RELATIONAL DATA BASE MANAGEMENT SYSTEM

Introduction of RDBMS

The relational database was first defined in June 1970 by Edgar Codd, of IBM's San Jose Research Laboratory. Codd's view of what qualifies as an RDBMS is summarized in Codd's 12 rules. A relational database has become the predominant type of database.

A relational database is a type of database that stores and provides access to data points that are related to one another.

Use of RDBMS

The use of an RDBMS can be beneficial to most organizations; the systematic view of raw data helps companies better understand and execute the information while enhancing the decision-making process. The use of tables to store data also improves the security of information stored in the databases.

Advantages of RDBMS

- RDBMS is based on rows and columns (table). So it is easier to understand RDBMS work.
- RDBMS supports more than one user.
- As it is advanced it can easily handle huge amounts of data.
- Security is pretty good for RDBMS.

Most significant advantage of RDBMS

Flexibility: An RDBMS is flexible in allowing users to change or store data in the database. This is helpful when users want to update the information in the stored data set. Simplicity: Relational databases are one of the simplest models to store and retrieve large data items in an organized way.

Codd's 12 Rules

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.t c

What is a Primary Key

A Primary Key is the minimal set of attributes of a table that has the task to uniquely identify the rows, or we can say the tuples of the given particular table.

A primary key of a relation is one of the possible candidate keys which the database designer thinks it's primary. It may be selected for convenience, performance and many other reasons. The choice of the possible primary key from the candidate keys depend upon the following conditions.ackward Skip 10sPlay Video Forward Skip 10s

- **Minimal:** The primary key should be composed of the minimum number of attributes that satisfy unique occurrences of the tuples. So if one candidate key is formed using two attributes and another using a single attribute then the one with the single attribute key should be chosen as the primary key.
- **Accessible:** The primary key used should be accessible by anyone who wants to use it. The user must easily insert, access or delete a tuple using it.
- **NON NULL Value:** The primary key must have a non-null value for each tuple of the relation, which is required for the identification of the tuple.
- **Time Variant:** The values of the primary key must not change or become null during the time of a relation.
- **Unique:** The value of the primary key must not be duplicated in any of the tuples of a relation.

Syntax for creating primary key constraint:

The primary key constraint can be defined at the column level or table level.

At column level:

1. **<column_name><datatype>** Primary key;

At table level:

1. Primary key(**<column_name1**>[,column_name>]....);

Properties of a Primary Key:

- A relation can contain only one primary key.
- A primary key may be composed of a single attribute known as single primary key or more than one attribute known as composite key.
- A primary key is the minimum super key.
- The data values for the primary key attribute should not be null.
- Attributes which are part of a primary key are known as Prime attributes.
- Primary key is always chosen from the possible candidate keys.
- If the primary key is made of more than one attribute then those attributes are irreducible.
- We use the convention that the attribute that form primary key of relation is underlined.
- Primary key cannot contain duplicate values.
- Columns that are defined as LONG or LONG RAW cannot be part of a primary key.

Use of Primary Key

As defined above, a primary key is used to uniquely identify the rows of a table. Thus, a row that needs to be uniquely identified, the key constraint is set as the Primary key to that particular field. A primary key can never have a **NULL** value because the use of the primary key is to identify a value uniquely, but if no value will be there, how could it sustain. Thus, the field set with the primary key constraint cannot be NULL. Also, it all depends on the user that the user can add or delete the key if applied.

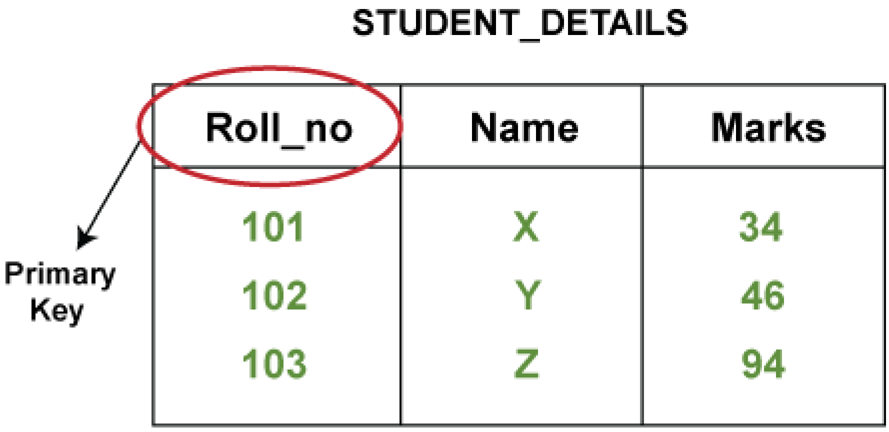
Understanding Primary Key

Let's discover some examples through which we can understand the role and use of a Primary key. Generally, in a database, we apply the primary key on those tuples or columns through which we need to uniquely identify the other database fields.

For example: When we store the registration details of the students in the database, we find the registration number field unique and assign the primary key to the field. Also, for an employee table, we set the primary key on the employee Id of the table.ertified by completing the course

Let's understand it practically:

Below is the table named **STUDENT_DETAILS**, where Roll_no, Name, and Marks are the specified attributes of it.



STUDENT_DETAILS

Roll_no	Name	Marks
101	X	34
102	Y	46
103	Z	94

Primary
Key

As we know that from these three attributes, the Roll_no attribute is the one that can uniquely identify other two attributes of the table as each student is provided with a unique roll number in every organization. So, we can set the primary key constraint on the Roll_no column.

What is a Foreign Key

A foreign key is the one that is used to link two tables together via the primary key. It means the columns of one table points to the primary key attribute of the other table. It further means that if any attribute is set as a primary key attribute will work in another table as a foreign key attribute. But one should know that a foreign key has nothing to do with the primary key.

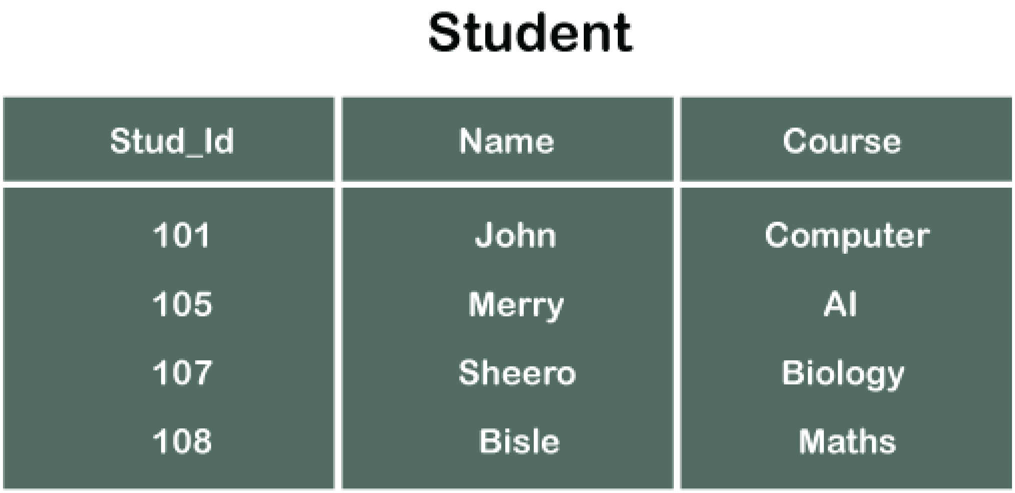
Use of Foreign Key

The use of a foreign key is simply to link the attributes of two tables together with the help of a primary key attribute. Thus, it is used for creating and maintaining the relationship between the two relations.

Example of Foreign Key

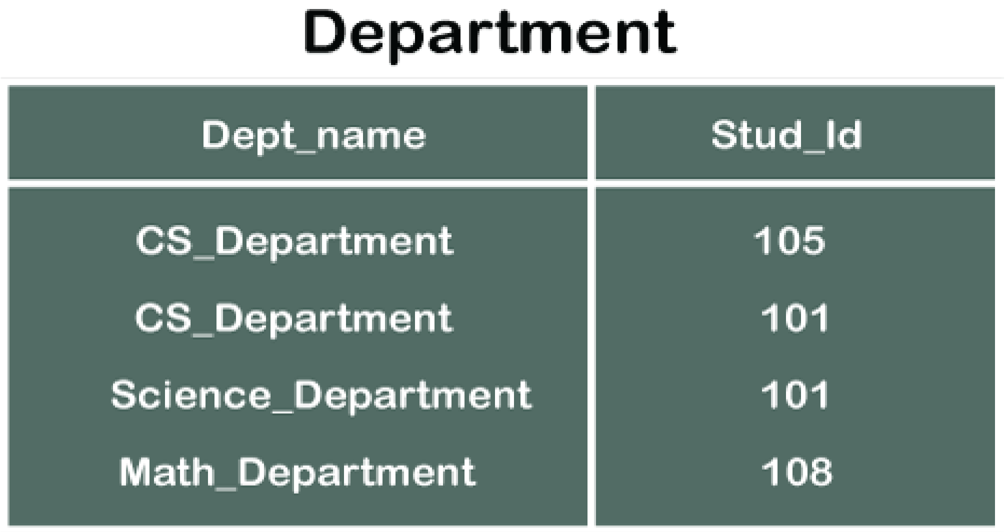
Let's discuss an example to understand the working of a foreign key.

Consider two tables **Student** and **Department** having their respective attributes as shown in the below table structure:



Student

Stud_Id	Name	Course
101	John	Computer
105	Merry	AI
107	Sheero	Biology
108	Bisle	Maths

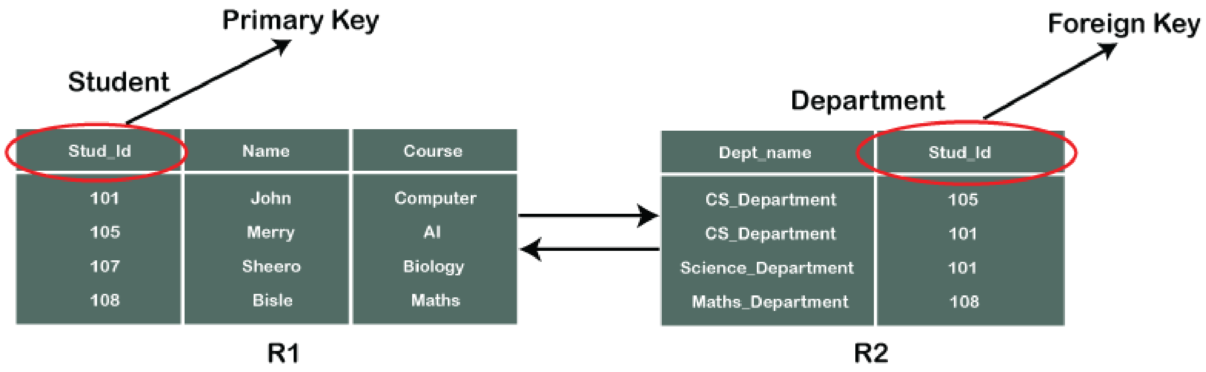


Department

Dept_name	Stud_Id
CS_Department	105
CS_Department	101
Science_Department	101
Math_Department	108

In the tables, one attribute, you can see, is common, that is **Stud_Id**, but it has different key constraints for both tables. In the Student table, the field Stud_Id is a **primary key** because it is uniquely identifying all other fields of the Student table. On the other hand, Stud_Id is a **foreign key** attribute for the Department table because it is acting as a primary key attribute for the Student table. It means that both the Student and Department table are linked with one another because of the Stud_Id attribute.

In the below-shown figure, you can view the following structure of the relationship between the two tables.



Stud_Id	Name	Course
101	John	Computer
105	Merry	AI
107	Sheero	Biology
108	Bisle	Maths

Dept_name	Stud_Id
CS_Department	105
CS_Department	101
Science_Department	101
Maths_Department	108

What is a Candidate Key

A candidate key is a subset of a super key set where the key which contains no redundant attribute is none other than a **Candidate Key**. In order to select the candidate keys from the set of super key, we need to look at the super key set.

Role of a Candidate Key

The role of a candidate key is to identify a table row or column uniquely. Also, the value of a candidate key cannot be Null. The description of a candidate key is "no redundant attributes" and being a "minimal representation of a tuple," according to the Experts.

How a Candidate key is different from a Primary Key

Although the purpose of both candidate and the primary key is the same, that is to uniquely identify the tuples, and then also they are different from each other. It is because, in a table, we can have one or more than one candidate key, but we can create only one primary key for a table. Thus, from the number of obtained candidate keys, we can identify the appropriate primary key. However, if there is only one candidate key in a table, then it can be considered for both key constraints.

Example of Candidate Key

Let's look at the same example took while discussing Super Key to understand the working of a candidate key.

We have an **EMPLOYEE_DETAIL** table where we have the following attributes:

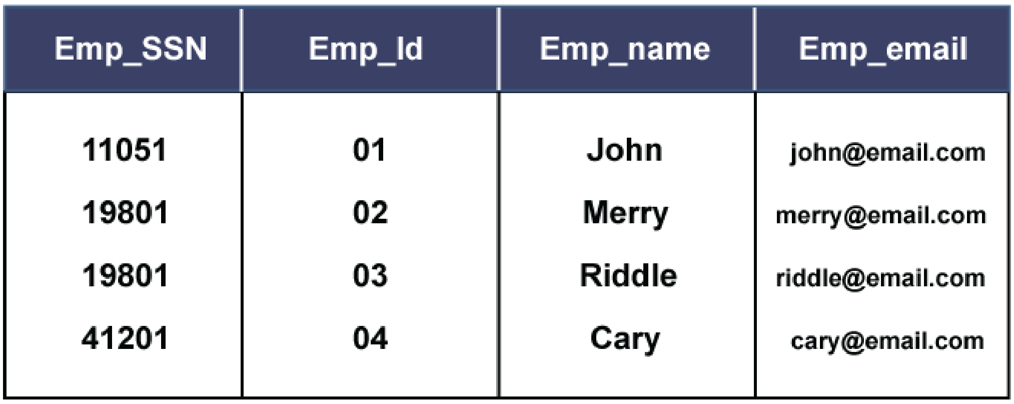
Emp_SSN: The SSN number is stored in this field.

Emp_Id: An attribute that stores the value of the employee identification number.

Emp_name: An attribute that stores the name of the employee holding the specified employee id.

Emp_email: An attribute that stores the email id of the specified employees.

The **EMPLOYEE_DETAIL** table is given below that will help you understand better:



Emp_SSN	Emp_Id	Emp_name	Emp_email
11051	01	John	john@email.com
19801	02	Merry	merry@email.com
19801	03	Riddle	riddle@email.com
41201	04	Cary	cary@email.com

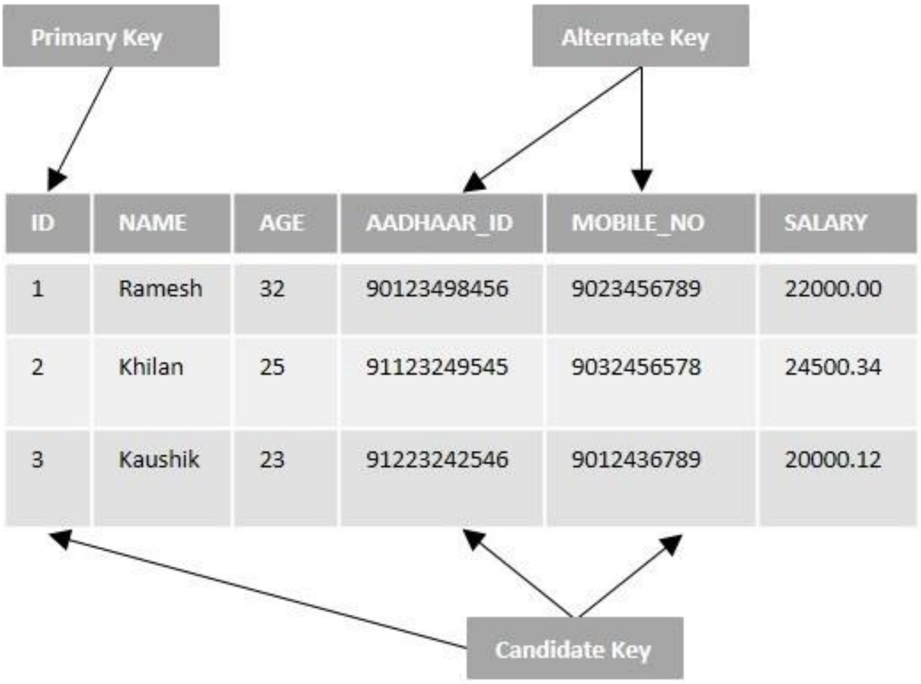
Alternate Key

The SQL Alternate Key

SQL **Alternate Keys** in a database table are candidate keys that are not currently selected as a primary key. They can be used to uniquely identify a tuple(or a record) in a table.

There is no specific query or syntax to set the alternate key in a table. It is just a column that is a close second candidate which could be selected as a primary key. Hence, they are also called secondary candidate keys.

If a database table consists of only one candidate key, that is treated as the primary key of the table, then there is no alternate key in that table.



ID	NAME	AGE	AADHAAR_ID	MOBILE_NO	SALARY
1	Ramesh	32	90123498456	9023456789	22000.00
2	Khilan	25	91123249545	9032456578	24500.34
3	Kaushik	23	91223242546	9012436789	20000.12

The details like id, mobile number and aadhaar number of a customer are unique, and we can identify the records from the CUSTOMERS table uniquely using their respective fields; ID, AADHAAR_ID and MOBILE_NO. Therefore, these three fields can be treated as candidate keys.

And among them, if one is declared as the primary key of the CUSTOMERS table then the remaining two would be alternate keys.

Features of Alternate Keys

Following are some important properties/features of alternate keys −

- The alternate key does not allow duplicate values.
- A table can have more than one alternate keys.
- The alternate key can contain NULL values unless the NOT NULL constraint is set explicitly.
- All alternate keys can be candidate keys, but all candidate keys can not be alternate keys.
- The primary key, which is also a candidate key, can not be considered as an alternate key.

Data Definition Language

Data Definition Language(DDL) is a subset of SQL and a part of DBMS(Database Management System). DDL consist of Commands to commands like CREATE, ALTER, TRUNCATE and DROP. These commands are used to create or modify the tables in SQL.

The five DDL commands in SQL:

- CREATE Command.
- DROP Command.
- ALTER Command.
- TRUNCATE Command.
- RENAME Command.

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER:** This is used to alter the structure of the database.
- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**: This is used to add comments to the data dictionary.
- **RENAME:** This is used to rename an object existing in the database.

DQL (Data Query Language)

CREATE Command

Create is a DDL SQL command used to create a table or a database in relational database management system.

Creating a Table

Create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the names and datatypes of various columns in the create command itself.

Following is the syntax,

CREATE TABLE <TABLE_NAME>

```
(    column_name1 datatype1,

    column_name2 datatype2,

    column_name3 datatype3,

    column_name4 datatype4

);
```

Create table command will tell the database system to create a new table with the given table name and column information.

Example for creating Table

```
CREATE TABLE Student(

    student_id INT,

    name VARCHAR(100),

    age INT);
```

The above command will create a new table with name **Student** in the current database with 3 columns, namely student_id, name and age. Where the column student_id will only store integer, name will hold upto 100 characters and age will again store only integer value.If you are currently not logged into your database in which you want to create the table then you can also add the database name along with table name, using a dot operator .For example, if we have a database with name **Test** and we want to create a table **Student** in it, then we can do so using the following query:

```
CREATE TABLE Test.Student(
    student_id INT,
    name VARCHAR(100),
    age INT);
```

Most commonly used datatypes for Table columns

Here we have listed some of the most commonly used datatypes used for columns in tables.

Datatype	**Use**
INT	used for columns which will store integer values.

FLOAT	used for columns which will store float values.
DOUBLE	used for columns which will store float values.
VARCHAR	columns which will be used to store characters and integers, basically a string.
CHAR	used for columns which will store char values(single character).
DATE	columns which will store date values.
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for **about me** section you can have a column of type TEXT.

Alter commands

The **ALTER TABLE statement in SQL** is used to add, remove, or modify columns in an existing table. The ALTER TABLE statement is also used to add and remove various constraints on existing tables.

ALTER TABLE ADD Column Statement in SQL

ADD is used to add columns to the existing table. Sometimes we may require to add additional information, in that case, we do not require to create the whole database again, **ADD** comes to our rescue.

ALTER TABLE ADD Column Statement Syntax:

ALTER TABLE table_name ADD (Columnname_1 datatype,
Columnname_2 datatype, …Columnname_n datatype);

The following SQL adds an “Email” column to the “Students” table:

ALTER TABLE ADD Column Statement Example:

ALTER TABLE Students

ADD Email varchar(25);

ALTER TABLE DROP Column Statement

DROP COLUMN is used to drop columns in a table. Deleting the unwanted columns from the table.

ALTER TABLE DROP Column Statement Syntax:

ALTER TABLE table_name DROP COLUMN column_name;
The following SQL drop an "Email" column to the "Students" table:

ALTER TABLE DROP Column Statement Example:

ALTER TABLE StudentsDROP COLUMN Email;

ALTER TABLE MODIFY Column Statement in SQL

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

ALTER TABLE MODIFY Column Statement Syntax:

ALTER TABLE table_name
MODIFY column_name column_type;

ALTER TABLE MODIFY Column Statement Example:

ALTER TABLE table_name MODIFY COLUMN column_name datatype;

SQL ALTER TABLE Queries

Suppose there is a student database:

ROLL_NO	NAME
1	Ram
2	Abhi
3	Rahul
4	Tanu

To ADD 2 columns AGE and COURSE to table Student.

Query: ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));

Output:

ROLL_NO	NAME	AGE	COURSE
1	Ram		
2	Abhi		
3	Rahul		
4	Tanu		

MODIFY column COURSE in table Student.

Query: ALTER TABLE Student MODIFY COURSE varchar(20);

After running the above query the maximum size of the Course Column is reduced to 20 from 40.

DROP column COURSE in table Student.

Query: ALTER TABLE Student DROP COLUMN COURSE;
Output:

ROLL_NO	NAME	AGE
1	Ram	
2	Abhi	
3	Rahul	
4	Tanu	

SQL commands are broadly classified into two types DDL, DML here we will be learning about DDL commands, and in DDL we will be learning about DROP and TRUNCATE in this article.
DDL Stands for Data Definition Language with the help of this DDL command we can add, remove, or modify tables within a database. Here we are to discuss DROP and TRUNCATE Commands So we are going to start with the DROP command first.

DROP

DROP is used to delete a whole database or just a table.

In this article, we will be learning about the DROP statement which destroys objects like an existing database, table, index, or view. A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Syntax

DROP object object_name ;

Case 1: To Drop a table

DROP TABLE table_name;

table_name: Name of the table to be deleted.

Case 2: To Drop a database

Syntax:

DROP DATABASE database_name;

database_name: Name of the database to be deleted.

TRUNCATE

The major difference between TRUNCATE and DROP is that truncate is used to delete the data inside the table not the whole table.

TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extent of a table for deallocation (empty for reuse). The result of this operation quickly removes all data from a table, typically bypassing several integrity-enforcing mechanisms. It was officially introduced in the SQL:2008 standard. The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).

Syntax

TRUNCATE TABLE table_name;

table_name: Name of the table to be truncated.

DATABASE name – student_data

Differences Between DROP and TRUNCATE

DROP	TRUNCATE
In the drop table data and its definition is deleted with their full structure.	It preserves the structure of the table for further use exist but deletes all the data.
Drop is used to eliminate existing complications and fewer complications in the whole database from the table.	Truncate is used to eliminate the tuples from the table.
Integrity constraints get removed in the DROP command.	Integrity constraint doesn't get removed in the Truncate command.

DROP	TRUNCATE
Since the structure does not exist, the View of the table does not exist in the Drop command.	Since the structure exists, the View of the table exists in the Truncate command.
Drop query frees the table space complications from memory.	This query does not free the table space from memory.
It is slow as there are so many complications compared to the TRUNCATE command.	It is fast as compared to the DROP command as there are fewer complications.

let’s consider the given database Student _data:

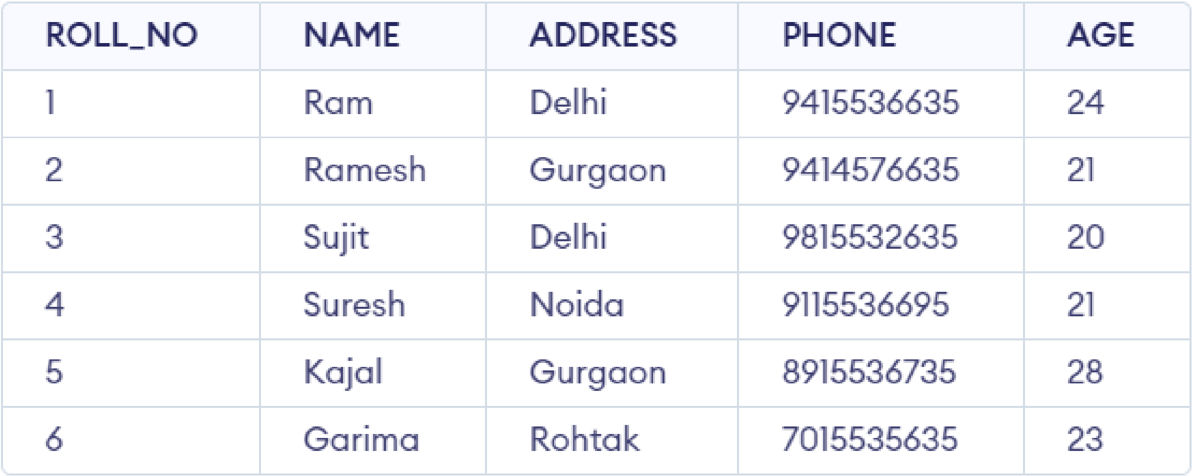
Query

```
CREATE TABLE Student _details (
ROLL_NO INT,
NAME VARCHAR(25),
ADDRESS VARCHAR(25),
PHONE INT ,
AGE INT); --
```

Inserting the data in Student Table

```
INSERT INTO Student _details(ROLL_NO,NAME,ADDRESS,PHONE,AGE) VALUES
(1,'Ram','Delhi',9415536635,24),
(2,'Ramesh','Gurgaon',9414576635,21),
(3,'Sujit','Delhi',9815532635,20),
(4,'Suresh','Noida',9115536695,21),
(5,'Kajal','Gurgaon',8915536735,28),
(6,'Garima','Rohtak',7015535635,23);
```

Output



ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Ram	Delhi	9415536635	24
2	Ramesh	Gurgaon	9414576635	21
3	Sujit	Delhi	9815532635	20
4	Suresh	Noida	9115536695	21
5	Kajal	Gurgaon	8915536735	28
6	Garima	Rohtak	7015535635	23

Student Table

Example1:
To delete the whole database
Query:
DROP DATABASE student_data;
After running the above query whole database will be deleted.

Example2:
To delete the whole table from the Database
Query:
DROP TABLE student_details;
After running the above query whole table from the database will be deleted.

Example3:
To truncate the Student_details table from the student_data database.
Query:
TRUNCATE TABLE Student_details;
After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

SQL DESC Statement (Describe Table)

SQL DESC statement use for describe the list of column definitions for specified table. You can use either DESC or DESCRIBE statement. both are return same result.

DESCRIBE statement to get following information:

- Column Name
- Column allow NULL or NOT NULL
- Datatype of the Column
- With database size precision and If NUMERIC datatype scale.

SQL DESC Syntax

SQL DESCRIBE Table Column use following syntax,

DESC table_name

SQL DESC Example

SQL> DESC users_info;

```
Name                    Null?                 Type
---------------------------- -------- ---------------------------


NO                     NOT NULL              NUMBER(3)
```

```
NAME                                    VARCHAR2(30)
ADDRESS                                 VARCHAR2(70)
CONTACT_NO                              VARCHAR2(12)
```

SQL DESCRIBE Syntax

```
DESCRIBE table_name
```

SQL DESCRIBE Syntax

```
SQL> DESCRIBE users_info;

Name                    Null?          Type
----------------------------- -------- ---------------------


NO                     NOT NULL       NUMBER(3)
NAME                                  VARCHAR2(30)
ADDRESS                               VARCHAR2(70)
CONTACT_NO                            VARCHAR2(12)
```

DML Commands in SQL

DML is an abbreviation of **Data Manipulation Language**.

The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.

Following are the four main DML commands in SQL:

1. SELECT Command
2. INSERT Command
3. UPDATE Command
4. DELETE Command

SELECT DML Command

SELECT is the most important data manipulation command in Structured Query Language. The SELECT command shows the records of the specified table. It also shows the particular record of a particular column by using the WHERE clause.

Syntax of SELECT DML command

1. **SELECT** column_Name_1, column_Name_2, ….., column_Name_N **FROM** Name_of_table;

Here, **column_Name_1, column_Name_2, ….., column_Name_N** are the names of those columns whose data we want to retrieve from the table.

If we want to retrieve the data from all the columns of the table, we have to use the following SELECT command:

```
SELECT * FROM table_name;
```

Examples of SELECT Command

Example 1: This example shows all the values of every column from the table.

```
SELECT * FROM Student;
```

This SQL statement displays the following values of the student table:

Student_ID	**Student_Name**	**Student_Marks**
Bcom1001	Abhay	85
Bcom1002	Anuj	75
Bcom1003	Bheem	60
Bcom1004	Ram	79
Bcom1005	Sumit	80

Example 2: This example shows all the values of a specific column from the table.

```
SELECT Emp_Id, Emp_Salary FROM Employee;
```

This SELECT statement displays all the values of **Emp_Salary** and **Emp_Id** column of **Employee** table:

Emp_Id	**Emp_Salary**
201	25000
202	45000

203	30000
204	29000
205	40000

Example 3: This example describes how to use the WHERE clause with the SELECT DML command.

Let's take the following Student table:

Student_ID	Student_Name	Student_Marks
Bcom1001	Abhay	80
Bcom1002	Ankit	75
Bcom1003	Bheem	80
Bcom1004	Ram	79
Bcom1005	Sumit	80

If you want to access all the records of those students whose marks is 80 from the above table, then you have to write the following DML command in SQL:

```
SELECT * FROM Student WHERE Stu_Marks = 80;
```

The above SQL query shows the following table in result:

Student_ID	Student_Name	Student_Marks
Bcom1001	Abhay	80
Bcom1003	Bheem	80

Bcom1005	Sumit	80

INSERT DML Command

INSERT is another most important data manipulation command in Structured Query Language, which allows users to insert data in database tables.

Syntax of INSERT Command

1. **INSERT INTO** TABLE_NAME (column_Name1 , column_Name2 , column_Name3 , ... column_NameN) **VALUES** (value_1, value_2, value_3, ... value_N) ;

Examples of INSERT Command

Example 1: This example describes how to insert the record in the database table.

Let's take the following student table, which consists of only 2 records of the student.

Stu_Id	Stu_Name	Stu_Marks	Stu_Age
101	Ramesh	92	20
201	Jatin	83	19

Suppose, you want to insert a new record into the student table. For this, you have to write the following DML INSERT command:

1. **INSERT INTO** Student (Stu_id, Stu_Name, Stu_Marks, Stu_Age) **VALUES** (104, Anmol, 89, 19);

UPDATE DML Command

UPDATE is another most important data manipulation command in Structured Query Language, which allows users to update or modify the existing data in database tables.

Syntax of UPDATE Command

1. **UPDATE** Table_name **SET** [column_name1= value_1, ….., column_nameN = value_N] **WHERE** CONDITION;

Here, 'UPDATE', 'SET', and 'WHERE' are the SQL keywords, and 'Table_name' is the name of the table whose values you want to update.

Examples of the UPDATE command

Product_Id	Product_Name	Product_Price	Product_Quantity
P101	Chips	20	20
P102	Chocolates	60	40
P103	Maggi	75	5
P201	Biscuits	80	20
P203	Namkeen	40	50

Example 1: This example describes how to update the value of a single field.

Let's take a Product table consisting of the following records:

Suppose, you want to update the Product_Price of the product whose Product_Id is P102. To do this, you have to write the following DML UPDATE command:

1. **UPDATE** Product **SET** Product_Price = 80 **WHERE** Product_Id = 'P102' ;

Example 2: This example describes how to update the value of multiple fields of the database table.

Let's take a Student table consisting of the following records:

Stu_Id	Stu_Name	Stu_Marks	Stu_Age
101	Ramesh	92	20
201	Jatin	83	19
202	Anuj	85	19
203	Monty	95	21

102	Saket	65	21
103	Sumit	78	19
104	Ashish	98	20

Suppose, you want to update Stu_Marks and Stu_Age of that student whose Stu_Id is 103 and 202. To do this, you have to write the following DML Update command:

1. **UPDATE** Student **SET** Stu_Marks = 80, Stu_Age = 21 **WHERE** Stu_Id = 103 AND Stu_Id = 202;

DELETE DML Command

DELETE is a DML command which allows SQL users to remove single or multiple existing records from the database tables.

This command of Data Manipulation Language does not delete the stored data permanently from the database. We use the WHERE clause with the DELETE command to select specific rows from the table.

Syntax of DELETE Command

1. **DELETE FROM** Table_Name **WHERE** condition;

Examples of DELETE Command

Example 1: This example describes how to delete a single record from the table.

Let's take a Product table consisting of the following records:

Product_Id	Product_Name	Product_Price	Product_Quantity
P101	Chips	20	20
P102	Chocolates	60	40
P103	Maggi	75	5
P201	Biscuits	80	20

P203	Namkeen	40	50

Suppose, you want to delete that product from the Product table whose Product_Id is P203. To do this, you have to write the following DML DELETE command:

1. **DELETE FROM** Product **WHERE** Product_Id = 'P202' ;

Example 2: This example describes how to delete the multiple records or rows from the database table.

Let's take a Student table consisting of the following records:

Stu_Id	Stu_Name	Stu_Marks	Stu_Age
101	Ramesh	92	20
201	Jatin	83	19
202	Anuj	85	19
203	Monty	95	21
102	Saket	65	21
103	Sumit	78	19
104	Ashish	98	20

Suppose, you want to delete the record of those students whose Marks is greater than 70. To do this, you have to write the following DML Update command:

DELETE FROM Student **WHERE** Stu_Marks > 70 ;

DCL (Data Control Language)

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

GRANT: This command gives users access privileges to the database.

Syntax:

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

REVOKE: This command withdraws the user's access privileges given by using the GRANT command.

Syntax:

```
REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
```

TCL (Transaction Control Language)

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions **here**. Hence, the following TCL commands are used to control the execution of a transaction:

BEGIN: Opens a Transaction.

COMMIT: Commits a Transaction.

Syntax:

```
COMMIT;
```

ROLLBACK: Rollbacks a transaction in case of any error occurs.

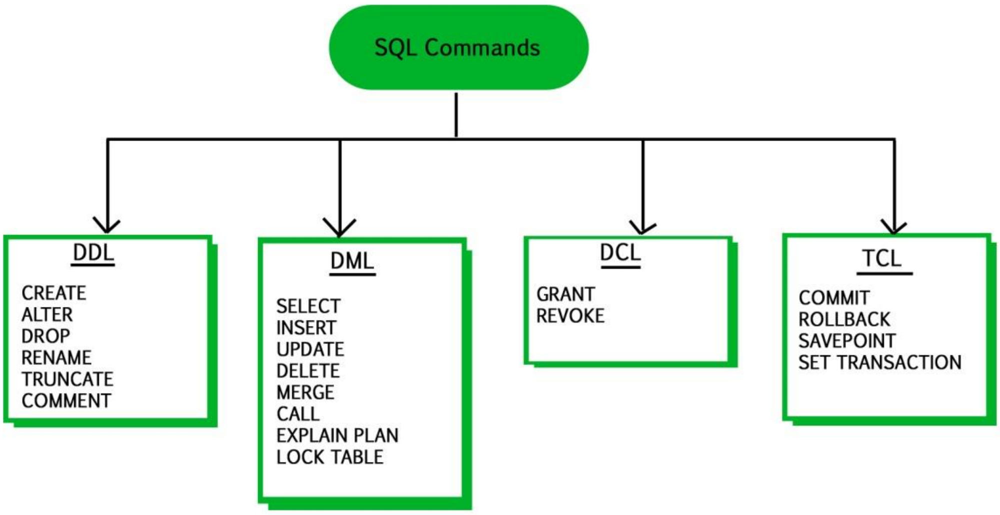
Syntax:

```
ROLLBACK;
```

SAVEPOINT: Sets a save point within a transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```



Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

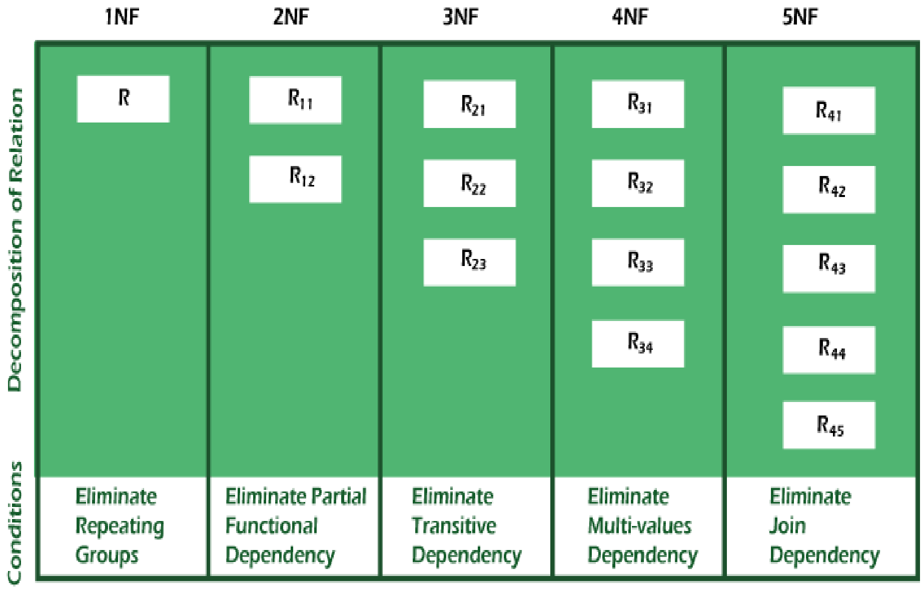
Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R11
R12
R21
R22
R23
R31
R32
R33
R34
R41
R42
R43
R44
R45
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.

- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Introduction to PL/SQL

PL/SQL stands for "Procedural Language extensions to the Structured Query Language".

SQL is a popular language for both querying and updating data in relational database management systems (RDBMS).

PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL. In addition, PL/SQL provides a more comprehensive programming language solution for building mission-critical applications on Oracle Databases.

PL/SQL is a highly structured and readable language. Its constructs express the intents of the code clearly. Also, PL/SQL is a straightforward language to learn.

PL/SQL is a standard and portable language for Oracle Database development. If you develop a program that executes on an Oracle Database, you can quickly move it to another compatible Oracle Database without any changes.

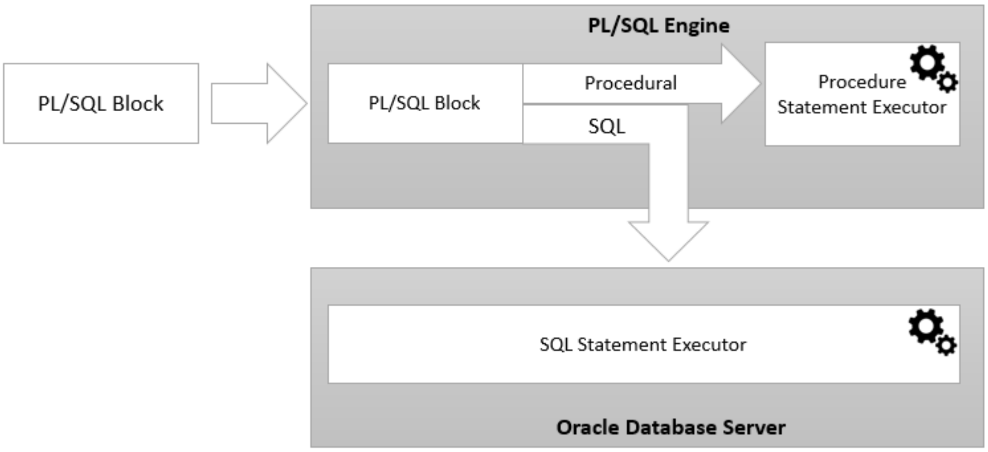
PL/SQL is an embedded language. PL/SQL only can be executed in an Oracle Database. It was not designed to be used as a standalone language like Java, C#, and C++. In other words, you cannot develop a PL/SQL program that runs on a system that does not have an Oracle Database.

PL/SQL is a high-performance and highly integrated database language. Besides PL/SQL, you can use other programming languages like Java, C#, and C++.

However, it is easier to write efficient code in PL/SQL than other programming languages when it comes to interacting with the Oracle Database. In particular, you can use PL/SQL-specific constructs like the FORALL statement that helps improve database performance.

PL/SQL architecture

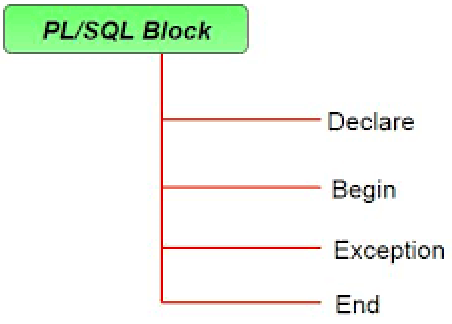
The following picture illustrates the PL/SQL architecture:



In this architecture, the PL/SQL engine compiles PL/SQL code into byte-code and executes the executable code. The PL/SQL engine can only be installed in an Oracle Database server or an application development tool such as Oracle Forms.

Once you submit a PL/SQL block to the Oracle Database server, the PL/SQL engine collaborates with the SQL engine to compile and execute the code. PL/SQL engine runs the procedural elements while the SQL engine processes the SQL statements.

Now you should have a basic understanding of PL/SQL programming language and its architecture. Let's create the first working PL/SQL anonymous block



Procedure

A **procedure** is a set of instructions which takes input and performs a certain task. In SQL, procedures do not return a value. In Java, procedures and functions are same and also called **subroutines**.In SQL, a procedure is basically a precompiled statement which is stored inside the database. Therefore, a procedure is sometimes also called a **stored procedure**. A procedure always has a name, list of parameters, and compiled SQL statements. In SQL, a procedure does not return any value.

CREATE [OR REPLACE] **PROCEDURE** procedure_name

[(parameter [,parameter])]

```
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [procedure_name];
```

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

Procedure created.

```
BEGIN
   insertuser(101,'Rahul');
   dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

Function

A **function**, in the context of computer programming languages, is a set of instructions which takes some input and performs certain tasks. In SQL, a function returns a value. In other words, a function is a tool in SQL that is used to calculate anything to produce an output for the provided inputs. In SQL queries, when a function is called, it returns the resulting value. It also controls to the calling function. However, in a function, we cannot use some DML statements like Insert, Delete, Update, etc.Also, a function can be called through a procedure. Based on definition, there are two types of functions namely, **predefined function** and **userdefined function**. Another important point about functions is that they may or may not return a value, i.e. a function can return a null valued as well.

Syntax to create a function:

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

1. **Function_name:** specifies the name of the function.
2. **[OR REPLACE]** option allows modifying an existing function.
3. The **optional parameter list** contains name, mode and types of the parameters.
4. **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
5. RETURN clause specifies that data type you are going to return from the function.
6. Function_body contains the executable part.
7. The AS keyword is used instead of the IS keyword for creating a standalone function.

Difference between Function and Procedure

Following are the important differences between SQL Function and SQL Procedure−

Key	Function	Procedure

Definition	A function is used to calculate result using given inputs.	A procedure is used to perform certain task in order.
Call	A function can be called by a procedure.	A procedure cannot be called by a function.
DML	DML statements cannot be executed within a function.	DML statements can be executed within a procedure.
SQL, Query	A function can be called within a query.	A procedure cannot be called within a query.
SQL, Call	Whenever a function is called, it is first compiled before being called.	A procedure is compiled once and can be called multiple times without being compiled.
SQL, Return	A function returns a value and control to calling function or code.	A procedure returns the control but not any value to calling function or code.
try-catch	A function has no support for try-catch	A procedure has support for try-catch blocks.
SELECT	A select statement can have a function call.	A select statement can't have a procedure call.
Explicit Transaction Handling	A function cannot have explicit transaction handling.	A procedure can use explicit transaction handling.

Exception Handling

program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using **WHEN others THEN** −

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Example

Let us write a code to illustrate the concept. We will be using the CUSTOMERS table

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

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

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

```
No such customer!

PL/SQL procedure successfully completed.
```

The above program displays the name and address of a customer whose ID is given. Since there is no customer with ID value 8 in our database, the program raises the run-time exception **NO_DATA_FOUND**, which is captured in the **EXCEPTION block**.

Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**. Following is the simple syntax for raising an exception −

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  statement;
END;
```

You can use the above syntax in raising the Oracle standard exception or any user-defined exception. In the next section, we will give you an example on raising a user-defined exception. You can raise the Oracle standard exceptions in a similar way.

User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure **DBMS_STANDARD.RAISE_APPLICATION_ERROR**.

The syntax for declaring an exception is −

```
DECLARE
  my-exception EXCEPTION;
```

Example

The following example illustrates the concept. This program asks for a customer ID, when the user enters an invalid ID, the exception **invalid_id** is raised.

```
DECLARE
  c_id customers.id%type := &cc_id;
  c_name customerS.Name%type;
  c_addr customers.address%type;
  -- user defined exception
  ex_invalid_id  EXCEPTION;
BEGIN
  IF c_id <= 0 THEN
```

```
      RAISE ex_invalid_id;
   ELSE
      SELECT  name, address INTO  c_name, c_addr
      FROM customers
      WHERE id = c_id;
      DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
      DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
   END IF;

EXCEPTION
   WHEN ex_invalid_id THEN
      dbms_output.put_line('ID must be greater than zero!');
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
/
```

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

```
Enter value for cc_id: -6 (let's enter a value -6)
old  2: c_id customers.id%type := &cc_id;
new  2: c_id customers.id%type := -6;
ID must be greater than zero!

PL/SQL procedure successfully completed.
```

Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important pre-defined exceptions −

Exception	**Oracle Error**	**SQLCODE**	**Description**
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.

COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.

SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

DDL COMMANDS

Create command

```
SQL> create table stu(sno number(3), name varchar2(20),mark1 number(3));

Table created.
```

Desc command

```
SQL> desc stu;

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNO                                                NUMBER(3)
 NAME                                               VARCHAR2(20)
 MARK1                                              NUMBER(3)
```

Alter command

```
SQL> alter table stu add(mark2 number(3));

Table altered.
```

```
SQL> desc stu;

 Name                               Null?                    Type

 --------------------------------------- -------- ---------------------------

 SNO                                                       NUMBER(3)

 NAME                                                      VARCHAR2(20)

 MARK1                                                     NUMBER(3)

 MARK2                                                     NUMBER(3)


SQL> alter table stu drop(mark1);

Table altered.

SQL> desc stu;

 Name                                     Null?                  Type

 --------------------------------------- -------- ---------------------------

 SNO                                                      NUMBER(3)

 NAME                                                     VARCHAR2(20)

 MARK2                                                    NUMBER(3)
```

Add command

```
SQL> alter table stu add(mark1 number(3));

Table altered.

SQL> desc stu;

 Name                              Null?                   Type

 --------------------------------------- -------- ---------------------------

 SNO                                                  NUMBER(3)

 NAME                                                 VARCHAR2(20)

 MARK2                                                NUMBER(3)

 MARK1                                                NUMBER(3)
```

Modify command

```
SQL> alter table stu modify(name varchar2(10));
```

```
Table altered.

SQL> desc stu;

 Name                           Null?                         Type

 ---------------------------------------- -------- ----------------------------

 SNO                                                       NUMBER(3)

 NAME                                                      VARCHAR2(10)

 MARK2                                                     NUMBER(3)

 MARK1                                                     NUMBER(3)
```

<u>Truncate command</u>

```
SQL> truncate table stu;

Table truncated.
```

<u>Rename command</u>

```
SQL> rename stu to stu1;

Table renamed.

SQL> desc stu1;

 Name                         Null?                        Type

 ---------------------------------------- -------- ----------------------------

 SNO                                                     NUMBER(3)

 NAME                                                    VARCHAR2(10)

 MARK2                                                   NUMBER(3)

 MARK1                                                   NUMBER(3)
```

<u>Drop command</u>

```
SQL> drop table stu1;

Table dropped.

SQL> desc stu1;

ERROR:

ORA-04043: object stu1 does not exist
```

DML COMMANDS

Table creation

```
SQL> create table Employee(Emp_id number(3),Emp_name varchar2(15),Emp_dept varchar2(15),Emp_salary number(6));

Table created.
```

Insert command

```
SQL> insert into Employee values(&Emp_id,'&Emp_name','&Emp_dept',&Emp_salary);

Enter value for Emp_id: 101

Enter value for Emp_name: Kaleeswari

Enter value for Emp_dept: Sales

Enter value for Emp_salary: 10000

old   1: insert into Employee values(&Emp_id,'&Emp_name','&Emp_dept',&Emp_salary)

new   1: insert into Employee values(101,'Kaleeswari','Sales',10000)

1 row created.


SQL> /

Enter value for Emp_id: 102

Enter value for Emp_name: Gayathri

Enter value for Emp_dept: Purchase

Enter value for Emp_salary: 15000

old   1: insert into Employee values(&Emp_id,'&Emp_name','&Emp_dept',&Emp_salary)

new   1: insert into Employee values(102,'Gayathri','Purchase',15000)

1 row created.


SQL> /

Enter value for Emp_id: 103
Enter value for Emp_name: Kaviya
Enter value for Emp_dept: Inventory
Enter value for Emp_salary: 15000
old   1: insert into Employee values(&Emp_id,'&Emp_name','&Emp_dept',&Emp_salary)
new   1: insert into Employee values(103,'Kaviya','Inventory',15000)
1 row created.
```

Select command

```
SQL> select*from Employee;
EMP_ID                 EMP_NAME              EMP_DEPT           EMP_SALARY
---------- -------------- -------------- -----------------------------------------------------------------
 101                   Kaleeswari                 Sales                      10000
 102                   Gayathri                   Purchase                   15000
 103                   Kaviya                     Inventory                   15000
```

Update command

```
SQL> update Employee set EMP_NAME='KALEESWARI'where EMP_ID=101;
1 row updated.
SQL> select*from Employee;
EMP_ID                   EMP_NAME                EMP_DEPT           EMP_SALARY
------------------------------------ --------------------------------------------------------------------
101                    KALEESWARI                   Sales                   10000
102                    Gayathri                     Purchase                15000
103                    Kaviya                       Inventory              15000
```

Delete command

```
SQL> delete from Employee;
```

```
3 rows deleted.

SQL> select*from Employee;

No rows selected
```

DCL COMMANDS

Grant

```
SQL> grant connect,dba,resource to commercesf identified by palani;

Grant succeeded.

SQL> connect commercesf/palani;

Connected.

SQL> select user from dual;

USER

-----------------------------

COMMERCESF
```

Revoke

```
SQL> revoke connect,dba,resource from commercesf;

Revoke succeeded.

SQL> connect system/manager;

Connected.

SQL> select user from dual;

USER

------------------------------

SYSTEM

SQL> create user college identified by apa;

User created.

SQL> grant create session to college;

Grant succeeded.
```

```
SQL>grant create table to college;

Grant succeeded.

SQL> connect college;

Enter password: ***

Connected.

SQL> select user from dual;

USER

-----------------------------

COLLEGE

SQL> connect system

Enter password: *******

Connected.
```

Table creation

```
SQL> create table Book (Book_no number(3),Book_name varchar2(15),Book_athor varchar2(15));

Table created.
```

Insert command

```
SQL> insert into Book values(101,'RDBMS','Alex');

1 row created.

SQL> insert into Book values(102,'VISUALBASIC','Lenin');

1 row created.

SQL> insert into Book values(103,'C++','Balagurusamy');

1 row created.
```

Select command

```
SQL> select*from Book;

BOOK_NO       BOOK_NAME     BOOK_ATHOR
```

```
---------- -------------- ----------------------------------------
   101                RDBMS                   Alex
   102                VISUALBASIC            Lenin
   103                C++                     Balagurusamy
```

Commit command

```
SQL> commit;
Commit complete.
```

Delete command

```
SQL> delete from Book where Book_name='C++';
1 row deleted.
SQL> select*from Book;
BOOK_NO              BOOK_NAME           BOOK_ATHOR
----------------------- ------------------------------------------------
    101                       RDBMS                   Alex
    102                       VISUALBASIC            Lenin
```

Rollback command

```
SQL> rollback;
Rollback complete.
SQL> select*from Book;
BOOK_NO           BOOK_NAME           BOOK_ATHOR
---------- -------------- ---------------------------------------------
   101                    RDBMS                   Alex
   102                    VISUALBASIC            Lenin
   103                    C++                     Balagurusamy
```

EXCEPTION HANDLING

```
SQL> set serveroutput on;

SQL> declare

n number(4);

d number(4);

begin

n:=&n;

d:=n/0;

exception

when zero_divide then

dbms_output.put_line('dividing by error exception is caught');

end;

11  /

Enter value for n: 4

old   5: n:=&n;

new   5: n:=4;

dividing by error exception is caught


PL/SQL procedure successfully completed.
```

SUM OF THE PROCEDURE

```
SQL> create or replace procedure sum(n1 IN number,n2 IN number)IS total number(6);

begin

total:=n1+n2;

dbms_output.put_line('The sum is:'||total);

end;

6  /

Procedure created.
```

```
SQL> set serveroutput on;

SQL> begin

sum(12,13);

end;

4  /

The sum is:25

PL/SQL procedure successfully completed.
```

FUNCTION CREATION

```
SQL> create table customers(cus_id number(6),cus_name varchar2(25),cus_age
number(6),cus_address varchar2(25),cus_salary number(6));

Table created.
```

Insert command

```
SQL> insert into customers

values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary);

Enter value for cus_id: 1

Enter value for cus_name: Kaleeswari

Enter value for cus_age: 22

Enter value for cus_address: palani

Enter value for cus_salary: 2000.00

old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)

new   2: values(1,'Kaleeswari',22,'palani',2000.00)

1 row created.


SQL> /

Enter value for cus_id: 2

Enter value for cus_name: Gayathri
Enter value for cus_age: 25
Enter value for cus_address: delhi
Enter value for cus_salary: 1500.00
old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)
new   2: values(2,'Gayathri',25,'delhi',1500.00)
1 row created.


SQL> /
Enter value for cus_id: 3
Enter value for cus_name: Nagalakshmi
Enter value for cus_age: 26
Enter value for cus_address: mumbai
Enter value for cus_salary: 6500.00
old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)
new   2: values(3,'Nagalakshmi',26,'mumbai',6500.00)
1 row created.

SQL> /
Enter value for cus_id: 4
Enter value for cus_name: Paritha
Enter value for cus_age: 23
Enter value for cus_address: madhurai
Enter value for cus_salary: 2000.00
old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)
new   2: values(4,'paritha',23,'madhurai',2000.00)
1 row created.
```

```
SQL> /

Enter value for cus_id: 5

Enter value for cus_name: Srinithi

Enter value for cus_age: 27

Enter value for cus_address: chennai

Enter value for cus_salary: 4500.00



old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)

new   2: values(5,'Srinithi',27,'chennai',4500.00)

1 row created.


SQL> /

Enter value for cus_id: 6

Enter value for cus_name: Aarthi

Enter value for cus_age: 25

Enter value for cus_address: covai

Enter value for cus_salary: 1500.00

old   2: values(&cus_id,'&cus_name',&cus_age,'&cus_address',&cus_salary)

new   2: values(6,'Aarthi',25,'covai',1500.00)

1 row created.
```

Select command

```
SQL> select*from customers;

CUS_ID        CUS_NAME          CUS_AGE     CUS_ADDRESS     CUS_SALARY
----------------------------------------------------------------------
1             Kaleeswari           22          palani           2000

2             Gayathri             25          delhi            1500
```

```
3           Nagalakshmi          26          mumbai          6500

4           Paritha              23          madhurai        2000

5           Srinithi             27          chennai         4500

6           Aarthi               25          covai           1500


6 rows selected.



SQL> create or replace function totalcustomers

return number IS

total number(2):=0;

begin

select count(*)into total

from customers;

return total;

end;

9  /

Function created.


SQL> set serveroutput on;

SQL> declare

c number(2);

begin

c:=totalcustomers();

dbms_output.put_line('Total no.of customers:'||c);

end;

7  /
```

```
Total no.of customers:6

PL/SQL procedure successfully completed.
```

QUESTION BANK

UNIT I:-

1. A **domain** is a set of all possible data values.

2. In formal relational term for table is called **Relation**.

3. In formal relational term for row or record is known as **Tuple**.

4. In formal relational term for rows are called **cardinality.**

5. In formal relational term for column or field is known as **attribute**.

6. In formal relational term for number of columns are called **Degree.**

7. The **Primary key** is referred as unique identifier.

8. The ERD stands for **Entity-Relationship Diagram**.

9. **Data** model makes easier to understand the meaning of the data.

10. A **Database** is a collection of data designed to be used by different people.

11. An IS stands for **Information Services**.

12. The RDBMS stands for **Relational Database Management System**.

13. The DBMS stands for **Database Management System**.

14. The **Conceptual** model represents a global view of the data.

15. The ERD is a **Visual** representation of E-R model.

16. The highest and the outermost layer is called **External or Logical level.**

17. The lowest level layer is called is called **Physical level.**

18. The RDBMS terminology principles were laid down by **Dr.E.F.Codd**.

19. The **External** level is the end user’s view of the data environment.

20. The physical components organized and stored are called **raw**data.

UNIT II:-

1. The set of basic objects are called **Entities.**

2. An entity is represented by a set of **Attributes**.

3. **Simple** attributes cannot be divided into subparts.

4. **Composite** attributes can be divided into subparts.

5. An entity which does not have a value for an attribute is called **Null** attribute.

6. The **Derived** attributes can be derived from the values of other related attributes or entities.

7. A **Relationship** is an association among several attributes.

8. An attribute which has a set of values for specific entities are called **Single valued** attributes.

9. An attribute which has different numbers of values are called **Multivalued** attribute.

10. A **Binary** relationship exits when two entities are associated.

11. A **Ternary** relationship exits when there are three entities associated.

12. The **n-ary** relationship set refers to nonbinary relationship set.

13. The **Rectangles** diagram represents the entity sets.

14. The **Ellipses** diagram represents the attributes.

15. The **Diamonds** diagram represents the relationship sets.

16. The **lines** diagram represents the link of entity set and relationship set.

17. The **Double ellipses** diagram represents the multivalved attributes.

18. The **Dashed ellipses** represent the derived attributes.

19. An entity set not having sufficient attributes is called **Weak entity** set.

20. When an entity is associated with another entity, the relationship is called as **one-to-one.**

UNIT III:-

1. The BCNF stands for **Boyce – Codd Normal Form**
2. The elimination of repeating groups is called **1NF**
3. The elimination of redundant data is called **2NF**
4. **Normalization** is the process of building database structures to store data.
5. An **intelligent** key is based upon data values.
6. A **key** uniquely identifies a row in a table.
7. The left hand side of a functional dependency is called **determinant**.
8. The right hand side of a functional dependency is called **dependent.**
9. The transitive dependencies have to be removed by **3NF.**
10. The multivalued dependencies are removed by **4 NF**.
11. The **intelligent keys and non intelligent keys** are two types of keys.
12. The repetition of information and inability to represent certain information are the **pitfalls** in relational database design
13. The goal of a relational database design is to generate a **set of relation** scheme.
14. The **careless** decomposition may lead to another form of bad design.
15. The **Transitive** dependency is a description of a type of functional dependency.
16. A **Functional** dependency describes the relationship between attributes in a relation.
17. The primary key consists of only one **Attribute**.
18. All the attributes in the relation are **Components** of the primary key
19. Each primary key may have any number of **Foreign** keys using the same values.
20. A **Foreign** key is a column in a table that uniquely identifies the records from a different table.

UNIT IV:-

1. DDL Stands for **Data Definition Language**.
2. In RDBMS rows are called the **Records**.
3. In RDBMS columns are called the **Fields.**
4. A record is a collection of related **Fields**.
5. **Altering** command is used to change the structure of a table.
6. The **Truncate** command is used to delete all records stored in a table.
7. The **Dropping** command is used to remove a table from the database.
8. The **View** command is used to view the structure of a table,
9. The **Data Manipulation Languages** are used to manipulate the existing objects.
10. The **Insert** command is used to add one or more rows to a table.
11. The **Select** command is used to retrieve the stored data from a table.
12. The **Distinct** command is used to avoid the selection of duplicate rows.
13. The **Update** command is used to update rows in a table.
14. The **Delete** command is used to delete rows from a table.
15. The **Rollback** command is used to undo all the changes made in the current transaction.
16. The **Commit** command is used to make all transaction changes permanent to the database.
17. TCL stands for **Transaction Control Language**.
18. All changes made to the database is defined as a **Transaction**.
19. The DCL stands for **Data Control Language**.
20. **Modify & Add** commands are the two types of Alter command.

UNIT-V:-

1. A **Procedure** is a module that performs one or more actions.

2. The portion of the procedure definition that comes before the IS keyword is called **Procedure header**.
3. The name of the procedure can be appended directly after the **END** keyword.
4. The **RETURN** statement is generally associated with a function.
5. The **PL/SQL** allows the use of return statement in a procedure.
6. A **Function** is a module that returns a value.
7. The portion of the function that comes before the IS keyword is called the **Function Header.**
8. The body of the function consists the **Declaration, Execution and Exception** sections.
9. The PL/SQL stands for **Procedural Language extensions to the Structured Query Language**.
10. **Date** data type, stores both dates and times.
11. CLOB stands for **Character Large Object**.
12. The **ROWID** represents the unique address of a row in its table.
13. A Cursor attribute takes the form **% attribute-name** and is appended to the name of a cursor or cursor variable.
14. The **FETCH** Command retrieves the next row from the cursor's result set.
15. In the PL/SQL language errors of any kind are treated as **Exceptions.**
16. The **System exceptions and programmer defined exceptions** are the two types of exceptions.
17. An Exception handlers must appear after all the **Executable** statements.
18. The **Exception** keyword indicates the start of the exception section and the individual exception.
19. The SQLERRM stands for the **Structured Query Language Error Message**.
20. The BLOB Stands for **Binary Large Object**.

Online Reference

S.No	Link
1	https://youtu.be/T7AxM7Vqvaw
2	https://youtu.be/3EJlovevfcA
3	https://youtu.be/q1kQ1vgW7D8
4	https://youtu.be/9RlxvHMg9PI