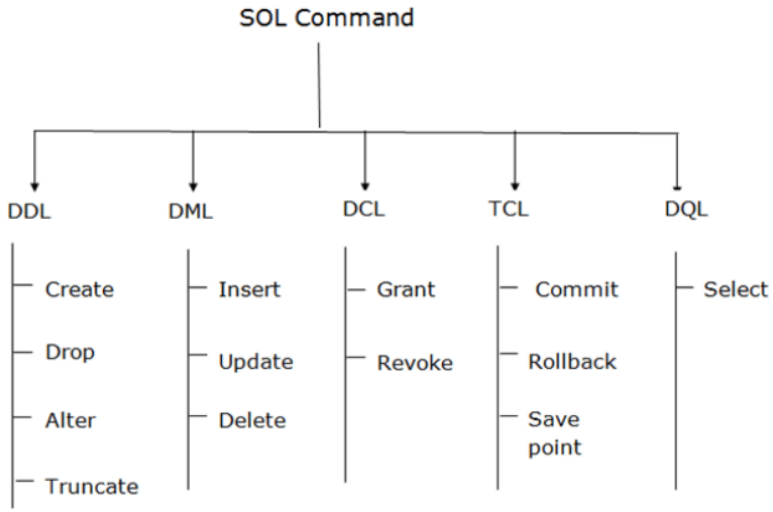
SQL Commands:

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DAT E);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

1. DROP TABLE ;

Example

1. DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE MODIFY(COLUMN DEFINITION....);

EXAMPLE

1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);
```

Or

```
1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, .... valueN);
```

For example:

```
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CO
   NDITION]
```

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
1. DELETE FROM table_name [WHERE condition];
```

For example:

```
1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

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

b. Revoke: It is used to take back permissions from the user.

Example

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

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
1. SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;
```

For example:

```
1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;
```

SQL Reports:

SQL Server Reporting Services (SSRS) is a reporting software that allows us to produce formatted reports with tables in the form of data, graph, images, and charts. These reports are hosted on a server that can be executed any time using parameters defined by the users. It is part of Microsoft SQL Server Services suite.

SQL Data Type:

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use which are listed below −

Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255

bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Note − Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
2	**varchar** Maximum of 8,000 characters.(Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only).
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

Sr.No.	DATA TYPE & Description
1	**nchar** Maximum length of 4,000 characters.(Fixed length Unicode)
2	**nvarchar** Maximum length of 4,000 characters.(Variable length Unicode)
3	**nvarchar(max)** Maximum length of 2E + 31 characters (SQL Server 2005 only).(Variable length Unicode)
4	**ntext** Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types

Sr.No.	DATA TYPE & Description
1	**binary** Maximum length of 8,000 bytes(Fixed-length binary data)
2	**varbinary** Maximum length of 8,000 bytes.(Variable length binary data)
3	**varbinary(max)** Maximum length of 2E + 31 bytes (SQL Server 2005 only). (Variable length Binary data)
4	**image** Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types

Sr.No.	DATA TYPE & Description
1	**sql_variant** Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
2	**timestamp** Stores a database-wide unique number that gets updated every time a row gets updated
3	**uniqueidentifier** Stores a globally unique identifier (GUID)
4	**xml** Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
5	**cursor** Reference to a cursor object
6	**table** Stores a result set for later processing

UNIT IV

PL/SQL

PL/SQL is a combination of SQL along with the procedural features of programming languages.

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.

Features of PL/SQL:

PL/SQL has the following features −

PL/SQL is tightly integrated with SQL.

It offers extensive error checking.

It offers numerous data types.

It offers a variety of programming structures.

It supports structured programming through functions and procedures.

It supports object-oriented programming.

It supports the development of web applications and server pages.

Advantages of PL/SQL:

PL/SQL has the following advantages −

SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.

PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.

PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.

PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.

Applications written in PL/SQL are fully portable.

PL/SQL provides high security level.

PL/SQL provides access to predefined SQL packages.

PL/SQL provides support for Object-Oriented Programming.

PL/SQL provides support for developing Web Applications and Server Pages.

PL/SQL Program Units:

A PL/SQL unit is any one of the following −

PL/SQL block

Function

Package

Package body

Procedure

Trigger

Type

Type body

PL/SQL stored Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

Passing parameters in procedure:

When we want to create a procedure or function, we have to define parameters .There is three ways to pass parameters in procedure:

IN parameters: The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.

OUT parameters: The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

INOUT parameters: The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

PL/SQL Create Procedure:

Syntax for creating procedure:

CREATE [OR REPLACE] **PROCEDURE** procedure_name
[(parameter [,parameter])]

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

Create procedure example

To insert record in user table. We 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.
```

PL/SQL program to call procedure

the code to call above created procedure:

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

Now, one record is inserted in the " ÜSER" table.

ID	Name
101	Rahul

PL/SQL Drop Procedure:

Syntax for drop procedure

DROP PROCEDURE procedure_name;

Example of drop procedure:

DROP PROCEDURE pro1;

Blocks in PL/SQL

In PL/SQL, All statements are classified into units that is called Blocks. PL/SQL blocks can include variables, SQL statements, loops, constants, conditional statements and exception handling. Blocks can also build a function or a procedure or a package.

PL/SQL block structure:

```
DECLARE
   Declaration statements;
BEGIN
   Execution statements;
 EXCEPTION
     Exception handling statements;
END;
/
```

PL/SQL Block sections:

1. Declaration section (optional).
2. Execution section (mandatory).
3. Exception handling section (optional).

Declaration section:

It is an optional section and starts with DECLARE keyword. It is used to declare the variables, constants, records and cursors etc.

Execution section:

Execution section starts with BEGIN keyword and ends with END keyword. It is a mandatory section. It is used to write the program logic code.

Note: Execution section must have one statement.

Exception handling section:

Execution section starts with EXCEPTION keyword. It is an optional section. It is used to handle the exceptions occurred in execution section.

Broadly, PL/SQL blocks are two types: Anonymous blocks and Named blocks.

1. Anonymous blocks:

In PL/SQL, That' s blocks which is not have header are known as anonymous blocks. These blocks do not form the body of a function or triggers or procedure.

Example:

Here a code example of find greatest number with Anonymous blocks.

```
   z number;
BEGIN
  IF x > y THEN
    z:= x;
  ELSE
    Z:= y;
  END IF;
  RETURN z;
END;
BEGIN
  a:= 10;
  b:= 100;
  c := findMax(a, b);
  dbms_output.put_line(' Maximum number in 10 and
100 is: ' || c);
END;
/
```

PL/SQL Trigger:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are 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).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

Generating some derived column values automatically

Enforcing referential integrity

Event logging and storing information on table access

Auditing

Synchronous replication of tables

Imposing security authorizations

Preventing invalid transactions

Creating Triggers:

The syntax for creating a trigger :

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSTEAD [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.

{BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.

{INSTEAD [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.

[OF col_name] − This specifies the column name that will be updated.

[ON table_name] − This specifies the name of the table associated with the trigger.

[REFERENCING OLD AS o NEW AS n] − This allows us to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.

[FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
```

```
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

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];

Here:

Function_name: specifies the name of the function.

[OR REPLACE] option allows modifying an existing function.

The **optional parameter list** contains name, mode and types of the parameters.

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.

RETURN clause specifies that 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.

PL/SQL Function Example:

```
create or replace function adder(n1 in number, n2 in number)
return number
is
```

PL/SQL function example using table

Let's take a customer table. This example illustrates creating and calling a standalone function. This function will return the total number of CUSTOMERS in the customers table.

Customers			
Id	**Name**	**Department**	**Salary**
1	alex	web developer	35000
2	ricky	program developer	45000
3	mohan	web designer	35000
4	dilshad	database manager	44000

Create Function:

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

After the execution of above code, we will get the following result.

```
Function created.
```

Calling PL/SQL Function:

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

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function we have to pass the required parameters along with function name and if function returns a value then we can store returned value.

Following program calls the function totalCustomers from an anonymous block:

```
DECLARE
  c number(2);
BEGIN
  c := totalCustomers();
```

```
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

After the execution of above code in SQL prompt, we will get the following result.

```
Total no. of Customers: 4
PL/SQL procedure successfully completed.
```

PL/SQL PACKAGES:

A package is a way of logically storing the subprograms like procedures, functions, exception or cursor into a single common unit.

A package can be defined as an object that is compiled and stored in the database.

Once it is compiled and stored in the database it can be used by all the users of database who have executable permissions on Oracle database.

Components of Package

Package has two basic components:

Specification:

It is the declaration section of a Package

Body:

It is the definition section of a Package.

Benefits of using Package

Following are some of the benefits of packages in PL/SQL:

REUSABILITY

Whenever a package is created, it is compiled and stored in the database. So, we write the code once which can be reused by other applications.

OVERLOADING

Two or more procedures or functions can be created in a package with the same name.

CREATING MODULES

A large application can be created by simply creating modules(or subprograms) clearly defined and easy to work.

IMPROVES PERFORMANCE

Package code gets loaded inside the SGA(system global area) of Oracle at first call itself due to which other subsequent calls will work very fast.

GLOBAL DECLARATION

If the objects(procedures, functions, variables, constants, exceptions, cursor etc) are declared globally in a package, they can be easily used when required.

Creation of PL/SQL Package:

Following are the steps to declare and use a package in PL/SQL code block:

STEP 1:

Package specification or declaration

It mainly comprises of the following:

* Package Name.

*Variable/constant/cursor/procedure/function/ exception declaration.

* This declaration is global to the package.

Syntax:

CREATE OR REPLACE PACKAGE <package_name> IS/AS

FUNCTION <function_name> (<list of arguments>)

RETURN <datatype>;

PROCEDURE <procedure_name> (<list of arguments>);

-- code statements

END <package_name>;

where,

CREATE OR REPLACE PACKAGE are keywords used to create a package

FUNCTION and PROCEDURE are keywords used to declare function and procedure while creating package.

<package_name>, <function_name>, <procedure_name> are user-defined names.

IS/AS are keywords used to declare package.

RETURN is a keyword specifying value returned by the function declared.

STEP 2: Package Body

It mainly comprises of the following:

* It contains the definition of procedure, function or cursor that is declared in the package specification.

* It contains the subprogram bodies containing executable statements for which package has been created

Syntax:

CREATE OR REPLACE PACKAGE BODY <package_name> IS/AS

FUNCTION <function_name> (<list of arguments>) RETURN <datatype>IS/AS

-- local variable declaration;

BEGIN

-- executable statements;

EXCEPTION

-- error handling statements;

END <function_name>;

PROCEDURE <procedure_name> (<list of arguments>)IS/AS

-- local variable declaration;

BEGIN

-- executable statements;

EXCEPTION

-- error handling statements;

END <procedure_name>;

END <package_name>;

Where,

CREATE OR REPLACE PACKAGE BODY are keywords used to create the package with a body.

FUNCTION and PROCEDURE are keywords used to define function and procedure while creating package.

<package_name>, <function_name>, <procedure_name> are user-defined.

IS/AS are keywords used to define the body of package, function and procedure.

RETURN is a keyword specifying value returned by the function defined.

DECLARE, BEGIN, EXCEPTION, END are the different sections of PL/SQL code block containing variable declaration, executable statements, error handling statements and marking end of PL/SQL block respectively where DECLARE and EXCEPTION part are optional.

Creating a package only defines it, to use it we must refer it using the package object.

Following is the syntax for referring a package object:

Packagename.objectname;

The Object can be a function, procedure, cursor, exception that has been declared in the package specification and defined in the package body and to access their executable statements above syntax is used.

ROLLNO	SNAME	AGE	COURSE
11	Anu	20	BSC
12	Asha	21	BCOM
13	Arpit	18	BCA
14	Chetan	20	BCA
15	Nihal	19	BBA

Example:We have a STUDENT table as specified below:Let's write a simple program to demonstrate the use of Package in PL/SQL.

PL/SQL code for package specification:

```
CREATE OR REPLACE PACKAGE pkg_student IS
    PROCEDURE
        updateRecord(sno student.rollno%type);
    FUNCTION deleteRecord(snm student.sname%type)
        RETURN boolean;
END pkg_student;
```

Package Created

PL/SQL code for package body:

```
 set serveroutput on;
CREATE OR REPLACE PACKAGE BODY pkg_student IS
    PROCEDURE
updateRecord(sno student.rollno%type) IS
        BEGIN
            Update student set age=23 where rollno=sno;
            IF  SQL%FOUND THEN
```

Example of exception handling:

Let's take a simple example to demonstrate the concept of exception handling. Here we are using the already created CUSTOMERS table.

SELECT* FROM COUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex.	21	Paris	28000
6	Sunita	20	Delhi	30000

```
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;
/
```

UNIT V

Locking Techniques in DBMS:

In order to ensure the correct execution of concurrent transactions, a number of concurrency control techniques are applied in DBMS. One of them is use of Locking Techniques.

Concurrency Control Schemes are the schemes which are used by DBMS to ensure that concurrent transactions do not interfere with each other.

Some of these Locking Techniques are:

* Locking

* Time Stamps, and

* Optimistic Protocols.

Introduction to Locking

* Locking is one of the most commonly used concurrency control schemes in DBMS.

* This works by associating a variable lock on the data items.

* This variable describes the status of the data item with respect to the possible operations that can be applied on it.

* The value of this variable is used to control the concurrent access and the manipulation of the associated data item.

* The concurrency control technique in which the value of the lock variable is manipulated is called locking.

* The technique of locking is one way to ensure Serializability in DBMS.

In DBMS, locking is the responsibility of a subsystem called lock manager.

Types of Locking Techniques

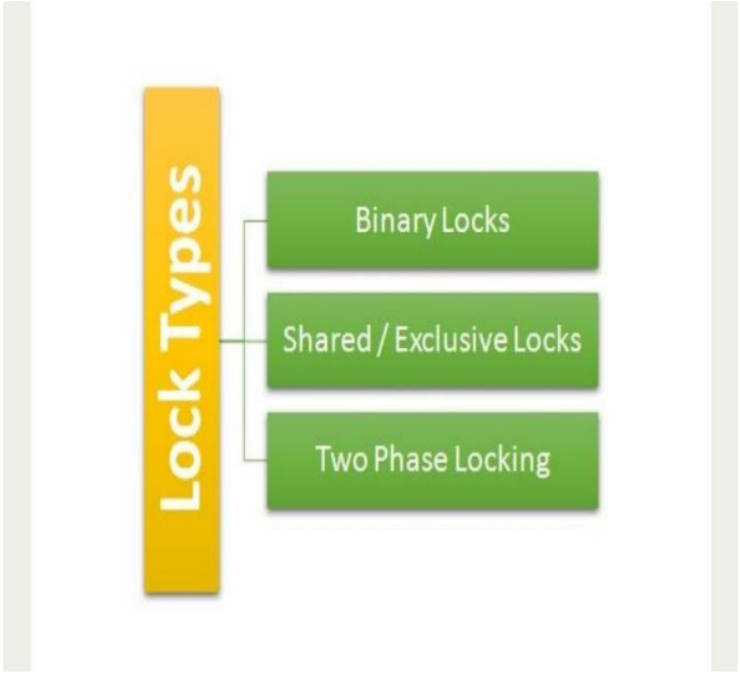
To control concurrency there are various types of locks which can be applied in DBMS.

Locking Techniques in DBMS

Locks are usually issued by the transactions before any operations are performed by them.

Binary Locks

A binary lock has two states or values associated



with each data item. These values are:

Locked – 1

Unlocked – 0

* If a data item is locked, then it cannot be accessed by other transactions i.e., other transactions are forced to wait until the lock is released by the previous transaction.

* But, if a data item is in the unlocked state, then, it can be accessed by any transaction and on access the lock value is set to locked state.

* These locks are applied and removed using Lock () and Unlock () operation respectively.

* In binary locks, at a particular point in time, only one transaction can hold a lock on the data item.No other transaction will be able to access the same data concurrently.

* Hence, Binary locks are very simple to apply but are not used practically.

Shared / Exclusive Locks

* In shared locks, multiple users are allowed to access the same data item with a read lock which is shared by them.

*But, in case when a transaction needs to write a data item, then an exclusive lock is applied on that data item. So here, we classify the locks as:

Shared Locks

Exclusive Locks

Shared Locks

* Shared locks are applied to a data item when the transaction requests a read operation on the data item.

* A shared lock will allow multiple transactions to only read the data item concurrently.

* As these locks are applied on read operation, they will not compromise on the consistency of the database.

Exclusive Locks

* Exclusive locks on the other hand are applied on the transactions which request a write operation on the data item.

* The transaction which is modifying the data item requests an exclusive lock on the data item and hence any other transaction which needs access to the data item has to wait until the lock applied by the previous transaction has been released by it.

But when exclusive locks are applied there are situations when a transaction enters into a wait state indefinitely. Such a state where a transaction cannot come out of the wait state is known as a **deadlock**.

Two Phase Locking

The Two Phase Locking Techniques guarantee Serializability in DBMS. A transaction is said to follow Two Phase Locking Protocol if all locking operations in the transaction precede the first unlock operation.

In this, locks are applied in two phases:

* Growing Phase

* Shrinking Phase

Growing Phase

* This phase is also known as the first phase or the expanding phase.

* It is in this phase that the transaction acquires all the locks needed by it but it cannot release any locks here.

Shrinking Phase

* This phase is also known as the second phase or the contracting phase.

* Here a transaction is not allowed to acquire any new locks but it can release the existing locks it holds.

* The Two Phase Locking Protocol helps solve problems of lost update, inconsistent analysis or dirty read too.

The Timestamp Ordering:

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.

The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a Read (X) operation:

If W_TS(X) >TS(Ti) then the operation is rejected.

If W_TS(X) <= TS(Ti) then the operation is executed.

Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:

If TS(Ti) < R_TS(X) then the operation is rejected.

If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Validation Techniques:

Validation Based Protocol in DBMS

Validation Based Protocol is also called Optimistic Concurrency Control Technique. This protocol is used in DBMS (Database Management System) for avoiding concurrency in transactions. It is called optimistic because of the assumption it makes, i.e. very less interference occurs, therefore, there is no need for checking while the transaction is executed.

In this technique, no checking is done while the transaction is been executed. Until the transaction end is reached updates in the transaction are not applied directly to the database. All updates are applied to local copies of data items kept for the transaction. At the end of transaction execution, while execution of the transaction, a validation phase checks whether any of transaction updates violate serializability. If there is no violation of serializability the transaction is committed and the database is updated; or else, the transaction is updated and then restarted.

Optimistic Concurrency Control is a three-phase protocol. The three phases for validation based protocol:

Read Phase:

Values of committed data items from the database can be read by a transaction. Updates are only applied to local data versions.

Validation Phase:

Checking is performed to make sure that there is no violation of serializability when the transaction updates are applied to the database.

Write Phase:

On the success of the validation phase, the transaction updates are applied to the database,

2. Ti begins its write phase after Tj completes its write phase, and the read_set of Ti should be disjoint with write_set of Tj.

3. Tj completes its read phase before Ti completes its read phase and both read_set and write_set of Ti are disjoint with the write_set of Tj.

Advantages:

1. Avoid Cascading-rollbacks: This validation based scheme avoid cascading rollbacks since the final write operations to the database are performed only after the transaction passes the validation phase. If the transaction fails then no updation operation is performed in the database. So no dirty read will happen hence possibilities cascading-rollback would be null.

2. Avoid deadlock: Since a strict time-stamping based technique is used to maintain the specific order of transactions. Hence deadlock isn't possible in this scheme.

Disadvantages:

1. Starvation: There might be a possibility of starvation for long-term transactions, due to a sequence of conflicting short-term transactions that cause the repeated sequence of restarts of the long-term transactions so on and so forth. To avoid starvation, conflicting transactions must be temporarily blocked for some time, to let the long-term transactions to finish.

Database Recovery Techniques in DBMS:

* **Database systems**, like any other computer system, are subject to failures but the data stored in it must be available as and when required.
* When a database fails it must possess the facilities for fast recovery.
* It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.
* There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations.

* Commitment always involves writing a commit entry to the log and writing the log to disk.
* At the time of a system crash, item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process

Undoing :

* If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction.
* This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to old-value.
* There are two major techniques for recovery from non-catastrophic transaction failures:

1.deferred updates and
2. immediate updates.

Deferred update:

* This technique does not physically update the database on disk until a transaction has reached its commit point.
* Before reaching commit, all transaction updates are recorded in the local transaction workspace.
* If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed.
* It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**

Immediate update :

* In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point.
* However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible.
* If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**

Caching/Buffering :

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

1. <Tn, City, 'Noida', 'Bangalore' >

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Database Security Issues:

The threats identified over the last couple of years are the same that continue to plague businesses today, according to Gerhart. The most common database threats include:

***Excessive privileges.** When workers are granted default database privileges that exceed the requirements of their job functions, these privileges can be abused, Gerhart said. " For example, a bank employee whose job requires the ability to change only account holder contact information may take advantage of excessive database privileges and increase the account balance of a colleague' s savings account." Further, some companies fail to update access privileges for employees who change roles within an organization or leave altogether.

***Legitimate privilege abuse.** Users may abuse legitimate database privileges for unauthorized purposes, Gerhart said.

***Database injection attacks.** The two major types of database injection attacks are SQL injections that target traditional database systems and NoSQL injections that target " big data" platforms. " A crucial point to realize here is that, although it is technically true that big data solutions are impervious to SQL injection attacks because they don' t actually use any SQL-based technology, they are, in fact, still susceptible to the same fundamental class of attack," Gerhart said. " In both types, a successful input injection attack can give an attacker unrestricted access to an entire database."

***Malware.** A perennial threat, malware is used to steal sensitive data via legitimate users using infected devices.

***Storage media exposure.** Backup storage media is often completely unprotected from attack, Gerhart said. " As a result, numerous security breaches have involved the theft of database backup disks and tapes. Furthermore, failure to audit and monitor the activities of administrators who have low-level access to sensitive information can put your data at risk. Taking the appropriate measures to protect backup copies of sensitive data and monitor your most highly privileged users is not only a data security best practice, but also mandated by many regulations," he said.

***Exploitation of vulnerable databases.** It generally takes organizations months to patch databases, during which time they remain vulnerable. Attackers

know how to exploit unpatched databases or databases that still have default accounts and configuration parameters. " Unfortunately, organizations often struggle to stay on top of maintaining database configurations even when patches are available. Typical issues include high workloads and mounting backlogs for the associated database administrators, complex and time-consuming requirements for testing patches, and the challenge of finding a maintenance window to take down and work on what is often classified as a business-critical system," Gerhart said.

***Unmanaged sensitive data.** Many companies struggle to maintain an accurate inventory of their databases and the critical data objects contained within them. " Forgotten databases may contain sensitive information, and new databases can emerge without visibility to the security team. Sensitive data in these databases will be exposed to threats if the required controls and permissions are not implemented," he said.

***The human factor.** The root cause for 30 percent of data breach incidents is human negligence, according to the Ponemon Institute Cost of Data Breach Study. " Often this is due to the lack of expertise required to implement security controls, enforce policies or conduct incident response processes," Gerhart said.

Database access control:

Database access control is a method of allowing access to company' s sensitive data only to those people (database users) who are allowed to access such data and to restrict access to unauthorized persons. It includes two main components: authentication and authorization.

Authentication is a method of verifying the identity of a person who is accessing your database. Note that authentication isn' t enough to protect data. An additional layer of security is required, authorization, which determines whether a user should be allowed to access the data or make the transaction he' s attempting. Without authentication and authorization, there is no data security.

Any company whose employees connect to the Internet, thus, every company today, needs some level of access control implemented.

Types of Access Control

Obsolete access models include Discretionary Access Control (DAC) and Mandatory Access Control (MAC). Role Based Access Control (RBAC) is the most common method today, and the most recent model is Attribute Based Access Control (ABAC).

Discretionary Access Control (DAC)

With DAC models, the data owner allows access. DAC is a means of assigning access rights based on user-specified rules.

Mandatory Access Control (MAC)

MAC was developed using a nondiscretionary model, in which people are granted access based on an information clearance. MAC is a policy in which access rights are assigned based on central authority regulations.

Role Based Access Control (RBAC)

RBAC grants access based on a user' s role and implements key security principles such as " least privilege" and " separation of privilege." Thus, someone attempting to access information can only access data necessary for their role.

Attribute Based Access Control (ABAC)

In ABAC, each resource and user are assigned a series of attributes. In this dynamic method, a comparative assessment of the user' s attributes, including time of day, position and location, are used to make a decision on access to a resource.

Working of Access Control:

Two-Factor Authentication

Two-factor authentication mechanisms based on emails and one-time passwords (OTP) which allow to access the target database. Database users should input database' s password and complete email-based or Google Authenticator based authentication to get access to the target database.

Database Access Restriction

DataSunrise features Data Security component which enables you to restrict access to a complete database or certain database objects depending on the following factors:

database that contains the income of each employee of company.

Query-1:

```
SELECT COUNT(*)
FROM EMP_SALARY
WHERE Emp-department = '3';
```

Query-2:

```
SELECT AVG(income)
FROM EMP_SALARY
WHERE Emp-id = '2';
```

Here, the **" Where"** condition can be manipulated by attacker and there is chance to access income of individual employees or confidential data of employee if he knows id/name of particular employee.

The possibility of accessing individual information from statistical queries is reduced by using the following measures –

1. **Partitioning of Database** – This means the records of database must be not be stored as bulk in single record. It must be divided into groups of some minimum size according to confidentiality of records.

   The advantage of Partitioning of database is queries can refer to any complete group or set of groups, but queries cannot access the subsets of records within a group. So, attacker can access at most one or two groups which are less private.
2. If no statistical queries are permitted whenever number of tuples in population specified by selection condition falls below some threshold.
3. Prohibit sequences of queries that refer repeatedly to same population of tuples.