NotesHuB

(NotesHub.co.in)

CSE: Advance Database Management Systems (ADBMS)

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Distributed DBMS Architectures

DDBMS architectures are generally developed depending on three parameters –

- **Distribution** – It states the physical distribution of data across the different sites.
- **Autonomy** – It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.
- **Heterogeneity** – It refers to the uniformity or dissimilarity of the data models, system components and databases.

Architectural Models

Some of the common architectural models are –

- Client - Server Architecture for DDBMS
- Peer - to - Peer Architecture for DDBMS
- Multi - DBMS Architecture

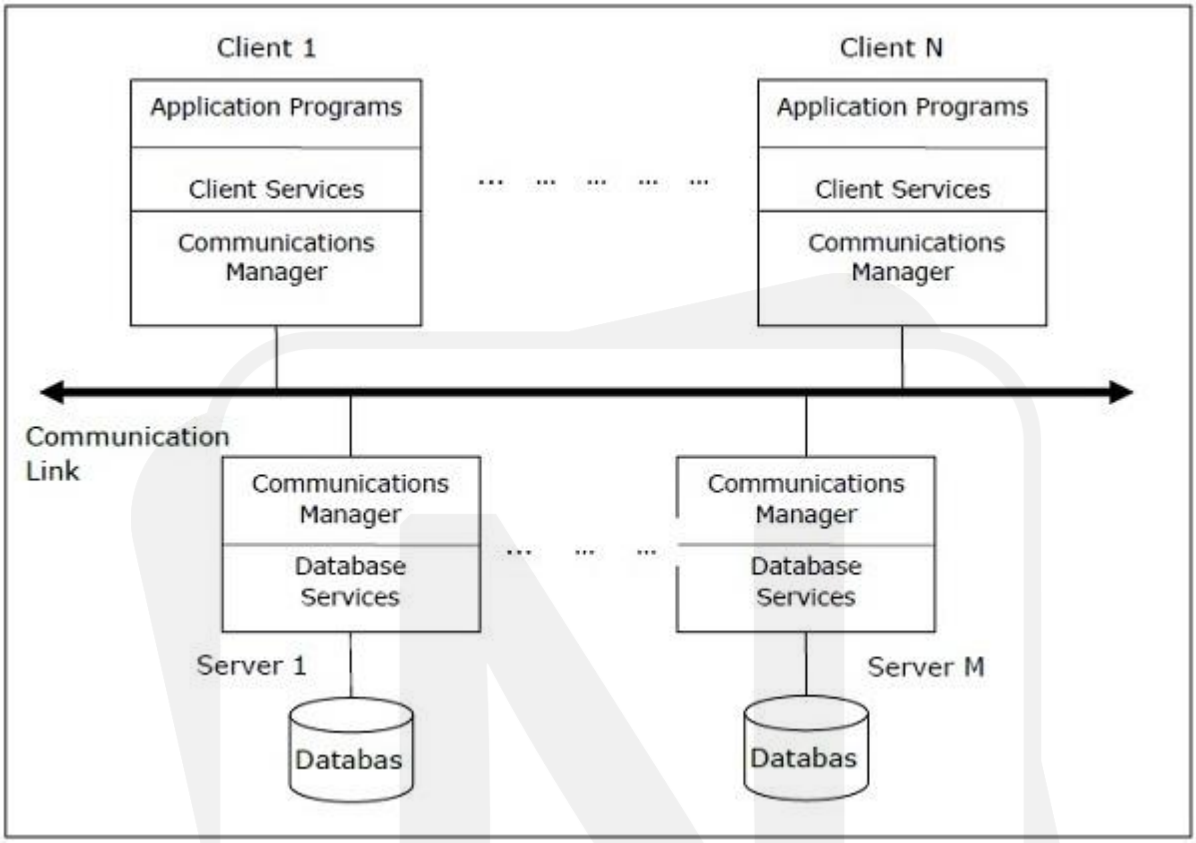
Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management.Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The two different client - server architecture are –

- Single Server Multiple Client

- Multiple Server Multiple Client (shown in the following diagram)

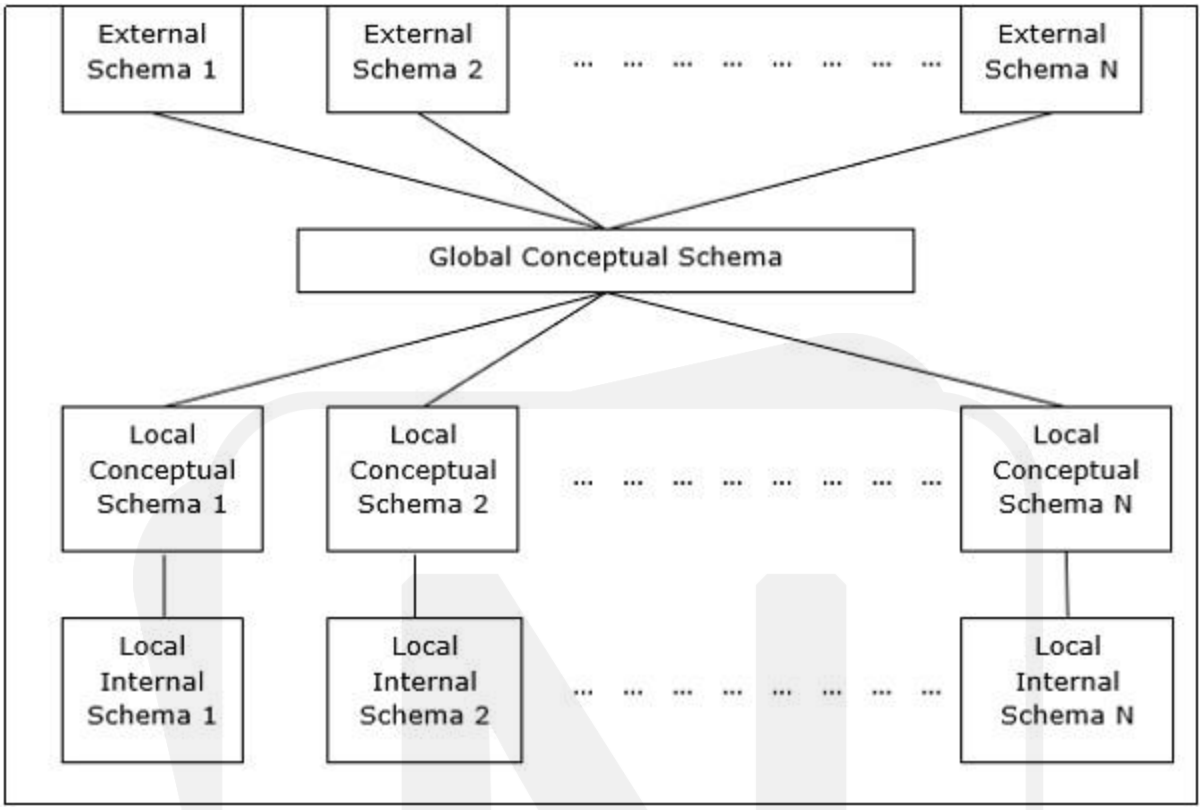


Peer- to-Peer Architecture for DDBMS

In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas –

- **Global Conceptual Schema** – Depicts the global logical view of data.
- **Local Conceptual Schema** – Depicts logical data organization at each site.
- **Local Internal Schema** – Depicts physical data organization at each site.
- **External Schema** – Depicts user view of data.



Multi - DBMS Architectures

This is an integrated database system formed by a collection of two or more autonomous database systems.

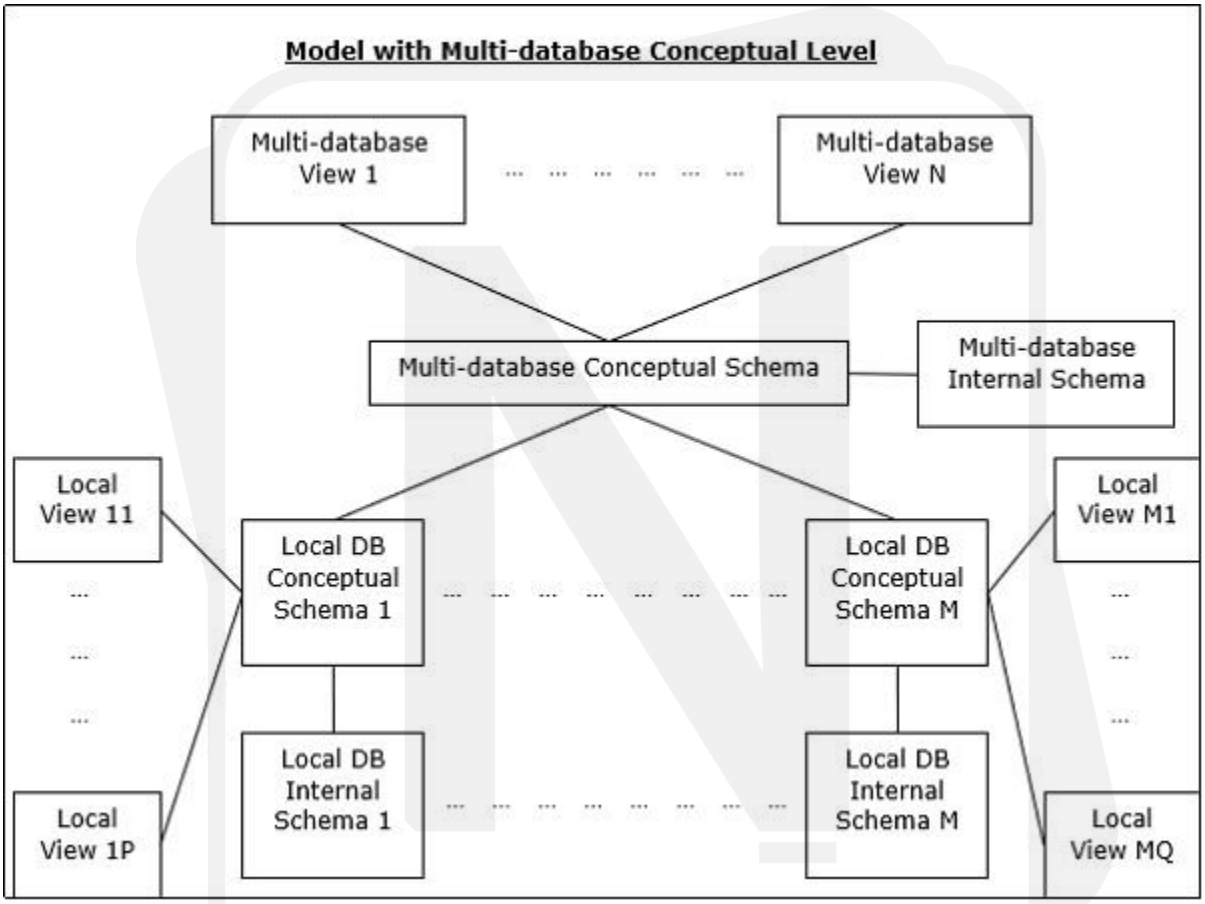
Multi-DBMS can be expressed through six levels of schemas –

- **Multi-database View Level** – Depicts multiple user views comprising of subsets of the integrated distributed database.
- **Multi-database Conceptual Level** – Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
- **Multi-database Internal Level** – Depicts the data distribution across different sites and multi-database to local data mapping.
- **Local database View Level** – Depicts public view of local data.
- **Local database Conceptual Level** – Depicts local data organization at each site.

- **Local database Internal Level** – Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS –

- Model with multi-database conceptual level.
- Model without multi-database conceptual level.



Advantages of distributed database:

1) In a distributed database, data can be stored in different systems like personal computers, servers, mainframes, etc.

2) A user doesn't know where the data is located physically. Database presents the data to the user as if it were located locally.

3) Database can be accessed over different networks.

4) Data can be joined and updated from different tables which are located on different machines.

5) Even if a system fails the integrity of the distributed database is maintained.

6) A distributed database is secure.

Disadvantages of distributed database:

1) Since the data is accessed from a remote system, performance is reduced.

2) Static SQL cannot be used.

3) Network traffic is increased in a distributed database.

4) Database optimization is difficult in a distributed database.

5) Different data formats are used in different systems.

6) Different DBMS products are used in different systems which increases in complexity of the system.

7) Managing system catalog is a difficult task.

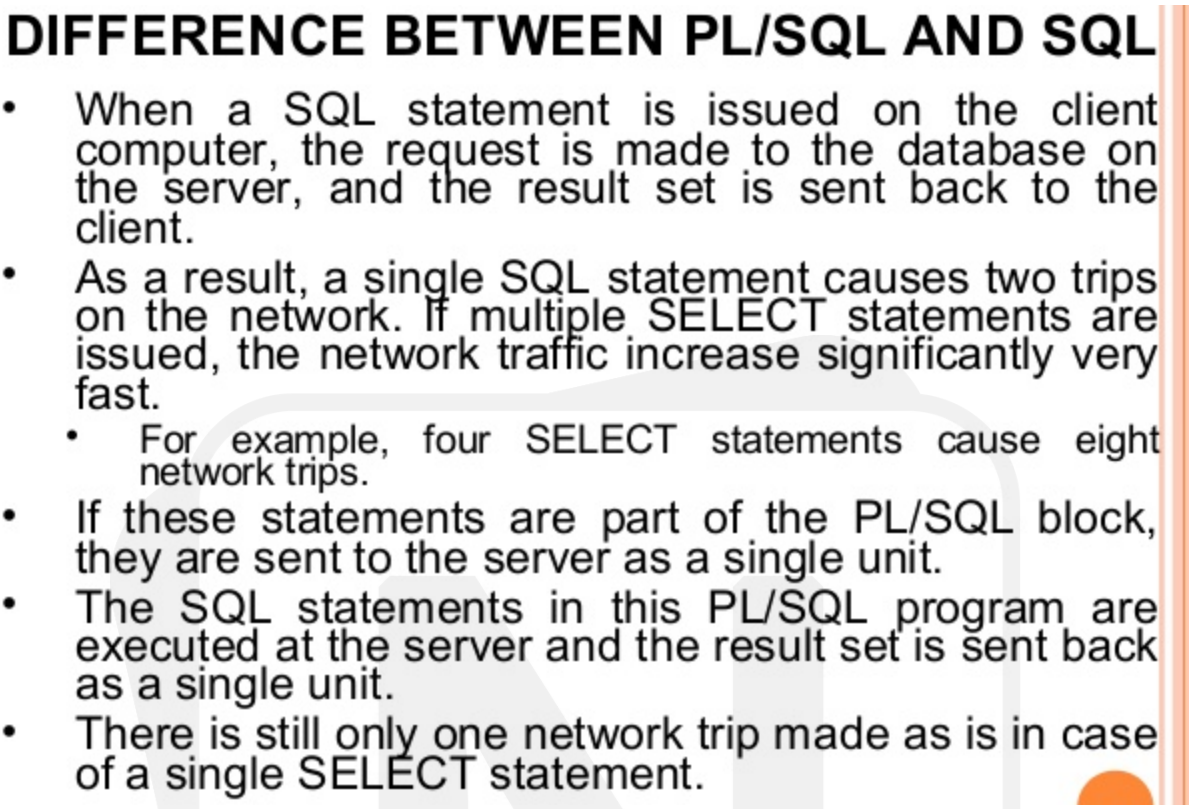
8) While recovering a failed system, the DBMS has to make sure that the recovered system is consistent with other systems.

9) Managing distributed deadlock is a difficult task.

What's New in Database Technologies & Tools?

Read from this link

https://www.hotscripts.com/blog/whats-database-technologies-tools/



What is PL/SQL?

PL/SQL stands for **Procedural Language** extension of SQL.

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

PL/SQL 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.

How to pass parameters in procedure:

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

1. **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.
2. **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.
3. **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

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:

```
1. create or replace procedure "INSERTUSER"
2. (id IN NUMBER,
3. name IN VARCHAR2)
4. is
5. begin
6. insert into user values(id,name);
7. end;
8. /
```

Output:

```
Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
1. BEGIN
2.   insertuser(101,'Rahul');
3.   dbms_output.put_line('record inserted successfully');
4. END;
5. /
```

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

ID	Name
101	Rahul

PL/SQL Drop Procedure

Syntax for drop procedure

```
1. DROP PROCEDURE procedure_name;
```

Example of drop procedure

```
1. DROP PROCEDURE pro1;
```

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:

```
1. CREATE [OR REPLACE] FUNCTION function_name [parameters]
2. [(parameter_name [IN | OUT | IN OUT] type [, ...])]
3. RETURN return_datatype
4. {IS | AS}
5. BEGIN
6.    < function_body >
7. 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.

The function must contain a return statement.

- 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

Let's see a simple example to **create a function**.

```
1. create or replace function adder(n1 in number, n2 in number)
2. return number
3. is
4. n3 number(8);    // Declaration
5. begin
```

```
6. n3 :=n1+n2;
7. return n3;
8. end;
9. /
```

Now write another program to **call the function**.

```
1. DECLARE
2.    n3 number(2);
3. BEGIN
4.    n3 := adder(11,22);
5.    dbms_output.put_line('Addition is: ' || n3);
6. END;
7. /
```

Output:

```
Addition is: 33
Statement processed.
0.05 seconds
```

Another PL/SQL Function Example

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return the maximum of two values.

```
1. DECLARE
2.    a number;
3.    b number;
4.    c number;
5. FUNCTION findMax(x IN number, y IN number)
6. RETURN number
7. IS
8.     z number;
9. BEGIN
10.   IF x > y THEN
11.     z:= x;
12.   ELSE
13.     Z:= y;
14.   END IF;
```

```
15.
16.    RETURN z;
17. END;
18. BEGIN   // Calling
19.    a:= 23;
20.    b:= 45;
21.
22.    c := findMax(a, b);
23.    dbms_output.put_line(' Maximum of (23,45): ' || c);
24. END;
25. /
```

Output:

```
Maximum of (23,45): 45
Statement processed.
0.02 seconds
```

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.

Create customers table and have records in it.

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:

```
1. CREATE OR REPLACE FUNCTION totalCustomers
```

```
2. RETURN number
3. IS
4.    total number(2) := 0;
5. BEGIN
6.    SELECT count(*) into total
7.    FROM customers;
8.     RETURN total;
9. END;
10. /
```

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

```
Function created.
```

Calling PL/SQL Function:

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. 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 you have to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function totalCustomers from an anonymous block:

```
1. DECLARE
2.    c number(2);
3. BEGIN
4.    c := totalCustomers();
5.    dbms_output.put_line('Total no. of Customers: ' || c);
6. END;
7. /
```

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

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

PL/SQL Recursive Function

You already know that a program or a subprogram can call another subprogram. When a subprogram calls itself, it is called recursive call and the process is known as recursion.

Example to calculate the factorial of a number

Let's take an example to calculate the factorial of a number. This example calculates the factorial of a given number by calling itself recursively.

```
1.  DECLARE
2.     num number;
3.     factorial number;
4.
5.  FUNCTION fact(x number)
6.  RETURN number
7.  IS
8.     f number;
9.  BEGIN
10.    IF x=0 THEN
11.       f := 1;
12.    ELSE
13.       f := x * fact(x-1);
14.    END IF;
15. RETURN f;
16. END;
17.
18. BEGIN
19.    num:= 6;
20.    factorial := fact(num);
21.    dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
22. END;
23. /
```

After the execution of above code at SQL prompt, it produces the following result.

```
Factorial 6 is 720
PL/SQL procedure successfully completed.
```

PL/SQL Drop Function

Syntax for removing your created function:

If you want to remove your created function from the database, you should use the following syntax.

1. **DROP FUNCTION** function_name;

PL/SQL packages are schema objects that groups logically related PL/SQL types, variables and subprograms.

A package will have two mandatory parts:

- Package specification
- Package body or definition

Package Specification

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

Package Body

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from code outside the package.

The CREATE PACKAGE BODY Statement is used for creating the package body.

Using the Package Elements

The package elements (variables, procedures or functions) are accessed with the following syntax:

```
package_name.element_name;
```

Example:

The following program provides a more complete package. We will use the CUSTOMERS table stored in our database with the following records:

```
Select*fromcustomers;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|3000.00|
|2|Khilan|25|Delhi|3000.00|
|3| kaushik  |23|Kota|3000.00|
|4|Chaitali|25|Mumbai|7500.00|
|5|Hardik|27|Bhopal|9500.00|
|6|Komal|22| MP        |5500.00|
+----+----------+-----+-----------+----------+
```

THE PACKAGE SPECIFICATION:

```
CREATE OR REPLACE PACKAGE c_package AS
--Adds a customer
PROCEDURE addCustomer(c_id   customers.id%type,
 c_namecustomers.name%type, // c_name is the name of variable and its type is name of
                                    customers.
   c_age  customers.age%type,
   c_addr customers.address%type,
   c_sal  customers.salary%type);

--Removes a customer
   PROCEDURE delCustomer(c_id  customers.id%TYPE);
--Lists all customers
   PROCEDURE listCustomer;

END c_package;
/
```

When the above code is executed at SQL prompt, it creates the above package and displays the following result:

```
Package created.
```

CREATING THE PACKAGE BODY:

```
CREATE OR REPLACE PACKAGE BODY c_package AS
   PROCEDURE addCustomer(c_id  customers.id%type,
      c_name customers.name%type,
      c_age  customers.age%type,
      c_addr  customers.address%type,
      c_sal   customers.salary%type)
   IS
BEGIN
      INSERT INTO customers (id,name,age,address,salary)
         VALUES(c_id, c_name, c_age, c_addr, c_sal);
END addCustomer;

   PROCEDURE delCustomer(c_id   customers.id%type) IS
BEGIN
       DELETE FROM customers
         WHERE id = c_id;
END delCustomer;

   PROCEDURE listCustomer IS
   CURSOR c_customers is
      SELECT  name FROM customers;
   TYPE c_list is TABLE OF customers.name%type;
   name_list c_list := c_list();
   counter integer :=0;
BEGIN
      FOR n IN c_customers LOOP
      counter := counter +1;
      name_list.extend;
      name_list(counter):= n.name;
      dbms_output.put_line('Customer('||counter||')'||name_list(counter));
```

```
END LOOP;
END listCustomer;
END c_package;
/
```

Above example makes use of **nested table** which we will discuss in the next chapter. When the above code is executed at SQL prompt, it produces the following result:

```
Package body created.
```

USING THE PACKAGE:

The following program uses the methods declared and defined in the package *c_package*.

```
DECLARE
   code customers.id%type:=8;
BEGIN
      c_package.addcustomer(7,'Rajnish',25,'Chennai',3500);
      c_package.addcustomer(8,'Subham',32,'Delhi',7500);
      c_package.listcustomer;
      c_package.delcustomer(code);
      c_package.listcustomer;
END;
/
```

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

```
Customer(1):Ramesh
Customer(2):Khilan
Customer(3): kaushik
Customer(4):Chaitali
Customer(5):Hardik
Customer(6):Komal
Customer(7):Rajnish
```

```
Customer(8):Subham
Customer(1):Ramesh
Customer(2):Khilan
Customer(3): kaushik
Customer(4):Chaitali
Customer(5):Hardik
Customer(6):Komal
Customer(7):Rajnish


PL/SQL procedure successfully completed
```

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

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

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations

- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

```
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;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} :This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

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

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

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

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
1.  CREATE OR REPLACE TRIGGER display_salary_changes
2.  BEFORE DELETE OR INSERT OR UPDATE ON customers
3.  FOR EACH ROW
4.  WHEN (NEW.ID > 0)
5.  DECLARE
6.    sal_diff number;
7.  BEGIN
8.    sal_diff := :NEW.salary  - :OLD.salary;
9.    dbms_output.put_line('Old salary: ' || :OLD.salary);
10.   dbms_output.put_line('New salary: ' || :NEW.salary);
```

```
11.    dbms_output.put_line('Salary difference: ' || sal_diff);
12. END;
13. /
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

Two types of Programmatic SQL

- **Embedded SQL statements**
  - SQL statements are embedded into the program source code and mixed with the host language statements. This allows users to write programs that access the database directly. A special precompiler modifies the source code to replace SQL statements with calls to DBMS routines. The source code can then be compiled and linked in the normal way.
- **Application Programming Interface (API)**
  - No need for any precompilation.
  - Provides a cleaner interface and generates more manageable code.

2 Types of Embedded SQL

- Static SQL
  - A complete embedded SQL statement.
  - Static SQL statements can be placed into stored procedures and can contain host variables.
- Dynamic SQL
  - With dynamic SQL statements, knowing the complete structure of an SQL statement before building the application is not necessary. Dynamic SQL statements allow run-time input to provide information about the database objects to query.

Static or Embedded SQL are **SQL** statements in an application that do not change at runtime and, therefore, can be hard-coded into the application. Dynamic**SQL** is **SQL** statements that are constructed at runtime; for example, the application may allow users to enter their own queries.

Dynamic SQL is a programming technique that enables you to build**SQL** statements dynamically at runtime. You can create more general purpose, flexible applications by using **dynamic SQL** because the full text of a **SQL** statement may be unknown at compilation.

S.No.	Static (embedded) SQL	Dynamic (interactive) SQL
1.	In static SQL how database will be accessed is predetermined in the embedded SQL statement.	In dynamic SQL, how database will be accessed is determined at run time.
2.	It is more swift and efficient.	It is less swift and efficient.
3.	SQL statements are compiled at compile time.	SQL statements are compiled at run time.
4.	Parsing, validation, optimization, and generation of application plan are done at compile time.	Parsing, validation, optimization, and generation of application plan are done at run time.
5.	It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non-uniformly.
6.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.
7.	It is less flexible.	It is more flexible.

Dynamic SQL refers to SQL code that is generated within an application or from the system tables and then executed against the database to manipulate the data. The SQL code is not stored in the source program, but rather it is generated based on user input. This can include determining not only what objects are involved, but also the filter criteria and other qualifiers that define the set of data being acted on.

Using this approach, we can develop powerful applications that allow us to create database objects and manipulate them based on user input. For example, suppose you are working on a web application that will include a function that presents the user with a screen that contains series of parameters to define the information, then that the application performs a search based on the parameters that have been filled in.

Without using Dynamic SQL, we would have to code the query to account for all the combinations of the various parameter fields.

```
Select * from Customer
Where ((CustNM is not null and CustNM like 'John Doe%') or CustNM is null) and
((Age is not null and Age = '') or Age is null) and
((Sex is not null and Sex = '') or Sex is null) and
((Cars is not null and Cars = 2) or Cars is null)
```

If there are only a few, it is okay, but when you have 10 or more parameters, you can end up with a complex query, particularly if you allow the user to specify conditions between parameters such as `AND`, `OR`, etc.

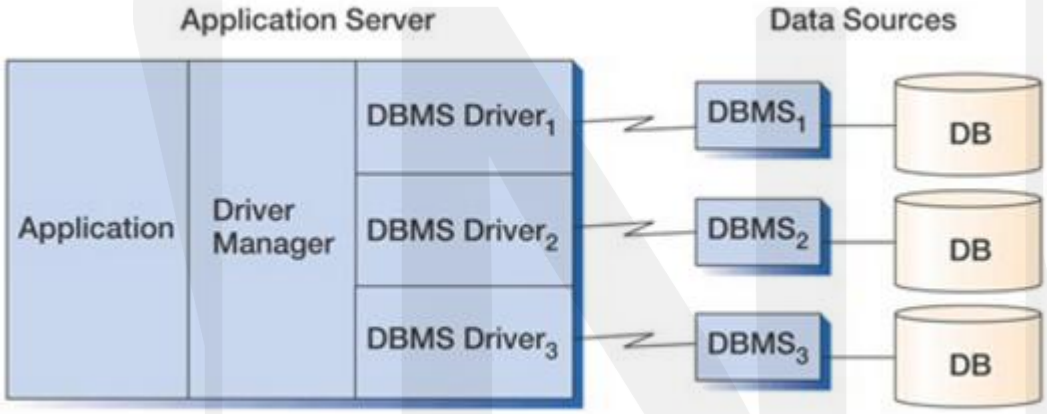
The more typical approach used by application developers is to use a routine that parses the fields within the client program and builds the `WHERE` clause to contain just the criteria needed. This results in SQL code created in the application that is based on the user input. In our applications, we can generate the query from these components to specify what we want to see and in what format.

```
Select * from Customer
```

Where CustName like 'John Doe%' and Cars = 2

- **ODBC** (**Open Database Connectivity**) standard provides a DBMS-independent means for processing relational database data
- It was developed in the early 1990s by an industry committee and has been implemented by Microsoft and many other vendors
- The goal is to allow a developer to create a single application that can access databases supported by different DBMS products without needing to be changed or recompiled

Figure 12.5 ODBC Architecture



Application can process a database using any of the three DBMS products.

- OBDC consists of data source, application program, drivermanager, and DBMS driver
- **Data source** is the database, its associated DBMS, operating system, and network platform
  - An ODBC data source can be a relational database, a file server, or a spreadsheet
- **Applications program** issues requests to create a connection with a data source

- **Driver manager** determines the type of DBMS for a given ODBC data source and loads that driver in memory
- **DBMS driver** processes ODBC requests and submits specific SQL statements to a given type of data source
  - A **single-tier driver** processes both ODBC calls and SQL statements
  - A **multiple-tier driver** processes ODBC calls, but passes the SQL requests to the database server

Parallel Database Architecture

We need certain architecture to handle the above said. That is, we need architectures which can handle data through data distribution, parallel query execution thereby produce good throughput of queries or Transactions. Figure 1, 2 and 3 shows the different architecture proposed and successfully implemented in the area of Parallel Database systems. In the figures, P represents Processors, M represents Memory, and D represents Disks/Disk setups.

1. Shared Memory Architecture

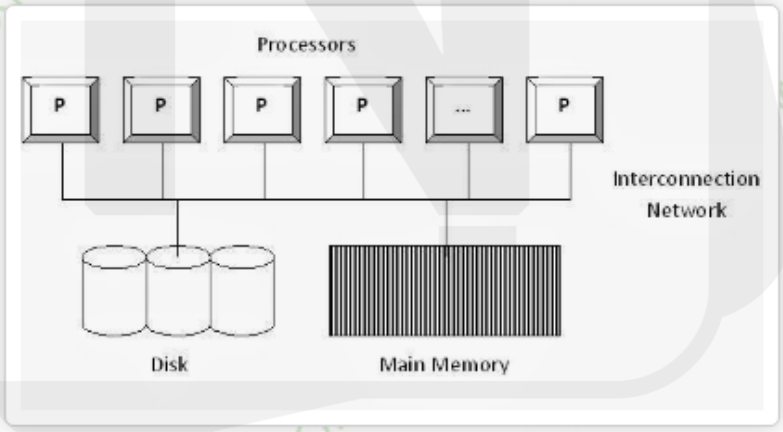


Figure 1 - Shared Memory Architecture

In Shared Memory architecture, single memory is shared among many processors as show in Figure 1. As shown in the figure, several processors are connected through an interconnection network with Main memory and disk setup. Here interconnection network is usually a high speed network (may be Bus, Mesh, or Hypercube) which makes data sharing (transporting) easy among the various components (Processor, Memory, and Disk).

2. Shared Disk Architecture

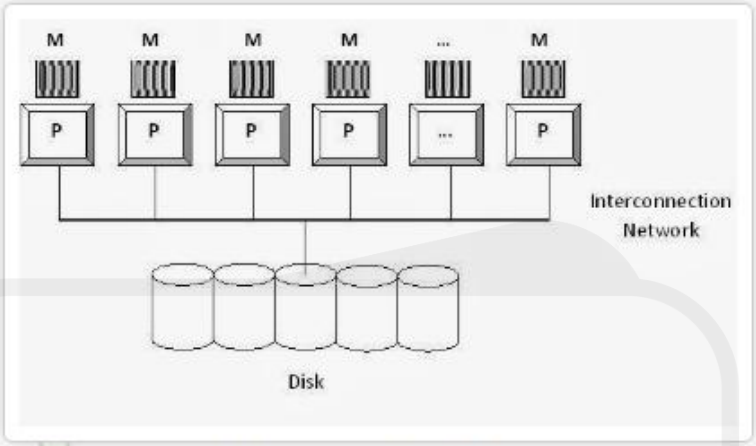


Figure 2 - Shared Disk Architecture

In Shared Disk architecture, single disk or single disk setup is shared among all the available processors and also all the processors have their own private memories as shown in Figure 2.

3. Shared Nothing Architecture

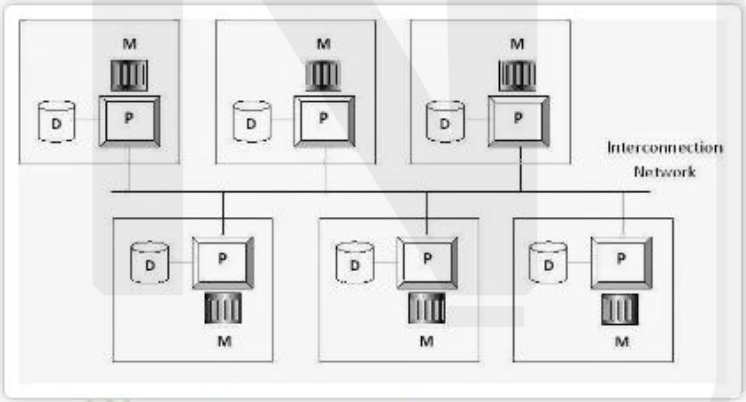


Figure 3 - Shared Nothing Architecture

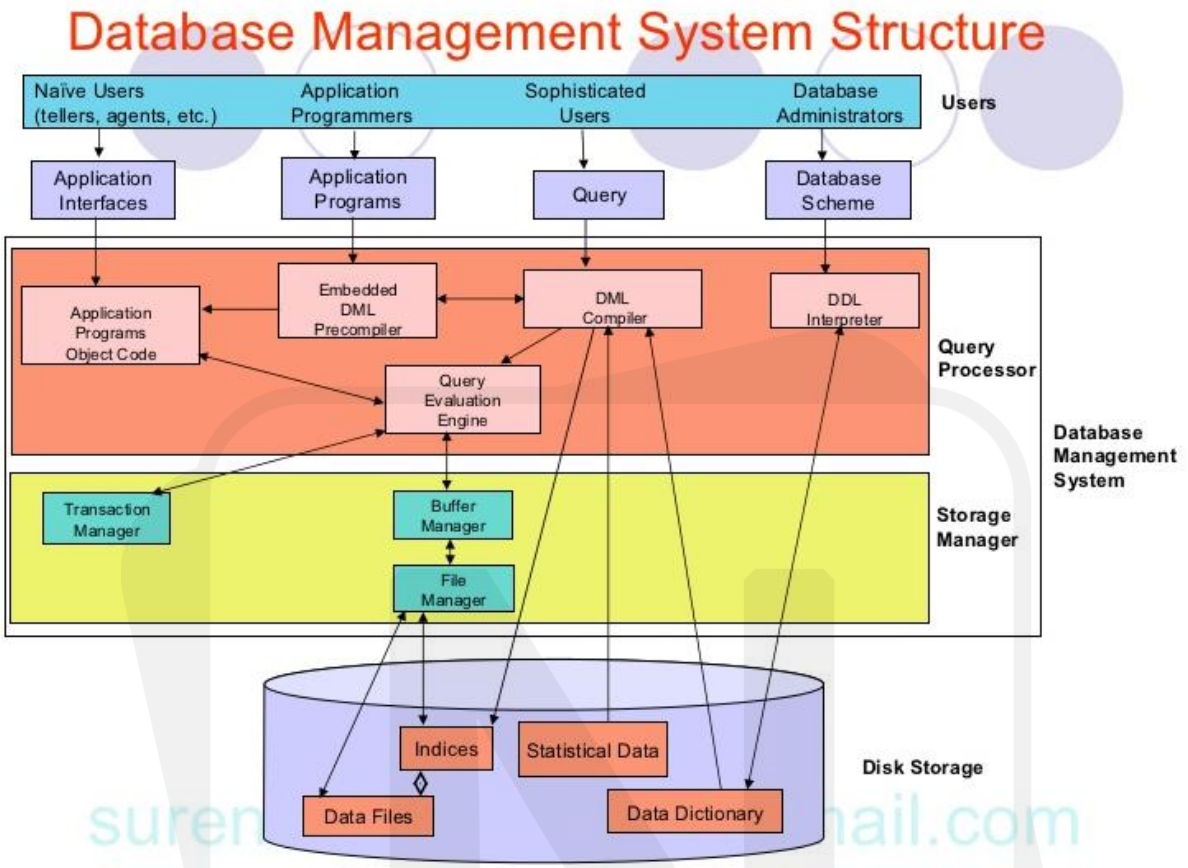
In Shared Nothing architecture, every processor has its own memory and disk setup. This setup may be considered as set of individual computers connected through high speed interconnection network using regular network protocols and switches for example to share data between computers. (This architecture is used in the Distributed Database System). In Shared Nothing parallel database system implementation, we insist the use of similar nodes that are Homogenous systems. (In distributed database System we may use Heterogeneous nodes)

DATABASE SYSTEM STRUCTURE

A Database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and query processor components.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in of data. A gigabyte is 1000 megabytes (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes), since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.

The Query Processor is important because it helps the database system simplify and facilitate access to data. High level views help to achieve this goal; with them, users of the system are not be burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.



Storage Manager

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low level file system commands. Thus, the storage manager is responsible for storing, retrieving and updating data in the database.

The storage manager components include:

- **Authorization and integrity manager**

Which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

- **Transaction manager**

Which ensures that the database remains in a consistent correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

- **File Manager**

Which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

- **Buffer manager**

Which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation:

- **Data files,** which store the database itself.
- **Data dictionary,** which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices,** which provide fast access to data items that hold particular values.

The Query Processor

The query processor components include

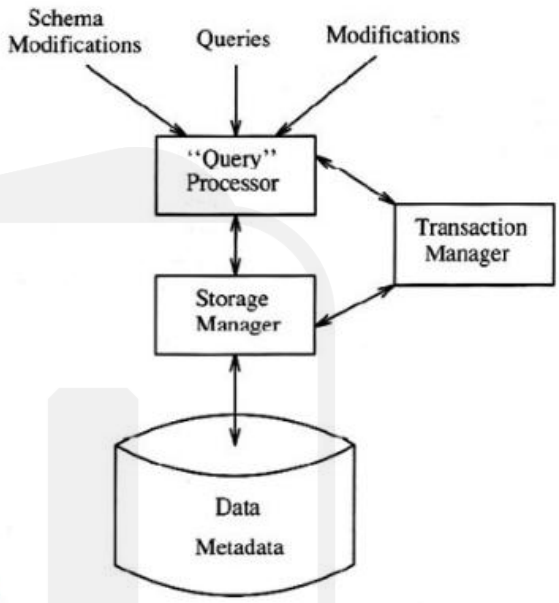
- **DDL interpreter,** which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler,** which translates DML statements in a query language into an evaluation plan consisting of low level instructions that the query evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is it picks the lowest cost evaluation plan from among the alternatives.

    - **Query evaluation engine,** which executes low level instructions generated by the DML compiler.
-

Query Processor

- **Query processor** handles: queries and modifications to the data.
  - Finds the best way to carry out a requested operation and
  - Issues commands to the storage manager that will carry them out.



- **E.g.** A bank has a DB with two relat.:
  Customers (name, SIN, address),
  Accounts (accountNo, balance, SIN)

  Query: "Find the balances of all accounts of which Sally is the owner."

```
SELECT Accounts.balance
FROM Customers, Accounts
WHERE Customers.SIN = Accounts.SIN
    AND Customers.name = 'Sally';
```

What is a Relational Model?

Relational model stores data in the form of tables. This concept purposed by Dr. E.F. Codd, a researcher of IBM in the year 1960s. The relational model consists of three major components:

1. The set of relations and set of domains that defines the way data can be represented (data structure).

2. Integrity rules that define the procedure to protect the data (data integrity).

3. The operations that can be performed on data (data manipulation).

A rational model database is defined as a database that allows you to group its data items into one or more independent tables that can be related to one another by using fields common to each related table.

Advanced SQL programming

Read from this link

https://advanced-sql-programming.blogspot.in/2010/01/advanced-sql-programming.html

DATA REPLICATION

Data Replication is the process of storing data in more than one site or node. This is necessary for improving the availability of data. There can be full replication, in which a copy of the whole database is stored at every site. There can also be partial replication, in which case, some fragment (important frequently· used fragments) of the database are replicated and others are not replicated. There are a number of advantages and disadvantages to replication.

Advantages & Disadvantages of Data Replication

There are following advantages of replication:

Availability

If one of the sites containing relation R fails, then the relation R can be obtained from another site. Thus, queries (involving relation R) can be continued to be processed in spite of the failure of one site.

Increased parallelism

The sites containing relation R can process queries (involving relation R) in parallel This leads to faster query execution.

Less Data Movement over Network

The more replicas of, a relation are there, the greater are the chances that the required data is found where the transaction is executing. Hence, data replication reduces movement of data among sites and. increases .speed of processing.

Disadvantages of Data Replication

There are following disadvantages of replication:

Increased overhead on update.

When an updation is required, a database system must ensure that all replicas are updated.

Require more disk space

Storing replicas of same data at different sites consumes more disk space.

Expensive

Concurrency control and recovery techniques will be more advanced and hence more expensive.

Security Considerations

Updated Software

It is mandatory to keep you software updated. It plays vital role in keeping your website secure.

SQL Injection

It is an attempt by the hackers to manipulate your database. It is easy to insert rogue code into your query that can be used to manipulate your database such as change tables, get information or delete data.

Cross Site Scripting (XSS)

It allows the attackers to inject client side script into web pages. Therefore, while creating a form It is good to endure that you check the data being submitted and encode or strip out any HTML.

Error Messages

You need to be careful about how much information to be given in the error messages. For example, if the user fails to log in the error message should not let the user know which field is incorrect: username or password.

Validation of Data

The validation should be performed on both server side and client side.

Passwords

It is good to enforce password requirements such as of minimum of eight characters, including upper case, lower case and special character. It will help to protect user’s information in long run.

Upload files

The file uploaded by the user may contain a script that when executed on the server opens up your website.

SSL

It is good practice to use SSL protocol while passing personal information between website and web server or database.

Database Security:

It is the mechanisms that protect the database against intentional or accidental threats.

Definition of Database Security

Database Security is defined as the process by which "Confidentiality, Integrity and Availability" of the database can be protected

Why need of database security?

If there is no security to database what happens???

Data will be easily corrupted

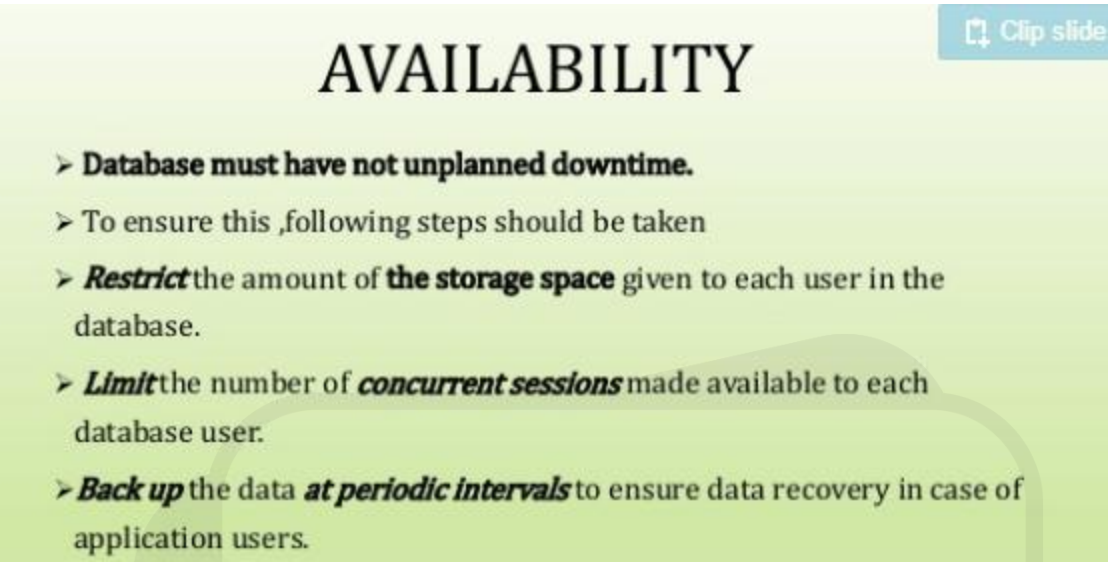
It is important ***to restrict access to the database from authorized users*** to protect sensitive data.

SECRECY / CONFIDENTIALITY

- It is protecting the database from unauthorized users.
- Ensures that users are allowed to do the things they are trying to do.
- Encryption is a technique or a process by which the data is encoded in such a way that only that authorized users are able to read the data.

INTEGRITY

- Protecting the database from authorized users.
- Ensures that what users are trying to do is correct.
- For examples,
  - An employee should be able to modify his or her own information.



RDBMS Implementation

Relational database technology remains the mechanism of choice for data storage in information management systems. BlazeLIMS supports Oracle and SQL Server, the two industry standards for RDBMS, giving the customer the choice of the system most suitable to the situation. This choice allows for cost effective installations ranging from 5 or fewer concurrent users up to hundreds of concurrent users:

- Oracle
  - Superior concurrency management.
  - More suitable for larger systems with more concurrent users.
  - More expensive.
  - More difficult to maintain.
- SQL Server
  - Less effective concurrency management - susceptible to more deadlocks.
  - More suitable for systems with fewer concurrent users.
  - Less expensive.
  - Royalty free low-end database engine.
  - Easier to maintain.
  - More friendly tools.

Perhaps more important than choice of database is how the database is used. BlazeLIMS has been implemented using the highest quality approach:

- Proper transaction bounding with brief transactions and minimum locking windows, using an optimistic locking model, provides superior concurrency management.
- Use of the most efficient API (native SQLNet for Oracle, ODBC for SQL Server).
- Server-only read/write transactionsinsures robust scaleable operation.
- Superior physical schema implementation reduces long term DBA maintenance/tuning costs.

- Proprietary automated query generation insures predictable response times in the face of common query optimizer weaknesses.

Query Optimization:-

- Query optimization is the process of selecting the most efficient query-evaluation plan from among the many strategies usually possible for processing a given query, especially if the query is complex.
- In simple scenes the query optimizer come up with query-evaluation plan that computes the same result as a given expression, but it is a least costly way of generating result.

Why?

- Faster processing of a query.
- Lesser cost per query.
- High performance of the system.
- Lesser stress on the database.
- Efficient use of database engine.
- Lesser memory is consumed.

INTEGRITY CONSTRAINTS OVER RELATION

INTRODUCTION

Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Constraints may apply to each attribute or they may apply to relationships between tables.

Integrity constraints ensure that changes (update deletion, insertion) made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

EXAMPLE- A brood group must be 'A' or 'B' or 'AB' or 'O' only (can not any other values else).

TYPES OF INTEGRITY CONSTRAINTS

Various types of integrity constraints are-

1. Domain Integrity
2. Entity Integrity Constraint
3. Referential Integrity Constraint
4. Key Constraints

1. Domain Integrity- Domain integrity means the definition of a valid set of values for an attribute. You define data type, length or size, is null value allowed , is the value unique or not for an attribute ,the default value, the range (values in between) and/or specific values for the attribute.

2. Entity Integrity Constraint- This rule states that in any database relation value of attribute of a primary key can't be null.

EXAMPLE- Consider a relation "STUDENT" Where "Stu_id" is a primary key and it must not contain any null value whereas other attributes may contain null value e.g "Branch" in the following relation contains one null value.

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	ECE
11255324	Ajay	
11255237	Raman	CSE
11255678	Aastha	ECE

3.Referential Integrity Constraint-It states that if a foreign key exists in a relation then either the foreign key value must match a primary key value of some tuple in its home relation or the foreign key value must be null.

The rules are:

1. You can't delete a record from a primary table if matching records exist in a related table.
2. You can't change a primary key value in the primary table if that record has related records.
3. You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
4. However, you can enter a Null value in the foreign key, specifying that the records are unrelated.

EXAMPLE-Consider 2 relations "stu" and "stu_1" Where "Stu_id " is the primary key in the "stu" relation and foreign key in the "stu_1" relation.

Relation "stu"

Stu_id	Name	Branch
11255234	Aman	CSE
11255369	Kapil	ECE
11255324	Ajay	ME
11255237	Raman	CSE
11255678	Aastha	ECE

Relation "stu_1"

Stu_id	Courc e	Durati on
112552 34	B.TEC H	4 years
112553 69	B.TEC H	4 years
112553 24	B.TEC H	4 years
112552 37	B.TEC H	4 years
112556 78	B.TEC H	4 years

Examples

Rule 1. You can't delete any of the rows in the ”stu” relation that are visible since all the ”stu” are in use in the “stu_1” relation.
Rule 2. You can't change any of the ”Stu_id” in the “stu” relation since all the “Stu_id” are in use in the ”stu_1” relation.
Rule 3. The values that you can enter in the” Stu_id” field in the “stu_1” relation must be in the” Stu_id” field in the “stu” relation.
Rule 4 You can enter a null value in the "stu_1" relation if the records are unrelated.

4.Key Constraints- A Key Constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple.
There are 4 types of key constraints-

1. Candidate key.
2. Super key
3. Primary key
4. Foreign key

QUERYING RELATIONAL DATA

INTRODUCTION

- There are several types of linguistic queries:
  - ‘Yes/No’ queries (Is the following declarative statement true?).
  - ‘W’ queries (who/what/when – fill in the correct answer).
  - ‘How/Why’ queries (ambiguous).
- *Erotetic* Logic was developed in an effort to deal systematically with such queries. Unfortunately it has proven unsatisfactory.
- A relational database query is a very special kind of question (I called it a list query). It may be paraphrased as “Give me a list of the entities satisfying such and such properties as well as certain of their properties”.

Example

Give me a list of the names of all students having gpa greater than 3.2 and their ages.

- This type of query bypasses the problem of logical description as the query is formulated in such a way that it specifies indirectly the operations to be performed by the system in order to provide an answer.
- The answer provided by the system is in the form of a relation.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be

either **unary** or **binary**.They accept relations as their input and yield relations as their output.Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows –

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like – =, ≠, ≥, < , >, ≤.

For example –

$$\sigma_{subject = "database"}(Books)$$

Output – Selects tuples from books where subject is 'database'.

$$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$$

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

$$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation – $\prod_{A1, A2, An} (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$$\prod_{subject, author} (Books)$$

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as –

$$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$$

Notation – r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold –

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$$\prod_{author} (Books) \cup \prod_{author} (Articles)$$

Output – Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$$\prod_{author}(Books) - \prod_{author}(Articles)$$

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$$\sigma_{author = 'tutorialspoint'}(Books \ X \ Articles)$$

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − $\rho_x(E)$

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

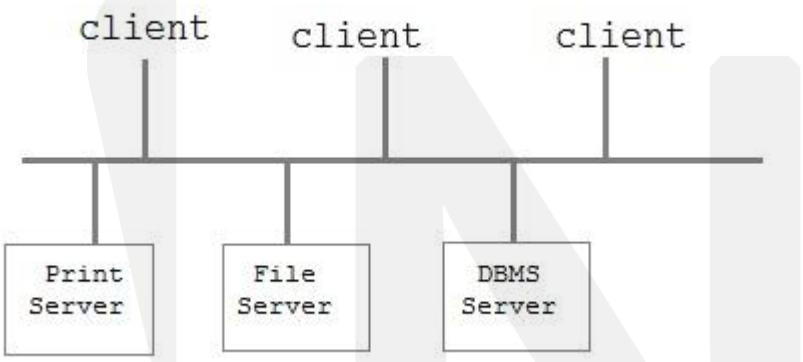
- Set intersection
- Assignment
- Natural join

Database Architecture

Database architecture is logically divided into two types.

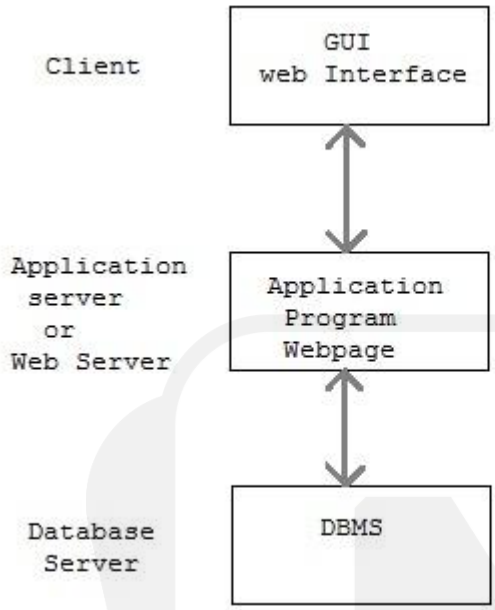
1. Logical two-tier Client / Server architecture
2. Logical three-tier Client / Server architecture

Two-tier Client / Server Architecture



Two-tier Client / Server architecture is used for User Interface program and Application Programs that runs on client side. An interface called ODBC(Open Database Connectivity) provides an API that allow client side program to call the dbms. Most DBMS vendors provide ODBC drivers. A client program may connect to several DBMS's. In this architecture some variation of client is also possible for example in some DBMS's more functionality is transferred to the client including data dictionary, optimization etc. Such clients are called **Data server**.

Three-tier Client / Server Architecture



Three-tier Client / Server database architecture is commonly used architecture for web applications. Intermediate layer called **Application server** or Web Server stores the web connectivty software and the business logic(constraints) part of application used to access the right amount of data from the database server. This layer acts like medium for sending partially processed data between the database server and the client.

Explain Inheritance and its types in Object oriented Data Model?

Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is – a" relationship.

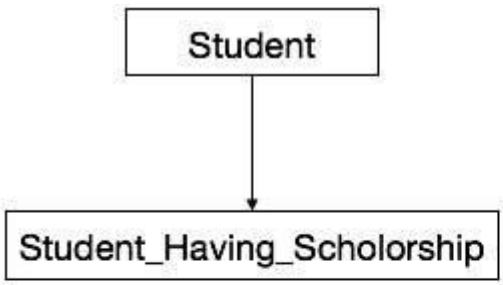
Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow "is – a" mammal.

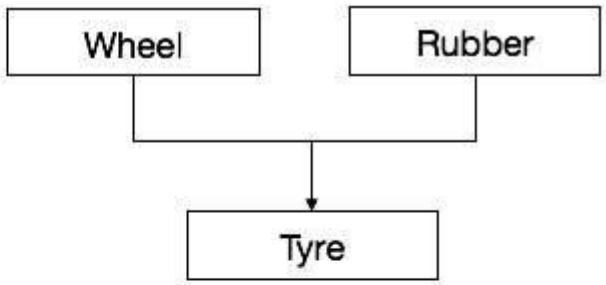
Types of Inheritance:

- **Single Inheritance** : A subclass derives from a single super-class.
- **Multiple Inheritance** : A subclass derives from more than one super-classes.
- **Multilevel Inheritance** : A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** : A combination of multiple and multilevel inheritance so as to form a lattice structure.

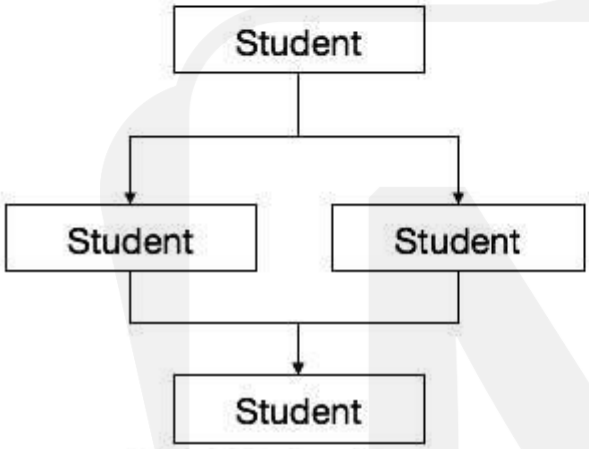
The following figure depicts the examples of different types of inheritance.



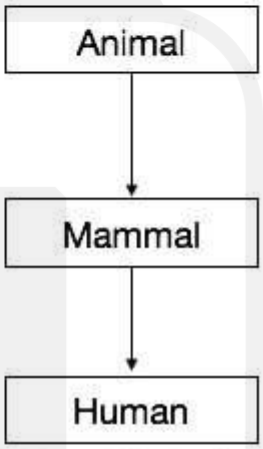
Single Inheritance



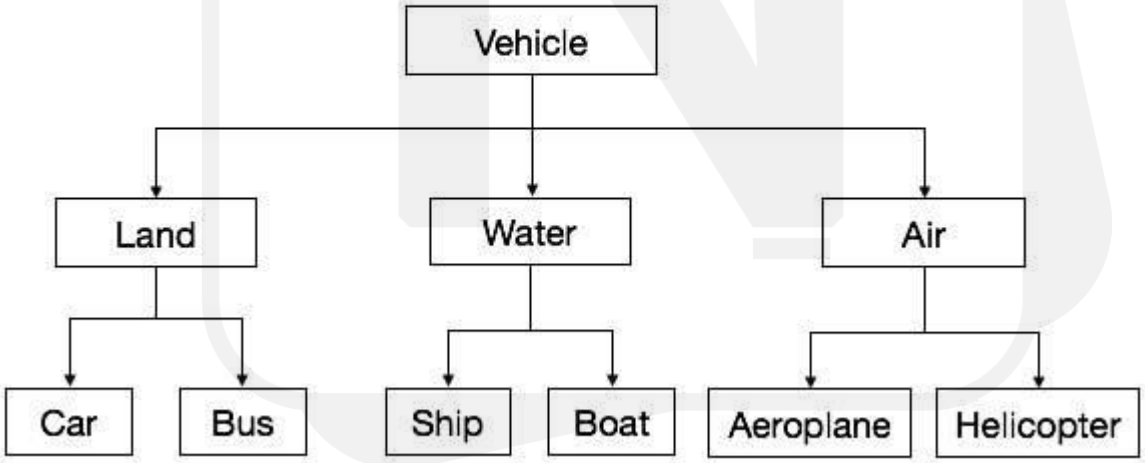
Multiple Inheritance



Hybrid Inheritance



Multi-level Inheritance



Hierarchical Inheritance

UNIT 3 and 4

Object-Relational Databases

3

- Relational DBMS (RDBMS) vendors recognized need to add object database features.
- The resulting database systems characterized as Object-Relational Database Systems or ORDBMS.
- SQL first specified by Chamberlin and Boyce (1974), underwent enhancements and standardization in 1989 and 1992. New standard, initially called SQL3 and later known as SQL:99
- Starting with SQL3 version, features from object databases were incorporated into the SQL standard.
- At first, these extensions were known as SQL/Object SQL standard 2008 for RDBMSs includes many ODBMS features - originally known as SQL/Object - now have merged into main SQL specification known as SQL/Foundation.

OODBMS	ORDBMS
It is created on the basis of persistent programming paradigm.	It is built by creating object-oriented extensions of a relational database system.
It supports ODL and OQL for defining and manipulating complex data types.	It supports an extended form of SQL.
It aims to achieve seamless integration with object-oriented programming languages such as C++, Java, or Smalltalk.	Such an integration is not required as SQL:1999 allows us to embed SQL commands in a host language.
Query optimization is difficult to achieve in these databases.	The relational model has a very strong foundation for query optimization, which helps in reducing the time taken to execute a query.
The query facilities of OQL are not supported efficiently in most OODBMS.	The query facilities are the main focus of ORDBMS. The querying in these databases is as simple as in relational database system, even for complex data types and multimedia data.
It is based on object-oriented programming languages; any error of data type made by programmer may affect many users.	It provides good protection against programming errors.

Temporal Database Concepts

- **Temporal databases** encompass all database applications that require some aspect of time when organizing their information (e.g., reservation system in hotels, airline, etc.).
- For temporal databases, time is considered to be an ordered sequences of points (e.g., seconds, minutes, day, etc.).
- Is SQL2, the temporal data types include DATE, TIME, TIMESTAMP, INTERVAL, and PERIOD
- A temporal database will store information concerning when certain events occur.
- Types of temporal information:
  - **Point events:** associated in the database with a **single time point** (e.g., 15/08/1998)**.**
  - **Duration events:** associated in the database with a specific **time period** (e.g., [15/08/1998, 15/08/2000]).

Forms of Temporal Database

Valid Time

The valid time of a database object is the time when the object is effective or holds (istrue) in reality. The time when the event occurred, took place in reality. For example, in a banking system, the payments and withdrawals made by a customer have a valid time associated with the time the customer performs the transaction at the bank. Another example would be, in a football competition, when the clubs won the competition i.e. the times when Arsenal won the F.A Cup competition.

Objects in the temporal database system will have a time component associated to it; this will hold either the valid time or the transaction time.

Transaction Time

A database object is stored in a database at some point in time. The transaction time of an object is the time when the object is stored in the database, the time that it is present in the database. For example, in a banking system, the transaction time of a withdrawal would be form the time the clerk entered the payment of withdrawal into the database to the time that it was made invalid in the database. Another example would be, in a company situation, an employee receives a pay rise but it comes into effect when the payroll clerk enters this salary rise into the database. Transaction time values cannot be after the current time.

WHAT IS MOBILE DATABASE?

- **A mobile database is a database that can be connected to by a mobile computing device over a mobile network.**
- **A database that is portable and physically separate from the corporate database server.**
- **But *Mobile Database* is capable of communicating with that corporate database server from remote sites allowing the sharing of corporate database.**

NEED FOR MOBILE DATABASE

- Need for data to be collected as it happens
- Business Today Means Anytime, Anywhere, Any Device
- Mobile users must be able to work without a wireless connection due to poor or even non-existent connections.
- Need to access data while the mobile user is disconnected from the network—wireless or otherwise.
- This capability is best implemented by incorporating persistent data storage using a mobile database in your application.

MOBILE DATABASE SYSTEM LIMITATIONS

- Limited wireless bandwidth
- Wireless communication speed
- Limited energy source (battery power)
- Less secured
- Vulnerable to physical activities
- Hard to make theft proof.

•Complex Data Types

Example1 : addresses

Atomic data item of type string

A better alternative structured data types

(street, address, city, state, postal code)

Example 2: phone numbers

Using normalization ?!

The alternative of normalization by creating a new relation is expensive and artificial for this example.

A library application

title	*author_array*	*publisher* (*name, branch*)	*keyword_set*
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

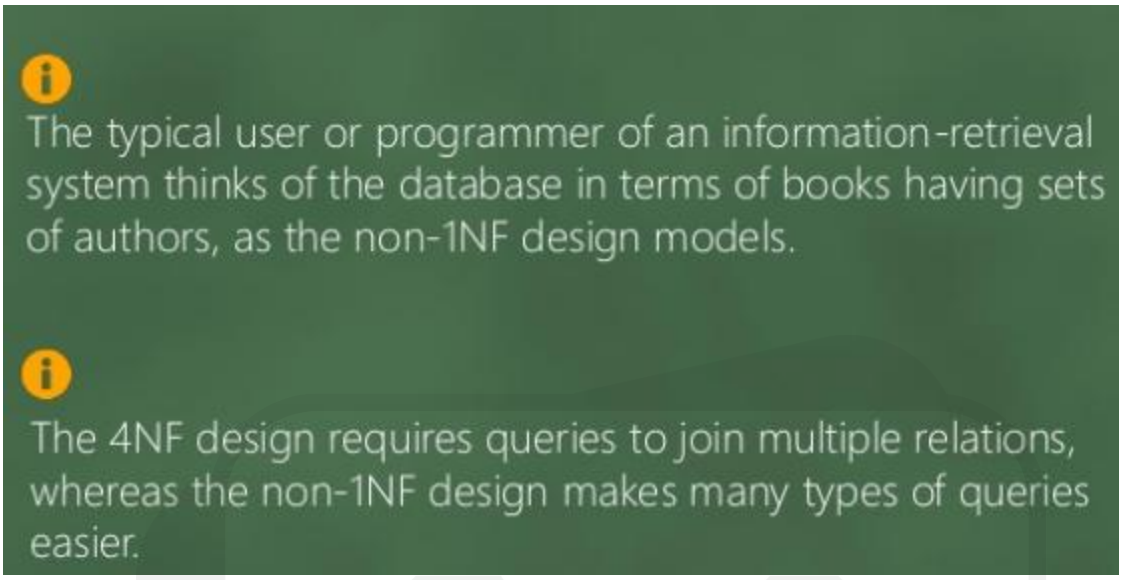
Several domains will be non-atomic

title	*author*	*position*
Compilers	Smith	1
Compilers	Jones	2
Networks	Jones	1
Networks	Frick	2

satisfies 4NF

title	*pub_name*	*pub_branch*
Compilers	McGraw-Hill	New York
Networks	Oxford	London

title	*keyword*
Compilers	parsing
Compilers	analysis
Networks	Internet
Networks	Web



What is the the need for Complex Data Type?

In recent years, there is need for more complex data types. For example, address field. While an address could be viewed as an atomic data item of type string, this view hide details such as street, city, state, pin_code. On the other hand, if an address were represented by breaking it into the components, writing queries would be more complicated. A better alternative is to use OODMBS which uses structured types, which allow a type address with subparts street, city, state, pincode. Consider another example; for representing multivalued attribute phone number, in RDBMS, it is necessary to create a new relation, which is expensive.

Structured Types and Inheritance in SQL

- Structured types can be declared and used in SQL

```
create type Name as
    (firstname      varchar(20),
     lastname       varchar(20))
    final

create type Address as
    (street         varchar(20),
     city           varchar(20),
     zipcode        varchar(20))
    not final
```

  - Note: **final** and **not final** indicate whether subtypes can be created

- Structured types can be used to create tables with composite attributes

```
create table customer (
    name        Name,
    address     Address,
    dateOfBirth date)
```

- Dot notation used to reference components: *name.firstname*

- User-defined row types

```
create type CustomerType as (
    name Name,
    address Address,
    dateOfBirth date)
    not final
```

- Can then create a table whose rows are a user-defined type

```
create table customer of CustomerType
```

Inheritance

- Suppose that we have the following type definition for people:

```
create type Person
    (name varchar(20),
    address varchar(20))
```

- Using inheritance to define the student and teacher types

```
create type Student
  under Person
  (degree       varchar(20),
   department   varchar(20))
create type Teacher
  under Person
  (salary       integer,
   department   varchar(20))
```

- Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

Object-Identity and Reference Types

- Define a type *Department* with a field *name* and a field *head* which is a reference to the type *Person*, with table *people* as scope:

```
create type Department (
        name varchar (20),
        head ref (Person) scope people)
```

- We can then create a table *departments* as follows

```
create table departments of Department
```

- We can omit the declaration **scope** people from the type declaration and instead make an addition to the **create table** statement:

```
create table departments of Department
    (head with options scope people)
```

How to Define the Structure of XML Document

- DTD (Document Type Definition)
  - a set of markup declarations that define a document type for an SGML-family markup language
- XML Schema
  - to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema
  - generally to use a prefix `:xsd` for XML Namespace
  - Comparing with DTD
    - XML syntax
    - various datatypes and user-define datatype
    - various content model and occurrence constraint
    - supporting XML Namespace

LiST
Linked Data & Semantic Web Technology

40

What is a DTD?

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And here is the file "note.dtd", which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
```

```
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

XML Document Schema

- Database schemas constrain what information can be stored, and the data types of stored values
- XML documents are not required to have an associated schema
- However, schemas are very important for XML data exchange
  - Otherwise, a site cannot automatically interpret data received from another site
- Two mechanisms for specifying XML schema
  - **Document Type Definition (DTD)**
    - Widely used
  - **XML Schema**
    - Newer, increasing use

Querying and Transforming XML Data

- Translation of information from one XML schema to another
- Querying on XML data
- Above two are closely related, and handled by the same tools
- Standard XML querying/translation languages
  - XPath
    - Simple language consisting of path expressions
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XQuery
    - An XML query language with a rich set of features

SAX (Simple API for XML)

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

SAX is an *event-driven* interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser.

SAX was developed collaboratively by members of the XML-DEV mailing list (currently hosted by OASIS). The original version of SAX, which was specific to Java, was the first API for XML in Java to gain broad industry support.

Applications of XML

- Database applications
- Document Mark-up(with HTML)
- Mathematical Mark-up language(MATHML)
- Messaging b/w different business platforms
- Channel definition Format (CDF)
- Metacontent definition
- Platform for Internet Context Selection (PICS)
- Platform for Privacy References Syntax Specification (P3P)
- Resource Description Format (RDF)
- Scaleable Vector Graphics (SVG)
- Synchronized Multemedia Integration Language (SMIL)

PostgreSQL is a powerful, open source object-relational database system(ORDBMS). It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

It is used to store data securely; supporting best practices and allow retrieving them when request is processed.

PostgreSQL (also pronounced as Post-gress-Q-L) is developed by the PostgreSQL Global Development Group (a worldwide team of volunteers). It is not controlled by any corporation or other private entity. It is open source and its source code is available free of charge.

PostgreSQL is cross platform and runs on many operating systems such as Linux, FreeBSD, OS X, Solaris, and Microsoft Windows etc.

PostgreSQL Features

- PostgreSQL runs on all major operating systems i.e. Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows etc.
- PostgreSQL supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).
- PostgreSQL supports a lot of features of SQL like Complex SQL queries, SQL Sub-selects, Foreign keys, Trigger, Views, Transactions, Multiversion concurrency control (MVCC), Streaming Replication (as of 9.0), Hot Standby (as of 9.0).
- In PostgreSQL, table can be set to inherit their characteristics from a "parent" table.
- You can install several extensions to add additional functionality to PostgreSQL.

PostgreSQL Data Types

A datatype specifies, what kind of data you want to store in the table field. While creating table, for each column, you have to use a datatype.

There are mainly three types of datatypes in PotgreSQL. Besides this, users can also create their own custom datatypes using CREATE TYPE SQL command.

Following are the mainly three types of datatypes in PostgreSQL:

- **Numeric datatype**
- **String datatype**
- **Date/time datatype**

Numeric Data Type

Numeric datatype is used to specify the numeric data into the table.

name	description	storage size	range
smallint	stores whole numbers, small range.	2 bytes	-32768 to +32767
integer	stores whole numbers.use this when	4 bytes	-2147483648 to +2147483647

	you want to store typical integers.		
bigint	stores whole numbers, large range.	8 bytes	-9223372036854775808 to 9223372036854775807
decimal	user-specified precision, exact	variable	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point.
numeric	user-specified precision, exact	variable	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point.
real	variable-precision, inexact	4 bytes	6 decimal digits precision.
double precision	variable-precision, inexact	8 bytes	15 decimal digits precision
serial	auto incrementing integer	4 bytes	1 to 2147483647
bigserial	large auto incrementing integer	8 bytes	1 to 9223372036854775807

String Data Type

String datatype is used to represent the string type values.

Datatype	Explanation
char(size)	Here size is the number of characters to store. Fixed-length strings. Space padded on right to equal size characters.
character(size)	Here size is the number of characters to store. Fixed-length strings.

	Space padded on right to equal size characters.
varchar(size)	Here size is the number of characters to store. Variable-length string.
character varying(size)	Here size is the number of characters to store. Variable-length string.
text	Variable-length string.

Date/Time Data Type

The date/time datatype is used to represent the columns using date and time values.

Name	Description	Storage size	Minimum value	Maximum value	Resolution
timestamp [(p)] [without time zone]	both date and time (no time zone)	8 bytes	4713 bc	294276 ad	1 microsecond / 14 digits
timestamp [(p)] with time zone	both date and time, with time zone	8 bytes	4713 bc	294276 ad	1 microsecond / 14 digits
date	date (no time of day)	4 bytes	4713 bc	5874897 ad	1 day
time [(p)] [without time zone]	time of day (no date)	8 bytes	00:00:00	24:00:00	1 microsecond / 14 digits

time [(p)] with time zone	times of day only, with time zone	12 bytes	00:00:00+1459	24:00:00-1459	1 microsecond / 14 digits
interval [fields] [(p)]	time interval	12 bytes	-178000000 years	178000000 years	1 microsecond / 14 digits

---

Some other data Types

Boolean type:

Name	Description	Storage size
boolean	it specifies the state of true or false.	1 byte

Monetary type:

Name	Description	Storage size	Range
money	currency amount	8 bytes	-92233720368547758.08 to +92233720368547758.07

Geometric Type:

Geometric data types represent two-dimensional spatial objects. The most fundamental type, the point, forms the basis for all of the other types.

Name	Storage size	Representation	Description
point	16 bytes	point on a plane	(x,y)
line	32 bytes	infinite line (not fully	((x1,y1),(x2,y2))

		implemented)	
lseg	32 bytes	finite line segment	((x1,y1),(x2,y2))
box	32 bytes	rectangular box	((x1,y1),(x2,y2))
path	16+16n bytes	closed path (similar to polygon)	((x1,y1),...)
path	16+16n bytes	open path	[(x1,y1),...]
polygon	40+16n	polygon (similar to closed path)	((x1,y1),...)
circle	24 bytes	circle	<(x,y),r> (center point and radius)

Oracle database (Oracle DB) is a relational database management system (RDBMS) from the Oracle Corporation. Originally developed in 1977 by Lawrence Ellison and other developers, Oracle DB is one of the most trusted and widely-used relational database engines.

The system is built around a relational database framework in which data objects may be directly accessed by users (or an application front end) through structured query language (SQL). Oracle is a fully scalable relational database architecture and is often used by global enterprises, which manage and process data across wide and local area networks. The Oracle database has its own network component to allow communications across networks. Oracle DB is also known as Oracle RDBMS and, sometimes, just Oracle.

IBM DB2 is a database product from IBM. It is a Relational Database Management System (RDBMS). DB2 is designed to store, analyze and retrieve the data efficiently. DB2 product is extended with the support of Object-Oriented features and non-relational structures with XML.

Initially, IBM had developed DB2 product for their specific platform. Since year 1990, it decided to develop a Universal Database (UDB) DB2 Server, which can run on any authoritative operating systems such as Linux, UNIX, and Windows.

SQL Standards

Oracle strives to comply with industry-accepted standards and participates actively in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Standards Organization (ISO), which is affiliated with the International Electrotechnical Commission (IEC). Both ANSI and the ISO/IEC have accepted SQL as the standard language for relational databases. When a new SQL standard is simultaneously published by these organizations, the names of the standards conform to conventions used by the organization, but the standards are technically identical.

The latest SQL standard was adopted in July 1999 and is often called SQL:99. The formal names of this standard are:

- ANSI X3.135-1999, "Database Language SQL", Parts 1 ("Framework"), 2 ("Foundation"), and 5 ("Bindings")
  - ISO/IEC 9075:1999, "Database Language SQL", Parts 1 ("Framework"), 2 ("Foundation"), and 5 ("Bindings")
- SQL/CLI : an updated definition of the extension Call Level Interface, originally published in 1995, also known as CLI-95 ISO/IEC 9075-3:1999
- SQL/PSM : an updated definition of the extension Persistent Stored Modules, originally published in 1996, also known as PSM-96 ISO/IEC 9075-4:1999

The SQL:2003 standard makes minor modifications to all parts of SQL:1999 (also known as SQL3), and officially introduces a few new features such as:[1]

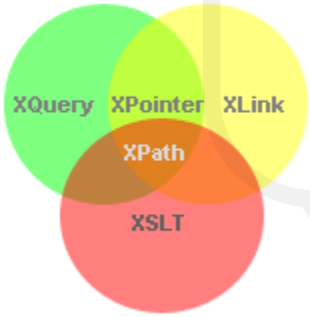
- XML-related features (SQL/XML)
- Window functions
- the sequence generator, which allows standardized sequences
- two new column types: auto-generated values and identity-columns
- the new MERGE statement
- extensions to the CREATE TABLE statement, to allow "CREATE TABLE AS" and "CREATE TABLE LIKE"
- removal of the poorly implemented "BIT" and "BIT VARYING" data types
- OLAP capabilities (initially added in SQL:1999) were extended with a window function.[2]

.

XML Specification

- XML Document Type Definitions (DTDs):
  - define the structure of "allowed" documents (i.e., valid written a DTD)
  - database schema
  - improve query formulation, execution, ...
- XML Schema
  - defines structure and data types
  - allows developers to build their own libraries of interchanged data types
- XML Namespaces
  - identify your vocabulary

What is XQuery?



- XQuery is ***the*** language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is a W3C Recommendation

XQuery Example

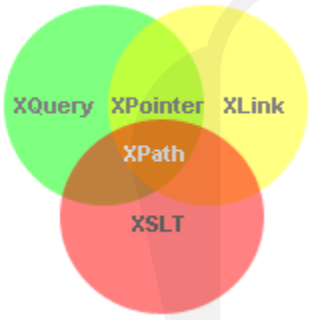
```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
```

```
order by $x/title
return $x/title
```

What is XPath?

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.



- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

XPath

XPath standards for XML Path language. XPATH is a major element in XSLT. It is used for selection and addressing elements and attributes in an XML document. It is basically ***a syntax used to describe parts of an XML document***. The root node is the XPath node that contains the entire document. The root node also contains comments or processing instructions that are outside the document element. Every element in the original XML document is represented by an XPath element node. At a minimum, an element node is the parent of one attribute node for each attribute in the XML source document. ***XPath is in fact a subset of XQuery.***

XQuery

XQuery stands for XML query. It was ***designed to query collections of XML data***. XQuery supports XPath natively therefore it can do everything that XPath can do. It supplements this with a ***SQL-like FLWOR(FOR, LET, WHERE, ORDER BY, RETURN) expression***. Its capabilities overlap with XSLT. XSLT was primarily ***conceived as a stylesheet language to render XML*** for the human reader on screen, XQuery was conceived as a database query language.***XSLT therefore is better in its handling of narrative documents with more flexible structure, while XQuery is stronger in its data handling.***

XPointer

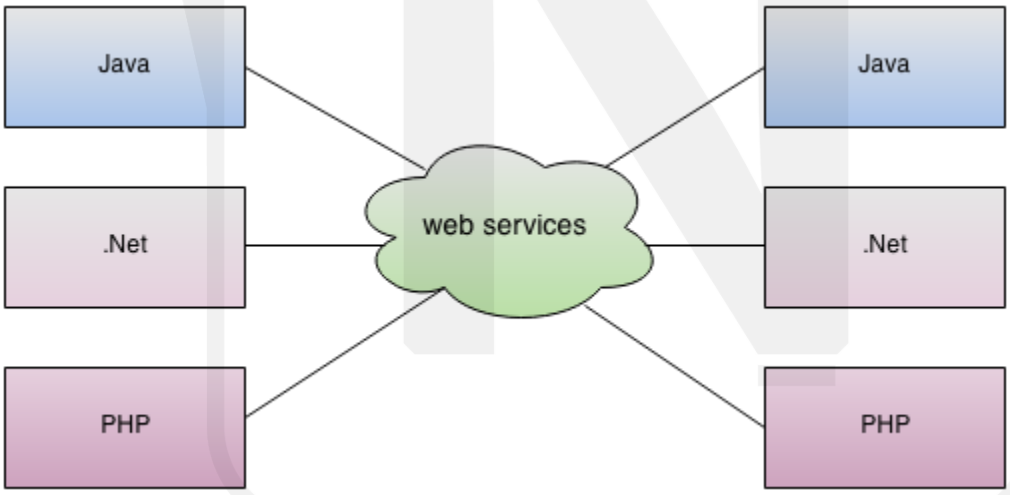
XPointer stands for XML Pointer language. The XPointer Framework is defined at http://www.w3.org/TR/xptrframework/. It is closely ***related to XPath*** and uses XPath expressions to find points inside parsed entities. It could be used to create a link from one document to an element inside another. It is similar to fragment identifier in HTML but more versatile.

WHAT IS WEB SERVICES?

A **Web Service** is can be defined by following ways:

- is a client server application or application component for communication.
- method of communication between two devices over network.
- is a software system for interoperable machine to machine communication.
- is a collection of standards or protocols for exchanging information between two devices or application.

Let's understand it by the figure given below:



As you can see in the figure, java, .net or PHP applications can communicate with other applications through web service over the network. For example, java application can interact with Java, .Net and PHP applications. So web service is a language independent way of communication.

SOAP Web Services

SOAP (Simple Object Access Protocol) is a protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence.

It is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between two applications.

SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

Advantages of Soap Web Services

WS Security: SOAP defines its own security known as WS Security.

Language and Platform independent: SOAP web services can be written in any programming language and executed in any platform.

Disadvantages of Soap Web Services

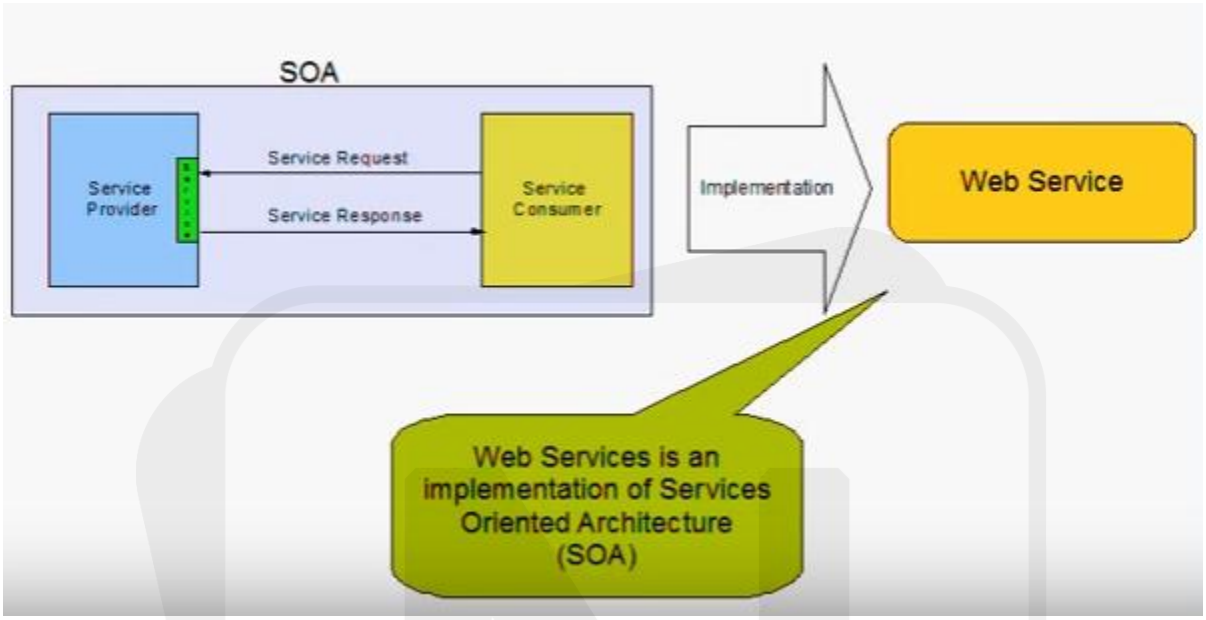
Slow: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.

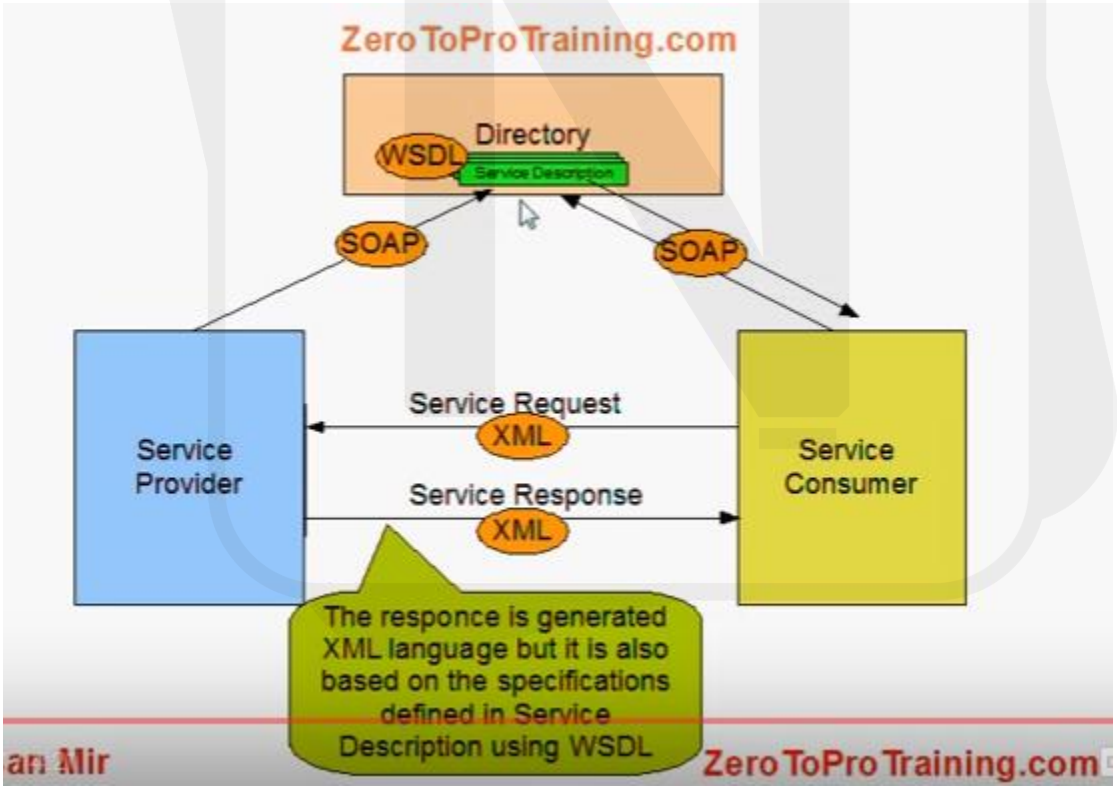
WSDL dependent: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

WHAT IS WSDL?

The Web Services Description Language (**WSDL** /ˈwɪzdəl/) is an XML-based interface definition language that is used for describing the functionality offered by a web service.

Other Diagrams for **Web services:-**





Web services play a complementary and dominant role in building global IS for today's dynamic business world. Web services are self-contained, modular applications that can be described, published, located and invoked over a network.

Web services perform functions ranging from simple requests to complicated business processes. Once a web service is developed, other applications and other web services can discover and invoke the deployed service through universal description, discovery and integration. The idea of web service is to leverage the advantages of the web as a platform to apply it to the application services. We use them, not just to the static information.