RDBMS -20UIT4CC7A
Unit 1

By
Zunaitha Sulthana AMS
Assistant Professor
Department of Computer Science & IT
Jamal Mohamed College
Tiruchirappalli-20

Content

- **Roles in Database Environment**
- **Advantages and Disadvantages**
- **The Three-Levels of ANSI-SPARC Architecture**
- **Database Languages**
- **Data Models**

Roll in the Database Environment

- **Data Administrator (DA)**
  - Database planning
  - Development and maintenance of standards, policies and procedures
- **Database Administrator (DBA)**
  - **Physical realization of the database**
  - **Physical database design and implementation**
  - **Security and integrity control**
  - **Maintenance of the operational system**
  - **Ensuring satisfactory performance of the applications for users**
- **Database Designers (Logical and Physical)**
- **Application Programmers**
- **End Users (naive and sophisticated)**

Database Design

- The structure of the database is determined during **database design**
- It can be an extremely complex task
- Need to think of the data first and then application -> paradigm shift

Roles in the Database Environment

- People: The fifth component in the DBMS environment
- There are 4 distinct types of people:
  - Data & Database Administrators
  - Database Designers
  - Application Developers
  - End-Users

Data & Database Administrators

- DA (data administrator) is responsible for the management of the data resource:
  - Database planning
  - Development & maintenance of standards
  - Policies and procedures
  - Conceptual/logical database design

Data & Database Administrators

- DBA (Database Administrator) is responsible for the physical realisation of the database:
  - Physical database design & implementation
  - Security & integrity control
  - Maintenance of operational system
  - Ensuring satisfactory performance of the applications for users

Database Designers

- Database designers is concerned with:
  - Identifying the data
  - Identifying relationship between entities and attributes
  - Identify the relationships between the data
  - Understand the constraints on the data (business rules)

Database Designers

- The work of the logical database designers can be split into two stages:
  - Conceptual database design
    - Independent of implementation details
    - Application programs
    - Programming languages
  - Logical database design
    - Specific data models
    - E.g.: relational, network, hierarchical or object-oriented

Database Designers

- **Physical database designer** decides how the logical database design is to be physically realised.
- It involves:
  - Mapping the logical database design into a set of tables & integrity constraints
  - Selecting specific storage structures and access methods for the data
  - Designing any security measures

Application Developers

- They worked from the specification produced by systems analysts
- Each program may contain statements that request the DBMS to perform some operation:
  - Retrieving data
  - Insert data
  - Delete data
  - Updating data

End Users

- Clients of the database
- Can be classified as:
  - Naïve users
    - Typically unaware of the DBMS
  - Sophisticated users
    - Familiar with the structure of the DBMS
    - May use a high-level query language to perform required operation

Advantages and Disadvantages

Advantages of DBMS

- Control of data redundancy
- Data consistency
- More information from the same amount of data
- Sharing of data
- Improved data integrity (constraints)
- Improved security (authentication, rights)

Advantages of DBMS

- Economy of scale (economical cost)
- Balance conflicting requirements
- Improved data accessibility and responsiveness (ad hoc queries)
- Increased productivity (developer)
- Improved maintenance through data independence

Disadvantages

- Complexity
- Size (disk space for DBMS)
- Cost of DBMS
- Additional hardware costs
- Cost of conversion
- Performance
- Higher impact of a failure

The Three-Levels of ANSI-SPARC Architecture

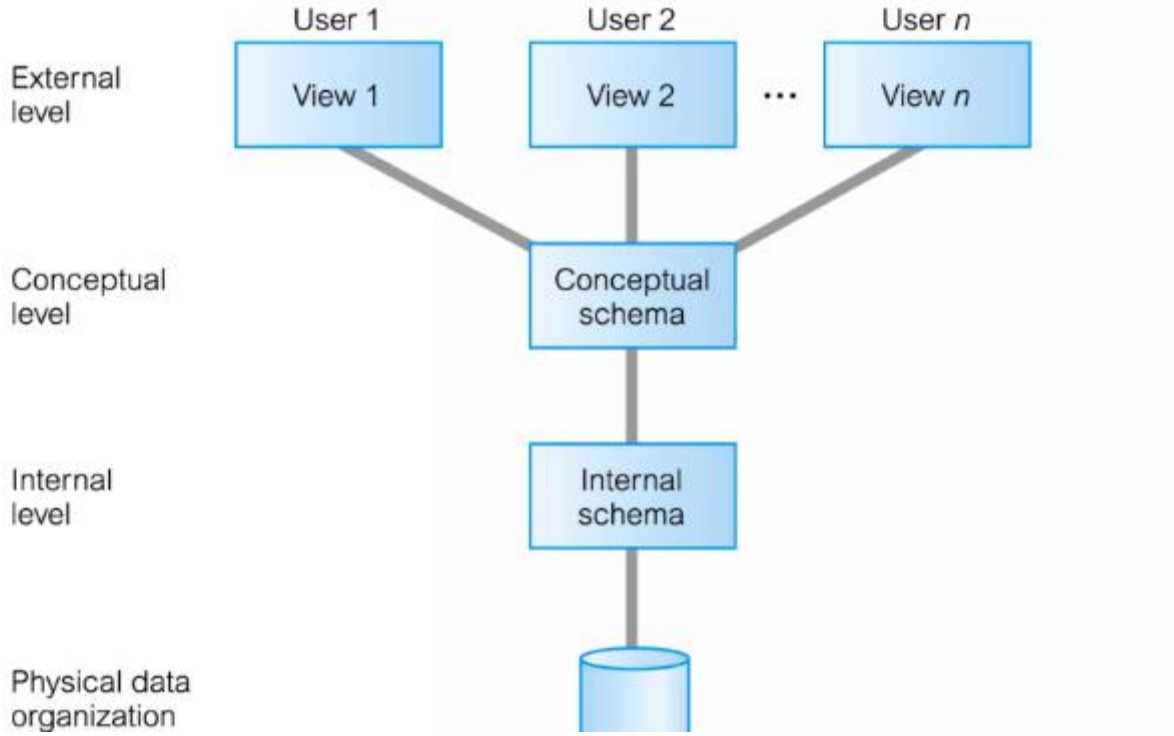
Objectives of Three-Level Architecture

- ANSI-SPARC Three Level Architecture
- All users should be able to access same data but have a different customized view
- A user's view is immune to changes made in other views
- Users should not need to know physical database storage details

Objectives of Three-Level Architecture..

- DBA should be able to change database storage structures without affecting the users' views
- Internal structure of database should be unaffected by changes to physical aspects of storage
- DBA should be able to change conceptual structure of database without affecting all Users

ANSI-SPARC Three-Level Architecture



Database

ANSI-SPARC Three-Level Architecture..

- **External Level**
  - **Users' view of the database**
  - **Describes that part of database that is relevant to a particular user**
  - **Different views may have different representation of same data (e.g. different date formats, age derived from DOB etc.)**

ANSI-SPARC Three-Level Architecture..

- **Conceptual Level**
  - **Community view of the database**
  - **Describes what data is stored in database and relationships among the data**
  - **Along with any constraints on data**
  - **Independent of any storage considerations**
    i. all entities, their attributes, and their relationships;
    ii. the constraints on the data;
    iii. semantic information about the data

    security and integrity information.

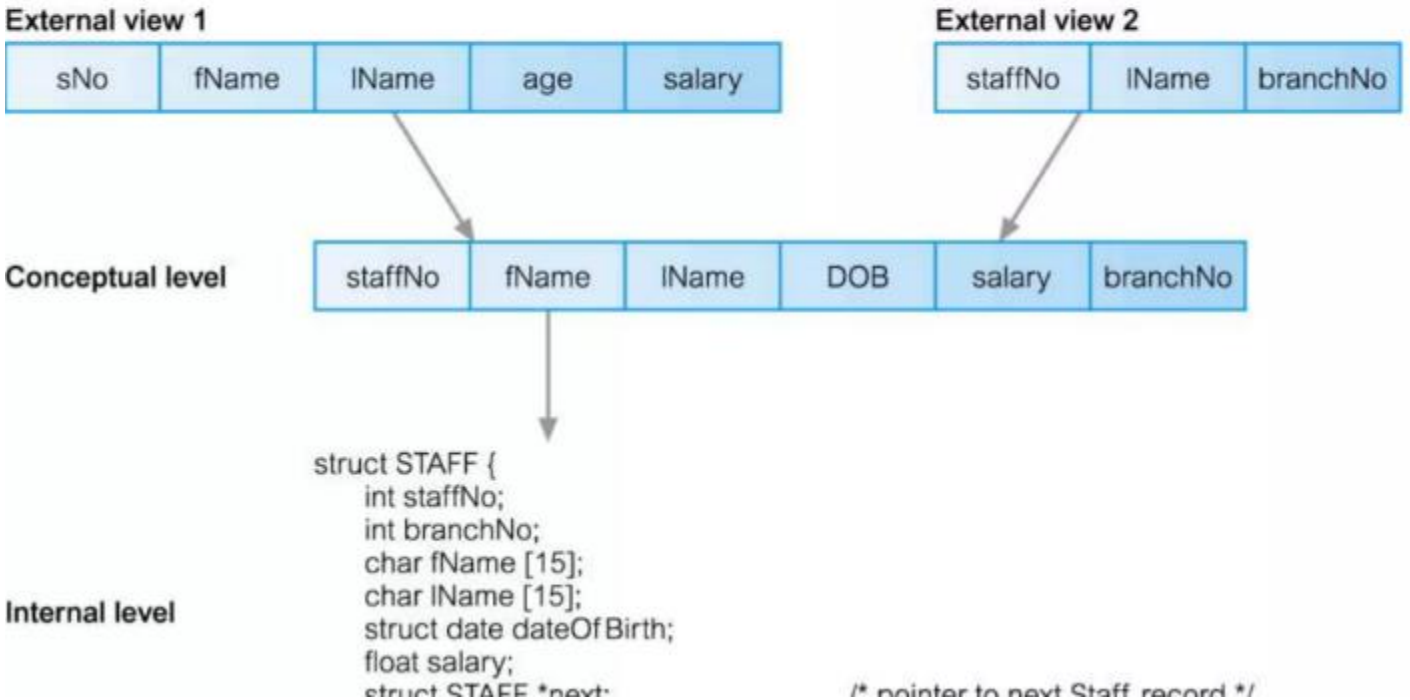
ANSI-SPARC Three-Level Architecture..

- **Internal Level**
- storage space allocation for data and indexes;
- record descriptions for storage (with stored sizes for data items);
- record placement;
- data compression and data encryption techniques.

ANSI-SPARC Three-Level Architecture..

- **Internal Level**
  - Physical representation of the database on the computer
  - Describes how the data is stored in the database
  - physical implementation of the database to achieve optimal runtime performance and storage space utilization
  - Data structures and file organizations used to store data on storage devices
  - Interfaces with the operating system access methods to place the data on the storage devices, build the indexes, retrieve the data, and so on

Differences between Three Levels of ANSI-SPARC Architecture



Schemas

- **External Schemas**
  - Also called subschemas
  - Multiple schemas per database
  - Corresponds to different views of data
- **Conceptual Schema**
  - Describes all the entities, attributes, and relationships together with integrity constraints
  - Only one schema per database

Mappings

- **The DBMS is responsible for mapping between these three types of schema:**
  - **The DBMS must check that each external schema is derivable from the conceptual schema, and it must use the information in the conceptual schema to map between each external schema and the internal schema**
- **Types of mappings**
  - **Conceptual/Internal mapping**
  - **External/Conceptual mapping**

Conceptual/Internal Mapping

- **Enables the DBMS to**
  - **Find the actual record or combination of records in physical storage that constitute a logical record in the conceptual schema,**
  - **Together with any constraints to be enforced on the operations for that logical record**
  - **It also allows any differences in entity names, attribute names, attribute order, data types, and so on, to be resolved**

External/Conceptual Mapping

- **Enables the DBMS to**
  - **Map names in the user's view on to the relevant part of the conceptual schema**

Instances

- **Database Schema**
  - Description of database (also called intension)
  - Specified during design phase
  - Remain almost static
- **Database Instance**
  - Data in the database at any particular point in time
  - Dynamic (changes with the time)
  - Also called an extension (or state) of database

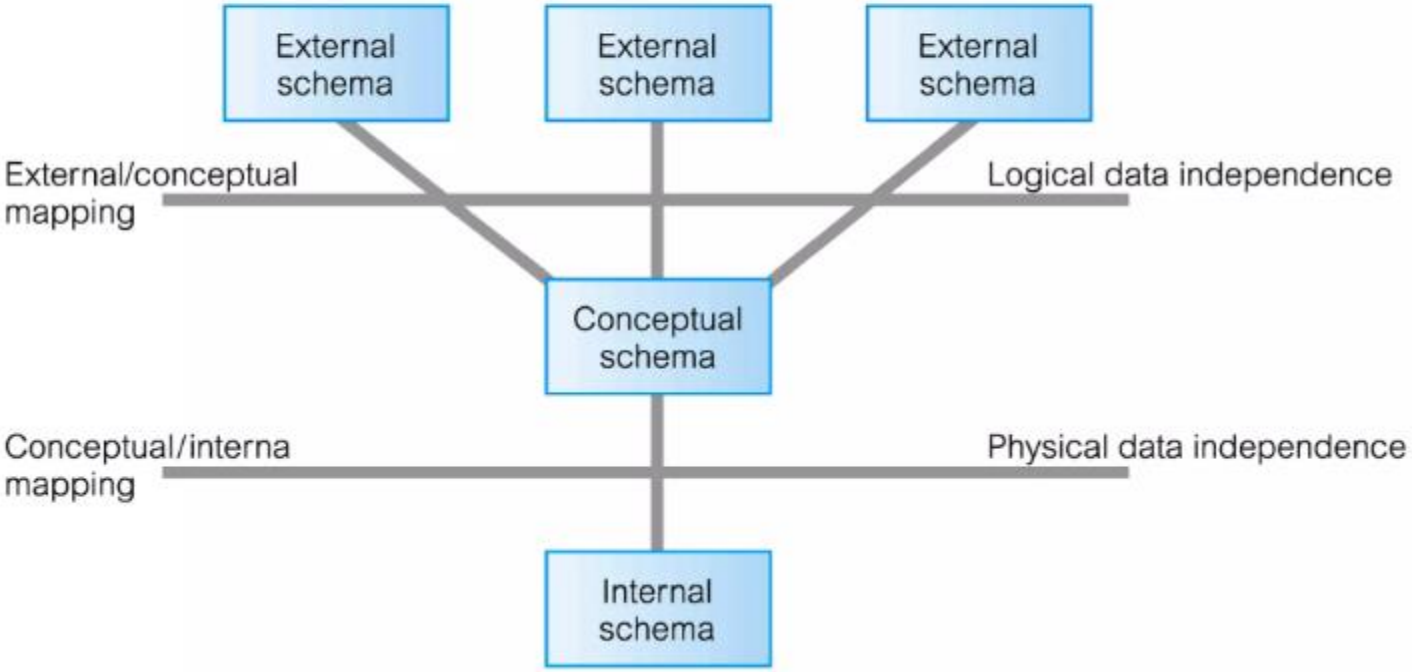
Data Independence

- **Logical Data Independence**
  - Refers to protection of external schemas to changes in conceptual schema
  - Conceptual schema changes (e.g. addition/removal of entities)
  - Should not require changes to external schema or rewrites of application programs

Data Independence

- **Physical Data Independence**
  - Refers to immunity of conceptual schema to changes in the internal schema
  - Internal schema changes (e.g. using different file organizations, storage structures, storage devices etc.)
  - Should not require change to conceptual or external schemas

Data Independence and the ANSI-SPARC Three-Level Architecture



Database Languages

Database Languages

- **Data sublanguage consist of two parts:**
  - **DDL (Data Definition Language)**
  - **DML (Data Manipulation Language)**
- **Data sublanguage**
  - **Does not include constructs for all computing needs such as iterations or conditional statements**
  - **Many DBMSs provide embedding the sublanguage in a high level programming language e.g. C, C++, Java etc.**
  - **In this case , these high level languages are called host languages**

Data Definition Language (DDL)

- Allows the DBA or user to describe and name entities, attributes, and relationships required for the application
- Plus any associated integrity and security constraints
- System catalog (data dictionary, data directory)
- Metadata (data about data, data description, data definitions)

Data Manipulation Language (DML)

- Provides basic data manipulation operations on data held in the database
  - Procedural DML
  - Non-Procedural DML

Procedural DML

- **Allows user to tell system exactly how to manipulate data**
  - **Operate on records individually**
  - **Typically, embedded in a high level language**
  - **Network or hierarchical DMLs**
  - **More work is done by user (programmer)**

Non-Procedural DML

- **Allows user to state what data is needed rather than how it is to be retrieved**
  - **Operate on set of records**
  - **Relational DBMS include e.g. SQL, QBE etc.**
  - **Easy to understand and learn than procedural DML**
  - **More work is done by DBMS than user**
  - **Provides considerable degree of data independence**
  - **Also called declarative languages**

Fourth Generation Languages (4GLs)

- **No clear consensus**
  - **Forms generators**
  - **Report generators**
  - **Graphics generators**
  - **Application generators**
  - **Examples : SQL and QBE**

Data Models

Data Model

- **Integrated collection of concepts for describing data, relationships between data, and constraints on the data in an organization**

Purpose of Data Model

- **To represent data in an understandable way**
  - **Represents the organization itself**
  - **Helps in unambiguous and accurate communication between between database designers and end-users about their understanding of the organizational data**

Components of a Data Model

- **A data model comprises:**
  - **A structural part**
  - **A manipulative part**
  - **Possibly a set of integrity rules**
  - **ANSI-SPARC architecture related models**
    - **External data model (Universe of Discourse)**
    - **Conceptual data model (DBMS independent)**
    - **Internal data model**

Categories of Data Models

- **Categories of data models include:**
  - **Object-based**
    - **Entity-Relationship**
    - **Semantic**
    - **Functional**
    - **Object-Oriented**
  - **Record-based**
    - **Relational Data Model**
    - **Network Data Model**
    - **Hierarchical Data Model**
  - **Physical**

Relational Model

The data in this model is kept in the form of a table that is two-dimensional. All of the data is kept in the form of rows and columns. Tables are the foundation of a relational paradigm.

Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them.

A set of attributes describe the entities.
For example, student_name, student_id describes the 'student' entity.
A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

Semantic Data Model
It is a data model defined on a higher level that captures the databases' semantic description, structure, and form.

Functional Data Models

Functional Data Models are a form of *Semantic Data Model* which appeared early in database history. They use the mathematical formalism of *function application* to represent and follow associations between data items. Functions are usually applied to variables whose values may be object identifiers or record instances.

EG : *name(town(P)) = P.town.name = "Aberdeen"*.

Object Oriented Data Model

In Object Oriented Data Model, data and their relationships are contained in a single structure which is referred as object in this data model. In this, real world problems are represented as objects with different attributes. All objects have multiple relationships between them. Basically, it is combination of Object Oriented programming and Relational Database Model

Relational Data Model

Branch

branchNo	street	city	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

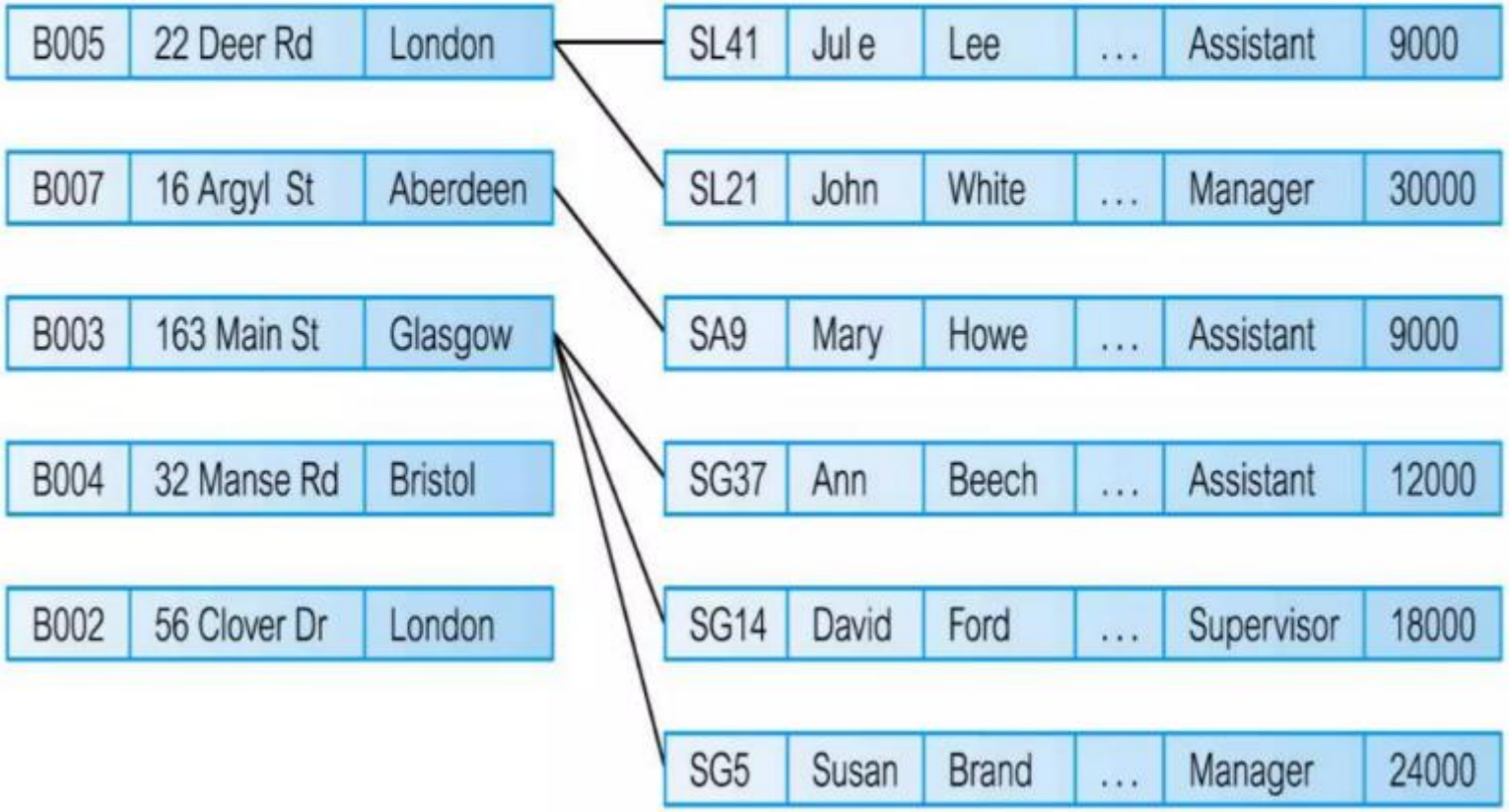
Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Network Model

The main difference between this model and the hierarchical model is that any record can have several parents in the network model. It uses a graph instead of a hierarchical tree.

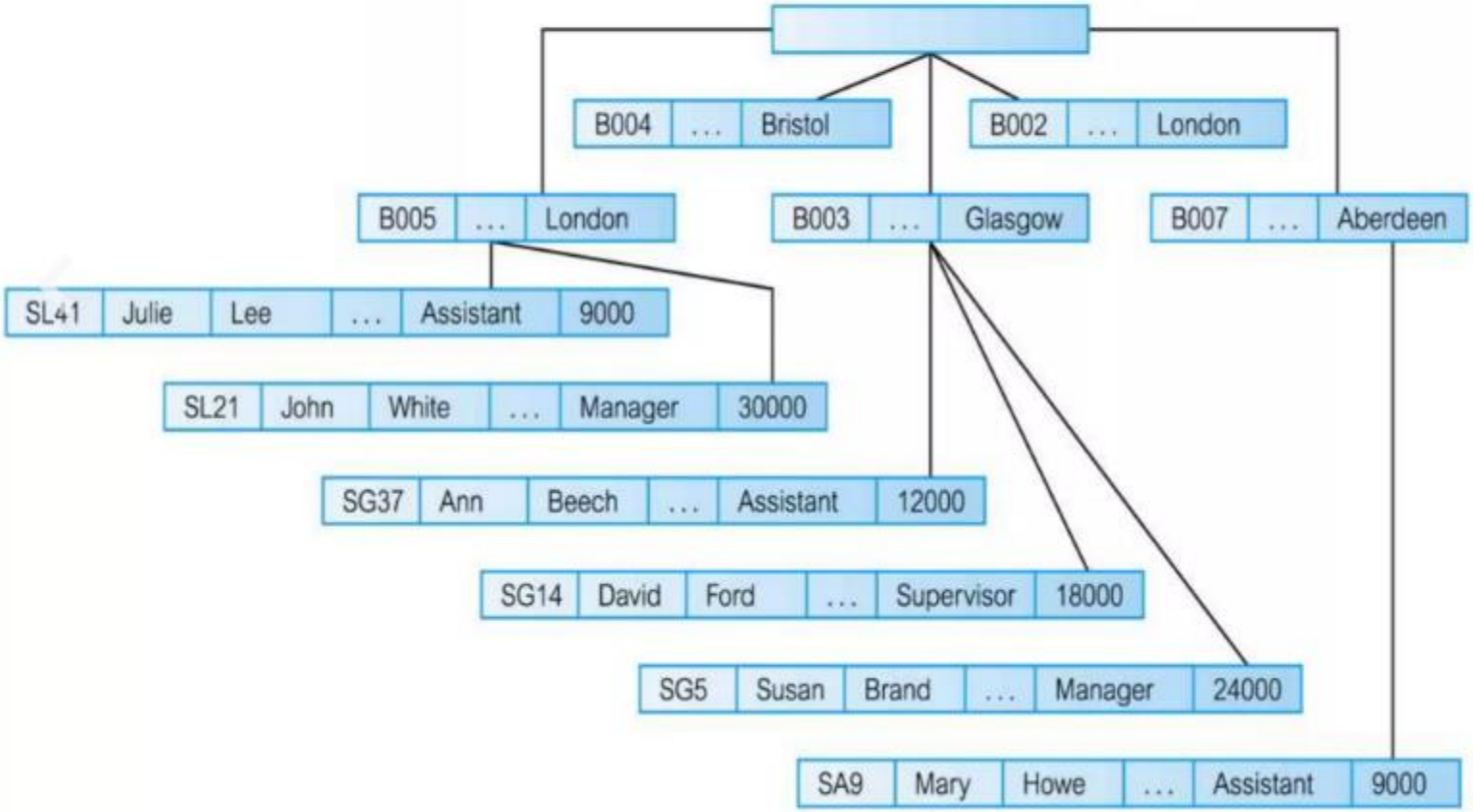
Network Data Model



Hierarchical Model

This concept uses a hierarchical tree structure to organize the data. The hierarchy begins at the root, which contains root data, and then grows into a tree as child nodes are added to the parent node.

Hierarchical Data Model



Physical Data Model

Physical data Models describes how data is stored in the computer , representing information such as record structures , record orderings and access paths.
Most common are unifying model and frame memory.

Conceptual Modeling

- **Conceptual modeling is process of developing a model of information use in an enterprise that is independent of implementation details**
  - **Should be complete and accurate representation of an organization's data requirements**
  - **Conceptual schema is the core of a system supporting all user views**
- **Conceptual vs. logical data model**

UNIT 5 PL/SQL

5.1 Introduction to PL/SQL

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.

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.

PL/SQL-Basic Syntax
PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts

Declarations
This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

Executable Commands

This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

Exception Handling

This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

```
DECLARE
<declarations section>
BEGIN
<executable command(s)>
EXCEPTION
<exception handling>
END;
```

Example 'Hello World'

```
DECLARE
message varchar2(20):= 'Hello, World!';
BEGIN
dbms_output.put_line(message);
END;
```

5.2 PL/SQL– Variables

The name of a PL/SQL variable consists of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters. By default, variable names are not case-sensitive. You cannot use a reserved PL/SQL keyword as a variable name.

Variable Declaration in PL/SQL

PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

The syntax for declaring a variable is

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Initializing Variables in PL/SQL

Whenever you declare a variable, PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either ofthe following

The DEFAULT keyword The assignment operator

For example −

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

You can also specify that a variable should not have a NULL value using the NOT NULL constraint. If you use the NOT NULL constraint, you must explicitly assign an initial value for that variable.

It is a good programming practice to initialize variables properly otherwise, sometimes programs would produce unexpected results. Try the following example which makes use of various types of variables

```
DECLARE
    a integer := 10;
    b integer := 20;
    c integer;
    f real;
BEGIN
    c := a + b;
    dbms_output.put_line('Value of c: ' || c);
    f := 70.0/3.0;
    dbms_output.put_line('Value of f: ' || f);
END;
/
```

Variable Scope in PL/SQL

PL/SQL allows the nesting of blocks, i.e., each program block may contain another inner block. If a variable is declared within an inner block, it is not accessible to the outer block. However, if a variable is declared and accessible to an outer block, it is also accessible to all nested inner blocks.

There are two types of variable scope

- Local variables − Variables declared in an inner block and not accessible to outer blocks.
- Global variables − Variables declared in the outermost block or a package.

Following example shows the usage of Local and Global variables in its simple form –

```
DECLARE
    -- Global variables
    num1 number := 95;
    num2 number := 85;
BEGIN
    dbms_output.put_line('Outer Variable num1: ' || num1);
    dbms_output.put_line('Outer Variable num2: ' || num2);

DECLARE
-- Local variables
    num1 number := 195;
    num2 number := 185;
BEGIN
    dbms_output.put_line('Inner Variable num1: ' || num1);
    dbms_output.put_line('Inner Variable num2: ' || num2);
END;
END;
```

5.3 PL / SQL Data Types

Category	**Description**
Scalar	Single values with no internal components, such as a NUMBER, DATE, or BOOLEAN.
Large Object LOB	Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
Composite	Data items that have internal components that can be accessed individually. For example, collections and records.
Reference	Pointers to other data items.

Scalar Data types:

CHAR	fixed length character data	Char,varchar,raw nchar, long
VARCHAR 2	variable character length data	
NUMBER	Fixed or floating point numbers of vitually any size	DECIMALprec, scale, NUMERICpre, secale, FLOAT, INT, INTEGER,

		SMALLINT, REAL
BINARY INTEGER	Integers	
DATE	date values	YEAR,MONTH, DAY,HOUR,MINUTE,SECOND
BOOLEAN	TRUE or FALSE Note: there is no BOOLEAN data type in database table.	BINARY_INTEGER, BINARY_FLOAT, BINARY_DOUBLE

Following is a valid declaration:

```
DECLARE
    num1 INTEGER;
    num2 REAL;
    num3 DOUBLE PRECISION;
BEGIN
    null;
END;
/
```

Composite Data types:

Table

1. Is similar but not the same as a database table
2. Must contain only one column of any scalar datatype
3. Is like a one-dimensional array of any size
4. Has its elements indexed with a binary integer column called the primary key of the table

Record

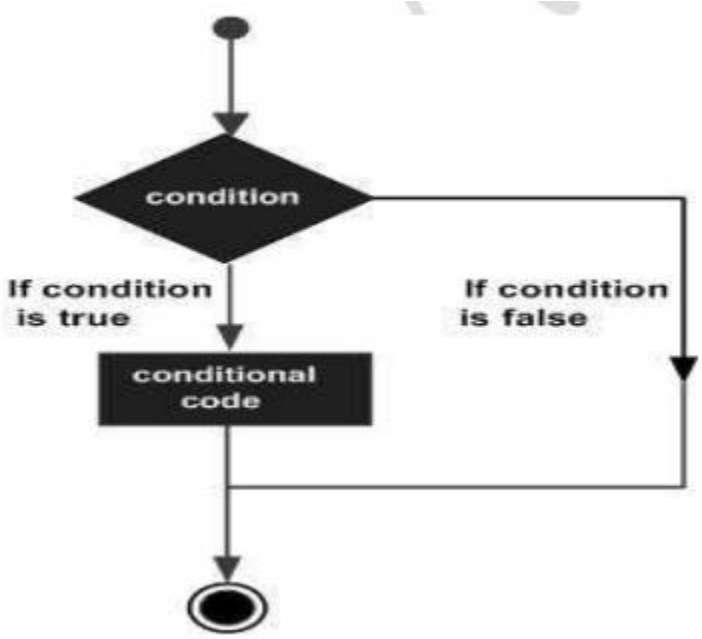
1. Contains uniquely defined columns of different data types,
2. Enables us to treat dissimilar columns that are logically related as a single unit.

5.4 PL/SQL Control Structures

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

1. if then statement
2. if then else statements
3. nested if-then statements
4. if-then-elseif-then-else ladder

Following is the general form of a typical conditional (i.e., decision making) structure found in most of the programming languages



PL/SQL programming language provides following types of decision-making statements.

Syntax
IF condition
THEN
Statement: {It is executed when condition is true}
END IF;

Eg:

```
declare
        num1 number:= 10;
        num2 number:= 20;

begin

        if num1 > num2 then
        dbms_output.put_line('num1 small');
end if;

        dbms_output.put_line('I am Not in if');

end;
```

EG :

```
declare
        num1 number:= 10;
        num2 number:= 20;
        num3 number:= 20;

begin
        if num1 < num2 then
        dbms_output.put_line('num1 small num2');
          if num1 < num3 then
         dbms_output.put_line('num1 small num3 also');
          end if;
         end if;
    dbms_output.put_line('after end if');
end;
```

5.5 PL/SQL - Cursors

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor

holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time.

There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.
- Programmers cannot control the implicit cursors and the information in it.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement.
- For INSERT operations, the cursor holds the data that needs to be inserted.
- For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.
- Recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.
- The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS, designed for use with the FORALL statement.
- The following table provides the description of the most used attributes –
- %FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECTINTO statement returned one or more rows. Otherwise, it returns FALSE.

- %NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
- %ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
- %ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

```
DECLARE
    total_rows number(2);
```

```
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
        dbms_output.put_line('no customers selected');
ELSIF sql%found THEN
        total_rows := sql%rowcount;
        dbms_output.put_line( total_rows || ' customers selected ');
END IF;
END;
```

When the above code is executed at the SQL prompt, it produces the following result −
6 customers selected
PL/SQL procedure successfully completed.

Explicit Cursors
Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block.
It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −
CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor
Declaring the cursor defines the cursor with a name and the associated SELECT statement.
For example
−CURSOR c_customers IS SELECT id, name, address FROM customers;

Opening the Cursor
Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.
For example, we will open the above defined cursor as follows
− OPEN c_customers;

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows

- FETCH c_customers INTO c_id, c_name, c_addr;

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows

- CLOSE c_customers;

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
        c_id customers.id%type;
        c_name customerS.No.ame%type;
        c_addr customers.address%type;
        CURSOR c_customers is
SELECT id, name, address FROM customers;
BEGIN
        OPEN c_customers;
        LOOP
         FETCH c_customers into c_id, c_name, c_addr;
                EXIT WHEN c_customers%notfound;
                dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
        END LOOP;
        CLOSE c_customers;
END;
```

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

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP
PL/SQL procedure successfully completed.
```

5.6 Iterative Control Statement

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages

```
BEGIN
  FOR i IN 1 .. 10 LOOP
    DBMS_OUTPUT.PUT_LINE('Iteration # ' || i);
  END LOOP;
END;
```

```
Iteration # 1
Iteration # 2
Iteration # 3
Iteration # 4
Iteration # 5
Iteration # 6
Iteration # 7
Iteration # 8
Iteration # 9
Iteration # 10
```

A WHILE LOOP statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

```
WHILE condition LOOP
  sequence_of_statements
END LOOP;
```

Example

```
DECLARE
  a number(2) := 10;
BEGIN
  WHILE a < 20 LOOP
    dbms_output.put_line('value of a: ' || a);
    a := a + 1;
  END LOOP;
END;
/
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
```

```
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

PL/SQL procedure successfully completed.
```

5.7 PL/SQL Exception Handling

This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

```
DECLARE
  <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

The 'Hello World' Example

```
DECLARE
        message varchar2(20):= 'Hello, World!';
BEGIN
        dbms_output.put_line(message);
END;
```

5.8 PL/SQL-Triggers

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

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

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 is
− CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [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.

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

Example
The following program creates a row-level trigger for the customers table that would fire for INSERT orUPDATE or DELETE operations performed on the CUSTOMERS table.
This trigger will display the salary difference between the old values and new values

```
– 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;
/
```

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

5.9 PL/SQL- Procedures

A subprogram is a program unit/module that performs a particular task. These subprograms are combined to form larger programs.
This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.

A subprogram can be created –

- At the schema level
- Inside a package
- Inside a PL/SQL block

At the schema level, subprogram is a standalone subprogram. It is created with the CREATEPROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.
A subprogram created inside a package is a packaged subprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters.

PL/SQLprovides two kinds of subprograms −

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

Parts of a PL/SQLSubprogram

Declarative Part
It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
Executable Part
This is a mandatory part and contains statements that perform the designated action.
Exception-handling
This is again an optional part. It contains the code that handles run-time errors.
Creating a Procedure
A procedure is created with the CREATE OR REPLACE PROCEDUREstatement. The simplified

syntax for the CREATE OR REPLACE PROCEDURE statement is as follows

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

Where,

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

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line('Hello World!');
END;
/
```

When the above code is executed using the SQL prompt, it will produce the following result −
Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

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

The above procedure named 'greetings' can be called with the EXECUTE keyword as
EXECUTE greetings;

The above call will display − Hello World
PL/SQL procedure successfully completed.
The procedure can also be called from another PL/SQL block
− BEGIN
greetings;
END;
/
The above call will display − Hello World
PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is
− DROP PROCEDURE procedure-name;
You can drop the greetings procedure by using the following statement
− DROP PROCEDURE greetings;

Methods for Passing Parameters

Actual parameters can be passed in three ways −

- Positional notationfindMin(a, b, c, d);
- Named notation

findMin(x => a, y => b, z => c, m => d);

- Mixed notation

findMin(a, b, c, m => d);
