

**ASSIGNMENT NO: 2**

Name: Sourav Shailesh Toshniwal

Class: TY CSE-8 AIEC-1

Roll no.: 2213047

Batch: A

**AIM : Data Definition Commands for creating database and tables (relations)**

**Create a Table for Manufacturing industry / Hospital/ Company with min 5 columns add primary key.**

**Alter any one column from the above table.**

**Rename two columns from the above table**

**Truncate the table**

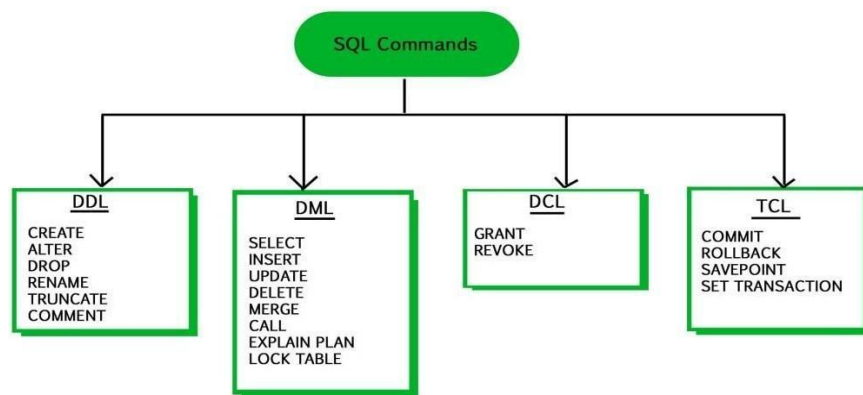
**Drop the table.**

**INDEX TERMS:** DDL,DML, PL/SQL Stored Procedure, functions

**THEORY**

**Different types of commands in SQL:**

- A). DDL commands: - To create a database objects
- B). DML commands: - To manipulate data of a database objects.
- C). DQL command: - To retrieve the data from a database.
- D). DCL command - To control the data of a database.

**DDL commands:**

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The schema for each

relation. The domain of values associated with each attribute. Integrity constraints And as we will see later, also other information such as The set of indices to be maintained for each relations.

These commands are:

- CREATE**: to create a new data structure.
- ALTER**: to change an existing data structure.
- DROP**: to remove an entire data structure.
- TRUNCATE** : to remove all rows from the table.
- RENAME** : to rename existing table

## Data Manipulation Language

The Data Manipulation Language (DML) is used to insert and modify database information.

Different DML statements includes:

- INSERT: to add records into the table
- UPDATE: to change column value in the table
- DELETE: to remove rows from the table

### Create Table

create table *student* (

*ID* varchar(5), *name* varchar(20) not null, *dept\_name* varchar(20), *tot\_cred* numeric(3,0),  
primary key (*ID*), foreign key (*dept\_name*) references *department*);

### Insert

insert into *instructor* values ('10211', 'Smith', 'Biology', 66000);

### Delete

Remove all tuples from the *student* relation : delete from *student*;

### Select Operation

List in alphabetic order the names of all instructors

select distinct *name* from *instructor* order by *name*;

### Aggregate Functions

#### Nested Sub Query

Definition of Nested Subquery:-

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within

Sr. No.	Aggregate Function	Use	Example
1.	COUNT ()	It is used to count number of records available.	Select count(*) from student; Select count(roll_no) from student;
2.	MAX ()	It returns Maximum value	Select max(salary) from employee;
3.	MIN ()	It returns Minimum value	Select min(salary) from employee;
4.	AVG ()	It returns Average value	Select avg(salary) from employee;
5.	SUM ()	It returns sum of all value	Select sum(salary) from employee;

the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.

### Sub queries with the SELECT Statement:

Sub queries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM table1 [, table2]
WHERE column_name OPERATOR
      (SELECT column_name [, column_name ]
      FROM table1 [, table2]
      [WHERE])
```

### Example of IN Operator

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, let us check following sub query with SELECT statement:

```
SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS
WHERE SALARY > 4500);
```

This would produce the following result

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

### Example of NOT IN Operator

Now, let us check following sub query with SELECT statement:

```
SELECT * FROM CUSTOMERS WHERE ID NOT IN (SELECT ID FROM CUSTOMERS
WHERE SALARY > 4500);
```

This would produce the following result

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
6	Komal	22	MP	4500.00

### Joins:

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol. There are different types of joins available in SQL

#### a) Inner Join:

The most important and frequently used of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax: `SELECT table1.column1, table2.column2...FROM table1 INNER JOIN table2 ON table1.common_field=table2.common_field;`

#### b) Left Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table. This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax: `SELECT table1.column1, table2.column2...FROM table1 LEFT JOIN table2 ON table1.common_field=table2.common_field;`

#### c) Right Join:

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table. This means that

a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax: `SELECT table1.column1, table2.column2...FROM table1 RIGHTJOIN table2 ONtable1.common_field=table2.common_field;`

#### **d) Full Join:**

The SQL FULL JOIN combines the results of both left and right outer joins. The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side. MySQL does not support full join so we have to use the set operation 'Union All' for left outer join and the right outer join on those two tables. Syntax: `<Left Outer Join Query> Union All <Right Outer Join Query>` e)Self Join: The SQL 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. Here, the WHERE clause could be any given expression based on your requirement.

Syntax: `SELECT a.column_name, b.column_name... FROM table1a, table1b WHERE a.common_field = b.common_field;`

#### **f) Cartesian or Cross Join:**

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement. Syntax:`SELECTtable1.column1, table2.column2...FROM table1,table2[, table3]`

### ***PL/SQL provides two kinds of subprograms-***

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

A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

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

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

Executing a Standalone Procedure A stand alone procedure can be called in two ways

- 1) Using the EXECUTE keyword
- 2) Calling the name of the procedure from a PL/SQL block

**Function :** A function is same as a procedure except that it returns a value.

#### **Creating a Function**

A standalone function is created using the CREATE FUNCTION statement. The simplified

syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

```
CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN|OUT |INOUT]
type[, ...])] RETURN return_datatype {IS |AS}
BEGIN <function_body>
END [function_name];
```

Where, function-name specifies the name of the function. [OR REPLACE] option allows the modification of an existing function. 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. Calling a 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. A called function performs the defined task and when its return statement is executed or when the last end statement is reached, it returns the program control back to the main program.

### Exercise :

Evaluate query for following questions and perform operations **Classicmodels** database.

- 1) Report those payments greater than 100000;
- 2) Report total payments for October 28, 2004;
- 3) What is the minimum payment received?
- 4) How many employees are there in the company?
- 5) List the product lines that contain 'Cars'
- 6) How many products in each product line? (stored procedure)
- 7) What are the names of executives with VP or Manager in their title?
- 8) What is the average percentage markup of the MSRP on buyPrice?
- 9) Which orders have a value greater than \$5000?
- 10) List the value of 'On Hold' orders
- 11) Report the number of orders 'On Hold' for each customer.
- 12) Who are the employees in Boston?
- 13) Report the products that have not been sold
- 14) Compute the commission for each sales representative, assuming the commission is 5% of the value of an order. sort by employee last name and first name

### 2. Data Definition Commands for creating database and tables (relations)

- A. Create a Table for Manufacturing industry / Hospital/ Company with min 5 columns add primary key.
- B. Alter any one column from the above table.
- C. Rename two columns from the above table
- D. Truncate the table
- E. Drop the table.

Code:

use TYAIEC;

```
create table hospital (Hospital_registration_no varchar(20), Hospital_name varchar(50),
```



```
Hospital_address varchar(70));
```

```
alter table hospital modify Hospital_registration_no varchar(13);
```

```
insert into hospital values ('SAFNVS@WEK1D2', 'Joshi Hospital', 'Opposite to Ambikanagar Bus stop, Kedgaon, Ahmednagar');
```

```
insert into hospital values ('ABCNVS@WEK1D2', 'Pandit Hospital', 'Opposite to Pune bus stand, Ahmednagar');
```

```
alter table hospital rename column Hospital_registration_no to Hospital_Reg_no;
```

```
alter table hospital rename column Hospital_address to Hospital_location;
```

```
update hospital set Hospital_name='Ambika Nursing Home' where Hospital_name='Joshi Hospital';
```

```
truncate table hospital;
```

```
drop table hospital;
```

```
select * from hospital;
```

Output:

	Hospital_Reg_no	Hospital_name	Hospital_location
▶	SAFNVS@WEK1D2	Ambika Nursing Home	Opposite to Ambikanagar Bus stop, Kedgaon, A..
	ABCNVS@WEK1D2	Pandit Hospital	Opposite to Pune bus stand, Ahmednagar

### FAQs:

1. What do you understand by a subquery? When is it used?

Ans: A subquery, also known as a nested query or inner query, is a query that is embedded or nested within another SQL query. It is used to retrieve data that will be used by the main (outer) query to perform further operations, filtering, or comparisons. Subqueries are an essential component of SQL and are used for various purposes, including:

1. Filtering Data: Subqueries can be used to filter the results of the outer query based on conditions or criteria that involve data from another table. For example, you can use a subquery to retrieve a list of customers who have placed orders in the past month.

2. Data Retrieval: Subqueries can retrieve specific pieces of data that are needed for the main query. For instance, you can use a subquery to find the highest or lowest value in a column within a table.

3. Comparison: Subqueries are often used to compare values in the outer query to values returned by the subquery. For example, you can use a subquery to check if a given value exists in another table or to compare a value with the maximum value in another table.

4. Aggregation: Subqueries can perform aggregate functions such as counting, summing, or averaging data in a related table, and the results can be used in the main query. For instance, you can use a subquery to calculate the average order value for a specific customer.

5. Testing Conditions: Subqueries can be used to test conditions and return a Boolean result that is used in the main query's WHERE clause. This is useful for complex conditional logic.

6. Correlated Subqueries: In some cases, a subquery can reference columns from the outer query, creating a correlated subquery. Correlated subqueries are used when the result of the subquery depends on the values of the outer query. For example, finding all employees whose salaries are greater than the average salary in their department.

Here's a simple example of a subquery:

```
SELECT customer_name
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM orders
    WHERE order_date >= '2023-08-01'
);
```

In this example, the subquery retrieves a list of customer IDs who placed orders in August 2023, and the outer query retrieves the names of those customers based on their IDs.

2. What is difference between FUNCTION and PROCEDURE in PL/SQL

Ans: In PL/SQL (Procedural Language/Structured Query Language), both functions and procedures are database objects that allow you to encapsulate a series of SQL and PL/SQL statements into a single named unit. However, they have distinct differences in terms of their purposes and how they are used:

Procedure:

1. Purpose: Procedures are primarily used for performing an action or a series of actions. They are often used for tasks such as data manipulation, data validation, or any other database operation that doesn't necessarily return a value.

2. Return Type: Procedures do not return a value explicitly. They are used to perform actions or operations and may modify the data or perform other tasks within the database.

3. Usage: You can call a procedure from within another PL/SQL block, like another procedure, function, or anonymous block. Procedures are used for their side effects or actions, such as updating data or sending emails.

4. Example:

```
``psql
CREATE OR REPLACE PROCEDURE UpdateEmployeeSalary (
    emp_id NUMBER,
    new_salary NUMBER
) AS
BEGIN
    -- SQL statements to update the salary for the employee
    UPDATE employees
```

```
SET salary = new_salary
WHERE employee_id = emp_id;
END;
'''
```

Function:

1. Purpose: Functions are used to return a single value. They are typically used for calculations, data retrieval, or any operation where you need a result returned to the calling program.
2. Return Type: Functions have a return type, and they must return a value of that type using the 'RETURN' keyword. The return value can be used in expressions or assigned to variables.
3. Usage: Functions can be called from SQL statements (e.g., SELECT statements) or within PL/SQL blocks. They are often used to compute and return values based on input parameters.
4. Example:

```
CREATE OR REPLACE FUNCTION CalculateTotalSalary (
    emp_id NUMBER
) RETURN NUMBER AS
    total_salary NUMBER;
BEGIN
    -- SQL statements to calculate and return the total salary
    SELECT SUM(salary) INTO total_salary
    FROM employees
    WHERE department_id = emp_id;

    RETURN total_salary;
END;
```

In summary, the main difference between a function and a procedure in PL/SQL is that functions return a value while procedures do not. Functions are used for computations and data retrieval, whereas procedures are used for performing actions and operations. Depending on your specific requirements, you would choose either a function or a procedure to meet your programming needs.

3. What is the difference between cross joins and natural joins?

Ans:

SR.NO.	NATURAL JOIN	CROSS JOIN
1.	Natural Join joins two tables based on same attribute name and datatypes.	Cross Join will produce cross or cartesian product of two tables .
2.	In Natural Join, The resulting table will contain all the attributes of both the tables but keep only one copy of each common column	In Cross Join, The resulting table will contain all the attribute of both the tables including duplicate columns also
3.	In Natural Join, If there is no condition specifies then it returns the rows based on the common column	In Cross Join, If there is no condition specifies then it returns all possible pairing of rows from both the tables whether they are matched or unmatched
4.	SYNTAX: SELECT * FROM table1 NATURAL JOIN table2;	SYNTAX: SELECT * FROM table1 CROSS JOIN table2;

**Conclusion:**

Out come of the experiment is understanding of

1. Implement DDL,DML Statements, aggregate functions, all types of Joins.
2. Implement Sub-Queries, PLSQL Stored procedure & Functions