL> SELECT * FROM STD;
            SNO          SNAME                 SNO            SNAME
            -----        -------               -----          -------
            1            kumar                  3             ramu
            2            ravi                   5             lalitha
            3            ramu                  9               devi
```

```
SQL> SELECT * FROM student UNION SELECT * FROM std;
            SNO          SNAME
            ----         ------
             1           kumar
             2           ravi
             3           ramu
             5           lalitha
             9           devi
```

10. INTERSECT: This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them.

Syntax: SELECT field_1, field_2, ... FROM relation_1 WHERE

(Condition) INTERSECT SELECT field_1, field_2,.. FROM relation_2 WHERE(Condition);

Example: SQL> SELECT * FROM student INTERSECT SELECT * FROM std;

```
SNO         SNAME
----        -------
1           Kumar
3           ramu
```

11. MINUS: This query is used to display all the rows in relation_1, which are not having in the relation_2.

Syntax: SELECT field_1, field_2,......FROM relation_1

WHERE (Condition) MINUS SELECT field_1,field_2,.....

FROM relation_2 WHERE (Conditon);

SQL> SELECT * FROM student MINUS SELECT * FROM std;

```
SNO         SNAME
----        -------
2           RAVI
```

Note: Different Where Condition

1) Comparison Condition

Operator	**Meaning**
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>	Not Equal to
Between ……AND……..	Between two values
IN (set)	Match any of a list of values
LIKE	Match a Character Pattern
IS NULL	Is a null Value

2) Logical Conditions

Operator	**Meaning**
AND	Return TRUE if both component condition are true
OR	Return TRUE if either component condition is true
NOT	Return TRUE if the following condition is false

Viva Questions:

1. **What is the difference between having and where clause?**

Ans. ***WHERE clause*** can be used with - Select, Insert, and Update statements, where as ***HAVING clause*** can only be used with the Select statement.

WHERE filters rows before aggregation (GROUPING), whereas, ***HAVING*** filters groups, after the aggregations are performed.

Aggregate functions cannot be used in the ***WHERE clause***, unless it is in a sub query contained in a **HAVING clause**, whereas, aggregate functions can be used in Having clause.

2. **Define UNION, MINUS, UNION ALL, INTERSECT?**

Ans. **UNION -** The values of the first query are returned with the values of the second query eliminating duplicates.

MINUS - The values of the first query are returned with duplicates values of the second query removed from the first query.

UNION ALL - The values of both queries are returned including all duplicates

INTERSECT - Only the duplicate values are returned from both queries.

Experiment No. 5

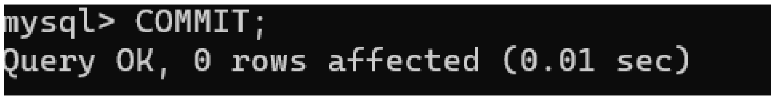
TRANSACTIONAL CONTROL LANGUAGE (T.C.L) COMMANDS

TRANSACTIONAL CONTROL LANGUAGE (T.C.L): A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

1. COMMIT: This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

Syntax: SQL>COMMIT;

Example: SQL>COMMIT;

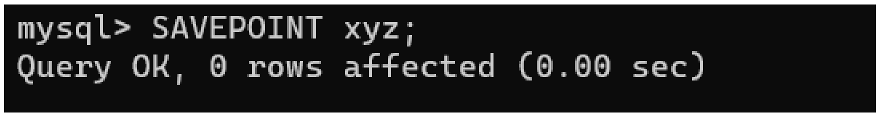


```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

2. SAVEPOINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter rollback. Thus, save point is used in conjunction with rollback.

Syntax: SQL>SAVEPOINT ID;

Example: SQL>SAVEPOINT xyz;



```
mysql> SAVEPOINT xyz;
Query OK, 0 rows affected (0.00 sec)
```

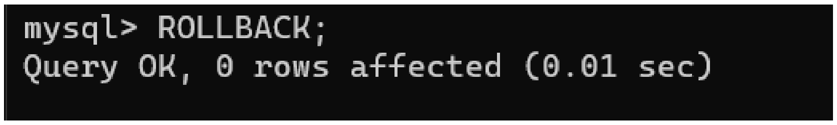
3. ROLLBACK: A rollback command is used to undo the current transactions. We can rollback the entire transaction so that all changes made by SQL statements are undo (or) rollback a transaction to a savepoint so that the SQL statements after the savepoint are rollback.

Syntax: ROLLBACK (current transaction can be rollback)

ROLLBACK to save point ID;

Example: SQL>ROLLBACK;

SQL>ROLLBACK TO SAVEPOINT xyz;



```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

Viva Questions:

1. **What is meant by TCL, What are its commands?**

Ans. A Transaction Control Language (TCL) is a computer language and a subset of SQL, used to control transactional processing in a database.

Commit command

Commit command is used to permanently save any transaaction into database.

Rollback command

This command restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction.

Savepoint command

Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

2. **What is a transaction? What are ACID properties?**

Ans. A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program. Transactions have the following four standard properties, usually referred to by the acronym ACID:

- **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.

- **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation:** enables transactions to operate independently of and transparent to each other.
- **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

Experiment No. 6

DATA CONTROL LANGUAGE (D.C.L) COMMANDS

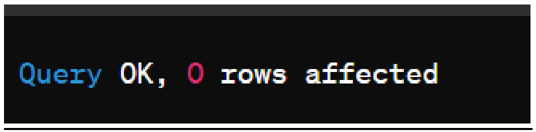
DATA CONTROL LANGUAGE (D.C.L): DCL provides users with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base users to access the objects as per their requirement

1. GRANT: The GRANT command allows granting various privileges to other users and allowing them to perform operations within their privileges

For Example, if a uses is granted as 'SELECT' privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

Syntax: SQL>GRANT PRIVILEGES on object_name To user_name;

Example: SQL>GRANT SELECT, UPDATE on emp To hemanth;

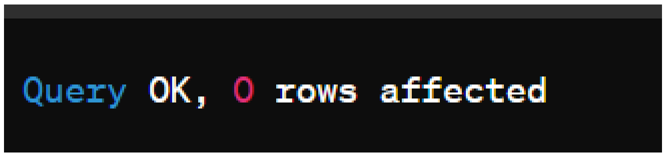


```
Query OK, 0 rows affected
```

2. REVOKE: To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

Syntax: SQL>REVOKE PRIVILEGES ON object-name FROM user_name;

Example: SQL>REVOKE SELECT, UPDATE ON emp FROM ravi;



```
Query OK, 0 rows affected
```

Viva Questions:

1. **Provide all privileges on employee table to User1.**

   Ans. GRANT SYSDBA on employee to USER1;

2. **Remove all privileges on employee table to User1.**

   Ans. REVOKE SYSDBA on employee to USER1;

Experiment No. 7

CONSTRAINTS IN DATABASE

Constraints are used to apply business rules for the database tables.

The constraints available in SQL are **Primary Key**, **Foreign Key, Not Null, Unique, Check.**

Constraints can be defined in two ways
1) The constraints can be specified immediately after the column definition. This is called column-level definition.
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

I) SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
Column_name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

- **column_name1, column_name2** are the names of the columns which define the primary Key.
- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

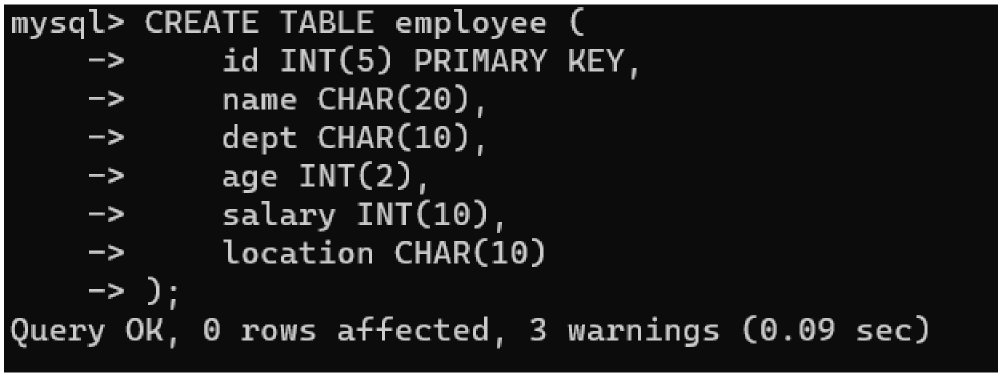
For Example: To create an employee table with Primary Key constraint, the query would be like.

Primary Key at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

or

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```



```
mysql> CREATE TABLE employee (
    ->     id INT(5) PRIMARY KEY,
    ->     name CHAR(20),
    ->     dept CHAR(10),
    ->     age INT(2),
    ->     salary INT(10),
    ->     location CHAR(10)
    -> );
Query OK, 0 rows affected, 3 warnings (0.09 sec)
```

Primary Key at column level:

```
CREATE TABLE employee
( id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp_id_pk PRIMARY KEY (id)
);
```

II) SQL Foreign key or Referential Integrity:

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as foreign key.

Syntax to define a Foreign key at column level:

```
[CONSTRAINT constraint_name] REFERENCES Referenced_Table_name(column_name)
```

Syntax to define a Foreign key at table level:

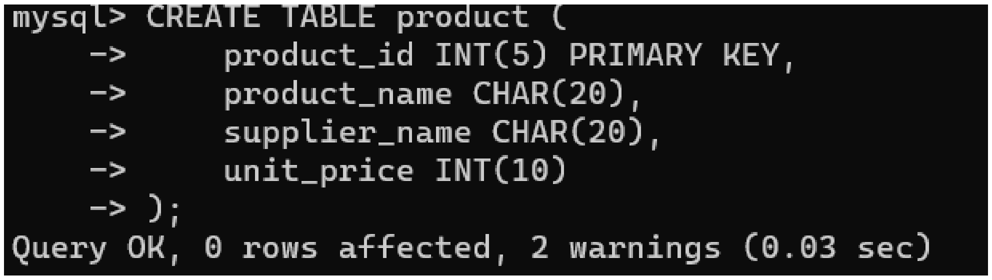
[CONSTRAINT constraint_name] FOREIGN KEY(column_name) REFERENCES referenced_table_name(column_name);

For Example:

a) Lets use the "product" table and "order_items".

Foreign Key at column level:

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```



```
mysql> CREATE TABLE product (
    ->     product_id INT(5) PRIMARY KEY,
    ->     product_name CHAR(20),
    ->     supplier_name CHAR(20),
    ->     unit_price INT(10)
    -> );
Query OK, 0 rows affected, 2 warnings (0.03 sec)
```

```
CREATE TABLE order_items
( order_id number(5) CONSTRAINT od_id_pk PRIMARY KEY,
product_id number(5) CONSTRAINT pd_id_fk REFERENCES, product(product_id),
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

Foreign Key at table level:

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT pd_id_fk FOREIGN KEY(product_id) REFERENCES product(product_id)
);
```

b) If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id),
salary number(10),
location char(10)
);
```

III) SQL Not Null Constraint:

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

Syntax to define a Not Null constraint:

```
Attribute_name (size) NOT NULL
```

For Example: To create a employee table with Null value, the query would be like

```
CREATE TABLE employee
( id number(5),
name char(20) NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

IV) SQL Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

```
[CONSTRAINT constraint_name] UNIQUE (column_name)
```

For Example: To create an employee table with unique key, the query would be like,

Unique Key at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

or

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) CONSTRAINT loc_un UNIQUE
);
```

Unique Key at table level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location)
);
```

V) SQL Check Constraint:

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:

[CONSTRAINT constraint_name] CHECK (condition)

For Example: In the employee table to select the gender of a person, the query would be like

Check Constraint at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10)
);
```

Check Constraint at table level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1),
salary number(10),
location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F'))
);
```

Viva Questions:

1. **Explain the terms foreign key and unique.**

Ans. A foreign key is one table which can be related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table. A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns. A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key.There can be much unique constraint defined per table, but only one Primary key constraint defined per table.

2. **Which of the following can be addressed by enforcing a referential integrity constraint?**

Ans. a) All phone numbers must include the area code.

b) Certain fields are required (such as the email address, or phone number) before the record is accepted

c) Information on the customer must be known before anything can be sold to that customer

d) When entering an order quantity, the user must input a number and not some text. (i.e.:12 rather than ‘a dozen’)

Ans**. (C)**

Experiment No. 8

JOINS OPERATION ON TABLES

JOINS:

Join is a query in which data is returned from two or more tables.
How the join will be performed:
Step 1: Make the Cartesian product of the given tables.
Step 2: Check for the equality on common attributes for the given tables.

1) Natural join:

It returns the matching rows from the table that are being joined.
Syntax:
select <attribute> from TN1 Natural Join TN2;

2) Equi join:

It returns the matching rows from the tables that are being joined.
Syntax:
select <tablename.attribute> from TN1,TN2 where TN1.attribute=TN2.attribute;

3) Left outer join:

It returns all the rows from the table1 even when they are unmatched.
Syntax:
select<tablename.attribute> from TN1 left outer join TN2 on TN1.attribute=TN2.attribute;

OR

select<tablename.attribute> from TN where TN1.attribute=(+)TN2.attribute;

4) Right outer join:

It returns all the rows from the table2 even when they are unmatched.
Syntax:
Select <tablename.attribute> from TN1 right outer join TN2 on TN1.attribute=TN2.attribute.

OR

Select <tablename.attribute> from TN where TN1.attribute(+)=TN2.attribute.

5) Full outer join:

It is the combination of both left outer and right outer join.
Syntax:
select<tablename.attribute> from TN1 full outer join TN2 on TN1.attribute=TN2.attribute.

Viva Questions:

1. **What Is The Difference Between Join And Union?**

Ans. Joins and Unions can be used to combine data from one or more tables. The difference lies in how the data is combined.

In simple terms, **joins combine data into new columns**. If two tables are joined together, then the data from the first table is shown in one set of column alongside the second table's column in the same row.

Unions combine data into new rows. If two tables are "unioned" together, then the data from the first table is in one set of rows, and the data from the second table in another set. The rows are in the same result.

2. **What Is The Difference Between Cross Join And Natural Join?**

Ans. The cross join produces the cross product or Cartesian product of two tables. The natural join is based on all the columns having same name and data types in both the tables.

Experiment No. 9

SUBQUERIES

Subqueries

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including:

- The WHERE clause
- The HAVING clause
- The FROM clause

In the syntax:

operator includes a comparison condition such as >, =, or IN

Note: Comparison conditions fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL).

The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

Subquery Syntax

Select Select_list from table where expr operator (select Select_list from table);

Example

```
SQL>  SELECT last_name, job_id   FROM employees

WHERE  job_id =   (SELECT job_id FROM employees WHERE  employee_id = 141);
```

```
Last_Name                    Job_id

Shayam                       Manager
Mohan                        Manager
Rahul                        Manager
Prince                       Manager
```

VIEWS

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

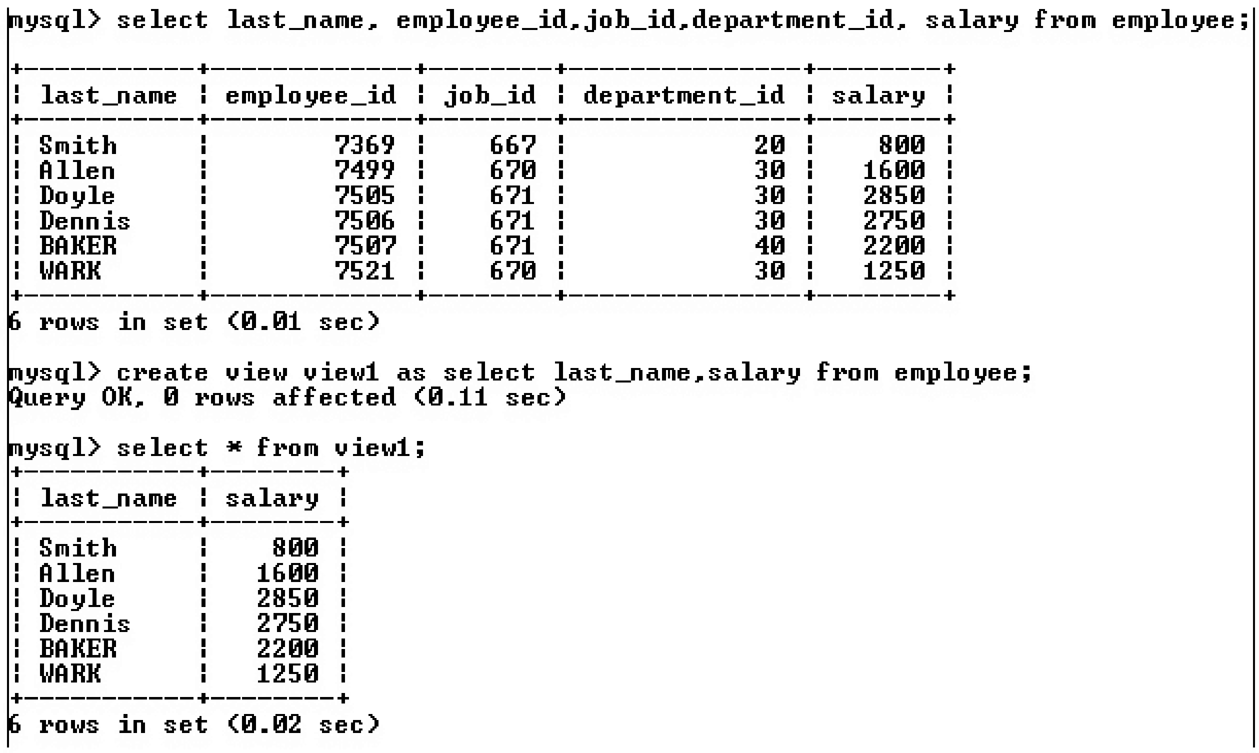
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

VIEW Syntax

CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name

Example



```
mysql> select last_name, employee_id,job_id,department_id, salary from employee;

+-----------+-------------+--------+---------------+--------+
| last_name | employee_id | job_id | department_id | salary |
+-----------+-------------+--------+---------------+--------+
| Smith     |        7369 |    667 |            20 |    800 |
| Allen     |        7499 |    670 |            30 |   1600 |
| Doyle     |        7505 |    671 |            30 |   2850 |
| Dennis    |        7506 |    671 |            30 |   2750 |
| BAKER     |        7507 |    671 |            40 |   2200 |
| WARK      |        7521 |    670 |            30 |   1250 |
+-----------+-------------+--------+---------------+--------+
6 rows in set (0.01 sec)

mysql> create view view1 as select last_name,salary from employee;
Query OK, 0 rows affected (0.11 sec)

mysql> select * from view1;
+-----------+--------+
| last_name | salary |
+-----------+--------+
| Smith     |    800 |
| Allen     |   1600 |
| Doyle     |   2850 |
| Dennis    |   2750 |
| BAKER     |   2200 |
| WARK      |   1250 |
+-----------+--------+
6 rows in set (0.02 sec)
```

Viva Questions:

1. **Is the sub-query always executes before the execution of the main query.**

Ans. Yes

2. **Explain views.**

Ans. A view is a logical snapshot based on a table or another view. It is used for −

- Restricting access to data;
- Making complex queries simple;
- Ensuring data independency;
- Providing different views of same data.

Experiment No. 10

PL/SQL INTRODUCTION

- **PL/SQL** stands for **Procedural Language** extension of SQL.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).

A Simple PL/SQL Block:

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

PL/SQL Block consists of three sections:

- The Declaration section (optional).
- The Execution section (mandatory).
- The Exception (or Error) Handling section (optional).

Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

Exception Section:

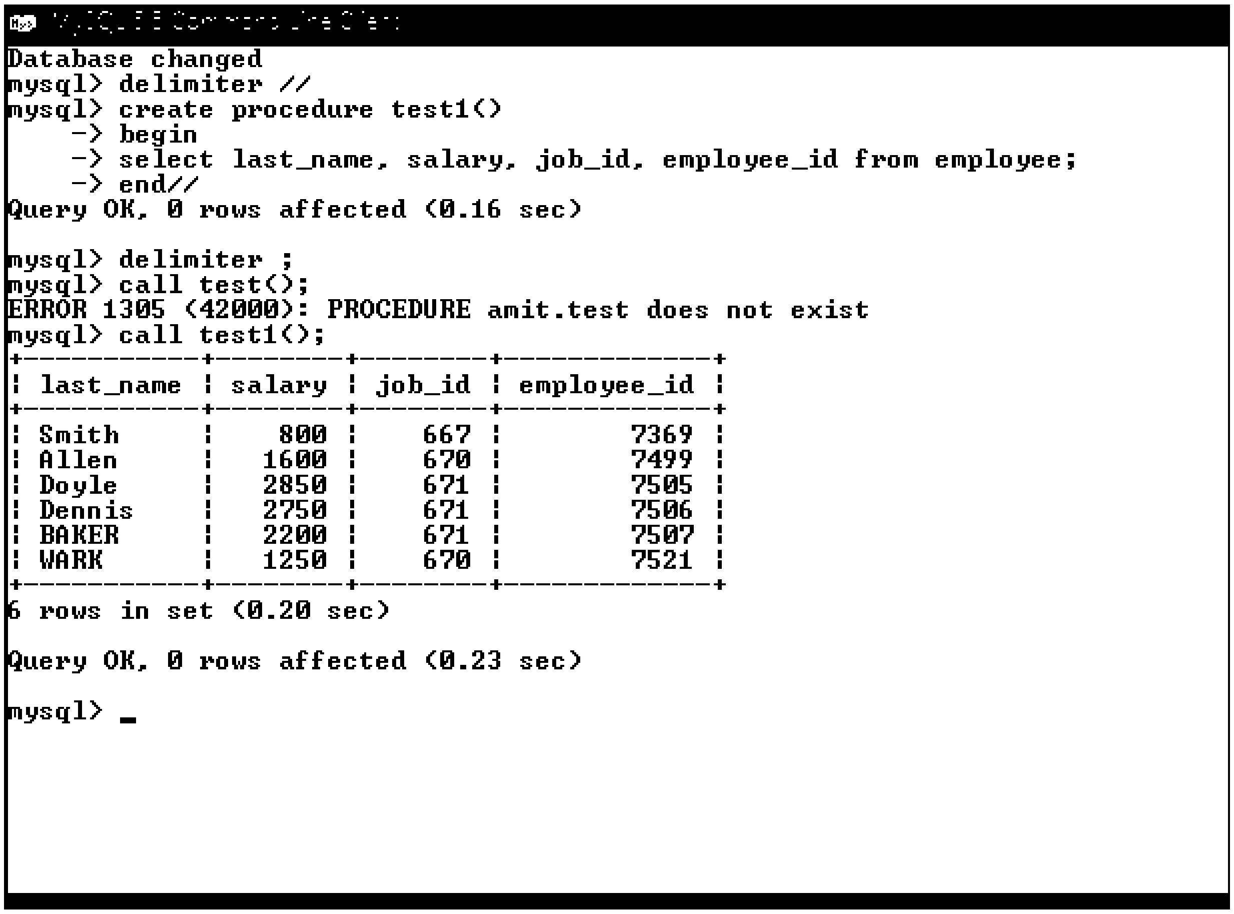
The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

Every statement in the above three sections must end with a semicolon **;** . PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

How a Sample PL/SQL Block Looks

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

PL/SQL Program



```
Database changed
mysql> delimiter //
mysql> create procedure test1()
    -> begin
    -> select last_name, salary, job_id, employee_id from employee;
    -> end//
Query OK, 0 rows affected (0.16 sec)

mysql> delimiter ;
mysql> call test();
ERROR 1305 (42000): PROCEDURE amit.test does not exist
mysql> call test1();
+-----------+--------+--------+-------------+
| last_name | salary | job_id | employee_id |
+-----------+--------+--------+-------------+
| Smith     |    800 |    667 |        7369 |
| Allen     |   1600 |    670 |        7499 |
| Doyle     |   2850 |    671 |        7505 |
| Dennis    |   2750 |    671 |        7506 |
| BAKER     |   2200 |    671 |        7507 |
| WARK      |   1250 |    670 |        7521 |
+-----------+--------+--------+-------------+
6 rows in set (0.20 sec)

Query OK, 0 rows affected (0.23 sec)

mysql> _
```

Viva Questions:

1. **What is the basic structure of PL/SQL?**

Ans.
```
[DECLARE]
   Declaration statements;
BEGIN
   Execution statements;
EXCEPTION
      Exception handling statements;
END;
```

2. **Explain the types of blocks in PL/SQL?**

Ans. **Anonymous Block**
- It is a block of codes without a name.
- It may contain a declaration part, an execution part, and exception handlers.

Stored Program Unit
- It is a block of codes with a name.
- It is similar to an anonymous block just that it can take parameters and return values.

Trigger
- It is a block of code that is implicitly fired based some specific event.

Experiment No. 11

PROCEDURE

A procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.
1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

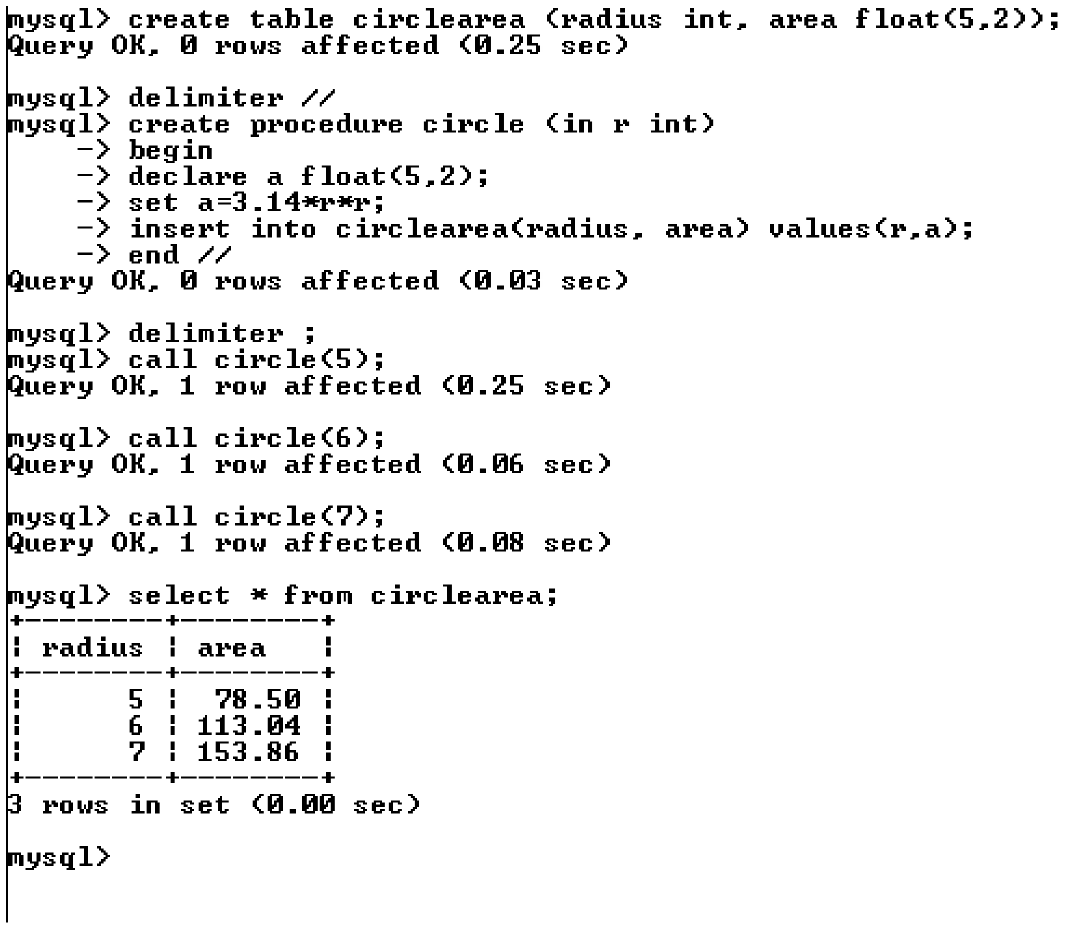
A procedure may or may not return any value.

<u>General Syntax to create a procedure:</u>

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]

IS

   Declaration section

BEGIN

   Execution section

EXCEPTION

  Exception section

END;
```

Procedure Example

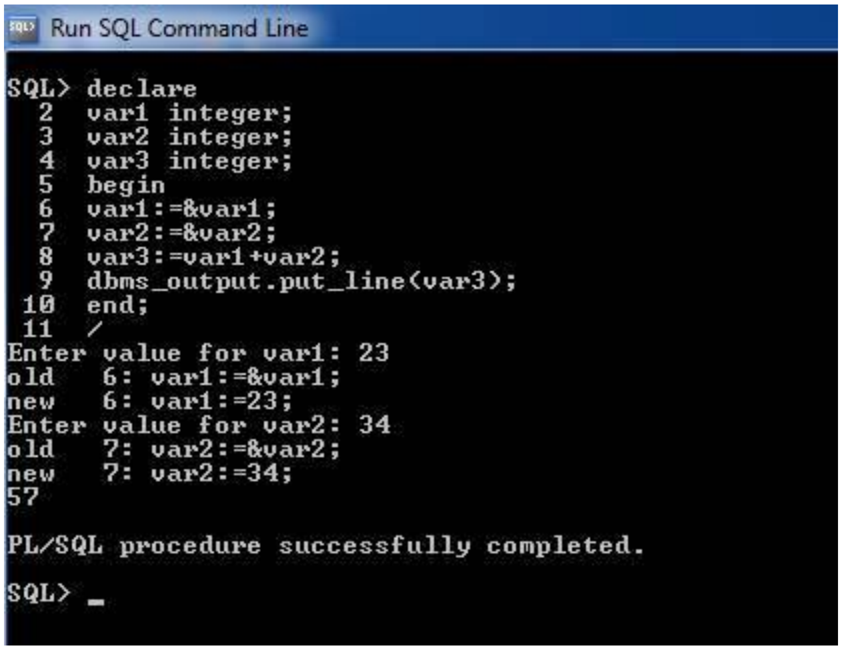
PL/SQL TO FIND AREA OF CIRCLE



```
mysql> create table circlearea (radius int, area float(5,2));
Query OK, 0 rows affected (0.25 sec)

mysql> delimiter //
mysql> create procedure circle (in r int)
    -> begin
    -> declare a float(5,2);
    -> set a=3.14*r*r;
    -> insert into circlearea(radius, area) values(r,a);
    -> end //
Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;
mysql> call circle(5);
Query OK, 1 row affected (0.25 sec)

mysql> call circle(6);
Query OK, 1 row affected (0.06 sec)

mysql> call circle(7);
Query OK, 1 row affected (0.08 sec)

mysql> select * from circlearea;
+--------+--------+
| radius | area   |
+--------+--------+
|      5 |  78.50 |
|      6 | 113.04 |
|      7 | 153.86 |
+--------+--------+
3 rows in set (0.00 sec)

mysql>
```

PL/SQL TO FIND ADDITION OF TWO NUMBERS

```
Declare
Var1 integer;
Var2 integer;
Var3 integer;
Begin
Var1:=&var1;
Var2:=&var2;
Var3:=var1+var2;
Dbms_output.put_line(var3);
End;
/
```



```
SQL> declare
  2  var1 integer;
  3  var2 integer;
  4  var3 integer;
  5  begin
  6  var1:=&var1;
  7  var2:=&var2;
  8  var3:=var1+var2;
  9  dbms_output.put_line(var3);
 10  end;
 11  /
Enter value for var1: 23
old   6: var1:=&var1;
new   6: var1:=23;
Enter value for var2: 34
old   7: var2:=&var2;
new   7: var2:=34;
57

PL/SQL procedure successfully completed.

SQL> _
```

Viva Questions:

1. Explain the difference between a FUNCTION, PROCEDURE.

Ans.

1. Procedure can performs one or more tasks where as function performs a specific task.
2. Procedure may or may not return value where as function should return one value.
3. We can call functions in select statement where as procedure we can’t.
4. We can call function within procedure but we cannot call procedure within function.
5. A FUNCTION must be part of an executable statement, as it cannot be executed independently where as procedure represents an independent executable statement.
6. Function can be called from SQL statement where as procedure can't be called from the SQL statement.
7. Function are normally used for computation where as procedure are normally used for executing business logic.
8. Stored procedure supports deferred name resolution where as function won’t support.
9. Stored procedure returns always integer value by default zero. whereas function returns type could be scalar or table or table value. 10. Stored procedure is precompiled execution plan where as function are not.

2. What is stored Procedure? What are the advantages of stored procedure?

Ans. Stored Procedure is a function consists of many SQL statements to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required.

Experiment No. 12

TRIGGER

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

The syntax for creating a trigger is:

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
trigger_timetrigger_event
  ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

- 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 would 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 would 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, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would 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.

Viva Questions:

1. **What are the types of triggers?**

Ans. There are three type of Triggers

1) DML

a) Instead of Trigger: Instead of trigger are fired in place of the triggering action such as an insert, update, or delete.
b) After Trigger: After trigger execute following the triggering action, such as an insert, update, or delete.

2) DDL Trigger: This type of trigger is fired against DDL statements like Drop Table, Create Table, Or Alter Table, DDL Triggers are always after Triggres.

3) Logon trigger: This type of trigger is fired against a LOGON event before a user session is established to the SQL Server.

2. **What are the uses of triggers?**

Ans. Triggers supplement the standard capabilities of Oracle to provide a highly customized database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours. You can also use triggers to:

- Automatically generate derived column values
- Prevent invalid transactions
- Enforce complex security authorizations
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access
- Modify table data when DML statements are issued against views
- Publish information about database events, user events, and SQL statements to subscribing applications