Unit - I: Database Concepts & Basics of DB2

Database Concepts and Definitions: Data, Information, metadata, Database. DBMS, Advantages of Database Management System.
Basics of DB2: Introduction to DB2, DB2 Environment, Architecture of DB2, DB2 Physical Objects
Data Base Installation, Understanding Command Line & GUI Features of IBM DB2, Usage of Control Center.

Database Concepts and Definitions:

Data:

- Data is one of the most critical assets of any business. Data is defined as collection of raw facts & figures about a place, person, and thing or object involving in the transactions of an organization and can be stored in a computer. These Facts & Figures that can be analyzed or used in an effort to gain knowledge or make some business decisions;
- Data can be represented in various forms like text, numbers, images, audio, video, graphs, document files, etc.
- Data is one of the important assets of the modern business; it becomes relevant based on the context.
- **Examples**: your name, age, height, weight, etc are some data related to you. A picture, image, file, pdf etc can also be considered data.

Information

- Information can be defined as processed data that increases the knowledge of end user and it is the result of processing raw data to reveal its meaning.
- Information is the set of data that has been organized for direct utilization of mankind, as information helps users in their decision making process. However, in common usage, the terms "data" and "information" are used synonymously.
- Good, accurate and timely information is used in decision making.
- The quality of data influences the quality of information.
- Information can be presented in the tabular form, bar graph or an image.
- **Examples**: Time Table, Merit List, Report card, Headed tables, printed documents,

Metadata

- Metadata is a special data that describes the characteristics or properties of the data.
- Metadata consists of name, data type, length, min, max, description, special constraints**.**

- Metadata allows the database designers and users understand what data exists and what data means.
- Metadata is generally stored in a repository.
- **For example**, data dictionaries and repositories provide information about the data elements in a database. Digital cameras store meta-data in the image files that include the date the photo was taken along with camera settings (see EXIF). Digital music files contain meta-data such as song title and artist name.

Database:

- Database can be defined as organized collection of logically related information so that it can be easily accessed, managed and updated.
- Databases support storage and manipulation of data it makes data management easy.
- Database can be of any size and complexity.
- Data are structured so as to be easily stored, manipulated, and retrieved by users. It can be organized into rows, columns and tables, and it is indexed to make it easier to find relevant information.
- In computing, databases are sometimes classified according to their organizational approach.
- There are many different kinds of databases, ranging from the most prevalent approach, the relational database, to a distributed database, cloud database or NoSQL database.
- Examples: Collection of information about all the students of a college, a big company can store the data of all activities of the organization which helps in decision making.

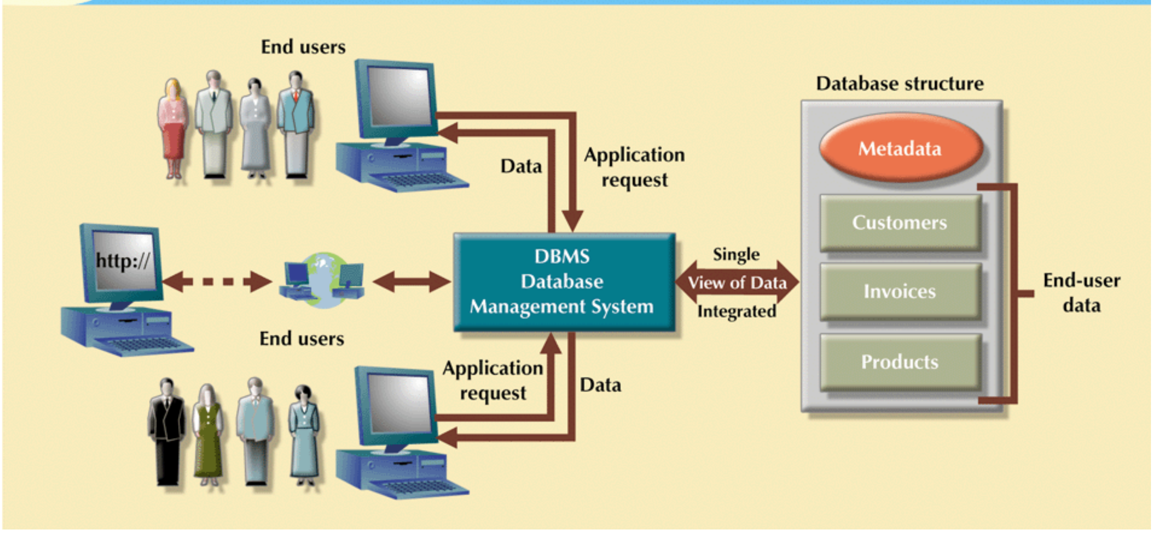
DBMS:

- Database management system (DBMS) can be defined as reorganized collection of logically related data and a set of programs used for creating, storing, updating and retrieval of data from the database. DBMS acts as a mediator between end-user and the database.
- In other words, a database management system, or simply DBMS, is a set of software tools that control access, organize, store, manage, retrieve and maintain data in a database. In practical use, the terms database, database server, database system, data server, and database management systems are often used interchangeably.
- DBMS is actually a software tool used to perform any kind of operation on data in database; it enables data to be shared and integrates many users' view of the data.
- DBMS eliminates most of file system's problems.
- A few examples of popular dbms - MySQL, Oracle, Sybase, MS-Access and IBM DB2 etc.

RDBMS:

- An RDBMS is a DBMS designed specifically for relational databases, therefore, RDBMSes are a subset of DBMSes.
- A relational database refers to a database that stores data in a structured format, using rows and columns. This makes it easy to locate and access specific values within the database. It is "relational" because the values within each table are related to each other.
- Most well known DBMS applications fall into the RDBMS category. Examples include Oracle Database, MySQL, Microsoft SQL Server, and IBM DB2.

FIGURE 1.2 The DBMS manages the interaction between the end user and the database



Types of DBMS:

1. **Hierarchical** - The hierarchical model organizes its data using a tree structure. The root of the tree is the parent followed by child nodes. A child node cannot have more than one parent, though a parent can have many child nodes.

2. **Network DBMS** - this type of DBMS supports many-to-many relations. This usually results in complex database structures. RDM Server is an example of a database management system that implements the network model.
   In 1969, CODASYL (Committee on Data Systems Languages) released its first specification about the network data model.
3. **Relational DBMS** - The relational data model is simple and elegant. It has a solid mathematic foundation based on sets theory and predicate calculus and is the most used data model for databases today. Examples of relational database management systems include MySQL, Oracle, and Microsoft SQL Server database.
4. **Object Oriented Relation DBMS** - The Object-Relational (OR) model is very similar to the relational model; however, it treats every entity as an object (instance of a class), and a relationship as an inheritance. Some features and benefits of an Object-Relational model are: Support for complex, user defined types

<u>Advantages of a DBMS:</u>

- **Improved data sharing:** A database is designed as a shared resource. Authorized users are granted permission to use the database, and each user is provided one or more user views. DBMS provides better access to data and better-managed data.

- **Improved data security:** When number of users increases to access the data, the risk of data security increases. But, DBMS provides a framework for better enforcement of data privacy and security policies. A database can be accessed only by proper authentication usually by verifying login and password.

- **Better data integration:** DBMS integrates the many different users' views into a single data repository. This gives clear picture of the organization's operations. It becomes much easier to see how actions in one segment of the company affect other segments.

- **Improved decision making:** Now a day business success depends on decision making which is based on quality information generated by databases. In DBMS, better-managed data and improved data access make it possible to generate quality information, on which better decisions are based.

- **Improved data access:** The DBMS makes it possible to produce quick answers to any queries. A query is a request or a question put to the DBMS for data manipulation or retrieval. Without any programming experience, one can retrieve and display data very easily. The language used to write queries is called Structured Query Language (SQL). For example, records from EMP table can be displayed using the query "SELECT * FROM EMP"

- **Minimized data inconsistency:** Data inconsistency exists when different versions of the same data appear in different places. In a DBMS, by eliminating this data redundancy, we can improve data consistency. For example, if a customer address is stored only once, updating that becomes simple.

- **Program-Data Independence:** The separation of data description (metadata) from the application programs that use the data is called data independence. With the database approach, data descriptions are stored in a central location called the repository. This allows an organization's data to change without changing the application programs that process the data.

Basics of DB2

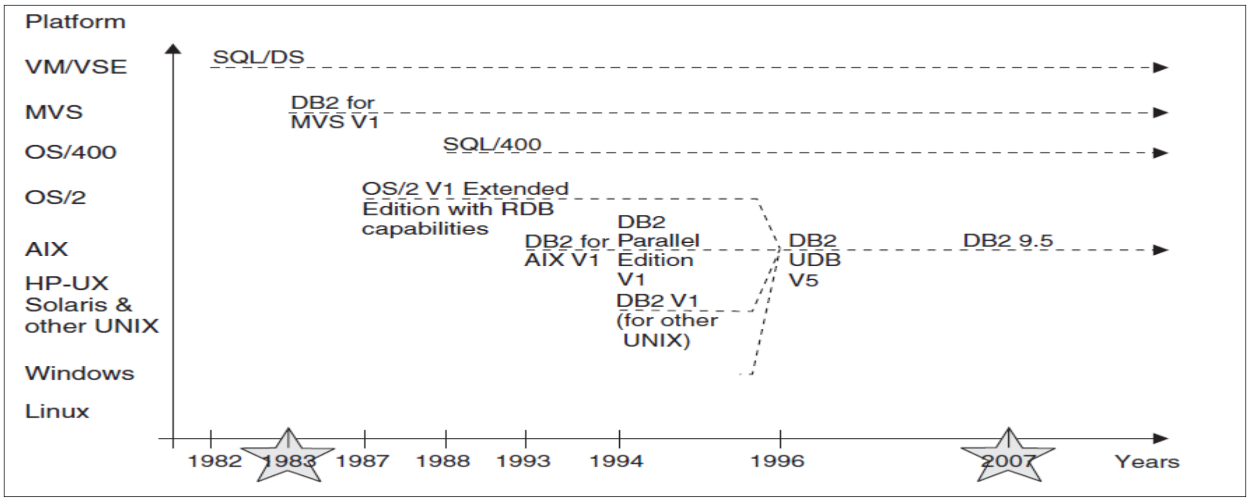
Introduction to DB2:

- DB2 is a product of IBM's Relational Database Management System for MVS (Multiple Virtual Storage – Mainframe Platform) Operating system.
- DB2 is designed to store, analyze and retrieve the data efficiently.
- DB2 database supports Object Oriented features and non relational structure with XML.
- IBM Db2 is a family of hybrid data management products offering a complete suite of AI-empowered capabilities designed to help you manage both structured and unstructured data on premises as well as in private and public cloud environments.
- DB2 is built on an intelligent common SQL engine designed for scalability and flexibility.

History of DB2:

The name DB2, or IBM Database 2, was first given to the Database Management System or DBMS in 1983 when IBM released DB2 on its MVS mainframe platform. It was originally released by IBM as the company's first commercially available relational database. Initially DB2 was developed for specific platform of IBM. In 1990, it was developed as a Universal Database (UDB) DB2 Server, which can run on any authoritative operating systems such as Linux, UNIX, and Windows.

"DB2" was used to indicate a shift from hierarchical databases—such as the Information Management System (IMS) popular at the time—to the new relational databases. DB2 development continued on mainframe platforms as well as on distributed platforms.1 Figure 1.1 shows some of the highlights of DB2 history.



DB2 Editions:

1. **DB2 Enterprise Server Edition (ESE):** A multi-user object relational database management system for complex configurations and large database needs for platforms ranging from Intel to UNIT to SMPs. ESE offers connectivity and integration for other enterprise DB2 and Informix data sources.
2. **DB2 Universal Database Connect Enterprise Edition:** Enables local and remote client applications to create, update, control, and manage DB2 databases and host systems using Structured Query Language (SQL), DB2 APIs (Application Programming Interfaces), ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), SQLJ (Embedded SQLJ for Java), or DB2 CLI (Call Level Interface). In addition, DB2 Connect supports Microsoft Windows data interfaces such as ActiveX Data Objects (ADO), Remote Data Objects (RDO), and Object Linking and Embedding (OLE) DB.
3. **DB2 Workgroup Server Edition (WSE):** A multi-user, object relational DBMS for applications and data shared in a PC LAN-based workgroup.
4. **DB2 Universal Database Personal Edition (PE):** A single-user, object-relational database management system for use on a PC.
5. **DB2 Universal Database Express Edition V8.2:** A specially tailored full feature relational database for small and medium business.
6. **DB2 Universal Database Connect Application Server Edition:** It is identical to the DB2 Connect Enterprise Server in its technology. It is designed for large 7 scale demanding environments.
7. **DB2 Universal Database Connect Unlimited Edition:** A unique package offering that allows complete flexibility of DB2 Connect deployment and simplifies product selection and licensing. This product contains both DB2 Connect Personal Edition and DB2 Connect Enterprise Edition with license terms and conditions that allow the unlimited deployment of any DB2 Connect product.

Figure illustrates the different editions and the types of servers they typically run on. By default, DB2 takes advantage of all the processing power it is given. The figure also shows that DB2 is a scalable product. With the exception of DB2 Everyplace, the functions, features, and benefits of an edition shown on the bottom of the figure are included in each subsequent edition as you move up the figure. The following sections provide more detail on the functionality of each edition.

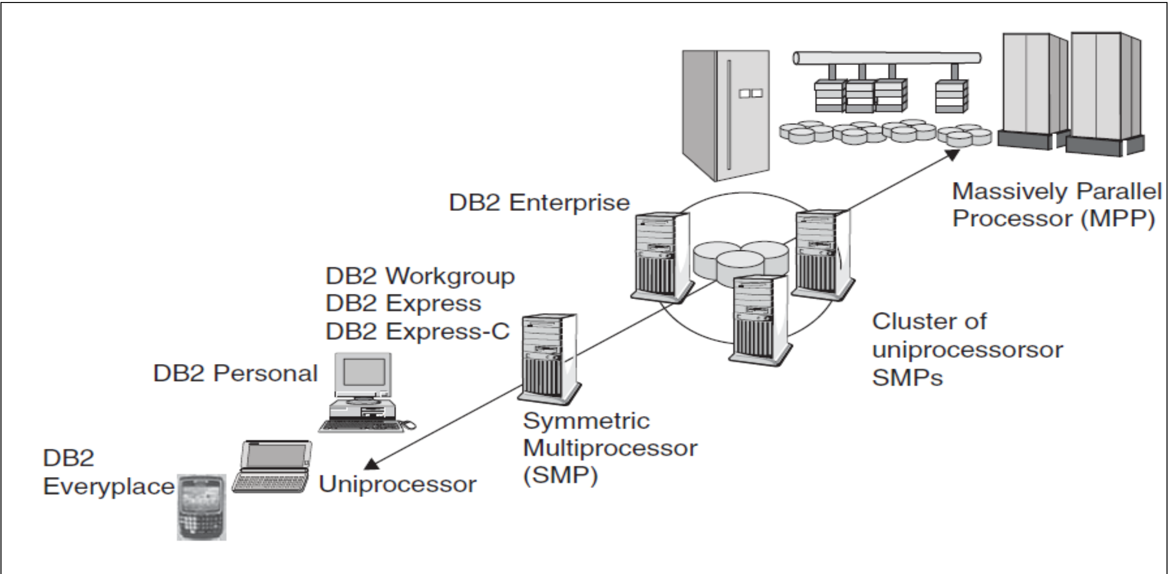
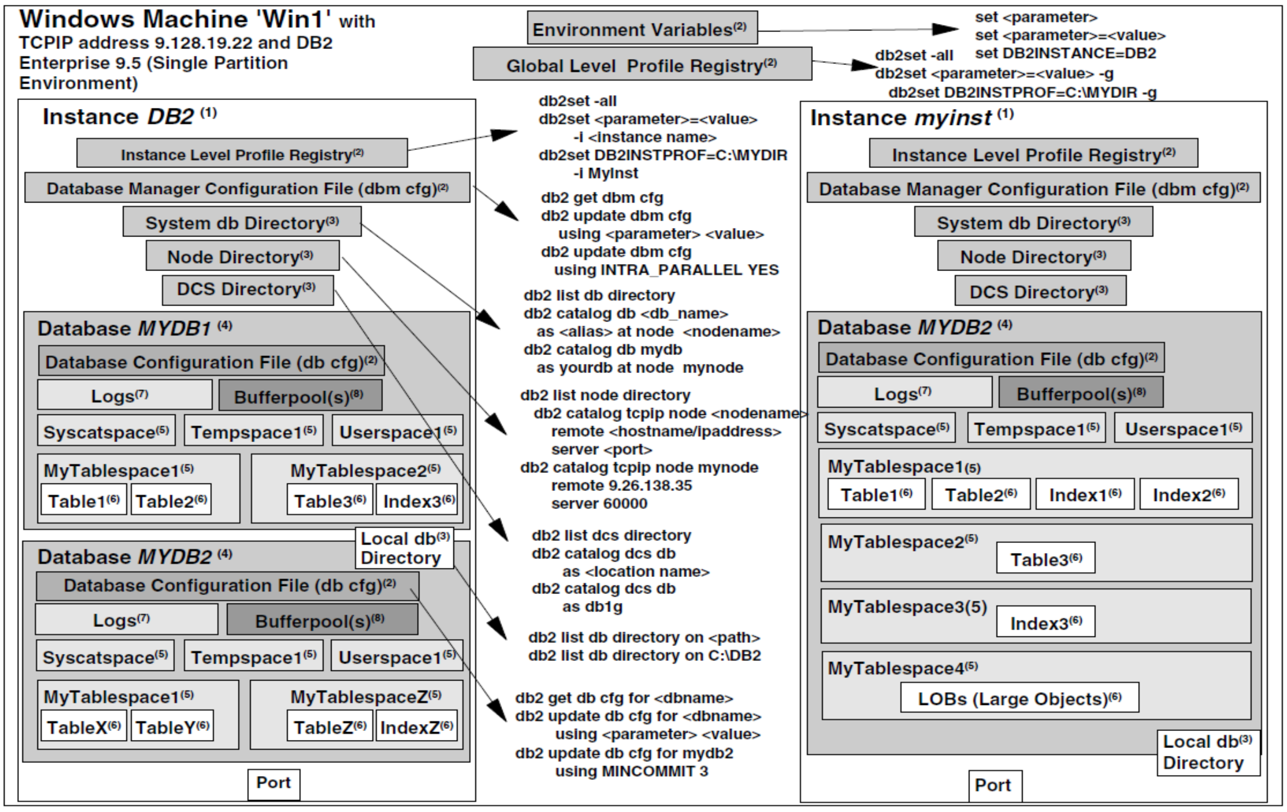

DB2 Enterprise
Massively Parallel Processor (MPP)
DB2 Workgroup
DB2 Express
DB2 Express-C
Cluster of uniprocessorsor SMPs
DB2 Personal
Symmetric Multiprocessor (SMP)
DB2 Everyplace
Uniprocessor

Figure 1.8 DB2 editions

DB2 Environment:

The following figure provides an overview of the DB2 environment. Consider the following when you review this figure:



```
set <parameter>
set <parameter>=<value>
set DB2INSTANCE=DB2

db2set -all
db2set <parameter>=<value> -g
  db2set DB2INSTPROF=C:\MYDIR -g

db2set -all
db2set <parameter>=<value>
      -i <instance name>
db2set DB2INSTPROF=C:\MYDIR
      -i MyInst

db2 get dbm cfg
db2 update dbm cfg
   using <parameter> <value>
db2 update dbm cfg
   using INTRA_PARALLEL YES

db2 list db directory
db2 catalog db <db_name>
  as <alias> at node  <nodename>
db2 catalog db mydb
  as yourdb at node  mynode

db2 list node directory
  db2 catalog tcpip node <nodename>
     remote <hostname/ipaddress>
     server <port>
db2 catalog tcpip node mynode
     remote 9.26.138.35
     server 60000

db2 list dcs directory
db2 catalog dcs db
      as <location name>
db2 catalog dcs db
      as db1g

db2 list db directory on <path>
db2 list db directory on C:\DB2

db2 get db cfg for <dbname>
db2 update db cfg for <dbname>
      using <parameter> <value>
db2 update db cfg for mydb2
      using MINCOMMIT 3
```

Figure 2.4 The DB2 environment

1. An Instance

In DB2, an instance provides an independent environment where databases can be created and applications can be run against them. Because of these independent environments, databases in separate instances can have the same name. For example, in above Figure the database called MYDB2 is associated to instance DB2, and another database called MYDB2 is associated to instance myinst. Instances allow users to have separate, independent environments for production, test, and development purposes.

When DB2 is installed on the Windows platform, an instance named DB2 is created by default. In the Linux and UNIX environments, if you choose to create the default instance, it is called db2inst1.

- To create an instance explicitly, use

  db2icrt instance_name
- To drop an instance, use

  db2idrop instance_name
- To start the current instance, use

  db2start
- To stop the current instance, use

  db2stop

When an instance is created on Linux and UNIX, logical links to the DB2 executable code are generated. For example, if the server in Figure 2.4 was a Linux or UNIX server and the instances DB2 and myinst were created, both of them would be linked to the same DB2 code. A logical link works as an alias or pointer to another program.

2. The Database Administration Server

The Database Administration Server (DAS) is a daemon or process running on the database server that allows for remote graphical administration from remote clients using the Control Center. If you don't need to administer your DB2 server using a graphical interface from a remote client, you don't need to start the DAS. There can only be one DAS per server regardless of the number of instances or DB2 install copies on the server. Note that the DAS needs to be running at the data server you are planning to administer remotely, not at the DB2 client. If you choose to administer your data servers using the DSAC, the DAS is not required.

- To start the DAS, use the following command:

  db2admin start
- To stop the DAS, use the following command:

db2admin stop

3. Configuration Files and the DB2 Profile Registries

Like many other RDBMSs, DB2 uses different mechanisms to influence the behavior of the database management system. These include:

- **a)** Environment variables
- **b)** DB2 profile registry variables
- **c)** Configuration parameters

a). Environment Variables:

Environment variables are defined at the operating system level. On Windows you can create a new entry for a variable or edit the value of an existing one by choosing **Control Panel** > **Sys-tem** > **Advanced Tab** > **Environment Variables.**

The DB2INSTANCE environment variable allows you to specify the current active instance to which all commands apply. If DB2INSTANCE is set to *myinst*, then issuing the command **CREATE DATABASE mydb** will create a database associated to instance *myinst*. If you wanted to create this database in instance *DB2*, you would first change the value of the DB2INSTANCE variable to *DB2*.

- To check the current setting of this variable, you can use any of these three commands:

  echo %DB2INSTANCE% (Windows only)

  set DB2INSTANCE

  db2 get instance

- For a list of all available instances in your system, issue the following command:

  db2ilist

b). the DB2 Profile Registry:

The word “registry” always causes confusion when working with DB2 on Windows. The DB2 profile registry variables, or simply the DB2 registry variables, have no relation whatsoever with the Windows Registry variables. The DB2 registry variables provide a centralized location where some key variables influencing DB2’s behavior reside.

The DB2 Profile Registry is divided into four categories:

- The DB2 instance-level profile registry
- The DB2 global-level profile registry
- The DB2 instance node-level profile registry

- The DB2 instance profile registry

The first two are the most common ones. The main difference between the global-level and the instance-level profile registries, as you can tell from their names, is the scope to which the variables apply. Global-level profile registry variables apply to all instances on the server. As we can see from Figure, this registry has been drawn outside of the two instance boxes. Instance-level profile registry variables apply to a specific instance.

- To view the current DB2 registry variables, issue the following command from the CLP:

  db2set -all

  You may get output like this:

  DB2INSTPROF=C:\PROGRAM FILES\SQLLIB

  DB2SYSTEM=PRODSYS

As you may have already guessed, **[i]** indicates the variable has been defined at the instance level, while **[g]** indicates that it has been defined at the global level.

The following are a few other commands related to the DB2 Registry variables.

- To view all the registry variables that can be defined in DB2, use this command:

  db2set –lr

c). Configuration Parameters:

Configuration parameters are defined at two different levels: the instance level and the database level. The variables at each level are different (not like DB2 registry variables, where the same variables can be defined at different levels).

At the instance level, variables are stored in the Database Manager Configuration file (dbm cfg). Changes to these variables affect all databases associated to that instance, which is why the figure shows a Database Manager Configuration file box defined per instance and outside the databases.

- To view the contents of the Database Manager Configuration file, issue the command:

  db2 get dbm cfg

- To update the value of a specific parameter, use

  db2 update dbm cfg using *parameter value*

  For example

  db2 update dbm cfg using INTRA_PARALLEL YES

Many of the Database Manager Configuration parameters are now “configurable online,” meaning the changes are dynamic—you don’t need to stop and start the instance.

At the database level, parameter values are stored in the Database Configuration file (db cfg). Changes to these parameters only affect the specific database the Database Configuration file applies to. In Figure we can see there is a Database Configuration file box inside each of the databases defined.

- To view the contents of the Database Configuration file, issue the command:

  db2 get db cfg for *dbname*

  For example

  db2 get db cfg for mydb2
- To update the value of a specific variable, use

  db2 update db cfg for *dbname* using *parameter value*

  For example

  db2 update db cfg for mydb2 using MINCOMMIT 3

Many of these parameters are configurable online, meaning that the change is dynamic, and you no longer need to disconnect all connections to the database for the change to take effect.

4. Connectivity and DB2 Directories

In DB2, directories are used to store connectivity information about databases and the servers on which they reside. There are four main directories and the corresponding commands to set up database and server connectivity are also included; however, many users find the Configuration Assistant graphical tool very convenient to set up database and server connectivity.

a). System Database Directory

The system database directory (or system db directory) is like the main “table of contents” that contains information about all the databases to which you can connect from your DB2 server. As you can see from Figure, the system db directory is stored at the instance level.

To list the contents of the system db directory, use the command:

db2 list db directory

Any entry from the output of this command containing the word *Indirect* indicates that the entry is for a local database, that is, a database that resides on the data server on which you are working.

Any entry containing the word *Remote* indicates that the entry is for a remote database—a database residing on a server other than the one on which you are currently working.

- To enter information into the system database directory, use the **catalog** command:

  db2 catalog db *dbname* as *alias* at node *nodename*
- For example

  db2 catalog db mydb as yourdb at node mynode

b). Local Database Directory

The local database directory contains information about databases residing on the server where you are currently working. Figure shows the local database directory overlapping the data-base box. This means that there will be one local database directory associated to all of the data-bases residing in the same location (the drive on Windows or the path on Linux or UNIX). The local database directory does not reside inside the database itself, but it does not reside at the instance level either; it is in a layer between these two.

To list the contents of the local database directory, issue the command:

db2 list db directory on *drive* / *path*

where **drive** can be obtained from the item *Database drive* (Windows) or **path** from the item *Local database directory* (Linux or UNIX) in the corresponding entry of the system db directory.

c). Node Directory:

The node directory stores all connectivity information for remote database servers. For example, if you use the TCP/IP protocol, this directory shows entries such as the host name or IP address of the server where the database to which you want to connect resides, and the port number of the associated DB2 instance.

- To list the contents of the node directory, issue the command:

  db2 list node directory
- To enter information into the node directory, use

  db2 catalog tcpip node *node_name*

  remote hostname or IP_address

  server service_name or port_number
- For example

  db2 catalog tcpip node mynode

remote 192.168.1.100

server 60000

d). Database Connection Services Directory:

The Database Connection Services (DCS) directory contains connectivity information for host databases residing on System z (z/OS or OS/390) or System i (OS/400) server. You need to have DB2 Connect software installed.

- To list the contents of the DCS directory, issue the following command:
  db2 list dcs directory
- To enter information into the DCS directory, use
  db2 catalog dcs db *dbname* as *location_name*
- For example
  db2 ctalog dcs db mydb as db1g

5. Databases

A database is a collection of information organized into interrelated objects like table spaces, tables, and indexes. Databases are closed and independent units associated to an instance. Because of this independence, objects in two or more databases can have the same name. For example, Figure shows a table space called *MyTablespace1* inside the database *MYDB1* associated to instance *DB2*. Another table space with the name *MyTablespace1* is also used inside the database *MYDB2*, which is also associated to instance *DB2*.

You create a database with the command **CREATE DATABASE.** This command automatically creates three table spaces, a buffer pool, and several configuration files, which is why this command can take a few seconds to complete.

6. Table Spaces

Table Spaces are logical objects used as a layer between logical tables and physical containers. **Containers** are where the data is physically stored in files, directories, or raw devices. When you create a table space, you can associate it to a specific buffer pool (database cache) and to specific containers.

Three table spaces—SYSCATSPACE (holding the Catalog tables), TEMPSPACE1 (system temporary space), and USERSPACE1 (the default user table space)—are automatically created when you create a database. SYSCATSPACE and TEMPSPACE1 can be considered system structures, as they are needed for the normal operation of your database. SYSCATSPACE contains the catalog tables containing **metadata** (data about your database objects) and must exist at all times. Some other RDBMSs call this structure a "data dictionary."

A system temporary table space is the work area for the database manager to perform operations, such as joins and overflowed sorts. There must be at least one system temporary table space in each database.

The USERSPACE1 table space is created by default, but you can delete it. To create a table in a given table space, use the **CREATE TABLE** statement with the **IN table_space_name** clause. If a table space is not specified in this statement, the table will be created in the first user-created table space. If you have not yet created a table space, the table will be created in the USERSPACE1 table space.

7. Tables, Indexes, and Large Objects

A **table** is an unordered set of data records consisting of columns and rows. An **index** is an ordered set of pointers associated with a table, and is used for performance purposes and to ensure uniqueness. Non-traditional relational data, such as video, audio, and scanned documents, are stored in tables as large objects (LOBs). Tables and indexes reside in table spaces. Chapter 8 describes these in more detail.

8. Logs

Logs are used by DB2 to record every operation against a database. In case of a failure, logs are crucial to recover the database to a consistent point.

9. Buffer Pools

A **buffer pool** is an area in memory where all index and data pages other than LOBs are processed. DB2 retrieves LOBs directly from disk. Buffer pools are one of the most important objects to tune for database performance.

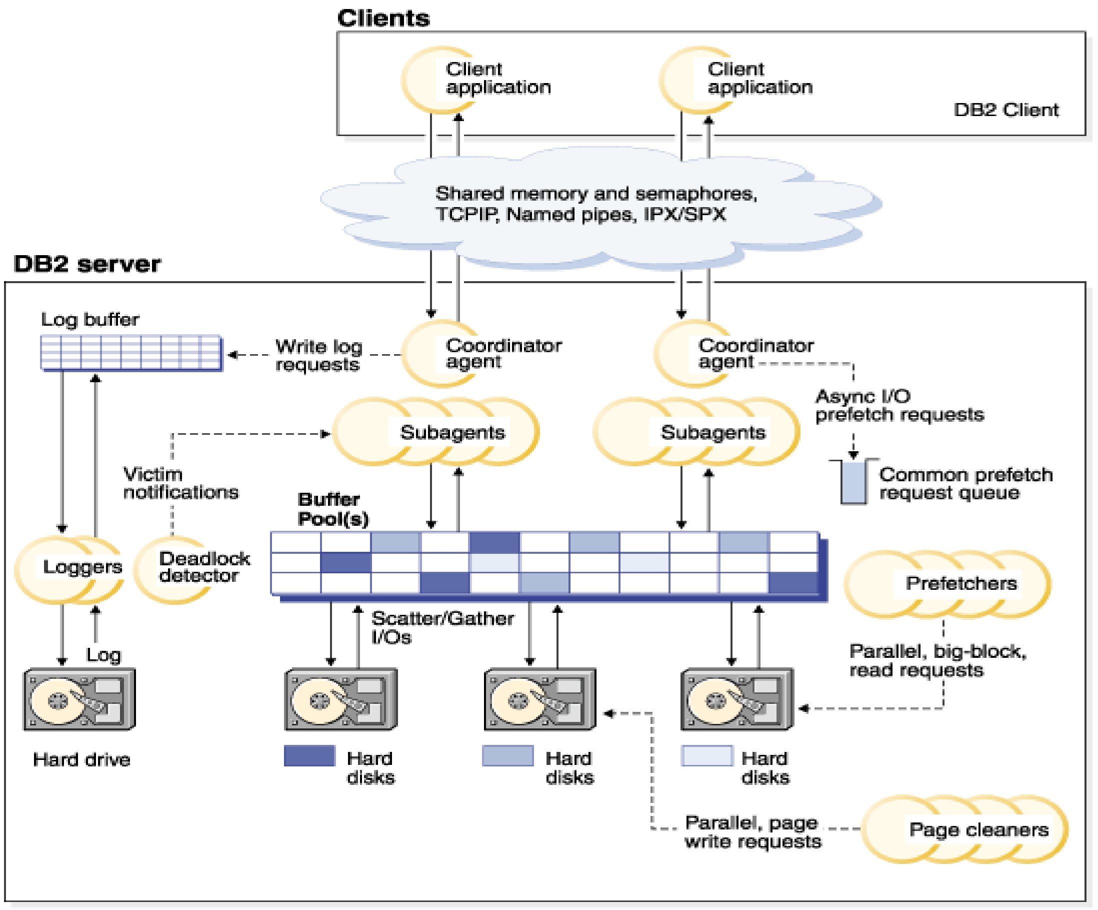
10. The Internal Implementation of the DB2 Environment

We have already discussed DB2 registry variables, configuration files, and instances. In this section we illustrate how some of these concepts physically map to directories and files in the Windows environment.

Architecture of DB2:

General information about DB2 Architecture and Processes can help you understand detailed information provided for specific topics. The following figure shows a general overview of the architecture and processes for IBM® DB2 Version 9.5.

Architecture and Processes Overview



On the client side, either local or remote applications, or both, are linked with the DB2 client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE) or TCP/IP.

On the server side, activity is controlled by engine dispatch able units (EDUs). In all figures in this section, EDUs are shown as circles or groups of circles. EDUs are implemented as threads on all platforms in Version 9.5. DB2 agents are the most common type of EDUs. These agents perform most of the SQL and XQuery processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process the client application requests. Multiple subagents can be assigned if the machine where the server resides has multiple processors or is part of a partitioned database. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit the many processors.
All agents and subagents are managed using a pooling algorithm that minimizes the creation and destruction of EDUs.

Buffer pools are areas of database server memory where database pages of user table data, index data, and catalog data are temporarily moved and can be modified. Buffer pools are a key determinant of database performance because data can be accessed much faster from memory than from disk. If more of the data needed by applications is present in a buffer pool, less time is required to access the data than to find it on disk.

The configuration of the buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed and how readily available it is to applications.

- **Prefetchers** retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter-read input operations to bring the requested pages from disk to the buffer pool. If you have multiple disks for storage of the database data, the data can be striped across the disks. Striping data lets the prefetchers use multiple disks at the same time to retrieve data.
- **Page cleaners** move data from the buffer pool back out to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages that have been

modified and write those changed pages out to disk. Page cleaners ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

- Without the independent prefetchers and the page cleaner EDUs, the application agents would have to do all of the reading and writing of data between the buffer pool and disk storage.

DB2 Physical Objects:

Logical Data Modeling and Physical Data Modeling are two tasks that we need to perform to design a DB2 database.

To design a database, you perform two general tasks. The first task is logical data modeling, and the second task is physical data modeling. In logical data modeling, you design a model of the data without paying attention to specific functions and capabilities of the DBMS that stores the data. In fact, you could even build a logical data model without knowing which DBMS you will use. Next comes the task of physical data modeling. This is when you move closer to a physical implementation. The primary purpose of the physical design stage is to optimize performance while ensuring the integrity of the data.

After completing the logical and physical design of your database, you implement the design.

- **Logical database design using entity-relationship modeling**: Before you implement a database, you should plan or design it so that it satisfies all requirements. This first task of designing a database is called logical design.
- **Logical database design with Unified Modeling Language**: UML modeling is based on object-oriented programming principals. UML defines a standard set of modeling diagrams for all stages of developing a software system.
- **Physical database design**: The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns.

Physical database design consists of defining database objects and their relationships.You can create the following database objects in a Db2® database:

- Tables

- Constraints
- Indexes
- Triggers
- Sequences
- Views
- Usage lists

We can use Data Definition Language (DDL) statements or tools such as IBM® Data Studio to create these database objects. The DDL statements are generally prefixed by the keywords CREATE or ALTER. Understanding the features and functionality that each of these database objects provides is important to implement a good database design that meets your current business's data storage needs while remaining flexible enough to accommodate expansion and growth over time.

Concepts common to most database objects

- **Tables**

  Tables are logical structures maintained by the database manager and are made up of columns and rows. Db2 databases store persistent data in tables, but there are also tables that are used for presenting results, summary tables and temporary tables.
- **Constraints**

  Within any business, data must often adhere to certain restrictions or rules. For example, an employee number must be unique. The database manager provides *constraints* as a way to enforce such rules.
- **Indexes**

  An *index* is a set of pointers that are logically ordered by the values of one or more keys. The pointers can refer to rows in a table, blocks in an MDC or ITC table, XML data in an XML storage object, and so on.
- **Triggers**

  A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*. Triggers are optional and are defined using the CREATE TRIGGER statement.

- **Sequences**

  A *sequence* is a database object that allows the automatic generation of values, such as cheque numbers. Sequences are ideally suited to the task of generating unique key values. Applications can use sequences to avoid possible concurrency and performance problems resulting from column values used to track numbers. The advantage that sequences have over numbers created outside the database is that the database server keeps track of the numbers generated. A crash and restart will not cause duplicate numbers from being generated.

- **Views**

  A *view* is an efficient way of representing data without the need to maintain it. A view is not an actual table and requires no permanent storage. A virtual table is created and used.

- **Cursors**

  A *cursor* is used in an application program to select a set of rows and then process that returned data one row at a time. When a SELECT statement in an embedded SQL application returns multiple rows of data, you need a mechanism that makes this returned data or result set available to your application program, one row after another.

- **Member subsets overview**

  A member subset is a database object that expresses a relationship between a database alias and a server list. The server list is composed of a set of members in a Db2 instance.

- **Usage lists**

  A *usage list* is a database object that records each DML statement section that references a particular table or index. A section is the executable form of the query. Statistics are captured for each statement section as it executes. Use usage lists when you want to determine which DML statements, if any, affected a table or index.

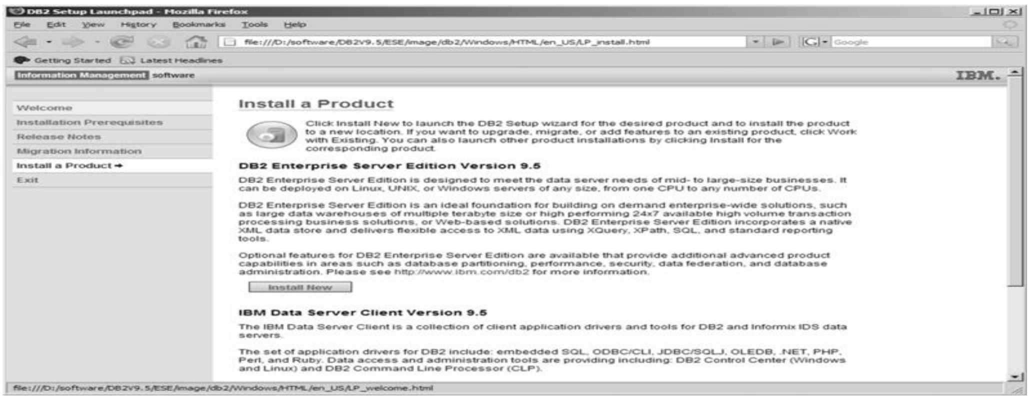
Data Base Installation:

To install DB2 on a Windows operating system, complete the following steps:

1. Log on with a user ID that has Administrator authority on Windows.
2. Insert the DB2 product DVD into the DVD drive or use the extracted DB2 installation image. Windows Autorun starts the launchpad.
3. If you are using Windows Autorun, the installation program should start in 15-30 seconds. If the installation program does not start, complete one of the following steps:

a. In a command prompt, to start the DB2 set up page, go to DVD-ROM\server_folder or go to *web_image_extraction\server_folder*and run the **setup.exe** command.
b. In Windows Explorer, go to DVD-ROM\server_folder or go to *web_image_extraction\server_folder*, and double-click the setup.exe file.

4. Where *server_folder* in the preceding steps is the location of the DB2 installation program.
5. In the DB2 Setup Launchpad, click **Install a Product**.


DB2 Setup Launchpad - Mozilla Firefox

file:///D:/software/DB2V9.5/ESE/image/db2/Windows/HTML/en_US/LP_install.html

Information Management software

Welcome
Installation Prerequisites
Release Notes
Migration Information
Install a Product
Exit

Install a Product

Click Install New to launch the DB2 Setup wizard for the desired product and to install the product to a new location. If you want to upgrade, migrate, or add features to an existing product, click Work with Existing. You can also launch other product installations by clicking Install for the corresponding product.

DB2 Enterprise Server Edition Version 9.5

DB2 Enterprise Server Edition is designed to meet the data server needs of mid- to large-size businesses. It can be deployed on Linux, UNIX, or Windows servers of any size, from one CPU to any number of CPUs.

DB2 Enterprise Server Edition is an ideal foundation for building on demand enterprise-wide solutions, such as large data warehouses of multiple terabyte size or high performing 24x7 available high volume transaction processing business solutions, or Web-based solutions. DB2 Enterprise Server Edition incorporates a native XML data store and delivers flexible access to XML data using XQuery, XPath, SQL, and standard reporting tools.

Optional features for DB2 Enterprise Server Edition are available that provide additional advanced product capabilities in areas such as database partitioning, performance, security, data federation, and database administration. Please see http://www.ibm.com/db2 for more information.

Install New

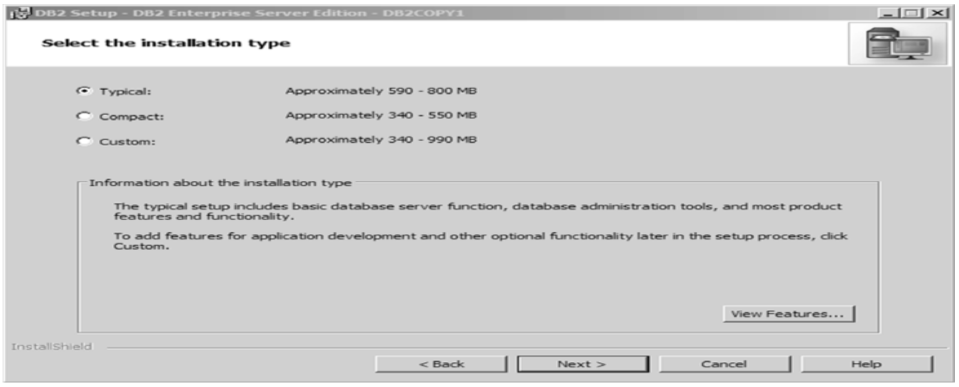
IBM Data Server Client Version 9.5

The IBM Data Server Client is a collection of client application drivers and tools for DB2 and Informix IDS data servers.

The set of application drivers for DB2 include: embedded SQL, ODBC/CLI, JDBC/SQLJ, OLEDB, .NET, PHP, Perl, and Ruby. Data access and administration tools are providing including: DB2 Control Center (Windows and Linux) and DB2 Command Line Processor (CLP).

file:///D:/software/DB2V9.5/ESE/image/db2/Windows/HTML/en_US/LP_welcome.html

6. Click **Install New** for **DB2 Version 10.5 Fix Pack 4 Workgroup, Enterprise and Advanced Editions**.
7. On the welcome page of the DB2 Setup wizard, click **Next**.
8. On the Software License Agreement page, review and accept the license agreement, and click **Next**.
9. On the Select the installation type page, click **Typical**, and click **Next**.


DB2 Setup - DB2 Enterprise Server Edition - DB2COPY1

Select the installation type

[✓ Typical:] Approximately 590 - 800 MB
Compact: Approximately 340 - 550 MB
Custom: Approximately 340 - 990 MB

Information about the installation type

The typical setup includes basic database server function, database administration tools, and most product features and functionality.

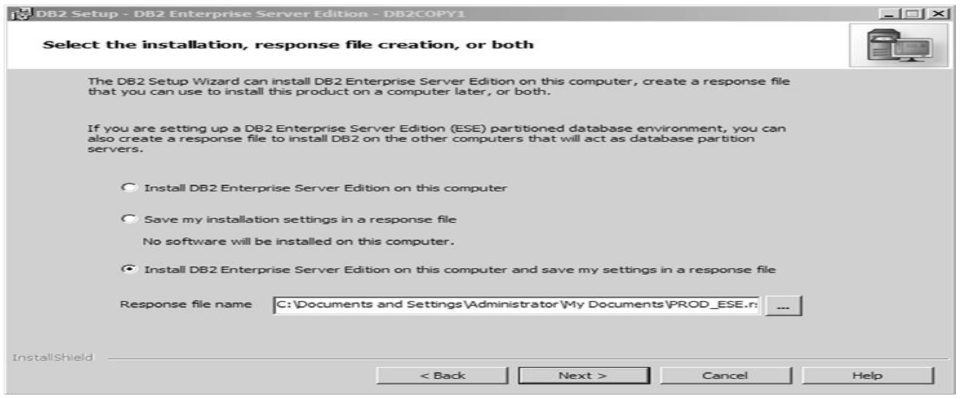
To add features for application development and other optional functionality later in the setup process, click Custom.

View Features...

InstallShield

< Back | Next > | Cancel | Help

10. On the Select the installation, response file creation, or both page, select **Install DB2 Server Edition on this computer and save my settings in a response file**.


DB2 Setup - DB2 Enterprise Server Edition - DB2COPY1

Select the installation, response file creation, or both

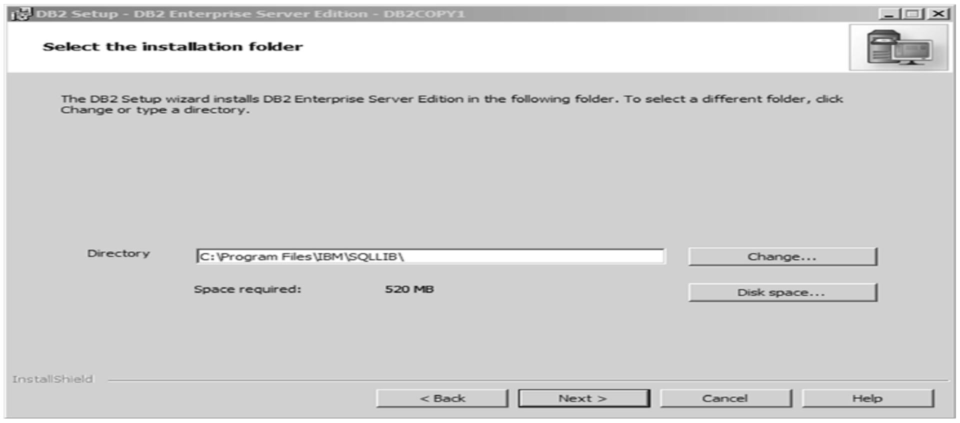
The DB2 Setup Wizard can install DB2 Enterprise Server Edition on this computer, create a response file that you can use to install this product on a computer later, or both.

If you are setting up a DB2 Enterprise Server Edition (ESE) partitioned database environment, you can also create a response file to install DB2 on the other computers that will act as database partition servers.

- ( ) Install DB2 Enterprise Server Edition on this computer
- ( ) Save my installation settings in a response file
  No software will be installed on this computer.
- (●) Install DB2 Enterprise Server Edition on this computer and save my settings in a response file

Response file name: C:\Documents and Settings\Administrator\My Documents\PROD_ESE.r:

11. Enter a response file name or accept the default, and click **Next**.
12. On the Select the installation folder page, enter a directory or accept the default, and click **Next**.


DB2 Setup - DB2 Enterprise Server Edition - DB2COPY1

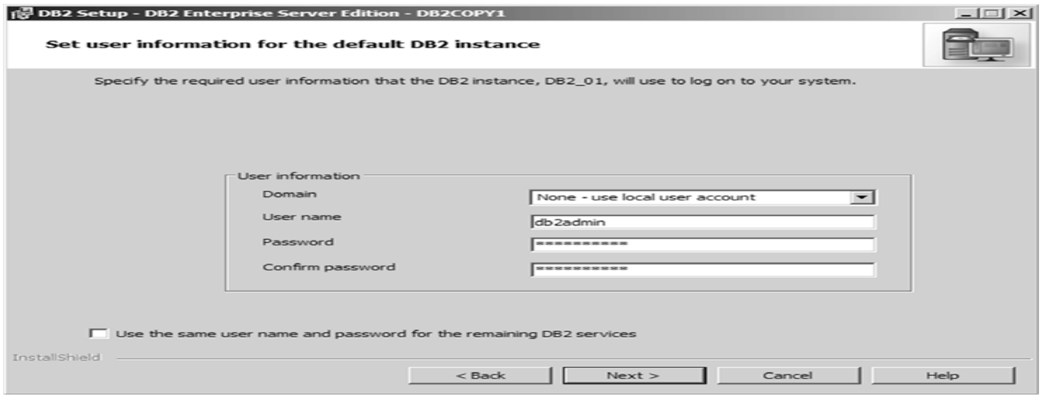
Select the installation folder

The DB2 Setup wizard installs DB2 Enterprise Server Edition in the following folder. To select a different folder, click Change or type a directory.

Directory: C:\Program Files\IBM\SQLLIB\

Space required: 520 MB

13. On the Select the IBM SSH server installation folder and startup option page, enter a directory or accept the default, and click **Next**.
14. On the Set user information for the DB2 Administration Server page, enter the following user information:
    a. Leave the **Domain** field blank.
    b. In the **User name** field, type the DB2 user ID that you want to use or accept the default.
    c. In the **Password** field, type a password and confirm the password by typing it again in the **Confirm password** field.



DB2 Setup - DB2 Enterprise Server Edition - DB2COPY1

Set user information for the default DB2 instance

Specify the required user information that the DB2 instance, DB2_01, will use to log on to your system.

User information

Domain: None - use local user account

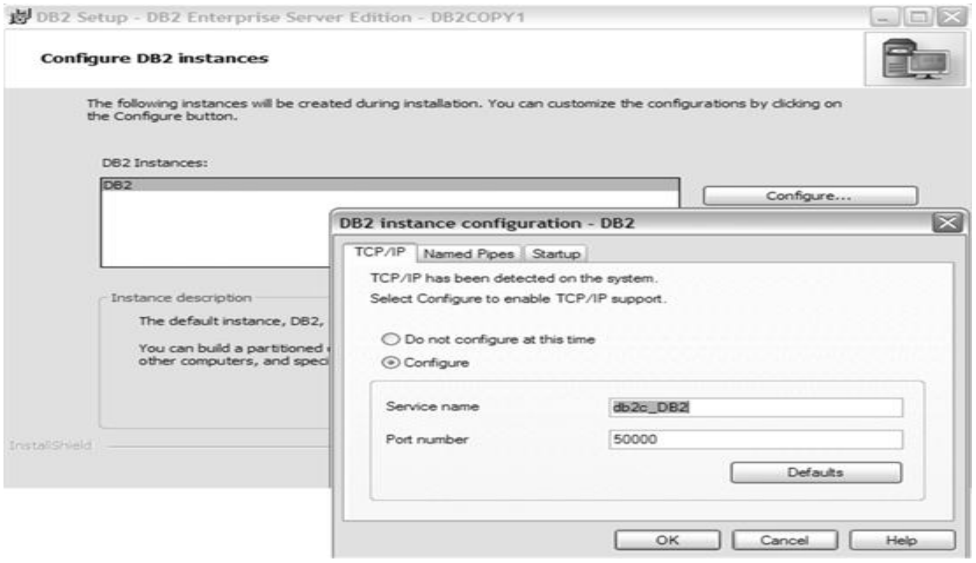
User name: db2admin

Password: ***********

Confirm password: ***********

Use the same user name and password for the remaining DB2 services

15. Select the **Use the same account for the remaining DB2 services** check box and click **Next**.
16. On the Configure DB2 instances page, click **Next**.



DB2 Setup - DB2 Enterprise Server Edition - DB2COPY1

Configure DB2 instances

The following instances will be created during installation. You can customize the configurations by clicking on the Configure button.

DB2 Instances: DB2

Instance description

The default instance, DB2,

You can build a partitioned other computers, and speci

DB2 instance configuration - DB2

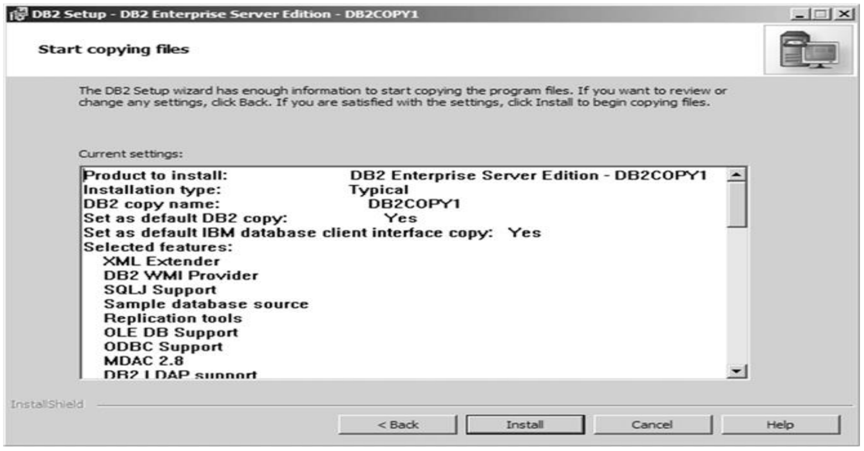
TCP/IP has been detected on the system.
Select Configure to enable TCP/IP support.

Do not configure at this time [✓ Configure]

Service name: db2c_DB2

Port number: 50000

17. On the Set up notifications page, clear the **Set up your DB2 server to send notifications** check box, and click **Next**.
18. On the Enable operating system security for DB2 objects page, accept the defaults, and click **Next**.
19. The default is to enable operating system security.
20. **Note:** If you installed DB2 before on this system, and the DB2ADMS group exists, when you click **Next**, the following message is displayed:
21. On the Start copying files and create response file page, review the current settings, and click **Finish**.



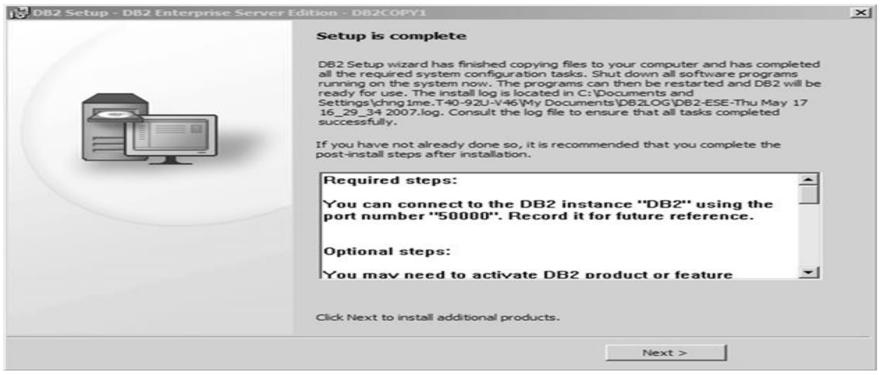
Start copying files

The DB2 Setup wizard has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Install to begin copying files.

Current settings:

```
Product to install:              DB2 Enterprise Server Edition - DB2COPY1
Installation type:               Typical
DB2 copy name:                     DB2COPY1
Set as default DB2 copy:             Yes
Set as default IBM database client interface copy:   Yes
Selected features:
   XML Extender
   DB2 WMI Provider
   SQLJ Support
   Sample database source
   Replication tools
   OLE DB Support
   ODBC Support
   MDAC 2.8
   DB2 LDAP support
```

22. On the Setup is complete page, review the information, and click **Next**.
23. On the Install additional products page, do not install more products, click **Finish**. The setup wizard closes and DB2 First Steps interface opens.



Setup is complete

DB2 Setup wizard has finished copying files to your computer and has completed all the required system configuration tasks. Shut down all software programs running on the system now. The programs can then be restarted and DB2 will be ready for use. The install log is located in C:\Documents and Settings\chng1me.T40-92U-V46\My Documents\DB2LOG\DB2-ESE-Thu May 17 16_29_34 2007.log. Consult the log file to ensure that all tasks completed successfully.

If you have not already done so, it is recommended that you complete the post-install steps after installation.

```
Required steps:

You can connect to the DB2 instance "DB2" using the
port number "50000". Record it for future reference.


Optional steps:

You may need to activate DB2 product or feature
```

Click Next to install additional products.

24. Restart the system.

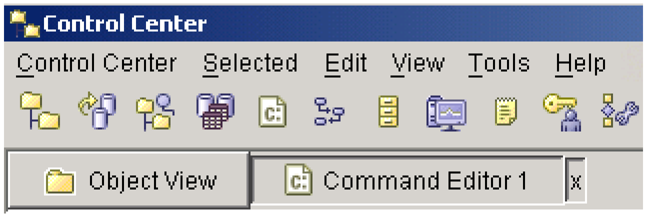
Understanding Command Line & GUI Features of IBM DB2:

The Command Editor interface:

The Command Editor is available as two different interfaces. It can be opened as part of the Control Center (embedded) or in a stand-alone view. Both versions offer the same set of functions and both allow you to open multiple Command Editors.

Embedded

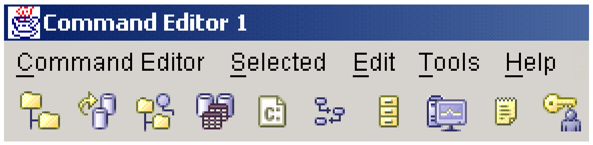
Using the Command Editor within the Control Center allows you to control the number of windows open on your desktop. The Command Editor opens as a tile inside the Control Center. Connections made by the Control Center are remembered by the Command Editor and can quickly be selected as targets for commands and SQL statements.



To open an embedded Command Editor, expand the Control Center object tree until you find a DB2 database for Linux, UNIX, and Windows, z/OS and OS/390 system or subsystem, or IMSplex. Right-click the object and select **Query** from the pop-up menu. A Command Editor tile opens in the Control Center.

Stand-alone

Using the stand-alone Command Editor allows you to execute commands and SQL statements without opening the Control Center.



To open a stand-alone Command Editor, select **Start** > **Programs** > **IBM DB2** > **Command Line Tools** > **Command Editor**.

Working in the Command Editor

The Command Editor is a notebook with a page for entering commands and pages for retrieving results. The appearance of this notebook changes depending on the target for your commands.

DB2 databases

The Command Editor appears as a three page notebook:

1. Commands page. Use this page to enter your commands and SQL statements. Command results are displayed in the output area of this page. The results of multiple SQL queries or queries executed as part of a script are also displayed in the output area. The results of a single SQL query can be returned on the Query Results page. You can also use this page to generate the access plan for SQL statements. The access plan appears in graphical form on the Access Plan page.

2. Query Results page. Use this page to view and work with the results of a single SQL query. Results are displayed in an editable table. Use the Command Editor page of the Tools Settings notebook to change this option.
3. Access Plan page. Use this page to view a graphical representation of the access plan for explainable SQL statements (for example, SELECT, INSERT, and UPDATE). An access plan indicates the order of operations for accessing requested data.

GUI Features of IBM DB2:

The graphical user interface (GUI) tools enable you to perform your job easily; however, you need to understand how the GUI works and become familiar with its features. Here are some basic GUI concepts you should be familiar with.

- **Start Menu:** The Start menu is the starting point for using the GUI Administration tools in Windows® operating environments. From the Start menu, select IBM DB2 —> General Administration Tools —> Control Center. You can also start other centers or tools such as the Journal, the Replication Center, and the Task Center from the Start menu as well as select the command line tools, the development tools, the monitoring tools, and the Information Center.
- **Windows:** Windows are panels that open and enable you to enter information pertaining to the action you want to perform, for example, you can type information into fields within a window. In the DB2 interface, fields that must be filled in are surrounded by a thick red border.
- **Notebooks**: A notebook is used to logically organize data into groups when there is too much information to fit on one page. The resulting pages are organized with tabs that reflect the particular page content.
- **Wizards**: Wizards are integrated into the administration tools. They assist you in completing a single task by stepping you through the task. To select a wizard, from the Control Center window, select Tools —> Wizards. The wizard task overview on the first page of the wizard lists any prerequisite steps and briefly describes every page of the wizard.
- **Advisors**: Advisors are integrated into the administration tools. They assist you with more complex tasks, such as tuning and performance tasks, by gathering information and recommending options that you may not have considered. You can accept or reject the advice of the advisor. To select an advisor, from the Control Center window, select Tools —> Wizards.
- **Launchpads**: Launchpads are integrated into the administration tools. They assist you in completing

high-level tasks by stepping you through a set of tasks in the right order. Launchpads can call wizards or other dialogs to accomplish the high-level task. To select a launchpad, from the Control Center window, select Tools —> Wizards.

- **Menu bars:** The Menu bar is a special panel that is displayed at the top of the window. It contains menu options that allow you to access drop-down menus. From the drop-down menus, you can select menu items. Items in the menu bar include actions that affect the objects in the center you are using. For example, menu items under Control Center include actions that affect the entire Control Center.
- **Toolbars:** Toolbars are panels that contain icons representing functions you can perform. Toolbars are located below the menu bar. To see a brief description of a tool, place your cursor over its icon and hover help will tell you what function each icon represents. Toolbars provide quick access to the functions you can perform. The functions can also be selected in the View menu. A Contents pane toolbar is located below the contents pane. It allows you to tailor the information in the contents pane.
- **Object trees:** Object trees display the system and database objects graphically in the left navigation pane, enabling you to see the hierarchical relationship between different objects and to work with those objects. You can expand the object tree to expose the objects that are contained within the object. The exposed objects are displayed as folders beneath the object. Each folder represents an object type. If the object tree is collapsed, the folders contained within the object no longer appear in the view of the object tree.
- **Infopops:** An infopop is a pop-up window that is displayed when a control in a window or notebook has focus and you press F1. Holding the mouse cursor over a control in a window or notebook also causes the infopop to display. Infopops contain a description of the field or control. They may also list restrictions and requirements or provide instructions. Infopops are disabled or re-enabled from the General page of the Tools Settings notebook.
- **Filtering:** Filtering enables you to work with a subset of displayed objects in the Control Center. Two forms of filtering exists. The first form of filtering allows you to create a customized view of objects which you would like to appear in the Contents pane of the Control Center. You select the subset of objects by right clicking the object folder in the object tree and selecting Filter —> Create. The Filter notebook opens allowing you to select which columns you would like to have in your customized view.

DB2 GUI tools. It explains what each tool is used for and why you might want to use it. It tells you how to invoke a tool and presents basic usage information. You can select the following tools from the toolbar:

- Control Center
- Replication Center
- Satellite Administration Center
- Data Warehouse Center
- Command Center
- Task Center
- Information Catalog Center
- Health Center
- Journal
- License Center
- Development Center
- Information Center

Usage of Control Center:

Among the many talents of the DB2 Control Center includes its capability to control the state of DB2 instances. More specifically, the DB2 Control Center allows you to start and stop an instance, and thereby take its related databases offline or bring them online. To perform an instance startup or shutdown, simply highlight the instance in the Control Center, as shown in Figure, and right-click or use the Selected menu to bring up the list of options.

An installation under Windows provides an IBM DB2 folder from the Start menu's All Programs list. Assuming that you retained the default DB2 Copy name of DB2COPY1, navigate through the menus DB2COPY1 (Default)

➤ General Administration Tools ➤ Control Center to launch the Control Center.

Start and Stop are pretty self-explanatory. When you stop an instance, you see a confirmation dialog box, which includes a useful option to disconnect any existing connections.

Tasks from the Control Center:

The following are some of the key tasks that you can perform with the Control Center:

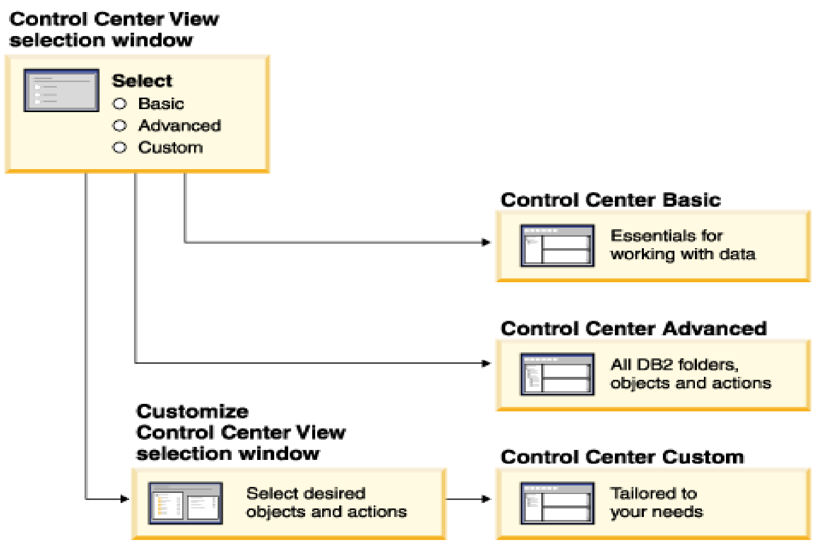
- Add DB2 database systems, federated systems, DB2 for z/OS and OS/390® systems, IMSysplexes, instances, databases, and database objects to the object tree.
- Manage database objects. You can create, alter, and drop databases, table spaces, tables, views, indexes, triggers, and schemas. You can also manage users.
- Manage data. You can load, import, export, and reorganize data. You can also gather statistics.
- Perform preventive maintenance by backing up and restoring databases or table spaces.
- Configure and tune instances and databases.
- Manage database connections, such as DB2 Connect™ servers and subsystems.
- Manage IMS systems.
- Manage DB2 UDB for z/OS and OS/390 subsystems.
- Manage applications.
- Analyze queries using Visual Explain to look at access plans.
- Launch other tools such as the Command Editor and the Health Center.

In many cases, advisors, launch pads, and wizards are available to help you perform these tasks quickly and easily.

The Control Center interface:

The Control Center interface is available in three different views:

- **Basic**. This view provides core DB2 database functionality, which includes the essential objects, such as databases, tables, and stored procedures.
- **Advanced**. This view displays all objects and actions available in the Control Center. This is the view that you should select if you are working in an enterprise environment and want to connect to DB2 for z/OS or IMS.
- **Custom**. This view gives you the ability to tailor the object tree and the object actions to your specific needs.


Control Center View selection window
Select
Basic
Advanced
Custom
Control Center Basic
Essentials for working with data
Control Center Advanced
All DB2 folders, objects and actions
Customize Control Center View selection window
Select desired objects and actions
Control Center Custom
Tailored to your needs

You can select or change your view by choosing Tools from the menu bar and selecting Customize the Control Center. You can then use your Control Center view to work with the various folders and the objects that they contain (the objects within a folder are called folder objects).

https://youtu.be/Pjvwm4rC5Ok - basics

https://youtu.be/O8CZXCpUyz8 - Architecture

https://youtu.be/2AtSEHC6iAQ - installation

https://youtu.be/Ns3h5wM7_64 - basics and sample db

https://youtu.be/a2UjLxVzxCQ - IBM DB2 Installation on Windows System
https://youtu.be/flMDot7Uqq8 - How to create a database in IBM DB2

UNIT - II

Data Modeling & Relational Database Design Concepts

Data Modeling: Data Model, Purpose of Data Models, Entity Relationship Model: Entities, Attributes & Relationships.

Relational Database Design Concepts: Defining a Relation, Keys, Entity Integrity and Referential Integrity Concepts, Functional dependencies, Normalization, Normal Forms, Codd's Rules, De-normalization.

Data Modeling

Data Model:

Database design focuses on how the database structure will be used to store and manage end-user data. Data modeling, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain. (A problem domain is a clearly defined area within the real-world environment, with a well-defined scope and boundaries that will be systematically addressed.) Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain.

- A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. A model's main function is to help you understand the complexities of the real-world environment.
- Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data.
- Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.
- Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data.
- Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding increases, so does the level of detail of the data model. When done properly, the final data model effectively is a "blueprint"

with all the instructions to build a database that will meet all end-user requirements.

An implementation-ready data model should contain at least the following components:

- A description of the data structure that will store the end-user data
- A set of enforceable rules to guarantee the integrity of the data
- A data manipulation methodology to support the real-world data transformations

The Purpose of Data Model: The major goals of using data model are -

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.

Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.
- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

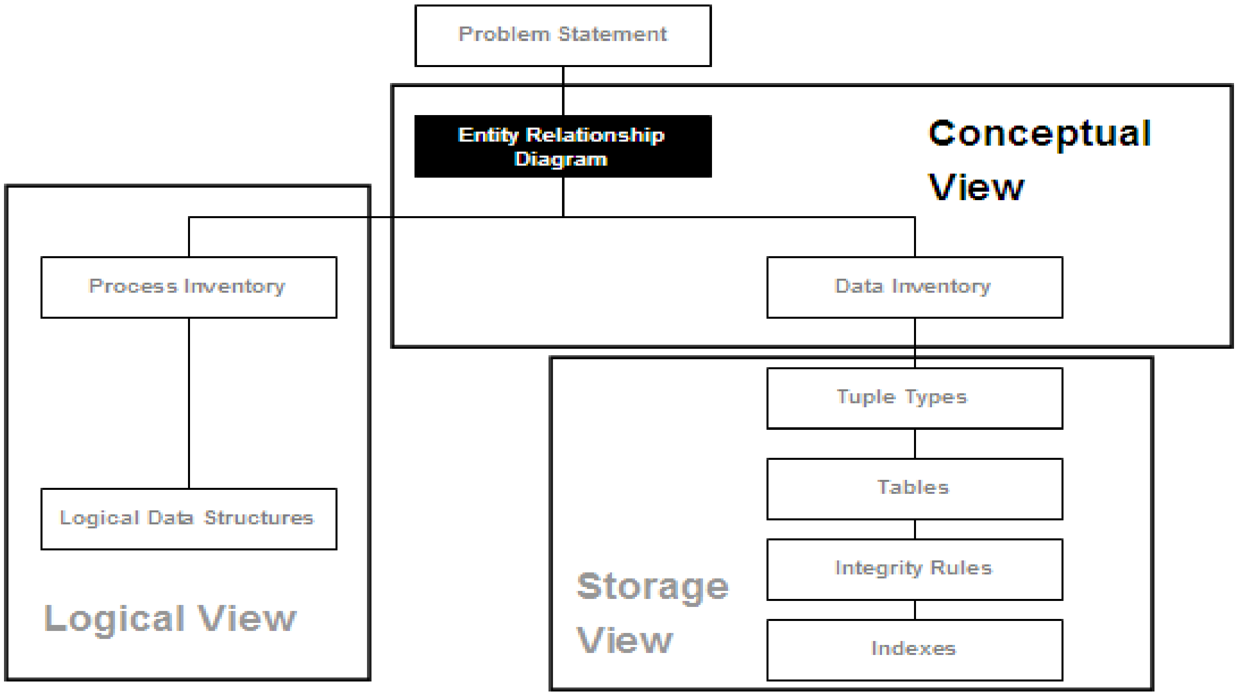
<u>Entity Relationship (ER) Model:</u>

In the mid 1970's, Peter Chen proposed the entity-relationship (E-R) data model. This was to be an alternative to the relational, CODASYL, and hierarchical data models. He proposed thinking of a database as a collection of instances of entities.

Entity-Relationship Model:

An **entity–relationship model (E-R model)** is a systematic way of describing and defining a business process. An -ER model is typically implemented as a database. The E-R model defines the conceptual view of a database, and is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the E-R Model creates entity set, relationship set, general attributes and constraints.

The development of an *entity-relationship model* for an application domain is the first step of the ***conceptual view*** during which the process - and implementation-independent architecture of the data of the application domain is established.



- The application domain, described by the problem statement, is scrutinized for its business object types, the relationships between them, and for business constraints. As a result of the scrutiny, an entity-relationship model is established visualizing and structuring the business object types of the application domains as entity types; illustrating the relationship types between the entity types resulting from the business relationship types of the application domain; and modeling the constraints for the entity types and relationship types imposed by the business constraints.
- The entity-relationship model is the basis for all subsequent design steps. A wrong assumption for the entity- relationship model will produce incorrect results for the subsequent steps. The

development of the entity-relationship model is not a one-time affair.

- The entity-relationship model is maintained constantly and changed as the subsequent steps reveal errors or discover undocumented business object types, business relationship types, or business constraints. If the problems found concern undocumented items of the application domain or items not properly described in the problem statement, the problem statement must be corrected as well. It should be corrected by the domain expert.

ER diagram-

- ER diagram or Entity Relationship diagram is a conceptual model that gives the graphical representation of the logical structure of the database.
- It shows all the constraints and relationships that exist among the different components.

Components of E-R Model - Entities, Attributes and Relationships:

The relational data model uses formal terms to define its concepts. We use formal relational terminology like: attribute, domain, tuple, relation, schema, candidate key, primary key and foreign key.

Entity:

- An entity set is a set of same type of entities.
- An entity refers to any object having-
    - Either a physical existence such as a particular person, office, house or car.
    - Or a conceptual existence such as a school or a company.
- An entity can be represented by rectangle symbol in E-R diagram
- An entity set may be of the following two types-
    - **Strong entity**:- An entity that has primary key of its own
    - **Weak entity**:-: an entity that depends for primary key on another table

Example-

Consider the following Student table-

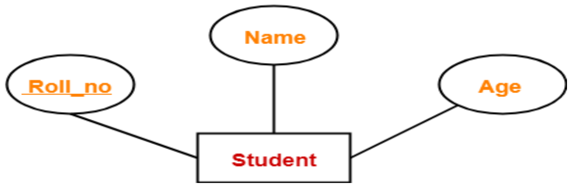
Roll_no	Name	Age
1	Akshay	20
2	Rahul	19
3	Pooja	20

4	Aarti	19

This complete table is referred to as “Student Entity Set” and each row represents an “entity”.

Representation of above table in ER Diagram-

The above table may be represented as ER diagram as-



Here,

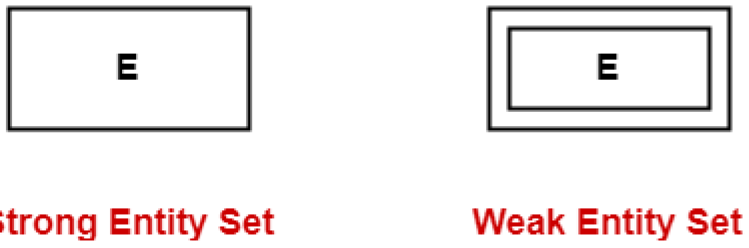
- Student is an entity , Roll_no, Name and Age are attributes .
- Roll_no is a primary key that can identify each entity uniquely.
- Thus, by using student’s roll number, a student can be identified uniquely.

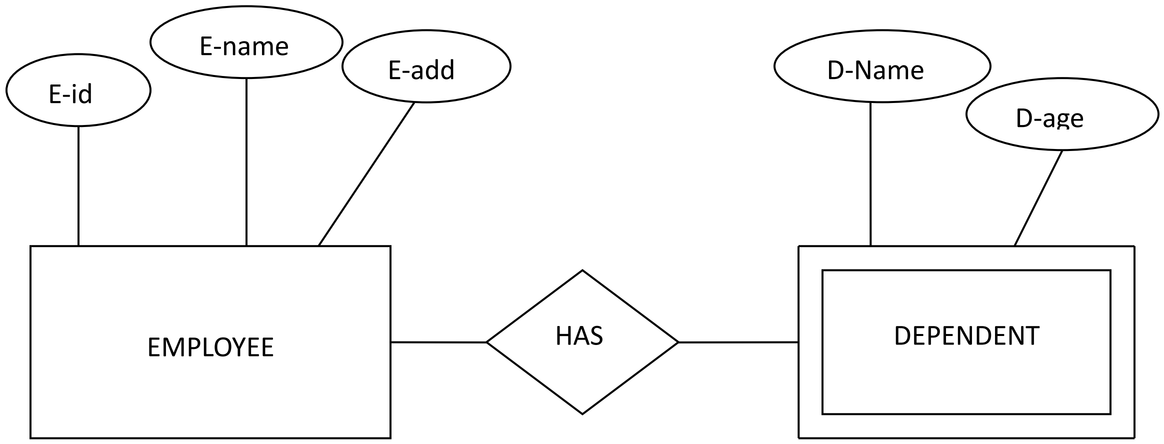
1. Strong Entity Set-

- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities. in other words A strong entity set possess its own primary key.
- Primary key of a strong entity set is represented by underlining it.
- It is represented using a single rectangle.

2. Weak Entity Set-

- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.
- In other words, a primary key does not exist for a weak entity set. However, it contains a partial key called as a **discriminator.**
- Discriminator can identify a group of entities from the entity set, Discriminator is represented by underlining with a dashed line.
- A weak entity set do not possess its own primary key.
- It is represented using a double rectangle.

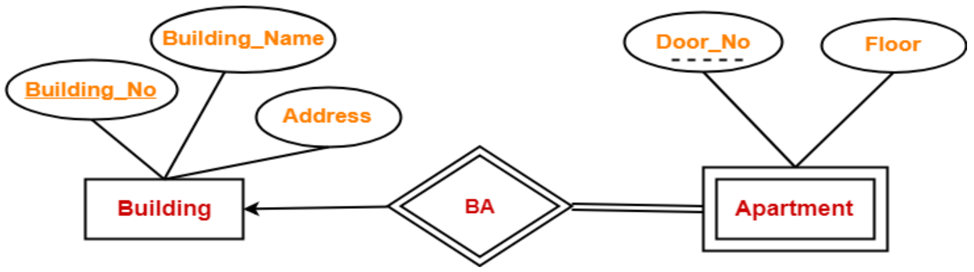




- In the above example the entity EMPLOYEE has an attribute E-id that can qualify as primary key therefore it is a strong entity.
- The entity DEPENDENT is not possessing any attribute that can qualify as a primary key therefore it is a weak entity
- As per the relational Database rules every entity should possess a primary key therefore primary key for DEPENDENT entity was build using the primary key of EMPLOYEE entity.

Example-

Consider the following ER diagram-



In this ER diagram,

- One strong entity set "**Building**" and one weak entity set "**Apartment**" are related to each other.
- Strong entity set "Building" has building number as its primary key.
- Door number is the discriminator of the weak entity set "Apartment".
- This is because door number alone cannot identify an apartment uniquely as there may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- It suggests that each apartment must be present in at least one building.
- Single line between Building and relationship set signifies partial participation
- It suggests that there might exist some buildings which has no apartment.

To uniquely identify any apartment,

First, building number is required to identify the particular building.

Secondly, door number of the apartment is required to uniquely identify the apartment.

Thus,

Primary key of Apartment:

= Primary key of Building + Its own discriminator

= Building number + Door number

Differences between Strong entity set and Weak entity set-

Strong entity set	**Weak entity set**
A single rectangle is used for the representation of a strong entity set.	A double rectangle is used for the representation of a weak entity set.
It contains sufficient attributes to form its primary key.	It does not contain sufficient attributes to form its primary key.
A diamond symbol is used for the representation of the relationship that exists between the two strong entity sets.	A double diamond symbol is used for the representation of the identifying relationship that exists between the strong and weak entity set.
A single line is used for the representation of the connection between the strong entity set and the relationship.	A double line is used for the representation of the connection between the weak entity set and the relationship set.
Total participation may or may not exist in the relationship.	Total participation always exists in the identifying relationship.

<u>Attribute:</u>

An attribute is a characteristic of data. A real-world data feature, modeled in the database, will be represented by an attribute. An attribute has to have a name, so you can refer to that feature, and the name has to be as relevant as possible for that feature.

For example: for a person the attributes can be: Name, gender, DateOfBirth. Informal terms used to define an attribute are: column in a table or field in a data file.

- Each table can have one or more attributes. Attributes are classified into different categories
    - Required and optional attributes
    - Primary key attribute
    - Composite and simple attribute
    - Single-valued or Multi-valued attributes
    - Derived attributes
    - Primary key(Identifier)

Required and Optional Attribute:

- **Required attribute**: An attribute that must have a value. These attributes does not allow NULL values
- **Optional attribute:** An attribute that may or may not have a value. These attributes allows NULL values

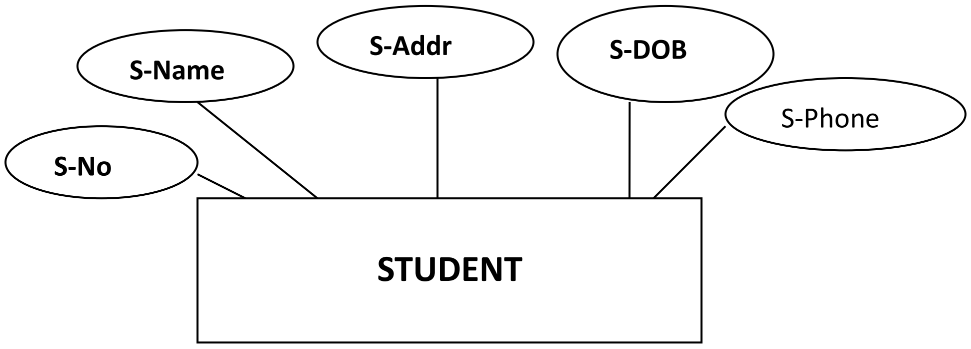


Figure 1

- In the above figure the attributes S-No, S-Name, S-Addr, S-DOB are required attributes and the attribute S-Phone is an optional attribute.

Primary Key Attribute (Identifier):

- One or more attributes that uniquely identify an entity instance
- Primary key attributes of an entity are underlined by solid line in ER diagrams.
- Each entity has only one primary key.
- It does not allow null values
- In the above example S-No is a primary key for STUDENT entity

Composite and Simple Attribute:

- **Composite attribute**: An attribute that can be further subdivided to yield additional attributes

- **Simple attributes**: An attribute that cannot be further subdivided

In the figure 1 S-Name attribute is a composite attribute so it is sub-divided into FName, MName, LName and the transformed entity is shown

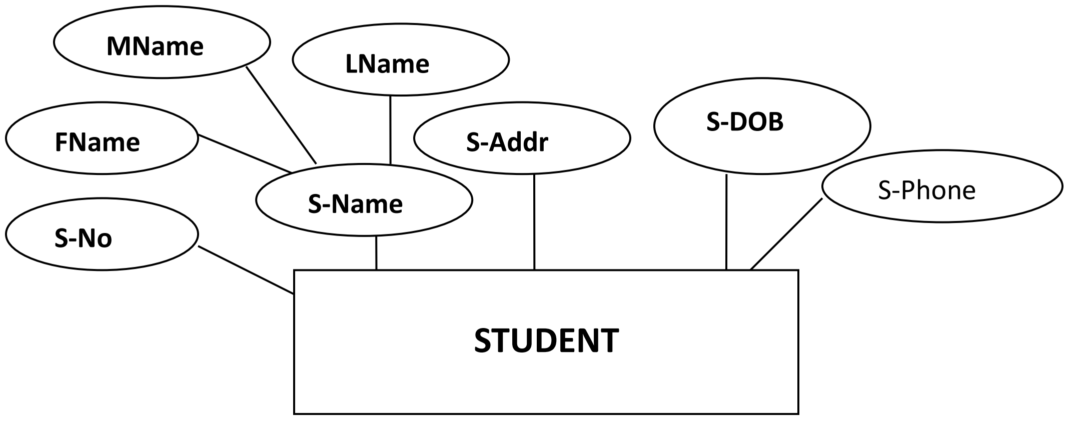
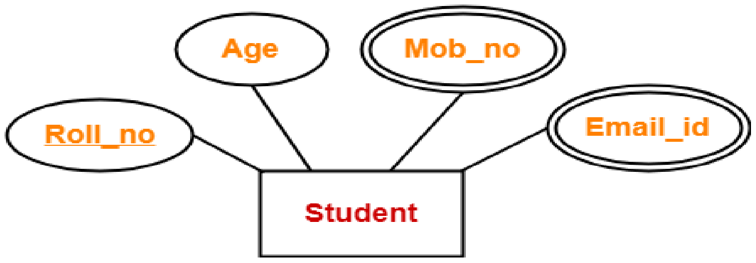


Figure 2

Single-valued and Multi-valued attributes:

- **Single-valued attribute** is an attribute that can take single value only.It need not be a simple attribute it can also be composite.
- **Multi-valued attributes** are attributes that can have many values**.** They are shown by a double line connecting the attribute to the entity.

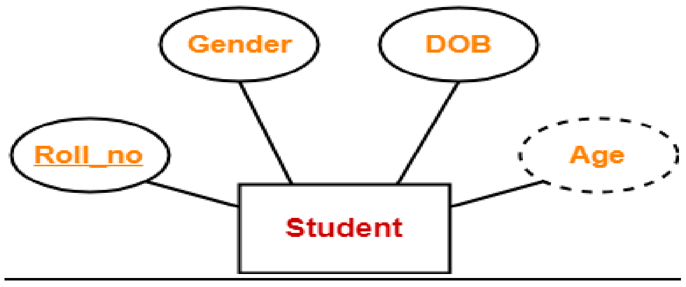
Example:



In the above example **mob_no and Email_id are** multi-valued attributes and the remaining attributes are single valued attributes

Derived attributes:

- An attribute whose value is derived from the other attributes value.
- They are referred as computed attributes
- Saves storage space but requires CPU cycles
- It can be represented by dashed oval symbol.



- In the above diagram **age** is derived attributed which is deriving from DOB. the value of the age is a derived from DOB therefore age is derived attribute.

Primary key of weak entity set

= Its own discriminator + Primary key of strong entity set

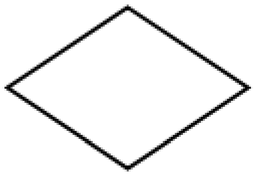
Relationships:

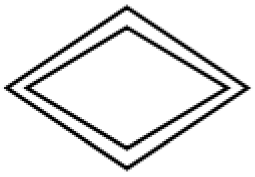
Relationship defines an association among several entities. A relationship set is a set of same type of relationships. When the basic data model components were introduced, three types of relationships among data were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1). The ER model uses the term connectivity to label the relationship types. The name of the relationship is usually an active or passive verb. A relationship set is a set of relationships of same type.

A relationship set may be of the following two types-

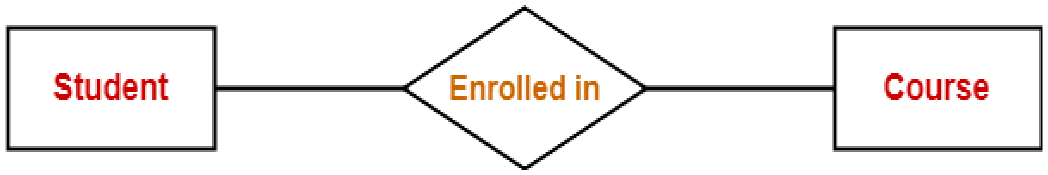
1.**Strong Relationship Set-** A strong relationship exists between two strong entity sets. It is represented using a diamond symbol.

2. Weak Relationship Set- A weak or identifying relationship exists between the strong and weak entity set. It is represented using a double diamond symbol.


Strong Relationship Set


Weak or Identifying Relationship Set

Example- 'Enrolled in' is a relationship that exists between entities **Student** and **Course**.

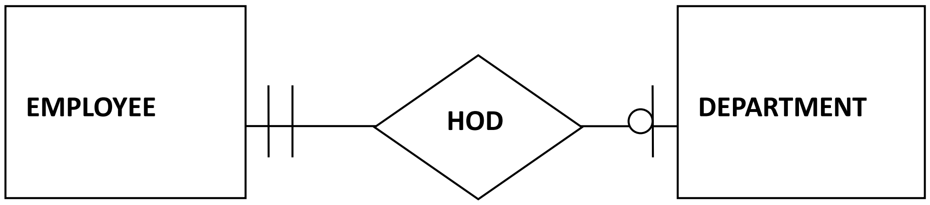


Connectivity and Cardinality of a Relationship:

The ER Model uses the term connectivity to label the type of relationship. There are three types of relationships based on cardinality

One-One Relationship (1:1):

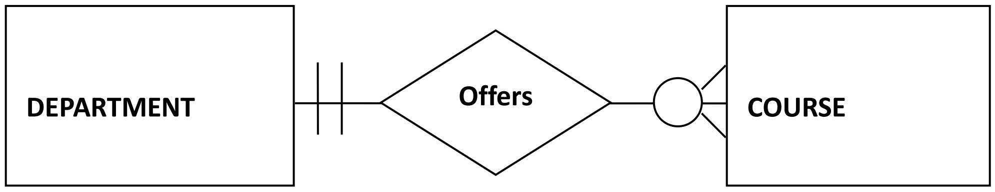
- Implies one instance of entity set can be related to only one instance of another entity set



- An employee can be head of only one DEPARTMENT
- A DEPARTMENT will have one and only one HOD

One-Many Relationship (1: M):

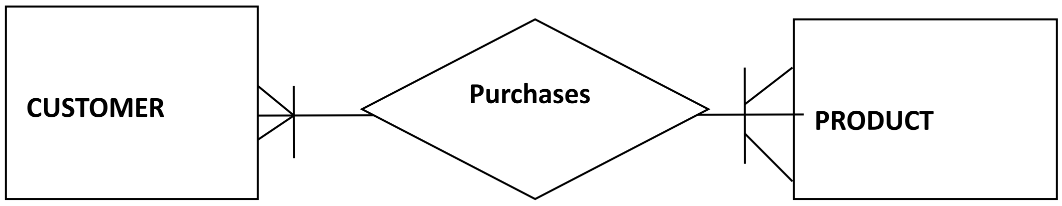
- Implies one instance of entity set can be related to many instances of another entity set



- The department offers zero or more courses
- Each course is offered by one and only one department.

Many- Many Relationship (M:N):

- Implies many instances of entity set can be related to many instances of another entity set.

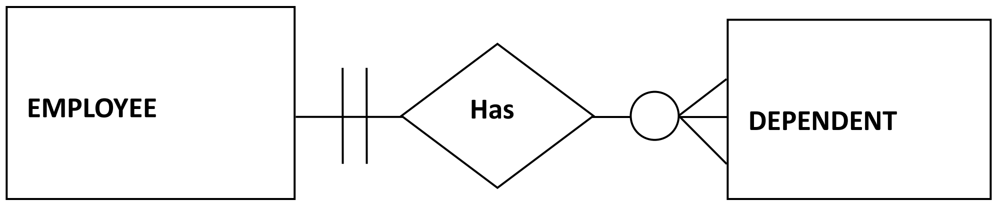


- A customer may purchase one or more products.
- A product can be purchased by more than one customer.

Existence Dependence:

- An entity is said to be existence dependent if it can exist in the database only when it is associated with another entity occurrence.
- In implementation terms, an entity is existence dependent if it has a mandatory foreign key.
- If an entity can exist apart from one or more related entities, then it is referred as existence-independent.

-



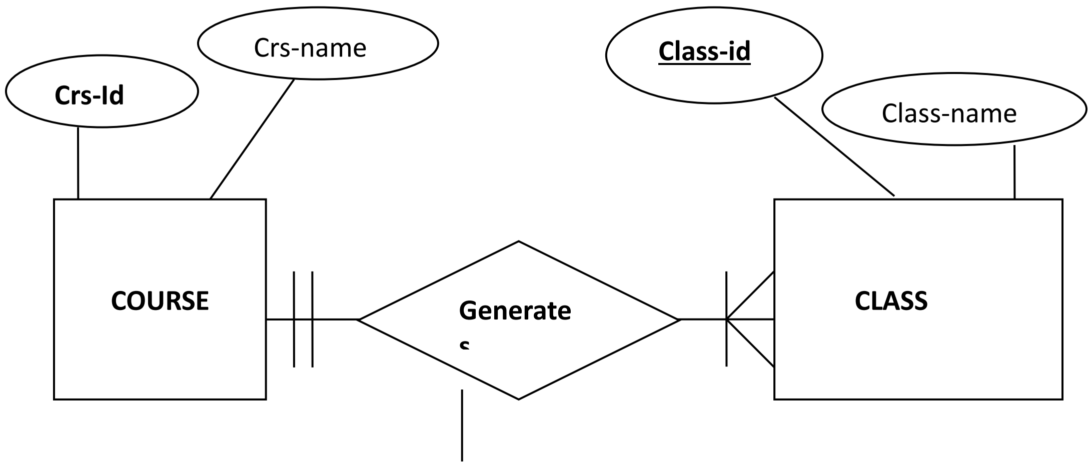
- In the above example the entity DEPENDENT is existence dependent on the entity EMPLOYEE.

Relationship Strength:

- Relationship strength is based on the primary key of the participating entities.

Weak(Non-identifying) Relationships

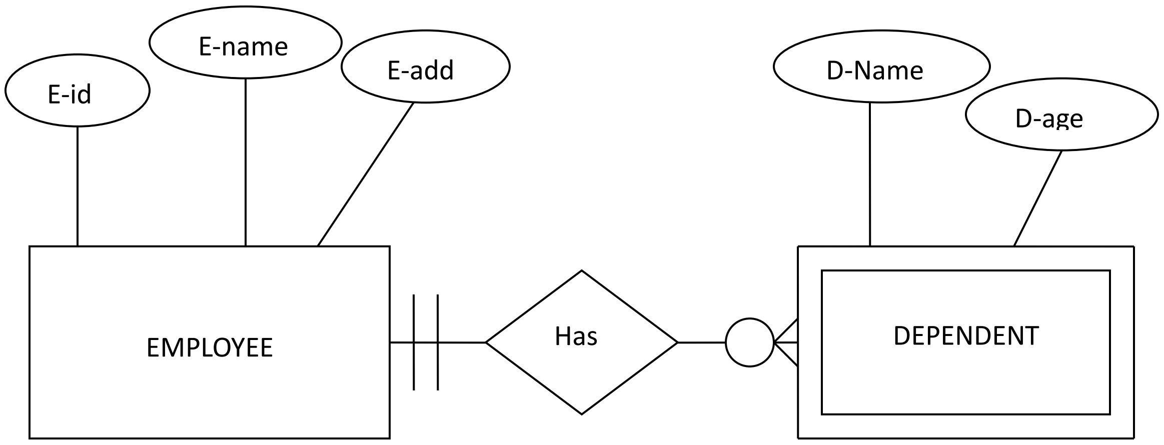
A weak relationship also known as a non-identifying relationship, exists if the participating entities have attributes that qualify as Primary Key.



In the above example both the entities are having attributes that qualify themselves as primary key, therefore the relationship GENERATES is weak relationship.

Strong (Identifying) Relationships:

A strong relationship also known as a identifying relationship, exists if one of the participating entity does not have any attributes that qualify themselves as Primary Key. So the primary key of such an entity is formed using the primary key of other participating entities.



In the above example the entity DEPENDENT does not have any attribute that can become primary key. So the attribute E-id of EMPLOYEE becomes the primary key for the DEPENDENT entity.

Relationship Participation in ER Diagram:

- Participation in an entity relationship is either optional or mandatory.
- **Optional participation** means that one entity occurrence does not require a corresponding entity occurrence in a particular relationship.
- **Mandatory participation** means that one entity occurrence requires a corresponding entity occurrence in a particular relationship.

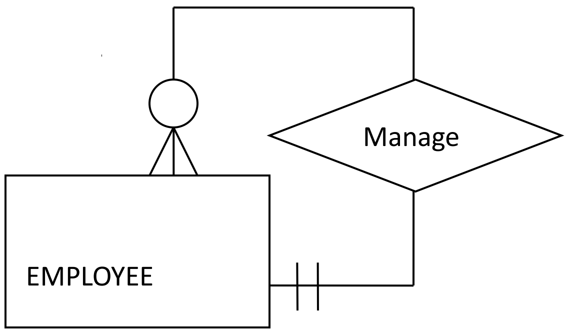
Relationship Degree:

- A relationship degree indicates the number of entities or participants associated with a relationship.
- There are three types of relationships based on the degree, they are
    - Unary relationship
    - Binary relationship
    - Ternary relationship

Unary Relationship:

- In unary relationship the entity has relationship with itself.

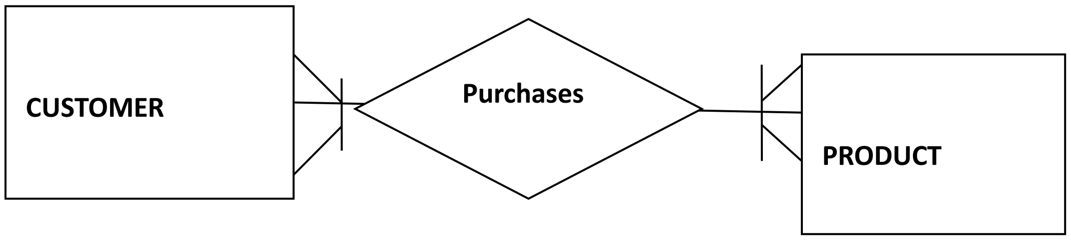
- One instance of the entity is related with another instance of the same entity.
- It is also called recursive relationship.



- Employee manages zero or more employees.
- Each employee is managed by only one manager

Binary Relationship:

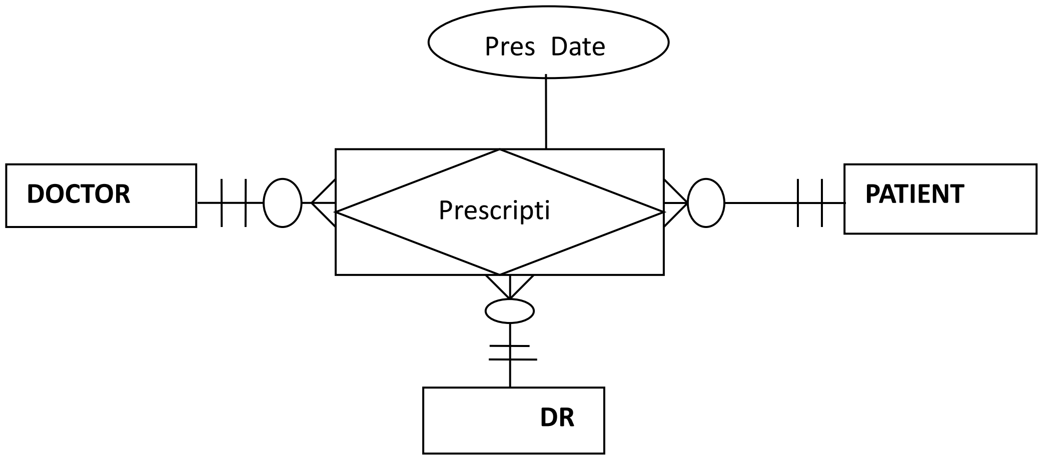
- A binary relationship exists when two entities are associated in a relationship.
- A binary relationship can be weak or strong based on the participating entities.



- Each customer can purchase one or more products.
- Each product can be purchased by more than one

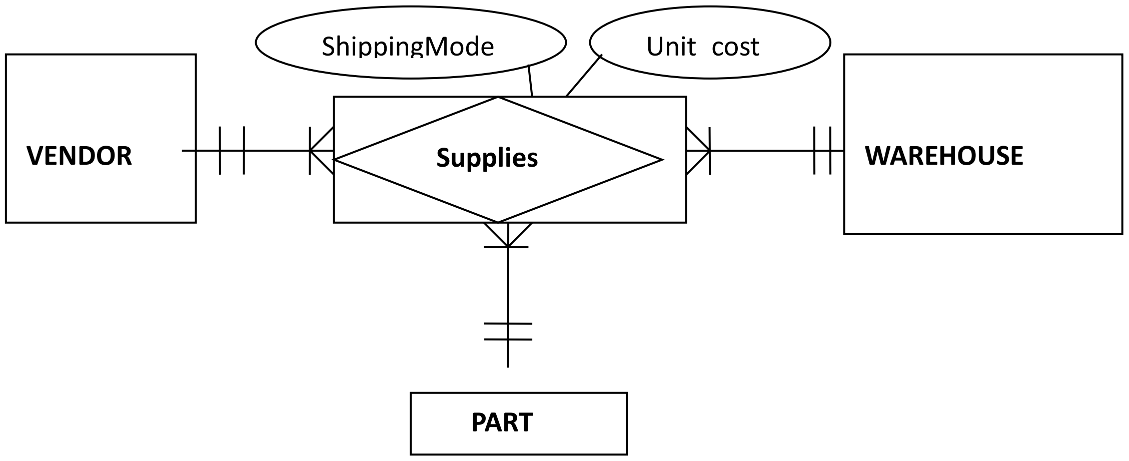
Ternary relationship: A simultaneous relationship that exists between instances of three entities is called ternary relationship.

Case - 1:



- A Doctor writes one or more prescriptions.
- A Patient may receive one or more prescriptions.
- A Drug may appear in one or more prescriptions.
- Prescription is an associative entity since many to many relationships exist between participating entities.

Case- 2:



- A Vendor may supply one or more Parts to a Warehouse.
- A Warehouse may receive one or more Parts from Vendor.
- A Part may be supplied by one or more Vendor to the Warehouse.
- Supplies is an associative entity since many to many relationships exist between participating entities.

Relational Database Design Concepts

Defining a Relation:

A relation is the core of the relational data. According to *introduction to database* systems a ***relation*** on domains **D1, D2, …, Dn** (not necessarily distinct) consists of a heading and a body.

- The ***heading*** consists of a fixed set of attributes **A1, A2, …, An**, such that each attribute **Ai** corresponds to exactly one of the underlying domains **Di (i=1, 2, …, n)**.
- The ***body*** consists of a time-varying set of tuples, where each tuple in turn consists of a set of attribute-value pairs **(Ai:vi) (i=1, 2, …, n)**, one such pair for each attribute **Ai** in the heading. For any given attribute-value pair **(Ai:vi)**, **vi** is a value from the unique domain **Di** that is associated with the attribute **Ai**.
- A ***relation degree*** is equivalent with the number of attributes of that relation. The relation from *Figure 2.2* has a degree of 6. A relation of degree one is called **unary**, a relation of degree two **binary**, a relation of degree three **ternary**, and so on. A relation of degree nis called **nary**.
- Relation ***cardinality*** is equivalent with the number of tuples of that relation. The relation from *Figure 2.2* has a cardinality equal to 5. The cardinality of a relation changes with time, whereas the degree does not change that often.

Keys:

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Example:

Employee ID	**FirstName**	**LastName**
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

In the above example, employee ID is a primary key because it uniquely identifies an employee record. In this table, no other employee can have the same employee ID.

Here, are reasons for using Keys in the DBMS system.

- Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
- Allows you to establish a relationship between and identify the relation between tables
- Help you to enforce identity and integrity in the relationship.

DBMS has following seven types of Keys each have their different functionality:

1. Super Key
2. Primary Key
3. Candidate Key
4. Foreign Key
5. Composite Key

1. Super Key-

- A super key is a set of attributes that can identify each tuple uniquely in the given relation.
- A super key is not restricted to have any specific number of attributes.
- Thus, a super key may consist of any number of attributes.

Example- Consider the following Student schema-

Student (rollno , name , gender , age , address , class , section)

Given below are the examples of super keys since each set can uniquely identify each student in the Student table-

- (rollno , name , gender , age , address , class , section)
- (class , section , rollno)
- (class , section , rollno , gender)
- (name , address)

NOTE- All the attributes in a super key are definitely sufficient to identify each tuple uniquely in the given relation but all of them may not be necessary.

2. Candidate Key-

A minimal super key is called as a candidate key.

OR

A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called as a candidate key.

Example-

Consider the following Student schema-

Student (rollno , name , gender , age , address , class , section)

Given below are the examples of candidate keys since each set consists of minimal attributes required to identify each student uniquely in the Student table-

- (class , section , rollno)
- (name , address)

NOTES-

All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.

- Removing any attribute from the candidate key fails in identifying each tuple uniquely.
- The value of candidate key must always be unique.
- The value of candidate key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which appears in some candidate key are called as **prime attributes**.

3. Primary Key-

A primary key is a candidate key that the database designer selects while designing the database.

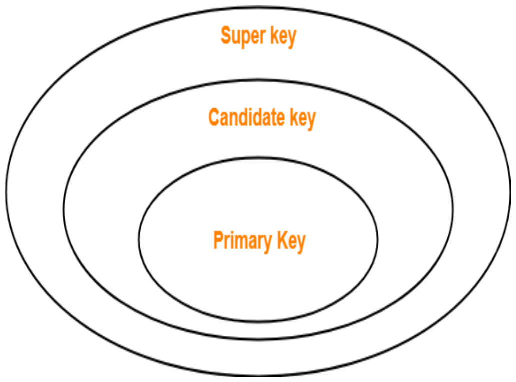
OR

Candidate key that the database designer implements is called as a primary key.

NOTES-

- The value of primary key can never be NULL.
- The value of primary key must always be unique.
- The values of primary key can never be changed i.e. no updating is possible.
- The value of primary key must be assigned when inserting a record.
- A relation is allowed to have only one primary key.

Remember-



4. Alternate Key-

Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.

OR

Unimplemented candidate keys are called as alternate keys.

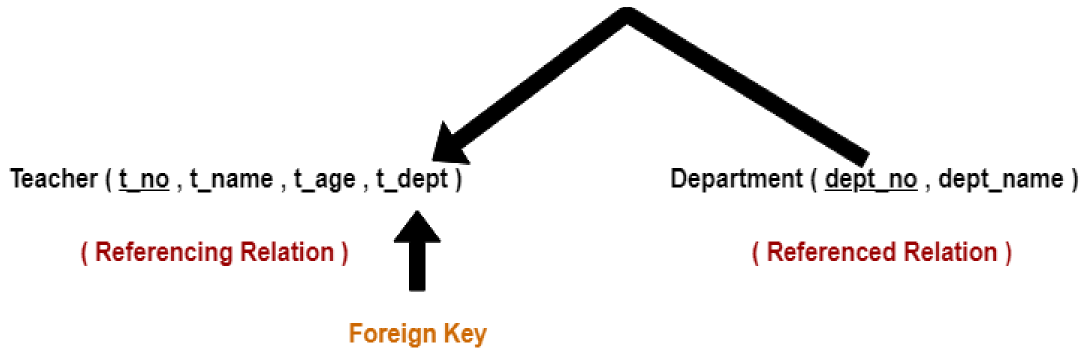
5. Foreign Key-

An attribute ‘X’ is called as a foreign key to some other attribute ‘Y’ when its values are dependent on the values of attribute ‘Y’.

- The attribute ‘X’ can assume only those values which are assumed by the attribute ‘Y’.
- Here, the relation in which attribute ‘Y’ is present is called as the **referenced relation**.
- The relation in which attribute ‘X’ is present is called as the **referencing relation**.
- The attribute ‘Y’ might be present in the same table or in some other table.

Example-

Consider the following two schemas-


Teacher (t_no , t_name , t_age , t_dept)
(Referencing Relation)
Foreign Key
Department (dept_no , dept_name)
(Referenced Relation)

Here, t_dept can take only those values which are present in dept_no in Department table since only those departments actually exist.

NOTES-

Foreign key references the primary key of the table.

- Foreign key can take only those values which are present in the primary key of the referenced relation.
- Foreign key may have a name other than that of a primary key.
- Foreign key can take the NULL value.
- There is no restriction on a foreign key to be unique.
- In fact, foreign key is not unique most of the time.
- Referenced relation may also be called as the master table or primary table.
- Referencing relation may also be called as the foreign table.

6. Composite Key-

A primary key comprising of multiple attributes and not just a single attribute is called as a composite key.

If a table do have a single columns that qualifies for a Candidate key, then you have to select 2 or more columns to make a row unique. Like if there is no EmployeeID or SSN columns, then you can make FullName + DateOfBirth as Composite primary Key. But still there can be a narrow chance of duplicate row.All super keys can't be candidate keys but its reverse is true. In a relation, number of super keys are more than number of candidate keys.

Example:

Student{ID, F_name, M_name, L_name, Age}

Here only ID can be primary key because the name, age and address can be same but ID can't be same.

Constraints in DBMS-

- Relational constraints are the restrictions imposed on the database contents and operations.
- They ensure the correctness of data in the database.

Entity Integrity and Referential Integrity Concepts:

In a relational data model, data integrity can be achieved using integrity rules or constraints. Those rules are general, specified at the database schema level, and they must be respected by each schema instance. If we want to have a correct relational database definition, we have to declare such constraints.

If a user attempts to execute an operation that would violate the constraint then the system must then either reject the operation or in more complicated situations, perform some compensating action on some other part of the database. This would ensure that the overall result is still in a correct state.

Entity Integrity Constraint-

Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation. This is because the presence of null value in the primary key violates the uniqueness property.

The justification for the entity integrity constraint is:

- Database relations correspond to entities from the real-world and by definition entities in the real-world are distinguishable, they have a unique identification of some kind .
- **Primary keys** perform the unique identification function in the relational model
- Therefore, a null **primary key** value would be a contradiction in terms because it would be saying that there is some entity that has no identity that does not exist.

Example-

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

Referential Integrity Constraint-

This constraint is enforced when a foreign key references the primary key of a relation. It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null. The ***referential integrity constraint*** says that if a relation **R2** includes a foreign key **FK** matching the **primary key PK** of other relation **R1**, then every value of **FK** in **R2** must either be equal to the value of **PK** in some tuple of **R1** or be wholly null (each attribute value participating in that **FK** value must be null). **R1** and **R2** are not necessarily distinct. The justification for referential integrity constraint is:

- If some tuple **t2** from relation **R2** references some tuple **t1** from relation **R1**, then tuple **t1** must exist, otherwise it does not make sense.
- Therefore, a given foreign key value must have a matching **primary key** value somewhere in the referenced relation if that foreign key value is different from null
- Sometimes, for practical reasons, it is necessary to permit the foreign key to accept null values

<u>Important Results-</u>

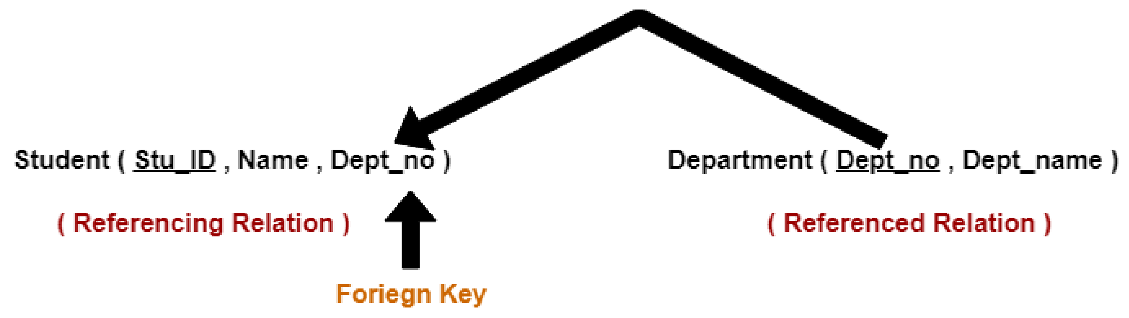
The following two important results emerges out due to referential integrity constraint-

- We cannot insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.
- We cannot delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

<u>Example-</u>

Consider the following two relations- 'Student' and 'Department'.

Here, relation 'Student' references the relation 'Department'.


Student (Stu_ID , Name , Dept_no)
(Referencing Relation)
Foriegn Key
Department (Dept_no , Dept_name)
(Referenced Relation)

Student

<u>STU_ID</u>	Name	Dept_no

S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	**D14**

Department

<u>**Dept_no**</u>	**Dept_name**
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation 'Student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department', no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

Functional Dependency:

Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs. The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address. Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

There are three Inference Rules (Properties)of functional dependency:

1. ***Reflexivity*****:** If **B** is a subset of attributes in set **A**, then **A** → **B**. (by **trivial FD**)
2. ***Augmentation*****:** If **A** → **B** and **C** is another attribute, then **AC** → **BC**

   Applying reflexivity to this rule, we can also say that, **AC** → **B**.
3. ***Transitivity*****:** If **A** → **B** and **B** → **C**, then **A** → **C**.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs

<u>**Normalization:**</u>

<u>**Normalization:-**</u> Normalization is a process for evaluating and correcting table's structures to minimize data redundancy thereby avoiding the occurring of data anomalies.

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and thereby avoiding data anomalies like Insertion, Update and Deletion. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

<u>**Need for Normalization**</u>

- To minimize data redundancy
- To avoid data anomalies resulting during insert, update or delete operations.

<u>**The Normalization Process**</u>

- The objective of normalization is to ensure that each table conforms to the concept of well-defined relations that satisfy the following characteristics
    - Each table represents a single subject
    - No data item will be unnecessarily stored in more than one table.
    - All nonprime attributes in a table are dependent on the primary key.
    - Each table should not exhibit insert, update and delete anomalies.
- Normalization process takes us through the steps that lead us through normal form to accomplish the above objective.

Anomalies in DBMS:

The problems arise from relations that are generated directly from user views are called anomalies. There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. The problems that we face when a table is not normalized.

- **Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.
- **Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

- **Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.
  To overcome these anomalies, we need to normalize the data.

Normal Forms:

Normalization works through a series of stages called normal form. Following are the different normal forms:-

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce Codd normal form(BCNF)

1. First Normal Form (1NF)

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row.

- It is performed by the three steps

1. Eliminate the repeating groups.
2. Identifying the primary key
3. Identify all the functional dependencies

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	**emp_name**	**emp_address**	**emp_mobile**
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	**emp_name**	**emp_address**	**emp_mobile**
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

2. Second normal form (2NF)

A relation or a table is in second normal form if it is satisfies the following conditions

- In should be in first normal from(1NF)
- Every non-key attribute should fully functionally dependent on the primary key.

If any relation or table is not satisfying the above conditions it is said not to be 2NF and the following steps are to convert the relation into the second normal form:-

a. Step:-1 write each key component on a separate line.
b. Step:-2 Assign corresponding dependent attributes
c. Step3: create a separate table for each determinant and its dependencies.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	**subject**	**teacher_age**
111	Maths	38

111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}

Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:
teacher_details table:

teacher_id	**teacher_age**
111	38
222	38
333	40

teacher_subject table:

teacher_id	**subject**
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

3. Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

Or

- A relation is in third normal form (3NF) if it satisfies the following conditions.
    1. The relation should be in 2 NF
    2. Transitive dependency between the attributes should not exist.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

4. Boyce Codd normal form (BCNF)

- It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

- A table is in Boyce-Codd Normal Form (BCNF) when every determinant in the table is a candidate key.
- When a table contains only one candidate key then 3NF and BCNF are equivalent.
- When a table contains more than one candidate key then the table need to be corrected so that it contains only one candidate key.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp

Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Normal forms in a nutshell

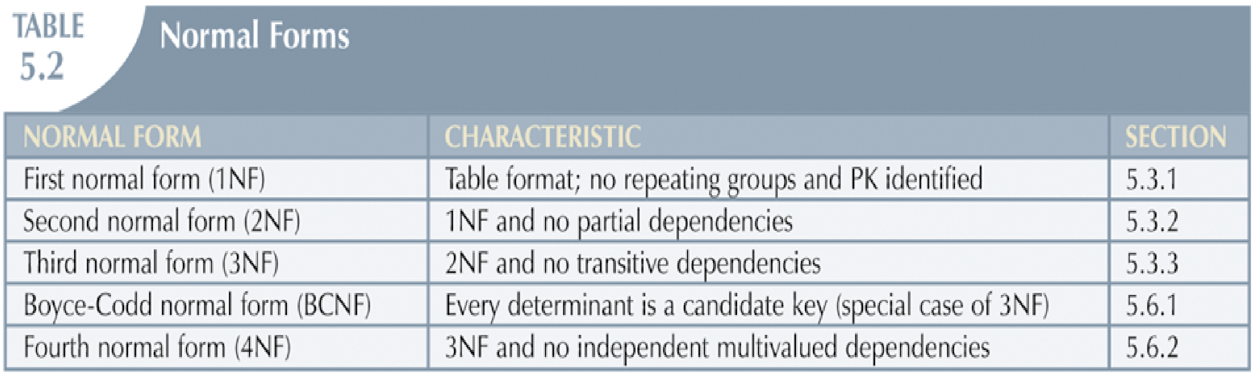


TABLE 5.2 Normal Forms

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format; no repeating groups and PK identified	5.3.1
Second normal form (2NF)	1NF and no partial dependencies	5.3.2
Third normal form (3NF)	2NF and no transitive dependencies	5.3.3
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	5.6.1
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	5.6.2

Codd's Rules:

- Dr E.F Codd was a Computer Scientist who invented **Relational model** for Database management. Based on relational model, **Relation database** was created.
- Codd proposed 13 rules popularly known as **Codd's 12 rules** to test DBMS's concept against his relational model.
- Codd's rule actually define what quality a DBMS requires in order to become a Relational Database Management System (RDBMS).
- Till now, there is hardly any commercial product that follows all the 13 Codd's rules. Even **Oracle** follows only eight and half out (8.5) of 13.

The Codd's 12 rules are as follows:

Rule 0:

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule

All information (including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2: Guaranteed Access

Each unique piece of data (atomic value) should be accessible by: **Table Name + primary key (Row) + Attribute (column)**.

NOTE: Ability to directly access via POINTER is a violation of this rule.

Rule 3: Systematic treatment of NULL

Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Primary key must not be null. Expression on **NULL** must give null.

Rule 4: Active Online Catalog

Database dictionary (catalog) must have description of **Database**. Catalog to be governed by same rule as rest of the database. The same query language to be used on catalog as on application database.

Rule 5: Powerful language

One well defined language must be there to provide all manners of access to data. Example: **SQL**. If a file supporting table can be accessed by any manner except SQL interface, then it's a violation to this rule.

Rule 6: View-Updation rule

All view that are theoretically updatable should be updatable by the system.

Rule 7: Relational Level Operation

There must be Insert, Delete, and Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8: Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table were renamed or moved from one disk to another, it should not affect the application.

Rule 9: Logical Data Independence

If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

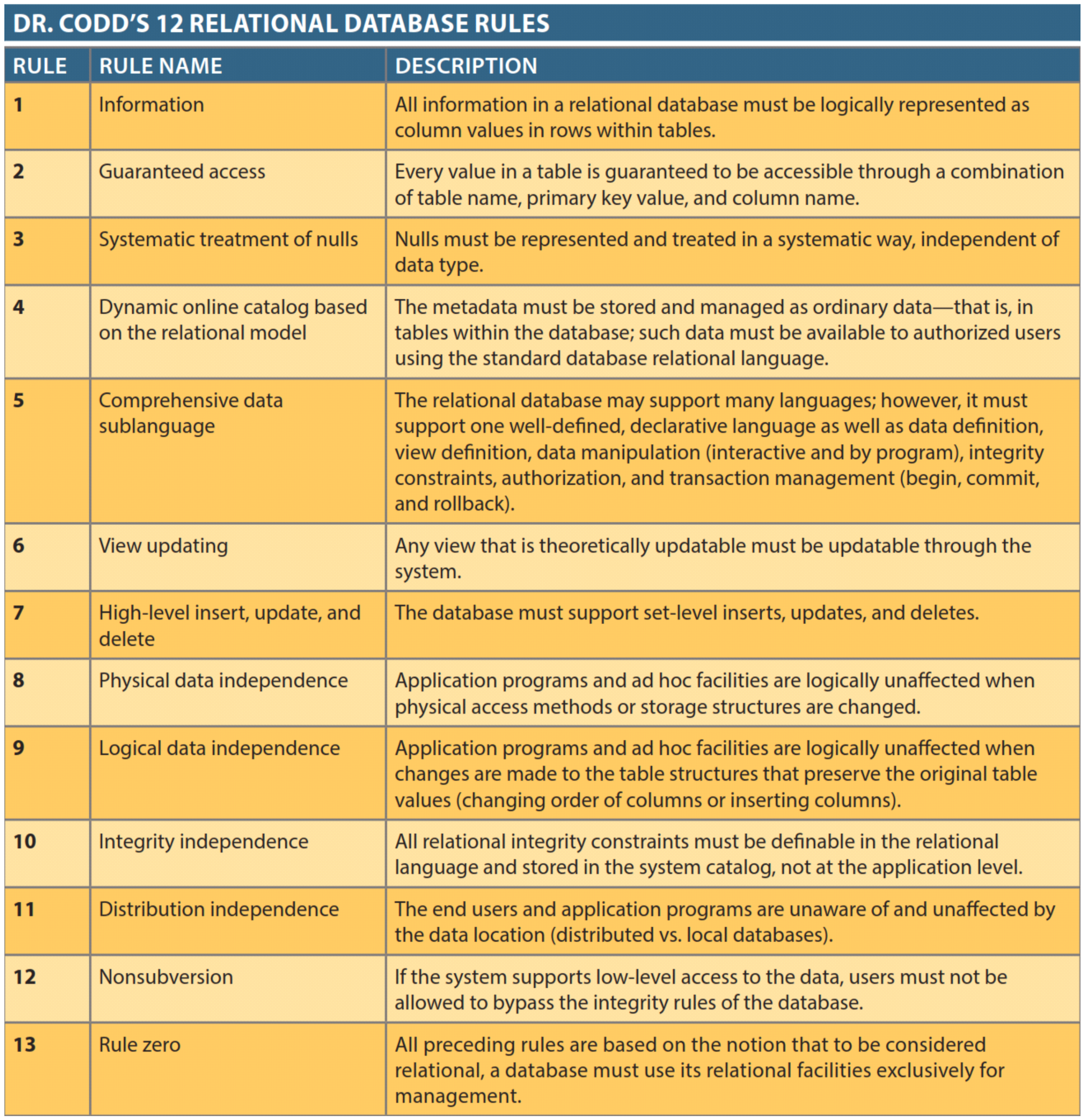
Rule 10: Integrity Independence

The database should be able to conforce its own integrity rather than using other programs. Key and Check constraints, trigger etc. should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Rule 11: Distribution Independence

A database should work properly regardless of its distribution across a network. This lays foundation of distributed database.

Rule 12: Non-subversion rule



DR. CODD'S 12 RELATIONAL DATABASE RULES

RULE	RULE NAME	DESCRIPTION
1	Information	All information in a relational database must be logically represented as column values in rows within tables.
2	Guaranteed access	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	Systematic treatment of nulls	Nulls must be represented and treated in a systematic way, independent of data type.
4	Dynamic online catalog based on the relational model	The metadata must be stored and managed as ordinary data—that is, in tables within the database; such data must be available to authorized users using the standard database relational language.
5	Comprehensive data sublanguage	The relational database may support many languages; however, it must support one well-defined, declarative language as well as data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
6	View updating	Any view that is theoretically updatable must be updatable through the system.
7	High-level insert, update, and delete	The database must support set-level inserts, updates, and deletes.
8	Physical data independence	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	Logical data independence	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).
10	Integrity independence	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
11	Distribution independence	The end users and application programs are unaware of and unaffected by the data location (distributed vs. local databases).
12	Nonsubversion	If the system supports low-level access to the data, users must not be allowed to bypass the integrity rules of the database.
13	Rule zero	All preceding rules are based on the notion that to be considered relational, a database must use its relational facilities exclusively for management.

De-Normalization:

The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of de-normalization so processing speed can be increased.

De-normalization is the process of taking a normalized database and modifying table structures to allow controlled redundancy for increased database performance.

1. It is a strategy which involves adding redundant data to a normalized database to reduce certain types of problems with database queries that combine data from various tables into a single table.
2. In many cases, denormalization involves creating separate tables or structures so that queries on one piece of information will not affect any other information tied to it.
3. The basic criteria for denormalization would be-
   - It should reduce the frequency of joins between the tables, and hence making the query faster.
   - Most of the cases, when we have joins on tables, full table scan is performed to fetch the data. Hence if the tables are huge, we can think of denormalization.
   - The column should not be updated more frequently. Also the column should very small to get rejoined with the table. Huge columns are again overhead to the table and cost of performance.
   - The developer should have very good knowledge of data, when he denormalizes it. He should know very clearly about all the factors, frequency of joins / access, updates, column and table size etc.

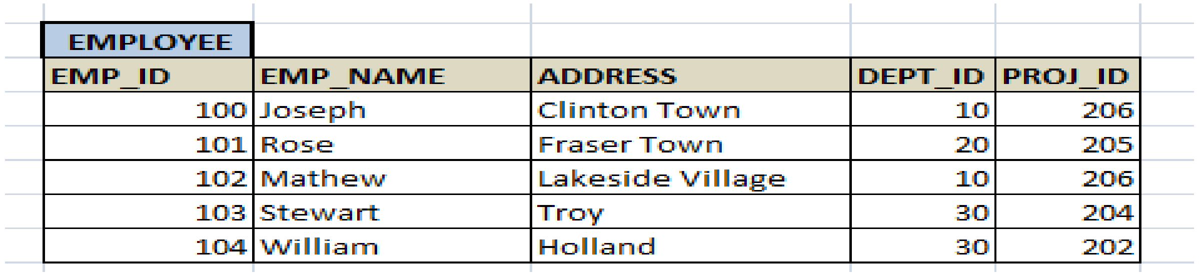
Methods of De-normalization:

1. Adding redundant columns- we can add redundant columns to eliminate frequent joins.
2. Adding Derived columns- it can help to eliminate joins and reduce the time needed to produce aggregate values.
3. Combining tables- Collapsing the two tables into one can improve performance by eliminating the join.
4. Repeating groups- These can be stored as nested table within the original table.
5. Creating extract tables- It allow users to access extract table directly.

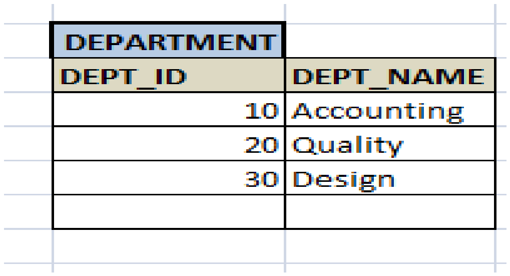
6. Partitioning relations- Instead of combining relations together, decompose them into a number of smaller and more manageable partitions.

Ex: Adding columns - In this method, only the redundant column which is frequently used in the joins is added to the main table. The other table is retained as it is.
For example, consider EMPLOYEE and DEPT tables. Suppose we have to generate a report where we have to show employee details and his department name. Here we need to have join EMPLOYEE with DEPT to get department name.

```
SELECT e.EMP_ID, e.EMP_NAME, e.ADDRESS, d.DEPT_NAME
FROM EMPLOYEE e, DEPT d
WHERE e.DEPT_ID = d.DEPT_ID;
```

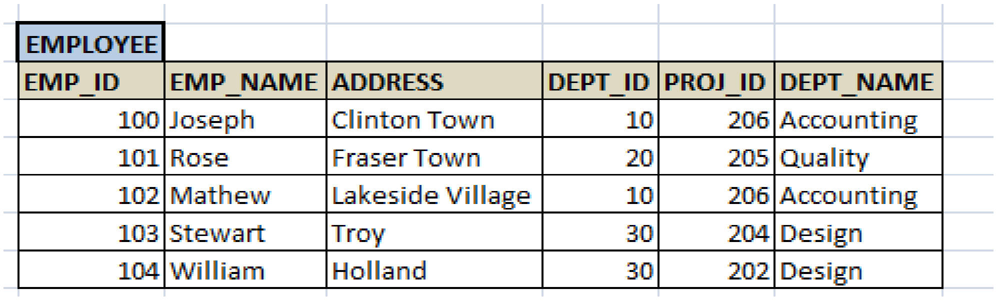


EMPLOYEE				
EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID
100	Joseph	Clinton Town	10	206
101	Rose	Fraser Town	20	205
102	Mathew	Lakeside Village	10	206
103	Stewart	Troy	30	204
104	William	Holland	30	202

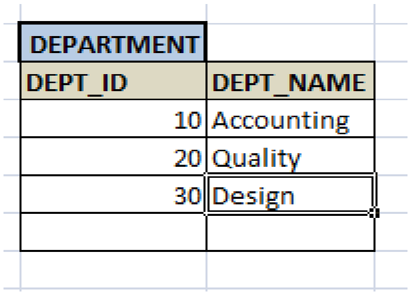


DEPARTMENT	
DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

But joining the huge EMPLOYEE and DEPT table will affect the performance of the query. But we cannot merge DEPT with EMPLOYEE. At the same time, we need to have a separate DEPT table with many other details, apart from its ID and Name. In this case, what we can do is add the redundant column DEPT_NAME to EMPLOYEE, so that it avoids join with DEPT and thus increasing the performance.



EMPLOYEE					
EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID	DEPT_NAME
100	Joseph	Clinton Town	10	206	Accounting
101	Rose	Fraser Town	20	205	Quality
102	Mathew	Lakeside Village	10	206	Accounting
103	Stewart	Troy	30	204	Design
104	William	Holland	30	202	Design



DEPARTMENT	
DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

```
SELECT e.EMP_ID, e.EMP_NAME, e.ADDRESS, e.DEPT_NAME
FROM EMPLOYEE e;
```

Now no need to join with DEPT to get the department name to get details. But it creates a redundancy of data on DEPT_NAME.

Advantages of Denormalization:

- Obviously, the biggest advantage of the denormalization process is increased performance.
- It makes retrieval of data easier to express and perform.
- Minimizing the need for joins.
- Reducing the no.of relations.
- Sometimes it makes the database easier to understand.

------<<<<<@@@@@>>>>>-------

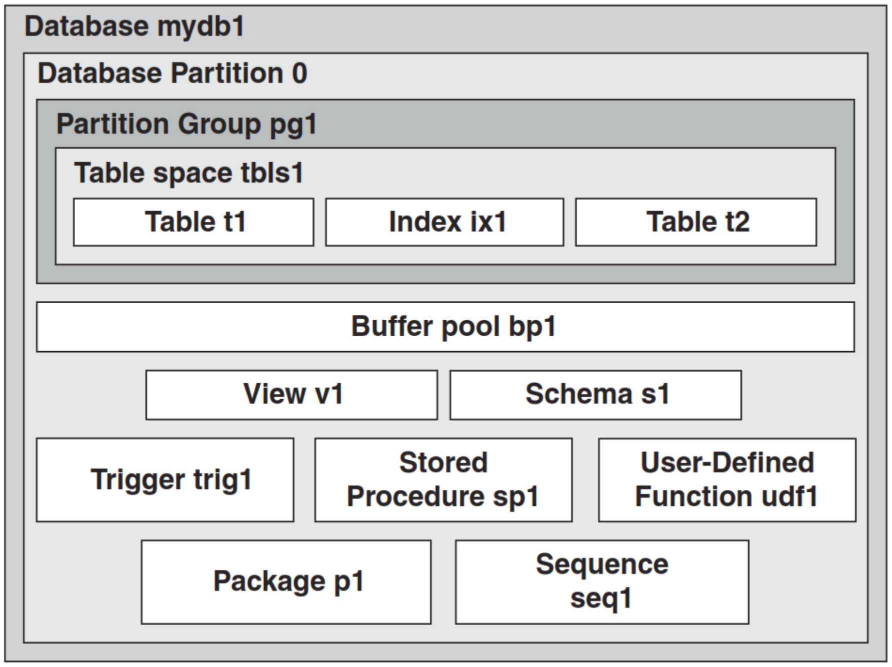
UNIT - III
Database Objects & DDL, DML & DCL Commands

Database Objects: Database, Tables, Table Spaces, Schema, Views, Indexes, Sequences.
DB2 Data Types, Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL).

Database Objects

Database Objects Overview:

A database is a collection of database objects. We can create a database on one or more database partitions. A database partition, as its name implies, is part of a database. The below Figure illustrates database objects in a database created in a single-partition environment (database partition 0).

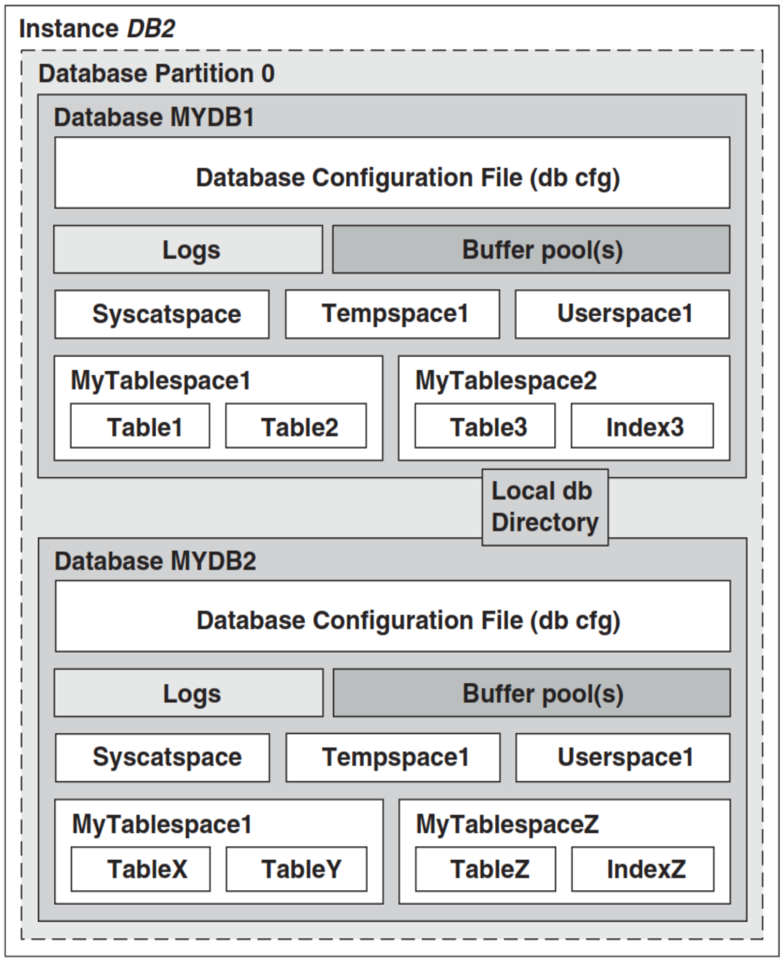


An overview of the DB2 database objects

1. Database:

A database is a collection of information organized into interrelated objects such as table spaces, partition groups, and tables. Each database is an independent unit containing its own system information, temporary space, transaction logs, and configuration files, as illustrated in Figure.

Figure shows two databases, **MYDB**1 and **MYDB2**, inside the **Instance DB2** in a single-partition environment (Database Partition 0). The box showing Database Partition 0 is included for completeness; in a single-partition environment you can ignore this box. Since databases are independent units, object names from different databases can be the same. For example, the name of the table space **MyTablespace1** is used in both databases in the figure.



A database and its objects

Figure also shows the three table spaces that DB2 creates by default when you create a database: **SYSCATSPACE, TEMPSPACE1,** and **USERSPACE1**.

To create a database, use the **CREATE DATABASE** command. To perform operations against database objects, we first need to connect to the database using the **CONNECT** statement.

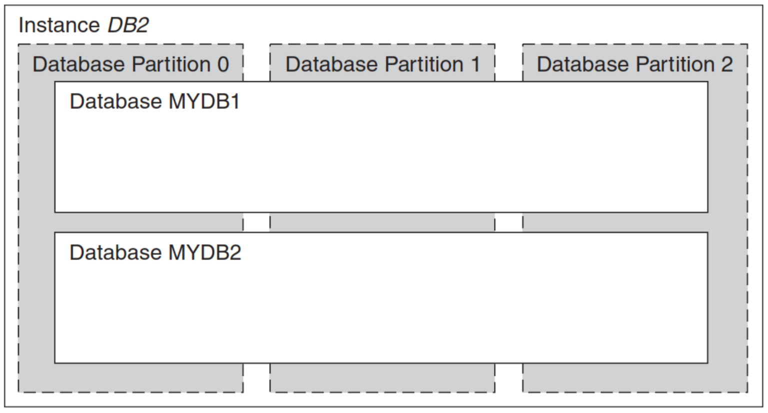
The following are commands to maintain databases in DB2

- **Find all databases for a certain instance: db2 list database directory**
- **Check for active databases: db2 list active databases**
- **Create database: db2 create db dbname**
- **Drop database: db2 drop db dbname**
- **Start/Activate database: db2 activate db dbname**
- **Deactivate database: db2 deactivate db**
- **Find table structure: db2 describe table**

Database Partitions:

We can create a single-partition or a multi-partition database, depending on your needs. In a multi-partition environment, a database partition (or simply partition) is an independent part of a database containing its own data, indexes, configuration files, and transaction logs. Database functions are also distributed between all the database partitions. This provides for unlimited scalability. Multiple database partitions can reside on one physical server; these are sometimes referred to as logical partitions sharing the resources of the machine.

Figure shows two databases, **MYDB1** and **MYDB2**, in a multi-partition environment with three database partitions.

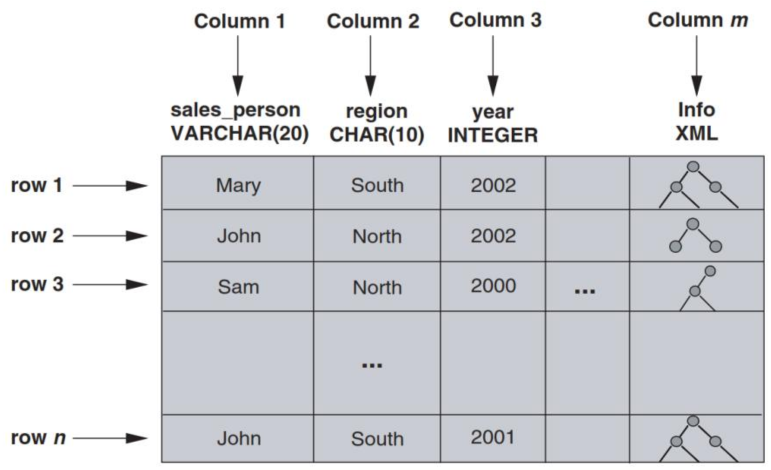


7.3 Database partitions

Partition Groups: A database partition group is a set of one or more database partitions for the same database. By grouping database partitions, we can perform database operations at the partition group level rather than individually on each partition. This allows for database administration flexibility.

2. Tables:

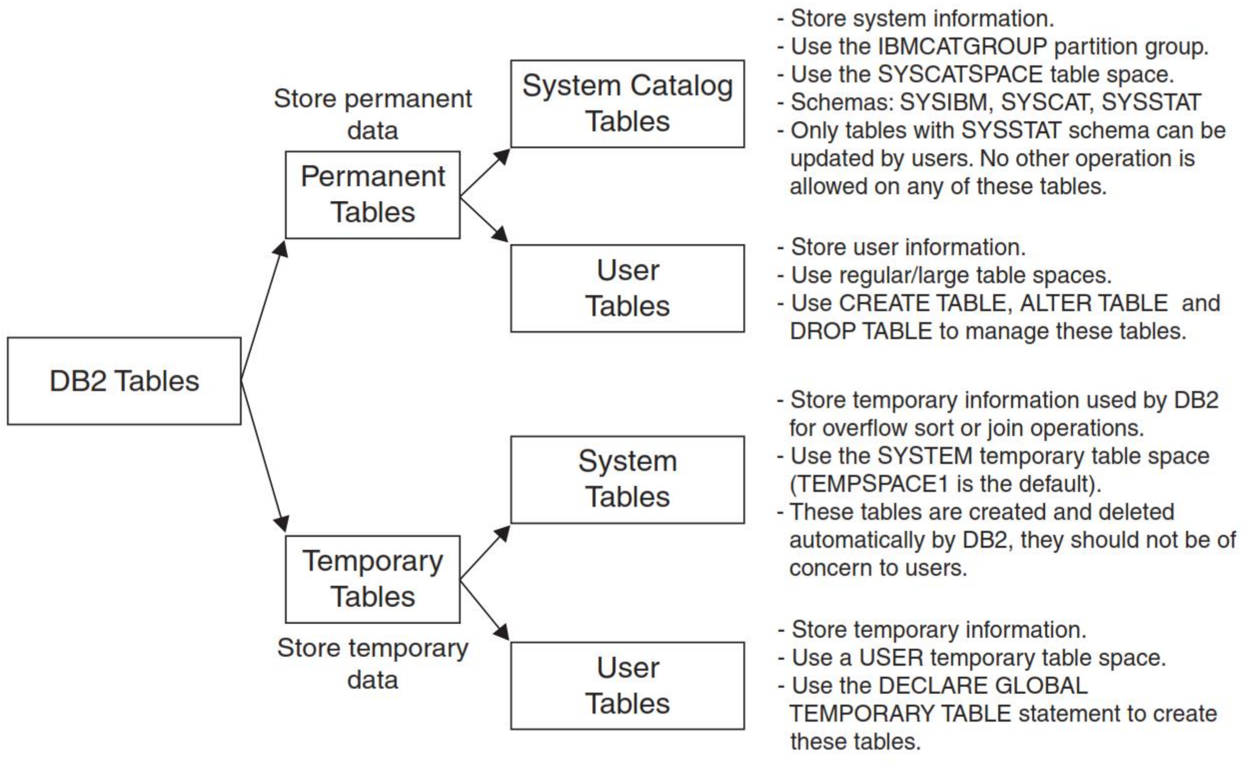
A table is an unordered set of records, consisting of rows and columns. Each column has a defined data type, and each row represents an entry in the table. Figure shows an example of a table with n rows and m columns. The sales_person column with a VARCHAR data type is the first column in the table, followed by the region column with a CHAR data type, and the year column with an INTEGER data type. The info column is the mth column in the table and has an XML data type.



ıre 7.9 An example of a table

Table Classification:

Tables in DB2 can be classified as illustrated in following figure 7.10.


DB2 Tables
Store permanent data
Permanent Tables
Temporary Tables
Store temporary data
System Catalog Tables
User Tables
System Tables
User Tables
- Store system information.
- Use the IBMCATGROUP partition group.
- Use the SYSCATSPACE table space.
- Schemas: SYSIBM, SYSCAT, SYSSTAT
- Only tables with SYSSTAT schema can be updated by users. No other operation is allowed on any of these tables.
- Store user information.
- Use regular/large table spaces.
- Use CREATE TABLE, ALTER TABLE and DROP TABLE to manage these tables.
- Store temporary information used by DB2 for overflow sort or join operations.
- Use the SYSTEM temporary table space (TEMPSPACE1 is the default).
- These tables are created and deleted automatically by DB2, they should not be of concern to users.
- Store temporary information.
- Use a USER temporary table space.
- Use the DECLARE GLOBAL TEMPORARY TABLE statement to create these tables.

igure 7.10 Classification of tables in DB2

a) System Catalog Tables:

DB2 automatically creates system catalog tables when a database is created. They always reside in the SYSCATSPACE table space. System catalog tables contain information about all the objects in the database.

For example, when you create a table space, its information will be loaded into one or more system catalog tables. When this table space is referenced during a later operation, DB2 checks the corresponding system catalog tables to see whether the table space exists and whether the operation is allowed. Without the system catalog tables, DB2 will not be able to function.

Some of the information contained in system catalog tables includes the following:

- Definitions of all database objects
- Column data types of the tables and views
- Defined constraints
- Object privileges
- Object dependencies

System catalog tables or views use the SYSIBM, SYSCAT, or SYSSTAT schemas.

- The SYSIBM schema is used for the base system catalog tables.
- The SYSCAT schema is used for views defined on the system catalog tables. DB2 users should normally query the SYSCAT views rather than the SYSIBM tables for information.
- The SYSSTAT schema is used for views containing information about database statistics and is also based on the system catalog tables.

b) User Tables:

User tables are used to store a user's data. A user can create, alter, drop, and manipulate user tables.

To create a user table, use the CREATE TABLE statement. You can specify the following:

- The name of the table
- The columns of the table and their data types
- The table spaces where you want the table, index, and long objects to be stored within the database
- The constraints you want DB2 to build and maintain on the table, such as referential constraints and unique constraints

The following example illustrates the creation of the table myemployees with four columns.

```
CREATE TABLE myemployees (
        empID    INT NOT NULL PRIMARY KEY,
        empname  VARCHAR(30)   NOT NULL,
        info XML,
        history  CLOB)
```

Once you have created a table, you cannot change the column names or data types; however, you are allowed to increase the length of VARCHAR columns or add new columns to the end of the table. You can do this with the **ALTER TABLE** statement. For example, to add the column address to the table myemployees, use this statement:

ALTER TABLE myemployees ADD COLUMN address CHAR(45)

We cannot remove a column from a table using the ALTER TABLE statement. If you want to remove a column from a table, you have two choices:

- Use a view to hide the column you want removed.
- Drop the table and recreate it.

To drop a table and all its contents, use the DROP TABLE statement, for example:

DROP TABLE myemployees

c) Partitioned Tables

With partitioned tables you can now create a table that can span multiple table spaces. In addition, queries can be directed automatically only to the partitions where the data resides. For example, if you partition a table based on the month, and a user runs a query that is calculating the total sales for March, the query need only access the data for March, not the data for any other month.

The table defined as:

```
CREATE TABLE sales(sdate DATE, customer INT)
     PARTITION BY RANGE(sdate)
        (STARTING '1/1/2006'  ENDING '3/31/2006',
         STARTING '4/1/2006'  ENDING '6/30/2006',
         STARTING '7/1/2006'  ENDING '9/30/2006',
         STARTING '10/1/2006' ENDING '12/31/2006')
```

would look like the following:

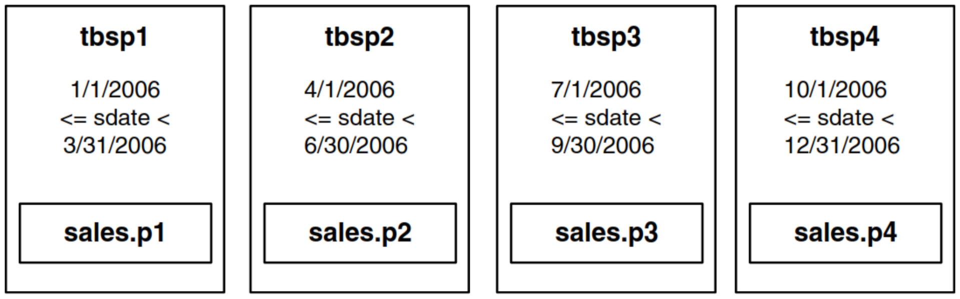

tbsp1
1/1/2006 <= sdate < 3/31/2006
sales.p1
tbsp2
4/1/2006 <= sdate < 6/30/2006
sales.p2
tbsp3
7/1/2006 <= sdate < 9/30/2006
sales.p3
tbsp4
10/1/2006 <= sdate < 12/31/2006
sales.p4

Figure 7.16 Table partitioned by date every three months

d) Default Values:

In the CREATE TABLE statement, you can use the DEFAULT clause for a given column to provide a default value for the column. This means that when you use an INSERT statement to insert a row that does not provide a value for the column, the default value specified in the DEFAULT clause will be used.

For example, let's say you create the table company with this statement:

CREATE TABLE company (

```
companyID       INTEGER,
companyName     VARCHAR(30),
city  VARCHAR(20) DEFAULT 'TORONTO'
)
```

Inserting a record with either of the following two statements provides the same result.

```
(1) INSERT INTO company  (companyID, companyName,  city)
                 VALUES (   111    ,  'cityOne' , DEFAULT)
(2) INSERT INTO company  (companyID, companyName)
                 VALUES (   111,      'cityOne' )
```

The following row would be inserted.

```
COMPANYID   COMPANYNAME                      CITY
----------- ------------------------------ --------------------
        111 cityOne                          TORONTO
```

e) Using NULL Values:

NULL values represent an unknown state. For example, let's review the contents of the table student, which contains NULL values.

```
NAME                 MARK
-------------------- -----------
Peter                        100
Mary                          60
John                           -
Raul                          80
Tom                            -
```

John and Tom were sick the day of the exam, therefore the teacher put NULL values for their marks. This is different than giving them a mark of zero. If you issue this statement:

SELECT avg(mark) as average FROM student

The result is:

```
AVERAGE
-----------
         80
```

f) Constraints:

Constraints allow you to create rules for the data in your tables. You can define four types of constraints on a table.

- A ***unique constraint*** ensures that no duplicate key values can be entered in the table.
- A ***referential constraint*** ensures that a value in one table must have a corresponding entry in a related table.
- A ***check constraint*** ensures that the values you enter into the column are within the rules specified when the table was defined.
- An ***informational constraint*** allows you to enforce or not enforce a constraint.

g) Table Compression:

We can compress tables to a certain extent by using the VALUE COMPRESSION clause of the CREATE TABLE statement. This clause tells DB2 that it can use a different internal format for the table rows so that they occupy less space.

In a sense, this clause turns on compression for the table; however, you need to specify another clause, COMPRESS SYSTEM DEFAULT, for each column that you want to compress. Only the columns whose values are normally NULL or the system default value of 0 can be compressed. Also, the data type must not be DATE, TIME, or TIMESTAMP. If the data type is a varying-length string, this clause is ignored. Here's an example:

```
CREATE TABLE company (
      company_ID  INTEGER  NOT NULL PRIMARY KEY,
      name CHAR(10),
      address  VARCHAR(30)  COMPRESS SYSTEM DEFAULT,
      no_employees  INTEGER   NOT NULL COMPRESS SYSTEM DEFAULT
      )
     VALUE COMPRESSION
```

h) Materialized Query Tables and Summary Tables:

Materialized query tables (MQTs) allow users to create tables with data based on the results of a query. The DB2 optimizer can later use these tables to determine whether a query can best be served by accessing an MQT instead of the base tables. Here is an example of an MQT:

```
CREATE SUMMARY TABLE my_summary
     AS  (SELECT  city_name, population
             FROM country A, city B
     WHERE  A.country_id = B.country_no)
             DATA INITIALLY DEFERRED
             REFRESH DEFERRED
```

The SUMMARY keyword is optional. The DATA INITIALLY DEFERRED clause indicates that DB2 will not immediately populate the my_summary MQT table after creation, but will populate it following the REFRESH TABLE statement:

```
     REFRESH TABLE my_summary
```

i) Temporary Tables:

Temporary tables can be classified as system or user tables. DB2 manages system temporary tables in the system temporary table space. DB2 creates and drops these tables automatically. Users don't have control over system temporary tables.

We create user temporary tables inside a user temporary table space. For example, the following statement creates a user temporary table space called usrtmp4k.

3. Table Spaces:

A table space is a logical object in your database. It is used to associate your logical tables and indexes to their physical storage devices (containers or storage paths) and physical memory (buffer pools). All tables and indexes must reside in table spaces.

1. Table Space Classification:

When you create a new database, the database manager creates some default tablespaces for database. These tables pace are used as a storage for user and temporary data. Each database must contain at least three table spaces as given here:

1. Catalog tablespace
2. User tablespace
3. Temporary tablespace

- Catalog tablespace: It contains system catalog tables for the database. It is named as **SYSCATSPACE** and it cannot be dropped.
- User tablespace: This tablespace contains user-defined tables. In a database, we have one default user tablespace, named as **USERSPACE1.** If you do not specify user-defined tablespace for a table at the time you create it, then the database manager chooses default user tablespace for you.
- Temporary tablespace: A temporary tablespace contains temporary table data. This tablespace contains system temporary tablespaces or user temporary tablespace.
  - **System Temporary Table Space:** This is used to store all the temporary data during SQL process such as Sorting, reorganizing, joining etc. There must be at least one Temporary TS per database; the default one created at the time of database creation is **TEMPSPACE1**.
  - **User temporary Table Space:** These can be used to store the declares temporary tables. These are not mandatory, can be created if necessary.

2. Tablespaces and storage management:

Tablespaces can be setup in different ways, depending on how you want to use them. You can setup the operating system to manage tablespace allocation, you can let the database manager allocate space or you can choose automatic allocation of tablespace for your data.

The following three types of managed spaces are available:

- **System-managed space (SMS)**: This type of table space is managed by the operating system and requires minimal administration.
- **Database-managed space (DMS)**: This type of table space is managed by the DB2database manager, and it requires some administration.
- **Automatic Storage:** This type of table space can be either SMS or DMS, and the space is managed by the DB2 database manager, but it requires minimal administration. The

database manager determines which containers are to be assigned to the table space, based upon the storage paths that are associated with the database.

To create a table space, use the **CREATE TABLESPACE** statement. A table space can be created with any of these page sizes: 4K, 8K, 16K, and 32K. A corresponding buffer pool of the same page size must exist prior to issuing this statement.
The following are commands to maintain databases in DB2

- **To list all tablespaces: list tablespaces;**
- **To list all tablespaces with details: list tablespaces show detail**;

4. Schemas:

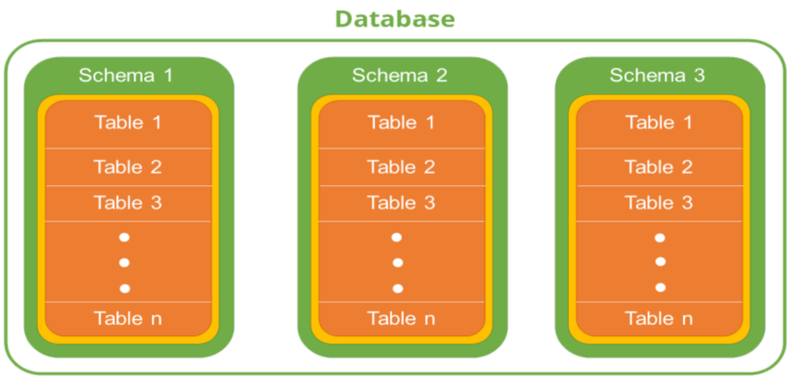
A schema is a database object used to logically group other database objects together.
Every database object name has two parts:

schema_name.object_name

This two-part name (also known as the fully qualified name) must be unique within the database.
Here are some examples:

db2admin.tab1
mary.idx1
sales.tblspace1

A schema can contain tables, functions, indices, tablespaces, procedures, triggers etc.
For example, you create two different schemas named as "Regular" and "Parttime" for an "employee" database. You can also create two different tables with the same name "Employee" where one table has regular information and the other has part time information of employee. It doesn't have actually two tables with the same name in spite they have two different schemas "Regular" and "Part time". It facilitates user to work with both without facing any problem. This feature is useful when there are constraints on the naming of tables.



To create the schema user1, use the CREATE SCHEMA statement as follows:

CREATE SCHEMA user1
Or, if you are connected to the database as *user1*, when you create the first new object using this connection without explicitly typing the schema name, DB2 will automatically create the schema *user1* and then the object. This assumes you have the appropriate authorization, in this case, the **IMPLICIT_SCHEMA** privilege.

The following statement creates the schema *user1*, followed by the table *table1*.
CREATE TABLE table1 (mycol int)
You are connected to the database as *user1*;
you can also create objects under a different schema.
In this case, explicitly indicate the schema name, for example:
CREATE TABLE newuser.table1 (mycol int)
This statement creates a table called table1 in the schema *new user*. If the schema doesn't already exist, it is created.

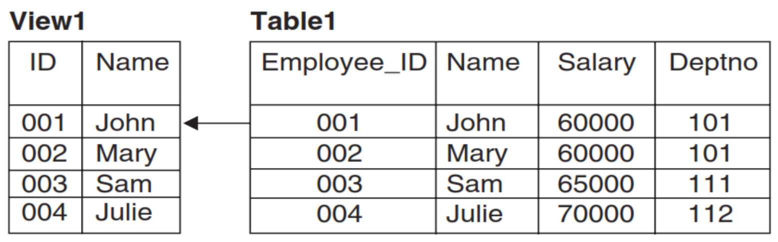
When you access a database object, you can omit the schema name. Let's say you are connected to the database as user1, and you issue the following statement:
SELECT * FROM table1
This statement references table user1.table1.

If the table you want to access is newuser.table1, you must explicitly include the schema name:
SELECT * FROM newuser.table1

You cannot alter a schema, but you can drop it (as long as no objects exist within the schema) and recreate it with the new definition.
Use the DROP SCHEMA statement to drop a schema:
DROP SCHEMA newuser RESTRICT
We must specify the RESTRICT keyword; it is part of the DROP SCHEMA syntax and serves as a reminder that you cannot drop a schema unless it is unused.

5. Views:

A **view** is a virtual table derived from one or more tables or other views. It is virtual because it does not contain any data, but a definition of a table based on the result of a **SELECT** statement. Figure 7.26 illustrates view *view1* derived from table *table1*.


View1

ID	Name
001	John
002	Mary
003	Sam
004	Julie

Table1

Employee_ID	Name	Salary	Deptno
001	John	60000	101
002	Mary	60000	101
003	Sam	65000	111
004	Julie	70000	112

A view derived from a table

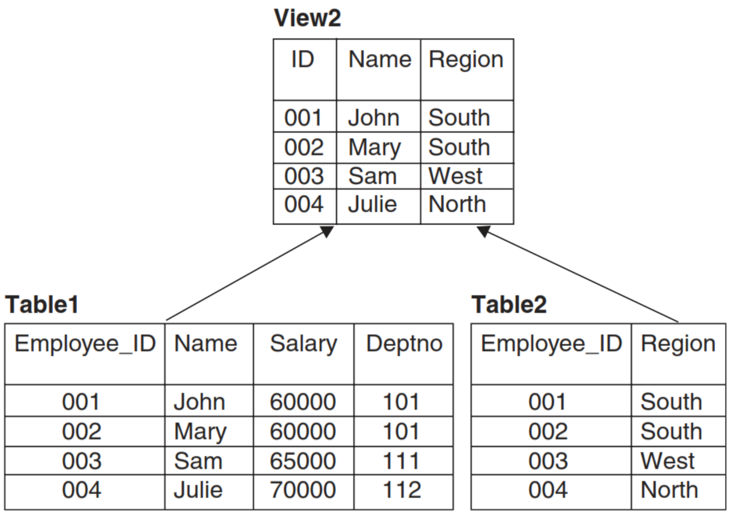
A view does not need to contain all the columns of the base table. Its columns do not need to have the same names as the base table, either. This is illustrated in Figure, where the view consists of only two columns, and the first column of the view has a different name than the corresponding column in the base table. This is particularly useful for hiding confidential information from users. You can create a view using the **CREATEVIEW** statement. For example, to create the view *view1* shown in Figure, issue this statement.

```
CREATE VIEW view1 (id, name)
    AS SELECT employee_id, name FROM table1
```

To display the contents of *view1*, use the following statement.

```
SELECT * FROM view1
```

You can also create views based on multiple tables. The below Figure shows a view created from two tables.



View2

ID	Name	Region
001	John	South
002	Mary	South
003	Sam	West
004	Julie	North

Table1

Employee_ID	Name	Salary	Deptno
001	John	60000	101
002	Mary	60000	101
003	Sam	65000	111
004	Julie	70000	112

Table2

Employee_ID	Region
001	South
002	South
003	West
004	North

A view derived from two tables

Here is the corresponding **CREATEVIEW** statement for the above Figure.

```
CREATE VIEW view2 (id, name, region)
        AS SELECT table1.employee_id, table1.name, table2.region
        FROM table1, table2
        WHERE table1.employee_id = table2.employee_id
```

With this statement we have combined the information of *table1* and *table2* into *view2*, while limiting access to the salary information.

When you create a view, its definition is stored in the **system catalog table SYSCAT.VIEWS**. This table contains information about each view such as its name, schema, whether or not it is read-only, and the SQL statement used to create the view.

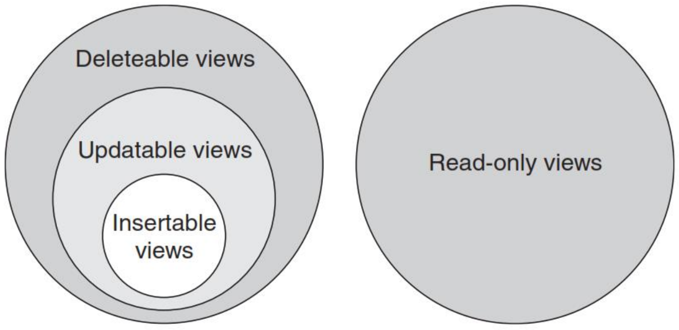
When a view is referenced in a query, DB2 reads and executes the view definition from the SYSCAT.VIEWS table, pulls the data from the base table, and presents it to the users.
To remove a view, use the **DROPVIEW** statement.
For example, to remove the view *view1* use **DROP VIEW view1**

1. **Views Classification:**

    Views are classified by the operations they allow. There are four classes of views:

    a) Deleteable views
    b) Updatable views
    c) Insertable views
    d) Read-only views

In the SYSCAT.VIEWS catalog table, when the value of the column READ-ONLY is Y, this indicates that the view is read-only; otherwise, it is either a deleteable, updatable, or insertable view. The Figure shows *view2* is a read-only view, but *view1* is not. Figure illustrates the relationship between the different types of views.



View classifications and relationships

a) **Deleteable Views:**
   A **deleteable view** allows you to execute the **DELETE** statement against it. All of the following must be true.
b) **Updatable Views:**
   An updatable view is a special case of a deleteable view. A view is updatable when at least one of its columns is updatable. All of the following must be true.

You can update *view1* using the **UPDATE** statement, and the changes will be applied to its base table.
For example, the following statement changes the value of column *employee_id* to 100 for records with the *name* value of Mary in *table1*.

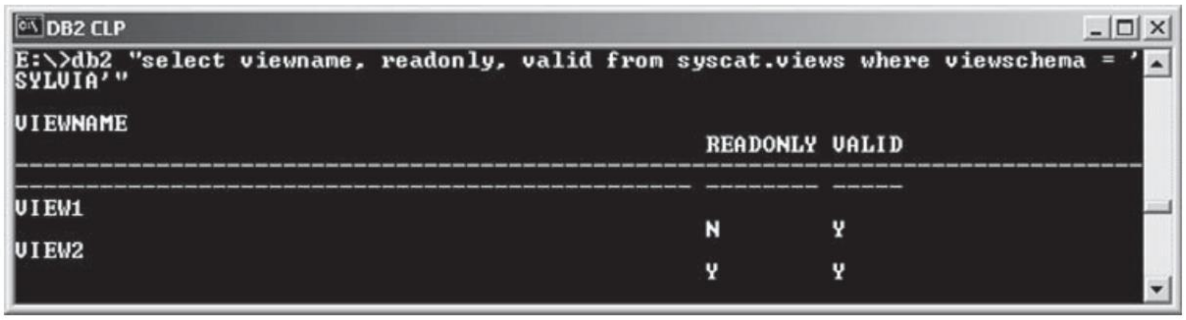
UPDATE view1 SET id='100' WHERE name = 'Mary';

c) **Insertable Views:**
   An insertable view allows you to execute the **INSERT** statement against it. A view is insertable when all of its columns are updatable. For example, *view1* fits this rule. The following statement will insert a row into *table1*, which is the base table of *view1*.

INSERT INTO view1 VALUES ('200', 'Ben');

d) Read-Only Views:

A read-only view is not deleteable. Its read-only property is also stored in the SYSCAT.VIEWS table as shown below.



```
E:\>db2 "select viewname, readonly, valid from syscat.views where viewschema = '
SYLVIA'"

VIEWNAME
                                                 READONLY VALID
---------------------------------------------------------------------------------
------------------------------------------------ -------- -----
VIEW1
                                                 N        Y
VIEW2
                                                 Y        Y
```

Figure 7.28 View definitions stored in the SYSCAT.VIEWS table

2. Nested Views:

Nested views are ones based on other views, for example:

CREATE VIEW view4
AS SELECT * FROM view3

In this example, *view4* has been created based on *view3*, which was used in earlier examples.

The **WITHCHECKOPTION** clause specified in *view3* is still in effect for *view4*; therefore, the following **INSERT** statement fails for the same reason it fails when inserting into *view3*.

INSERT INTO view4 VALUES ('007','Shawn',201)

When a view is defined with the **WITHCHECKOPTION** clause, the search condition is propagated through all the views that depend on it.

6. Indexes:

An index is a data structure that exists physically on disk. It consists of one or more columns in order and pointers to rows in the table. It is by definition a partial duplication of the data, but is tied to the data by the DBMS so that an index cannot be different than the data in the table

Indexes are database objects that are built based on one or more columns of a table. They are used for two main reasons:

- To improve query performance. Indexes can be used to access the data faster using direct access to rows based on the index key values.

- To guarantee uniqueness when they are defined as unique indexes.

1. **Working with Indexes:**

To create an index, use the CREATE INDEX statement. This statement requires at a minimum:

- The name of the index
- The name of the associated table
- The columns that make up the index (also known as index keys)

In addition, you can specify the following:

- Whether the index is unique (enforce uniqueness for the key values) or non-unique (allow duplicates)
- Which order DB2 should use to build and maintain the key values: ascending (ASC, the default) or descending (DESC) order
- Whether to create INCLUDE columns that are not part of the index key but are columns often retrieved by your queries

For example, let's consider the following statement.

```
CREATE UNIQUE INDEX company_ix
        ON company (company_ID ASC, name DESC)
        INCLUDE (no_employees)
```

This statement creates a unique index **company_ix**. This index is associated to the table company based on the columns **company_ID** in ascending order and name in descending order.

In addition, an INCLUDE column **no_employees** were added to the index definition. This column does not belong to the index key, that is, the index will not be built and maintained taking this column into consideration. Instead, an INCLUDE column is useful for performance reasons.

Once an index has been created, it cannot be modified. To add or remove a key column from the index, you must drop and recreate the index. To drop an index, use the DROP INDEX statement.

For example:

DROP INDEX index1

Indexes can improve query performance considerably; however, the more indexes you define on a table, the more the cost incurred when updating the table because the indexes will also need to be updated.

2. **Clustering Indexes:**

Clustered indexes are indexes where the actual data in the table is placed at least roughly in order of the index. If a clustered index exists on a table, DB2 will attempt to insert data in the order of the clustering index. A clustering index is created so that the index pages physically map to the data pages. That is, all the records that have the same index key are physically close together. Figure 7.21 illustrates how index1 works when created as a clustering index using the CLUSTER clause as follows.

CREATE INDEX index1 ON sales (sales_person) CLUSTER

In the figure, when you issue this query:

SELECT * FROM sales WHERE sales_person = 'Sam'

DB2 would still use index **index1** but it requires less I/O access to the disk because the desired data pages are clustered together on extents 4 and 5.

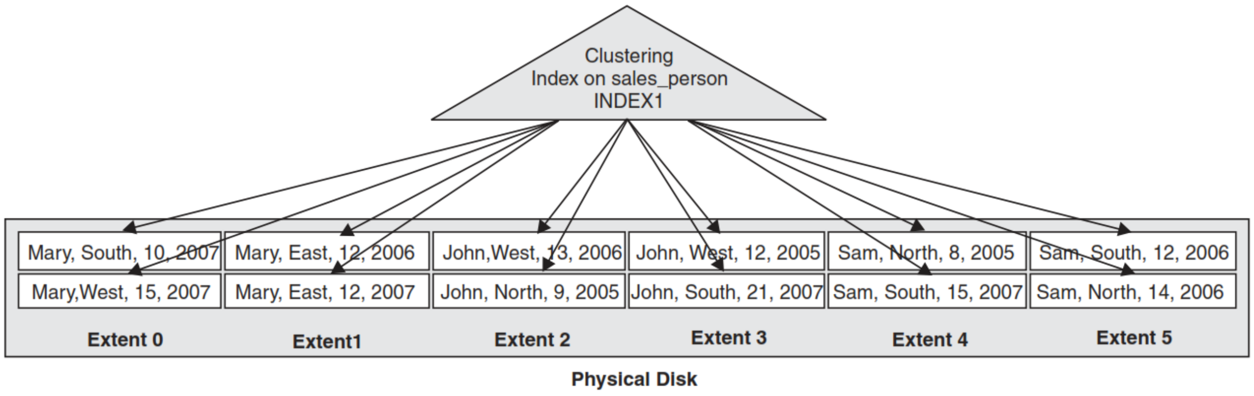


Figure 7.21 A clustering index on the sales_person column

Advantages of indexes::

To improve performance

To access data faster

To cluster the data

To make sure that particular row is unique

To provide index only to access data

7. Sequences:

A sequence is a database object that allows automatic generation of values. A sequence is a software function that generates integer numbers in either ascending or descending order, within a definite range. Unlike identity columns, this object does not depend on any table—the same sequence object can be used across the database. You use sequence for availing integer numbers say, for employee_id or transaction_id. A sequence can support SMALLINT, BIGINT, INTEGER, and DECIMAL data types. A sequence can be shared among multiple applications. A sequence is incremented or decremented irrespective of transactions. To create a sequence, use the **CREATE SEQUENCE** statement.

The following parameters are used for sequences:

- **Data type**: This is the data type of the returned incremented value. (SMALLINT, BIGINT, INTEGER, NUMBER, DOUBLE)
- **START WITH**: The reference value, with which the sequence starts.
- **MINVALUE**: A minimum value for a sequence to start with.
- **MAXVALUE**: A maximum value for a sequence.
- **INCREMENT BY**: step value by which a sequence is incremented.
- **Sequence cycling**: the **CYCLE** clause causes generation of the sequence repeatedly. The sequence generation is conducted by referring the returned value, which is stored into the database by previous sequence generation.

Example:

```
CREATE SEQUENCE myseq AS INTEGER
        START WITH 1 INCREMENT BY 1
        NO MAXVALUE
        NO CYCLE
        CACHE 5
```

This statement creates the sequence **myseq**, which is of type **INTEGER**. The sequence starts with a value of 1 and then increases by 1 each time it's invoked for the next value.
The **NO MAXVALUE** clause indicates there is no explicit maximum value in which the sequence will stop; therefore, it will be bound by the limit of the data type, in this case, **INTEGER**.
The **NO CYCLE** clause indicates the sequence will not start over from the beginning once the limit is reached. **CACHE 5** indicates five sequence numbers will be cached in memory, and the sixth number in the sequence would be stored in a catalog table. Sequence numbers are cached in memory for performance reasons; otherwise, DB2 needs to access the catalog tables constantly to retrieve the next value in line.
The following table shows different operation we can do on sequences:

Table 7.6 Other Statements Used with Sequences

Statement	Explanation
ALTER SEQUENCE	Alters the characteristics of a sequence, like the increment value
DROP SEQUENCE	Drops the sequence
NEXTVAL FOR *sequence_name* or **NEXT VALUE FOR *sequence_name***	Retrieves the next value generated in the sequence
PREVVAL FOR *sequence_name* or **PREVIOUS VALUE FOR *sequence_name***	Retrieves the previous value generated in the sequence

DB2 Data Types:

A **data type** indicates what type of data can be saved in a column or variableand how large it can be. DB2 data types are either:

1. Built-in data types
2. User-defined types (UDTs)

1. DB2 Built-in Data Types:

DB2 provides several built-in data types, which can be classified into the following categories:

a) Numeric
b) String
c) Datetime
d) Extensible Markup Language

Figure 7.8 summarizes the built-in data types supported in DB2.

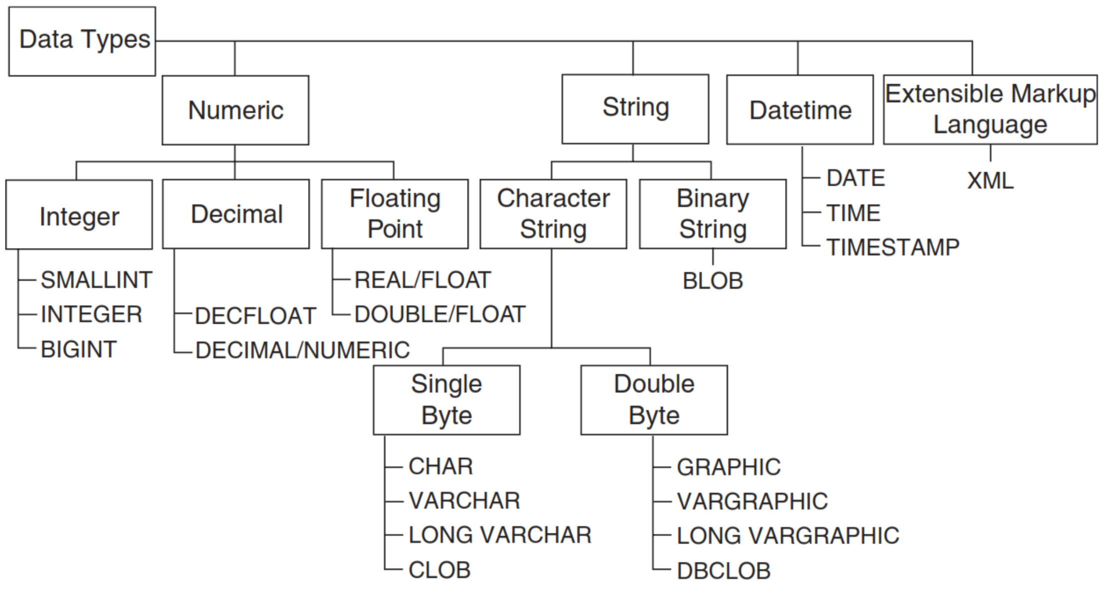


Figure 7.8 The DB2 built-in data types

a) Numeric Data Types:

The numeric data types include the following:

- **Small integer (SMALLINT)** - uses the least amount of storage in the database for each value. The data value range for a SMALLINT is –32768 to 32767. The precision for a SMALLINT is 5 digits to the left of the decimal. Each SMALLINT column value uses 2 bytes of database storage.
- **Integer (INT or INTEGER**) - An INTEGER uses twice as much storage as a SMALLINT but has a greater range of possible values. The data value range for an INTEGER data type is –2,147,483,648 to 2,147,483,647. The precision for an INTEGER is 10 digits to the left of the decimal. Each INTEGER column value uses 4 bytes of database storage.
- **Big integer (BIGINT**) - The BIGINT data type is available for supporting 64-bit integers. The value range for BIGINT is –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Since platforms include native support for 64-bit integers, processing large numbers with BIGINT is more efficient than processing with DECIMAL and more precise than using DOUBLE or REAL. Each BIGINT column value uses 8 bytes of database storage. The SMALLINT, INTEGER, and BIGINT data types do not allow any digits to the right of the decimal.
- **DECIMAL/NUMERIC -** A DECIMAL or NUMERIC data type is used for numbers with fractional and whole parts. The DECIMAL data is stored in a packed format. The precision is the total number of digits (ranging from 1 to 31 digits), and the scale is the number of digits in the fractional part of the number.
- **REAL/FLOAT** - data type is an approximation of a number. The approximation requires 32 bits or 4 bytes of storage. To specify a single-precision number using the REAL data type, you must define its length between 1 and 24.
- **DOUBLE/FLOAT** - data type is also an approximation of a number. The approximation

requires 64 bits or 8 bytes of storage. To specify a double-precision number using the FLOAT data type, you must define its length between 25 and 53.

- **Decimal floating-point (DECFLOAT) -** data type, new in DB2 9.5, is useful in business applications that deal with exact decimal values such as financial applications where monetary values are manipulated. It supports the 16-digit and 34-digit decimal floating-point encodings, corresponding to 8 bytes and 16 bytes of storages respectively.

b) String Data Types:

You can define string or character columns as either fixed length or variable length. The character

string data types include the following:

- **Character (CHAR)** - A CHAR column is stored as a fixed-length field in the database; if the string you enter is shorter than the defined length of the column, the string will be padded with blanks. A fixed-length character column can have a maximum length of 254 characters. If no length is specified, DB2 will use the default length of 1 character.
- **Variable character (VARCHAR) -** A VARCHAR column stores only the characters entered for the string, and its maximum size closely corresponds to the page size for the table. For example, for a table created in a table space with a 32K page size, the maximum length of a VARCHAR string is 32,672 characters.
- **Long variable character (LONG VARCHAR) -** A LONG VARCHAR column also stores only the characters entered for the string, but it does not store them in the data object with the other columns. LONG VARCHAR is a special data type stored in a separate long object. The maximum length of a LONG VARCHAR string is 32,700 characters.
- **GRAPHIC -** GRAPHIC data types use 2 bytes of storage to represent a single character. There are three types:
  - GRAPHIC: fixed length with a maximum of 127 characters
  - VARGRAPHIC: varying length with a maximum of 16,336 characters
  - LONG VARGRAPHIC: varying length with a maximum of 16,350 characters
- When a VARCHAR data type’s maximum size of 32,672 bytes is not enough to hold your data, use large objects. Large objects (LOBs) can store data greater than 32K up to 2GB in size. They are typically used to store information such as an audio file, or a picture. Though XML documents can also be stored as LOBs, with DB2 9 we recommend you to use the XML data type instead in order to take advantage of pure XML technology.

  Three kinds of LOB data types are provided with DB2:
  - Binary large object (BLOB)
  - Single-byte character large object (CLOB)
  - Double-byte character large object (DBCLOB)

BLOB - store variable-length data in binary format and are ideal for storing video or audio information in your database. This data type has some restrictions; for example, you cannot sort by this type of column.

CLOB - CLOBs store large amounts of variable-length single-byte character set (SBCS) or multibyte character set (MBCS) character strings, for example, large amounts of text such as white papers or long documents.

DBCLOBs store large amounts of variable-length double-byte character set (DBCS) character strings, such as large amounts of text in Chinese.

- **VARGRAPHIC& LONG VARGRAPHIC -** Similar to LONG VARCHAR and LONG VARGRAPHIC data types, LOBs are accessed directly from disk without going through the buffer pool, so using LOBs is slower than using other data types.

c) Date& Time Data Types

Date and time data types are special character data types used to store date and/or time values inspecific formats. These data types are stored in an internal format by DB2, and use an externalformat for the users. DB2 supports three datetime data types: DATE, TIME, and TIMESTAMP.

- The **DATE** type stores a date value (month, day, and year). Its external format is MMDD-YYYYor MM/DD/YYYY.
- The **TIME** type stores a time value (hour, minute, and second). Its external format isHH:MM:SS or HH.MM.SS.
- The **TIMESTAMP** type combines the DATE and TIME types but also stores the timedown to the nanosecond. Its external format is MM-DD-YYYY-HH.MM.SS.NNNNNN.

d) Extensible Markup Language Data Type:

The Extensible Markup Language data type, XML, is a data type that is part of the SQL standardwith XML extensions(SQL/XML). This newdata type allowsfor storing well-formedXMLdocuments natively(in a parsed-hierarchical manner) internally in a DB2 database.

2. User-Defined Types:

User-defined types (UDTs) allow database users to create or extend the use of data types to theirown needs. UDTs can be classified as **DISTINCT, STRUCTURE,** or **REFERENCE**.

A DISTINCT UDT can enforce business rules and prevent data from being used improperly.UDTs are built on top of existing DB2 built-in data types. To create a UDT, use the **CREATEDISTINCTTYPE** statement:

CREATE DISTINCT TYPE *type_name* AS *built-in_datatype* WITH COMPARISONS

For example, let’s say you create two UDTs, *celsius* and *fahrenheit*:

```
CREATE DISTINCT TYPE  celsius  AS integer WITH COMPARISONS
CREATE DISTINCT TYPE fahrenheit AS integer WITH COMPARISONS
```

The first statement creates a casting function named *celsius*, and the second statement creates acasting function named *fahrenheit*.

Now, let’s say you create a table using the newly created UDTs:

```
CREATE TABLE temperature
 (country  varchar(100),
  average_temp_c  celsius,
  average_temp_f  fahrenheit
 )
```

Table *temperature* keeps track of the average temperature of each country in the world in bothCelsius and Fahrenheit. If you would like to know which countries have an average temperaturehigher than 35 degrees Celsius, you can issue this query:

```
SELECT country FROM temperature WHERE average_temp_c > 35
```

Would this query work? At first, you may think it will, but remember that *average_temp_c* has data type *celsius*, while 35 is an INTEGER. Even though *celsius* was created based on the INTEGER built-in
data type, this comparison cannot be performed as is. To resolve this problem, use the casting function generated with the creation of the *Celsius* UDT as shown below:

```
SELECT country FROM temperature WHERE average_temp_c > celsius(35)
```

UDTs enforce business rules by preventing illegitimate operations. For example, the followingquery will not work:

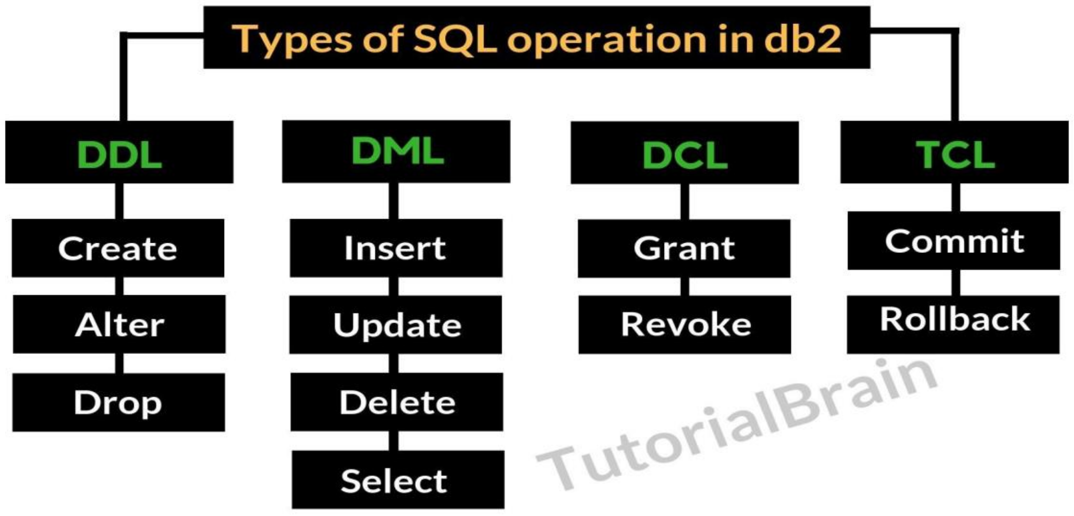
```
SELECT country FROM temperature WHERE average_temp_c = average_temp_f
```

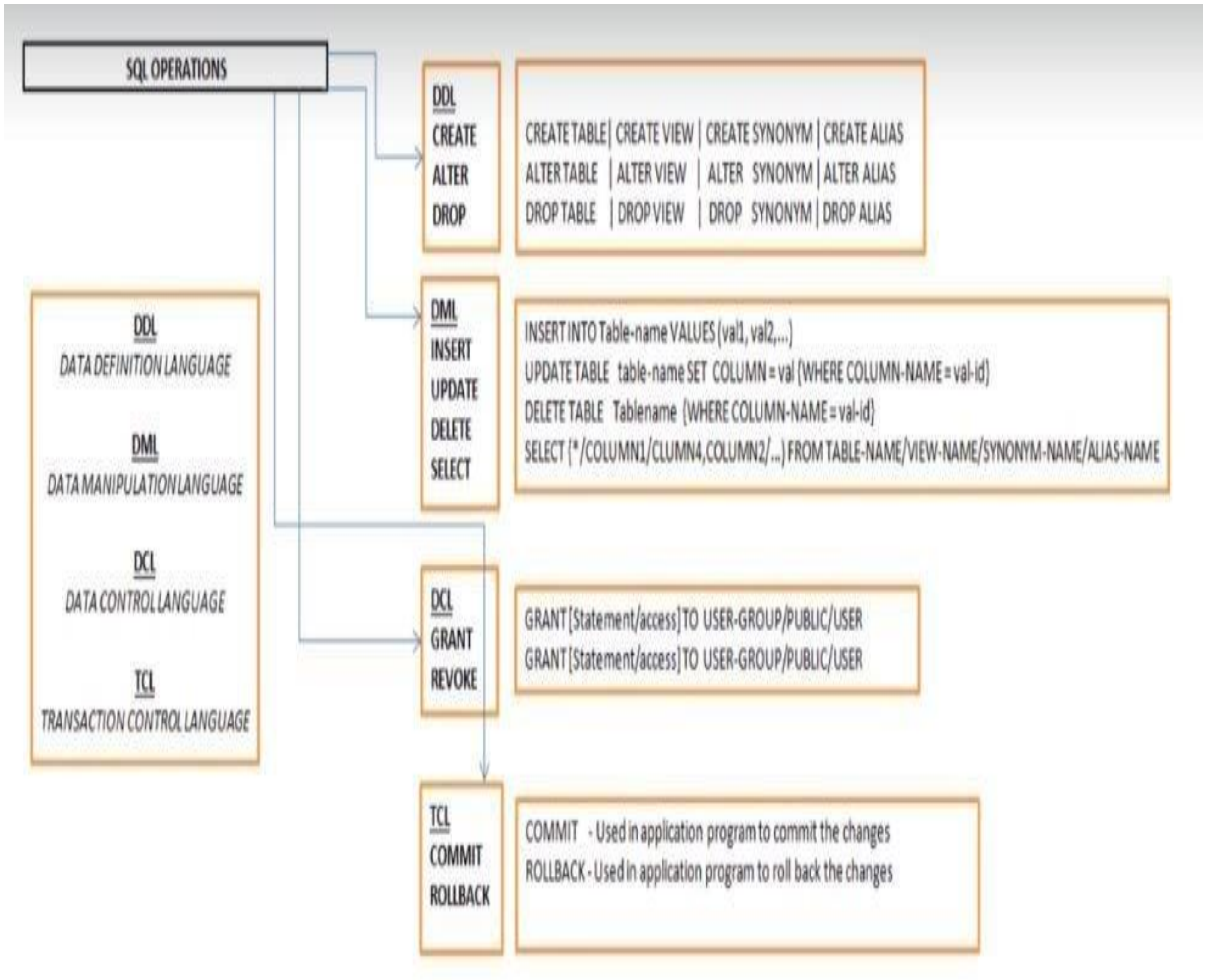
Because column *average_temp_c* and *average_temp_f* are of different data types, this query willresult in an error. If UDTs had not been created and the INTEGER built-in data type had beenused instead for both columns, the query would have worked.

Choosing the Proper Data Type: It is important to choose the proper data type because this affects performance and disk space.To choose the correct data type, you need to understand how your data will be used and its possiblevalues.SQL statements allow you to work with the relational and XML data stored in your database. The statements are applied against the database you are connected to, not against the entire DB2 environment.

SQL (Structured Query Language):

There are different classes of SQL statements.




SQL OPERATIONS
DDL
DATA DEFINITION LANGUAGE
DML
DATA MANIPULATION LANGUAGE
DCL
DATA CONTROL LANGUAGE
TCL
TRANSACTION CONTROL LANGUAGE
DDL
CREATE
ALTER
DROP
CREATE TABLE | CREATE VIEW | CREATE SYNONYM | CREATE ALIAS
ALTER TABLE | ALTER VIEW | ALTER SYNONYM | ALTER ALIAS
DROP TABLE | DROP VIEW | DROP SYNONYM | DROP ALIAS
DML
INSERT
UPDATE
DELETE
SELECT
INSERT INTO Table-name VALUES (val1, val2,...)
UPDATE TABLE table-name SET COLUMN = val (WHERE COLUMN-NAME = val-id)
DELETE TABLE Tablename (WHERE COLUMN-NAME = val-id)
SELECT (*/COLUMN1/CLUMN4,COLUMN2/...) FROM TABLE-NAME/VIEW-NAME/SYNONYM-NAME/ALIAS-NAME
DCL
GRANT
REVOKE
GRANT [Statement/access] TO USER-GROUP/PUBLIC/USER
GRANT [Statement/access] TO USER-GROUP/PUBLIC/USER
TCL
COMMIT
ROLLBACK
COMMIT - Used in application program to commit the changes
ROLLBACK - Used in application program to roll back the changes

Data Definition Language (DDL):

Data Definition Language (DDL) statements are used to define the database structure or schema. Data Definition Language understanding with database schemas and describes how the data

should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE belongs to the DDL. DDL is about "metadata".
For example:

```
CREATE INDEX ix1 ON t1 (salary)
ALTER TABLE t1 ADD hiredate DATE
DROP VIEW view1
```

DDL includes commands such as CREATE, ALTER and DROP statements. DDL is used to CREATE, ALTER OR DROP the database objects (Table, Views, Users).
Data Definition Language (DDL) are used different statements:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- RENAME - rename an object

The Create Table Command

The create table command defines each column of the table uniquely. Each column has minimum of three attributes.

- Name
- Data type
- Size(column width).

Each table column definition is a single clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statement is terminated with a semicolon.

The Structure of Create Table Command

Table name is Student

Column name	**Data type**	**Size**
Reg_no	varchar2	10
Name	Char	30
DOB	Date	
Address	varchar2	50

Example:

```
CREATE TABLE Student
        (Reg_no varchar2(10),
         Name char(30),
         DOB date,
         Address varchar2(50));
```

The DROP Command

Syntax:

```
DROP TABLE <table_name>
```

Example:

```
DROP TABLE Student;
```

It will destroy the table and all data which will be recorded in it.

The TRUNCATE Command

Syntax:

```
TRUNCATE TABLE <Table_name>
```

Example:

```
TRUNCATE TABLE Student;
```

The RENAME Command

Syntax:

```
RENAME <OldTableName> TO <NewTableName>
```

Example:

```
RENAME <Student> TO <Stu>
```

The old name table was **Student** now new name is the **Stu.**

The ALTER Table Command

By The use of ALTER TABLE Command we can **modify** our exiting table.

Adding New Columns

Syntax:

```
ALTER TABLE <table_name>
        ADD (<NewColumnName><Data_Type>(<size>),......n)
```

Example:

```
ALTER TABLE Student ADD (Age number(2), Marks number(3));
```

The Student table is already exist and then we added two more columns **Age** and **Marks** respectively, by the use of above command.

Dropping a Column from the Table

Syntax:

ALTER TABLE <table_name> DROP COLUMN <column_name>

Example:

ALTER TABLE Student DROP COLUMN Age;

This command will drop particular column

Modifying Existing Table

Syntax:

ALTER TABLE <table_name> MODIFY (<column_name><NewDataType>(<NewSize>))

Example:

ALTER TABLE Student MODIFY (Name Varchar2(40));

The Name column already exist in Student table, it was char and size 30, now it is modified by Varchar2 and size 40.

Restriction on the ALTER TABLE

Using the ALTER TABLE clause the following tasks cannot be performed.

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

Data Manipulation Language (DML):

Data Manipulation Language (DML) statements are used for managing data within schema objects DML deals with data manipulation, and therefore includes most common SQL statements such SELECT, INSERT, etc. DML allows to add / modify / delete data itself. DML is used to manipulate with the existing data in the database objects (insert, select, update, delete).

DML Commands:

1.INSERT

2.SELECT

3.UPDATE

4.DELETE

Select -Viewing Data in the Table (Select Command)

Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The SELECT SQL verb is used to achieve this.

All Rows and All Columns

Syntax: **SELECT * FROM Table_name;**

eg: **Select * from Student;** It will show all the table records.

SELECT First_name, DOB FROM STUDENT WHERE Reg_no = 'S101';

This Command will show one row. Because you have given condition for only one row and particular records. If condition which has given in WHERE Clause is true then records will be fetched otherwise it will show **no records selected.**

Eliminating Duplicates:

A table could hold duplicate rows. In such a case, you can eliminate duplicates.

Syntax: SELECT DISTINCT col, col, .., FROM table_name;

eg : **SELECT DISTINCT * FROM Student;**

or : **SELECT DISTINCT first_name, city, pincode FROM Student;**

It scans through entire rows, and eliminates rows that have exactly the same contents in each column.

Sorting DATA:

The Rows retrieved from the table will be sorted in either **Ascending** or **Descending** order depending on the condition specified in select statement, the Keyword has used **ORDER BY.**

SELECT * FROM Student ORDER BY First_Name;

it will in show records as alphabetical order from A to Z ascending order. If you want Descending order means Z to A then used **DESC** Keyword at last.

eg : **SELECT first_name, city,pincode FROM Student**
ORDER BY First_name DESC;

Insert command:

To insert a new row into a table, you use the `INSERT` statement. The following shows the syntax:

INSERT INTOtable_name (column_list)VALUES(value_list);

Or

INSERT INTOtable_name VALUES(value_list);

In this syntax:

- First, specify the name of the table to which you want to insert a new row after the `INSERT INTO` keywords followed by comma-separated column list enclosed in parentheses.
- Then, specify the comma-list of values after the `VALUES` keyword. The values list is also surrounded by parentheses. The order of values in the value list must be corresponding to the order of columns in the column list.

If you don’t specify a column of a table in the column list, you must ensure that Db2 can provide a value for insertion or an error will occur.

Ex:

```
INSERT INTO lists(list_name,
created_at)VALUES('Monthly Digest', DEFAULT);
```

Update Statement:it is used to update the values in a table.

Syntax: UPDATE table-name SET Column-name = value

Ex: update table set b='y' where b='x'

Update student set marks1=67 where marks2=56;

Data Control Language (DCL):

DCL is the abstract of Data Control Language. Data Control Language includes commands such as GRANT, and concerns with rights, permissions and other controls of the database system. DCL is used to grant / revoke permissions on databases and their contents. DCL is simple, but MySQL permissions are a bit complex. DCL is about security. DCL is used to control the database transaction.DCL statement allow you to control who has access to specific object in your database.

1. GRANT
2. REVOKE

GRANT: It provides the user's access privileges to the database. In the MySQL database offers both the administrator and user a great extent of the control options. By the administration side of the process includes the possibility for the administrators to control certain user privileges over the

It creates an entry in the security system that allows a user in the current database to work with data in the current database or execute specific statements.

Syntax :

Syntax:GRANT names of privileges/ALL **ON** database object name **TO**user name;

Normally, a database administrator first uses CREATE USER to create an account, then GRANT to define its privileges and characteristics.

For example:

CREATE USER 'arjun'@'localhost' IDENTIFIED BY 'mypass';
GRANT ALL ON db1.* TO 'arjun'@'localhost';
GRANT SELECT ON child TO 'arjun'@'localhost';
GRANT USAGE ON *.* TO 'arjun'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;

REVOKE : The REVOKE statement enables system administrators and to revoke the privileges from MySQL accounts.

Syntax :

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [, user] ...
```

For example:

```
mysql> REVOKE INSERT ON *.* FROM 'arjun'@'localhost';
```

TCL Operations in DB2

It is a TRANSACTION CONTROL LANGUAGE. and it is not used directly in DB2, rather it is used in COBOL DB2 program inside the application program. Once you have created the COBOL DB2 program inside the program we can use DCL operation.

In TCL we have two statements:

i) **COMMIT**– It is used in an application program to commit the changes. For Example, if you want to COMMIT changes at a particular point, you can just give COMMIT in the program.

ii) **ROLLBACK**– It is used in the application program to roll back the changes. ROLLBACK has used to blackout the changes.

For Example, we want to back out the changes due to some condition or due to an event.

****

UNIT - IV

Retrieving Data& Functions in DB2

Retrieving Data from multiple tables: Joins, Union operations in DB2, Grouping, Sub Queries, DB2 Functions and Procedures, Scalar Functions, Column Functions, Row functions.

Retrieving Data from multiple tables:

The related tables of a large database are linked through the use of foreign and primary keys or what are often referred to as common columns. The ability to join tables will enable you to add more meaning to the result table that is produced. For 'n' number tables to be joined in a query, minimum (n-1) join conditions are necessary. Based on the join conditions, Oracle combines the matching pair of rows and displays the one which satisfies the join condition.

JOINS:

Joins are classified as below

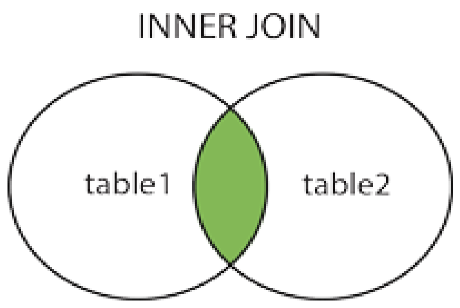
- Inner Join - (also known as an equijoin or a simple join) - Creates a join by using a commonly named and defined columns based on some condition
- Natural join – It is a variation of inner join with a small change in its syntax. No condition is required to join the tables.
- Self-join - Joins a table to itself.
- Outer join - Includes records of a table in output when there's no matching record in the other table.
- Cartesian join (also known as a Cartesian product or cross join) - Replicates each row from the first table with every row from the second table. Creates a join between tables by displaying every possible record combination.

1. Inner Join:

The INNER JOIN is one of the join clauses that allow you to query data from two or more related tables. The INNER JOIN clause combines each row from the first table with every row from the second table, keeping only the rows in which the join condition evaluates to true. The following shows the syntax of joining two tables using the INNER JOIN clause:

SELECT column list FROM Table1 INNER JOIN Table2 ON join condition;

In this syntax, the join_condition is a Boolean expression that evaluates to true, false, and unknown. Typically, it matches the values of the columns in the table Table1 with the values of the columns in the table Table2 using the equality operator (=).



class table

ID	NAME
1	abhi
2	adam
3	alex
4	anu

class_info table

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Result Table

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

Query: SELECT class.ID, class.Name, class_info.Address FROM class
INNER JOIN class_info **ON** class.ID=class_info.ID;

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "class" table that do not have matches in "class_info", these orders will not be shown.

2. Natural Join

- Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.
- **Natural Join: Guidelines**
  - The associated tables have one or more pairs of identically named columns.
  - The columns must be the same data type.
  - Don't use ON clause in a natural join.
- Natural Join Syntax : Select * from table1 NATURAL JOIN table2;

Example of Natural JOIN:

Query: SELECT * FROM ***class*** **NATURAL JOIN** ***class_info***;

In the above example, both the tables being joined have ID column(same name and same datatype), hence the records for which value of ID matches in both the tables will be the result of Natural Join of these two tables.

class table

ID	NAME
1	abhi
2	adam
3	alex
4	anu

class_info table

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Result Table

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

There are some limitations regarding the NATURAL JOIN. You cannot specify a LOB column with a NATURAL JOIN. Also, columns involved in the join cannot be qualified by a table name or alias.

Join with USING Clause

Using Natural joins, Oracle implicitly identify columns to form the basis of join. Many situations require explicit declaration of join conditions. In such cases, we use USING clause to specify the joining criteria. Since, USING clause joins the tables based on equality of columns, it is also known as Equijoin. They are also known as Inner joins or simple joins.

Syntax:

```
SELECT <column list>
FROM   TABLE1   JOIN   TABLE2
USING (column name)
```

Consider the below SELECT query, EMPLOYEES table and DEPARTMENTS table are joined using the common column DEPARTMENT_ID.

```
SELECT E.first_nameNAME,D.department_name DNAME
FROM employees E JOIN departments D
USING (department_id);
```

3. Self Join

A SELF-JOIN operation produces a result table when the relationship of interest exists among rows that are stored within a single table. In other words, when a table is joined to itself, the join is known as Self Join.

Consider EMPLOYEES table, which contains employee and their reporting managers. To find manager's name for an employee would require a join on the EMPLOYEES table itself. This is a typical candidate for Self Join.

```
SELECT e1.FirstNameManager,e2.FirstNameEmployee
FROM employees e1 JOIN employees e2
ON (e1.employee_id = e2.manager_id)
ORDER BY e2.manager_id DESC;
```

4. Outer Joins

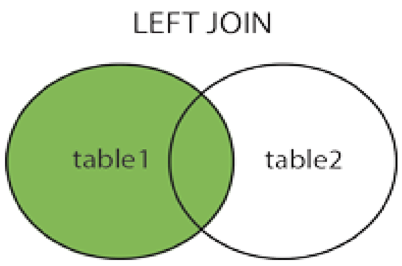
An Outer Join is used to identify situations where rows in one table do not match rows in a second table, even though the two tables are related.

- There are three types of outer joins: the LEFT, RIGHT, and FULL OUTER JOIN.
- They all begin with an INNER JOIN, and then they add back some of the rows that have been dropped.
- A LEFT OUTER JOIN adds back all the rows that are dropped from the first (left) table in the join condition, and output columns from the second (right) table are set to NULL.
- A RIGHT OUTER JOIN adds back all the rows that are dropped from the second (right) table in the join condition, and output columns from the first (left) table are set to NULL.
- The FULL OUTER JOIN adds back all the rows that are dropped from both the tables.

5. Left Outer Join

- A LEFT OUTER JOIN adds back all the rows that are dropped from the first (left) table in the join condition, and output columns from the second (right) table are set to NULL. The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.
- ***syntax***:

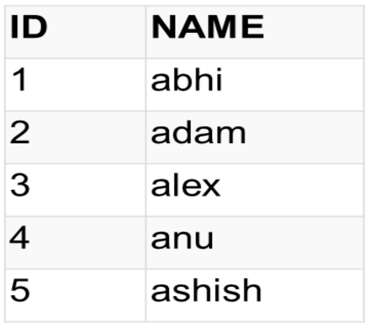
Select column-list from Table1 LEFT OUTER JION Table2 ON Table1.column-name = Table2.column-name;



Example of Left Outer JOIN:
Left Outer Join query will be:

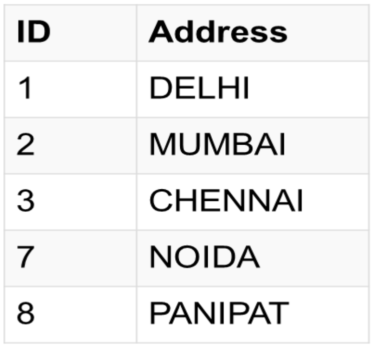
Select * from class LEFT OUTER JOIN class_info ON (class.ID =class_info.ID);

class table



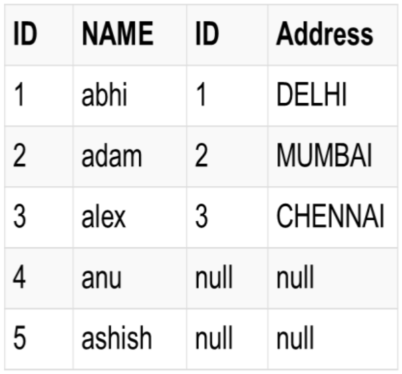
ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

class_info table



ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Result Table

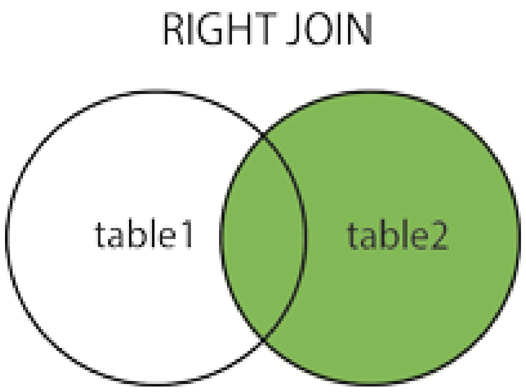


ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null

6. Right Outer Join

- A RIGHT OUTER JOIN adds back all the rows that are dropped from the second (right) table in the join condition, and output columns from the first (left) table are set to NULL.
- The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.
- ***syntax*:**

Select column-list from Table1 RIGHT OUTER JION Table2 ON Table1.column-name =Table2.column-name;



Example of Left Outer JOIN:

Right Outer Join query will be:

Select * from class RIGHT OUTER JOIN class_info ON (class.ID =class_info.ID);

class table

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

class_info table

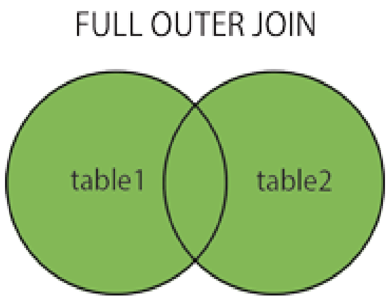
ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Result Table

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

7. Full Outer Join

- The FULL OUTER JOIN adds back all the rows that are dropped from both the tables.
- The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.
- ***syntax*:**
  Select column-list from Table1 FULL OUTER JION Table2 ON Table1.column-name =Table2.column-name;



Example of Left Outer JOIN:
Full Outer Join query will be:
Select * from class FULL OUTER JOIN class_info ON (class.ID =class_info.ID);

class table

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

class_info table

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Result Table

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
null	null	7	NOIDA
null	null	8	PANIPAT

8. Cartesian product or Cross join

- A cross join, also known as a Cartesian product join, returns a result table where each row from the first table is combined with each row from the second table.
- The number of rows in the result table is the product of the number of rows in each table. If the tables involved are large, this join can take a very long time.
- For two entities A and B, A * B is known as Cartesian product. A Cartesian product consists of all possible combinations of the rows from each of the tables. Therefore, when a table with 10 rows is joined with a table with 20 rows, the Cartesian product is 200 rows (10 * 20 = 200).For example, joining the employee table with eight rows and the department table with three rows will produce a Cartesian product table of 24 rows (8 * 3 = 24).
- A Cartesian product result table is normally not very useful. In fact, such a result table can be terribly misleading.

Example: Suppose that the following tables exist.

Table A

ACOL1	**ACOL2**
A1	AA1
A2	AA2
A3	AA3

ACOL1	**ACOL2**
BCOL1	**BCOL2**
B1	BB1
B2	BB2

Table B

The following two select statements produce identical results.

```
SELECT * FROM A CROSS JOIN B
SELECT * FROM A, B
```

The result table for either of these SELECT statements looks like this.

ACOL1	**ACOL2**	**BCOL1**	**BCOL2**
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2
A3	AA3	B1	BB1

ACOL1	ACOL2	BCOL1	BCOL2
A3	AA3	B2	BB2

Unions:

The UNION operator is used to combine the results of two or more SELECT queries into a single result set. The union operation is different from using joins that combine columns from two tables. The union operation creates a new table by placing all rows from two source tables into a single result table, placing the rows on top of one another. The UNION operation eliminates the duplicate rows from the combined result set, by default.

These are basic rules for combining the result sets of two SELECT queries by using UNION:

- The number and the order of the columns must be the same in all queries.
- The data types of the corresponding columns must be compatible.

When these criteria are met, the tables are *union-compatible*:

Syntax:

```
SELECT column_list FROM table1_name
UNION SELECT column_list FROM table2_name;
```

Example:

The **First** table

ID	Name
1	abhi
2	adam

The **second** table

ID	Name
2	adam
3	Chester

Union SQL query will be:

select * from First UNION select * from second;

The result will be

ID	NAME

1	abhi
2	adam
3	Chester

UNION ALL:

The SQL UNION ALL operator is used to combine the result sets of 2 or more SELECT statements. It does not remove duplicate rows between the various SELECT statements (all rows are returned). Each SELECT statement within the UNION ALL must have the same number of fields in the result sets with similar data types.

What is the difference between UNION and UNION ALL?

- UNION removes duplicate rows.
- UNION ALL does **not** remove duplicate rows.

Syntax: The syntax for the UNION ALL operator in SQL is:

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
UNION ALL
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

Note:

- There must be same number of expressions in both SELECT statements
- The corresponding expressions must have the same data type in the SELECT statements. For example: *expression1* must be the same data type in both the first and second SELECT statement.
- See also the UNION operator.

Example:

The **First** table

ID	Name
1	abhi
2	adam

The **second** table

ID	**Name**
2	adam
3	Chester

Union SQL query will be:

select * from First UNION ALL select * from second;

The result will be

ID	**NAME**
1	Abhi
2	Adam
2	Adam
3	Chester

Grouping:

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:

```
SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

Example of Group by in a Statement

Consider the following **Emp** table.

eid	**name**	**age**	**salary**

401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

If we want to find **name** and **age** of employees grouped by their **salaries** or in other words, we will be grouping employees based on their salaries, hence, as a result, we will get a data set, with unique salaries listed, along side the first employee's name and age to have that salary.

group by is used to group different row of data together based on any one column. SQL query for the above requirement will be,

SELECT name, age FROM Emp GROUP BY salary

Result will be,

name	**Age**
Rohan	34
Shane	29
Anu	22

Example of Group by in a Statement with WHERE clause

Consider the following **Emp** table

eid	**Name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000

404	Scott	44	9000
405	Tiger	35	8000

SQL query will be,

```
SELECT name, salary FROM Emp WHERE age > 25 GROUP BY salary;
```

Output:

name	Salary
Rohan	6000
Shane	8000
Scott	9000

We must remember that Group By clause will always come at the end of the SQL query, just like the Order by clause.

HAVING Clause:

We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we cannot use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

- HAVING filters records that work on summarized GROUP BY results.
- HAVING applies to summarized group records, whereas WHERE applies to individual records.
- Only the groups that meet the HAVING criteria will be returned.
- HAVING requires that a GROUP BY clause is present.
- WHERE and HAVING can be in the same query.

Syntax:

```
SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
```

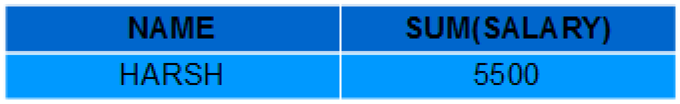
```
HAVING condition
ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

Example:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME
HAVING SUM(SALARY)>3000;
```

Output:



NAME	SUM(SALARY)
HARSH	5500

As you can see in the above output only one group out of the three groups appears in the result-set as it is the only group where sum of SALARY is greater than 3000. So we have used HAVING clause here to place this condition as the condition is required to be placed on groups not columns.

Example: List the number of customers in each country. Only include countries with more than 10 customers.

SELECT COUNT(Id), Country
FROM Customer
GROUP BY Country
HAVING COUNT(Id) > 10

Results: 3 records

Count	Country
11	France
11	Germany
13	USA

ORDER BY:

- SELECT returns records in no particular order.
- To ensure a specific order use the ORDER BY clause.
- ORDER BY allows sorting by one or more columns.

- Records can be returned in ascending or descending order.

The SQL ORDER BY syntax

SELECT column-names
FROM table-name
WHERE condition
ORDER BY column-names
Example - Sorting Results in Ascending Order

To sort your results in ascending order, you can specify the ASC attribute. If no value (ASC or DESC) is provided after a field in the ORDER BY clause, the sort order will default to ascending order. Let's explore this further.

In this example, we have a table called *customers* with the following data:

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com
6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

Enter the following SQL statement:

```
SELECT *
FROM customers
ORDER BY last_name;
```

There will be 6 records selected. These are the results that you should see:

customer_id	last_name	first_name	favorite_website
8000	Anderson	Paige	NULL
6000	Ferguson	Samantha	bigactivities.com
4000	Jackson	Joe	techonthenet.com
9000	Johnson	Derek	techonthenet.com
7000	Reynolds	Allen	checkyourmath.com

customer_id	last_name	first_name	favorite_website
5000	Smith	Jane	digminecraft.com

This example would return all records from the *customers* sorted by the *last_name* field in ascending order and would be equivalent to the following SQL ORDER BY clause:

```
SELECT *
FROM customers
ORDER BY last_name ASC;
```

Most programmers omit the ASC attribute if sorting in ascending order.

Example - Sorting Results in descending order

When sorting your result set in descending order, you use the DESC attribute in your ORDER BY clause. Let's take a closer look.

In this example, we have a table called *suppliers* with the following data:

supplier_id	supplier_name	city	state
100	Microsoft	Redmond	Washington
200	Google	Mountain View	California
300	Oracle	Redwood City	California
400	Kimberly-Clark	Irving	Texas
500	Tyson Foods	Springdale	Arkansas
600	SC Johnson	Racine	Wisconsin
700	Dole Food Company	Westlake Village	California
800	Flowers Foods	Thomasville	Georgia
900	Electronic Arts	Redwood City	California

Enter the following SQL statement:

```
SELECT *
FROM suppliers
WHERE supplier_id > 400
ORDER BY supplier_id DESC;
```

There will be 5 records selected. These are the results that you should see:

supplier_id	supplier_name	city	state
900	Electronic Arts	Redwood City	California
800	Flowers Foods	Thomasville	Georgia
700	Dole Food Company	Westlake Village	California
600	SC Johnson	Racine	Wisconsin
500	Tyson Foods	Springdale	Arkansas

This example would sort the result set by the *supplier_id* field in descending order.

Example - Using both ASC and DESC attributes

When sorting your result set using the SQL ORDER BY clause, you can use the ASC and DESC attributes in a single SELECT statement.

In this example, let's use the same *products* table as the previous example:

product_id	product_name	category_id
1	Pear	50
2	Banana	50
3	Orange	50
4	Apple	50
5	Bread	75
6	Sliced Ham	25
7	Kleenex	NULL

```
SELECT *
FROM products
WHERE product_id <> 7
ORDER BY category_id DESC, product_name ASC;
```

There will be 6 records selected. These are the results that you should see:

product_id	product_name	category_id
5	Bread	75

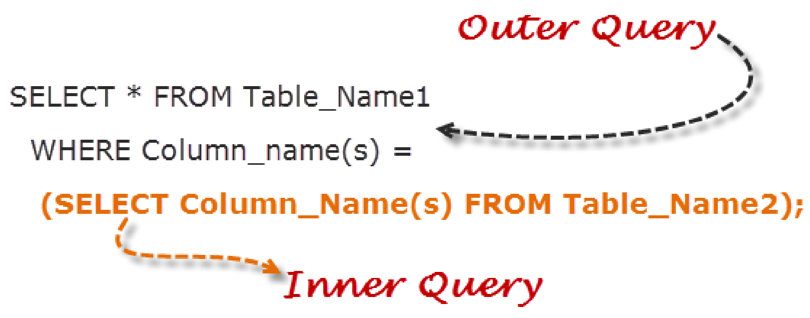
product_id	product_name	category_id
4	Apple	50
2	Banana	50
3	Orange	50
1	Pear	50
6	Sliced Ham	25

This example would return the records sorted by the *category_id* field in descending order, with a secondary sort by *product_name* in ascending order.

<u>Sub Queries</u>:

SQL Sub queries are the queries which are embedded inside another query. The embedded queries are called as **INNER** query & container query is called as **OUTER** query.
The subqueries are the queries which are executed inside of another query. The result SQL query totally depends on the result of a subquery. First, the INNER query gets executed & the result of an INNER query is passed as input to the outer query.

SQL Sub-Query Syntax:
Let's look at the basic syntax of the SQL Sub query command:



```
SELECT * FROM Table_Name1
  WHERE Column_name(s) =
  (SELECT Column_Name(s) FROM Table_Name2);
```

Three types of sub queries are supported in SQL are – Scalar, Row and Table sub queries.

- The **Scalar subquery** result returns only a single row and single column.
- The **Row subquery** result returns only a single row with single/multiple column(s).
- The **Table subquery** result returns can be return single/multiple row(s) or column(s).

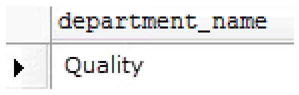
In the Sub query you may use the different operators to filter out the result like [=, >, =, <=, !=,]. These Sub queries can be used conjunction with INSERT, UPDATE and DELETE queries.

Suppose you want to find the name of the department in which employee_id = 100 is currently working on.

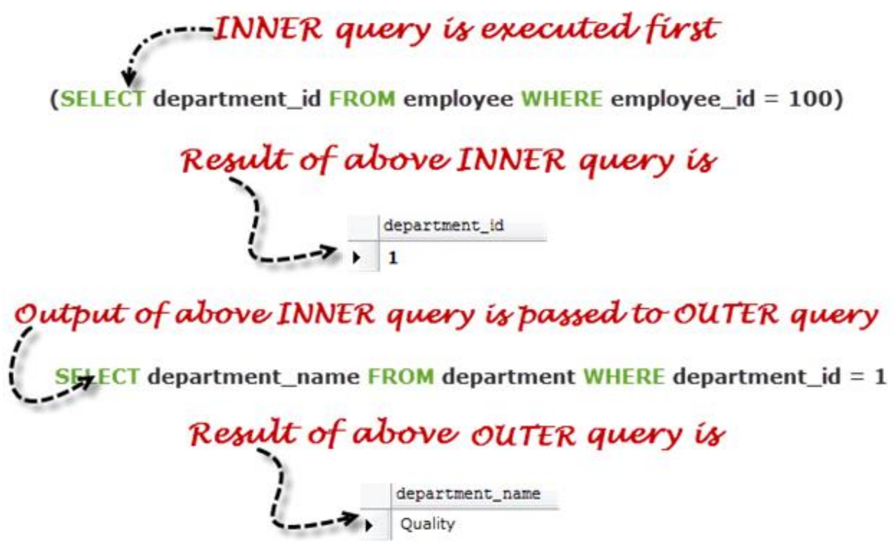
Let's see how this subquery is constructed & executed inside of another query:

SELECT department_name FROM department
WHERE department_id =
(SELECT department_id FROM employee WHERE employee_id = 100);

Following is the result upon executing the above SQL Sub query:



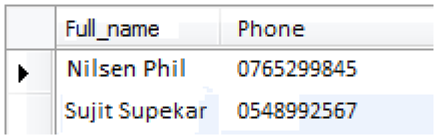
```
department_name
Quality
```


INNER query is executed first
(SELECT department_id FROM employee WHERE employee_id = 100)
Result of above INNER query is
department_id
1
Output of above INNER query is passed to OUTER query
SELECT department_name FROM department WHERE department_id = 1
Result of above OUTER query is
department_name
Quality

In above **Row Sub-Queries,** the result of INNER query can is returned only one value.
Let's take a look at the other Sub query type who returns can be return single/multiple row(s) or column(s) i.e. **Table sub-quer**y:
Suppose you want get list of employee's Name and Phone number who's working in other than Quality department & date of birth is not registered in Employee tracking system.

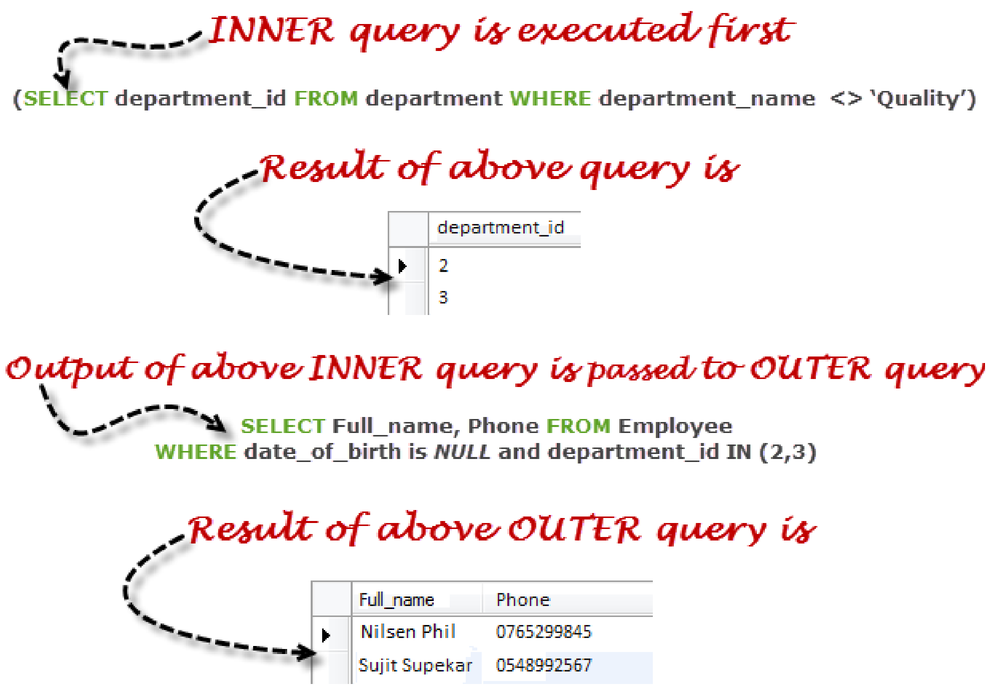
SELECT Full_name,Phone FROM Employee
WHERE date_of_birth is NULL and department_id IN
(SELECT department_id FROM department WHERE department_name <> 'Quality')

Following is the result upon executing the above SQL Sub query:



Full_name	Phone
Nilsen Phil	0765299845
Sujit Supekar	0548992567

So let’s discuss how the result of above query is calculated:


INNER query is executed first
Result of above query is
Output of above INNER query is passed to OUTER query
Result of above OUTER query is

```
(SELECT department_id FROM department WHERE department_name  <> 'Quality')
```

department_id
2
3

```
SELECT Full_name, Phone FROM Employee
WHERE date_of_birth is NULL and department_id IN (2,3)
```

Full_name	Phone
Nilsen Phil	0765299845
Sujit Supekar	0548992567

You can use multiple INNER queries inside INNER queries, the SQL supports INNER queries up to 32 levels.

In above examples we have seen INNER queries up to two levels; here we are seeing three level INNER query:

Sub queries contain two parts, one is INNER query & other is OUTER query. The result of INNER query is passed to OUTER query as input.

- Sub queries are simple & easy to understand. It can be easily broken down into logical steps, so it offers more flexibility.
- The Sub queries are used in conjunction with SELECT, INSERT, UPDATE & DELETE commands.
- In this article we have learnt about three types of SQL supb queries: scalar, row and table sub queries.
- In SQL server, The Nested query can be used up to 32 levels.
- As compare with Joins, the performance of Sub query is low. Joins are 500 times faster than Sub queries.

For performance issues, when it comes to getting data from multiple tables, it is strongly recommended to use JOINs instead of sub queries. Sub queries should only be used with good reason. So in the next article I am covering basics of Joins & what all types of Joins offered in the SQL server.

DB2 Functions and Procedures:
We can use functions and expressions to control the appearance and values of rows and columns in your result tables. DB2® offers many built-in functions, including aggregate functions and scalar functions.

Scalar Functions:

- A scalar function can be used wherever an expression can be used.
- The restrictions on the use of aggregate functions do not apply to scalar functions, because a scalar function is applied to single set of parameter values rather than two sets of values.
- The argument of a scalar function can be a function. However, the restrictions that apply to the use of expressions and aggregate functions also apply when an expression or aggregate function is used within a scalar function.
- As opposed to aggregate functions, scalar functions operate against a single value and return a single value.
- Scalar functions can be placed anywhere but Aggregate functions can be placed only in select statement or with the clause having.

Example: The following SELECT statement calls for the employee number, last name, and age of each employee in department D11 in the sample table DSN8A10.EMP. To obtain the ages, the scalar function YEAR is applied to the expression:

ABSVAL or ABS	Converts a value of any numeric data type to its absolute value.
ACOS	Returns the arc-cosine of the argument as an angle expressed in radians.
BLOB	Converts a string or ROWID data type into a value of data type BLOB.
CEILING or CEIL	Converts the argument, represented as any numeric data type, to the smallest integer value greater than or equal to the argument value.
CHAR	Converts a DB2 date, time, timestamp, ROWID, floating point, integer, or decimal value to a character value. For example `SELECT CHAR(HIREDATE, USA)`

	FROM DSN8810.EMP
	WHERE EMPNO = '000140';

CLOB	Converts a string or ROWID data type into a value of data type CLOB.
CONCAT	Converts two strings into the concatenation of the two strings.
DATE	Converts a value representing a date to a DB2 date. The value to be converted can be a DB2 timestamp, a DB2 date, a positive integer, or a character string.
DAY	Returns the day portion of a DB2 date or timestamp.
DAYOFMONTH	Similar to DAY except DAYOFMONTH cannot accept a date duration or time duration as an argument.
DAYOFWEEK	Converts a date, timestamp, or string representation of a date or timestamp into an integer that represents the day of the week. The value 1 represents Sunday, 2 Monday, 3Tuesday, 4 Wednesday, 5 Thursday, 6 Friday, and 7Saturday.
DAYS	Converts a DB2 date or timestamp into an integer value representing one more than the number of days since January 1, 0001.
DBCLOB	Converts a string or ROWID data type into a value of data type DBCLOB.
DECIMAL or DEC	Converts any numeric value, or character representation of a numeric value, to a decimal value.
EXP	Returns the exponential function of the numeric argument. The EXP and LOG functions are inverse operations.
FLOOR	Converts the argument, represented as any numeric data type, to the largest integer value less than or equal to the argument value.
GRAPHIC	Converts a string data type into a value of data type GRAPHIC.
GREATEST	Returns the maximum value in a supplied set of values. The argument values can be of any built-in data type other than CLOB, DBCLOB, BLOB, or ROWID.
HEX	Converts any value other than a long string to hexadecimal.

HOUR	Returns the hour portion of a time, a timestamp, or a duration.

IFNULL	-Returns the first argument in a set of two arguments that is not null. For example
	`SELECT EMPNO, IFNULL(WORKDEPT, 'N/A')`
	`FROM  DSN8810.EMP;`

	This SQL statement returns the value for WORKDEPT for all employees, unless WORKDEPT is null, in which case it returns the string 'N/A'.

LAST_DAY	Returns the last day of the month for the specified DB2 date or timestamp, or character representation of a date or timestamp.
LEAST	Returns the minimum value in a supplied set of values. The argument values can be of any built-in data type other than CLOB, DBCLOB, BLOB, or ROWID.

LEFT	Returns a string containing only the leftmost characters of the string in the first argument, starting at the position indicated by the second argument. For example
	`SELECT LEFT('THIS IS RETURNED', 4)`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement returns 'THIS', which is the four leftmost characters of the first argument.
LENGTH	Returns the length of any column, which may be null. Does not include the length of null indicators or variable character-length control values, but does include trailing blanks for character columns.

LOCATE	Returns the position of the first occurrence of the first string the second string. For example
	`SELECT LOCATE('I', 'CRAIG MULLINS')`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement returns the value 4, because the value 'I' first appears in position four within the searched string. It also appears in the

	11th position, but that is of no concern to the LOCATE function. Optionally, a third argument can be supplied indicating where the search should start. For example
	`SELECT LOCATE('I', 'CRAIG MULLINS', 7)`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement returns the value 11, because after position 7, the value 'I' first appears in the 11th position. When the third argument is not specified, LOCATE defaults to the beginning of the second string.
LOG or LN	Returns the natural logarithm of the numeric argument. The EXP and LOG functions are inverse operations.
LOG10	Returns the base 10 logarithm of the numeric argument.
LOWER or LCASE	Converts a character string into all lowercase characters.
LTRIM	Removes the leading blanks from a character string.
MICROSECOND	Returns the microsecond component of a timestamp or the character representation of a timestamp.
MINUTE	Returns the minute portion of a time, a timestamp, a character representation of a time or timestamp, or a duration.
MONTH	Returns the month portion of a date, a timestamp, a character representation of a date or timestamp, or a duration.

NEXT_DAY	Returns a timestamp indicating the first day of the week as specified in the second argument that is later than the date expression specified in the first argument. Valid values for the second argument are text representations of the days of the week; that is, MONDAY, TUESDAY, and so on. For example
	`SELECT NEXT_DAY(CURRENT DATE, 'FRIDAY')`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement returns a timestamp specifying the first Friday after today.
NULLIF	Returns a null when two specified expressions are equal; if not equal, the first expression is returned.

	This SQL statement returns the value 10; the value 'ADMIN'first appears in the 10th position.
POWER	Returns the value of the first argument raised to the power of the second argument.
RADIANS	Returns the number of radians for the numeric argument expressed in degrees.

REPLACE	Returns a character string with the value of the second argument replaced by each instance of the third argument in the first argument. For example
	`SELECT REPLACE('BATATA', 'TA', 'NA')`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement replaces all instances of 'TA' with 'NA'changing the character string 'BATATA' into 'BANANA'.

RIGHT	Returns a string containing only the rightmost characters of the string in the first argument, starting at the position indicated by the second argument. For example
	`SELECT RIGHT('RETURN ONLY THIS', 4)`
	`FROM  SYSIBM.SYSDUMMY1;`

	This SQL statement returns 'THIS', which is the four rightmost characters of the first argument.
ROUND	Rounds the first numeric argument to the number of places specified in the second argument.
RTRIM	Removes the trailing blanks from a character string.
SPACE	Returns a string of blanks whose length is specified by the numeric argument. The string of blanks is an SBCS character string.
SQRT	Returns the square root of the numeric argument.
SUBSTR	Returns the specified portion of a character column from any starting point to any ending point.

TRUNCATE or TRUNC	Converts the first numeric argument by truncating it to the right of the decimal place by the integer number specified in the second numeric argument. For example
	`SELECT TRUNC(3.014015,2)`
	`FROM  SYSIBM.SYSDUMMY1;`

UPPER or UCASE	Converts a character string into all uppercase characters.
VARCHAR	Converts a character string, date, time, timestamp, integer, decimal, floating point, or ROWID value into a corresponding variable character string representation.
VARGRAPHIC	Converts a character string to a graphic string.
WEEK	Returns an integer between 1 and 54 based on the week of the year in which a date, timestamp, or string representation of a date or timestamp falls. The assumption is that a week begins on Sunday and ends on Saturday. The value 1 represents the first week, 2 the second week, and so on.
YEAR	Returns the year portion of a date, a timestamp, or a duration.

Column Functions:

Column functions compute, from a group of rows, a single value for a designated column or expression. This provides the capability to aggregate data, thereby enabling you to perform statistical calculations across many rows with one SQL statement. To fully appreciate the column functions, you must understand SQL's set-level processing capabilities.

This list shows some rules for the column functions:

- Column functions can be executed only in SELECT statements.
- A column function must be specified for an explicitly named column or expression.
- Each column function returns only one value for the set of selected rows.
- If you apply a column function to one column in a SELECT statement, you must apply column functions to any other columns specified in the same SELECT statement, unless you also use the GROUP BY clause.
- Use GROUP BY to apply a column function to a group of named columns. Any other column named in the SELECT statement must be operated on by a column function.

- The result of any column function (except the COUNT and COUNT_BIG functions) will have the same data type as the column to which it was applied. The COUNT function returns an integer number; COUNT_BIG returns a decimal number.
- The result of any column function (except the COUNT and COUNT_BIG functions) can be null. COUNT and COUNT_BIG always return a numeric result.
- Columns functions will not return a SQLCODE of +100 if the predicate specified in the WHEREclause finds no data. Instead, a null is returned. For example, consider the following SQL statement:

```
SELECT  MAX(SALARY)
FROM   DSN8810.EMP
WHERE  EMPNO = '999999';
```

- There is no employee with an EMPNO of '999999' in the DSN8810.EMP table. This statement therefore returns a null for the MAX(SALARY). Of course, this does not apply to COUNT and COUNT_BIG, both of which always return a value, never a null.
- When using the AVG, MAX, MIN, STDDEV, SUM, and VARIANCE functions on nullable columns, all occurrences of null are eliminated before applying the function.
- You can use the DISTINCT keyword with all column functions to eliminate duplicates before applying the given function. DISTINCT has no effect, however, on the MAX and MIN functions.
- You can use the ALL keyword to indicate that duplicates should not be eliminated. ALL is the default.

A column function can be specified in a WHERE clause only if that clause is part of a subquery of a HAVING clause. Additionally, every column name specified in the expression of the column function must be a correlated reference to the same group.

The column functions are:

AVG, COUNT, COUNT_BIG, MAX, MIN, STDDEV, SUM, and VARIANCE.

AVG Function:

The AVG function computes the average of the values for the column or expression specified as an argument. This function operates only on numeric arguments. The following example calculates the average salary of each department:

```
SELECT  WORKDEPT, AVG(SALARY)
FROM   DSN8810.EMP
GROUP BY WORKDEPT;
```

The AVG function is the preferred method of calculating the average of a group of values. Although an average, in theory, is nothing more than a sum divided by a count, DB2 may not return equivalent values for AVG(COL_NAME) and SUM(COL_NAME)/COUNT(*). The reason is that the COUNT function will count all rows regardless of value, whereas SUM ignores nulls.

The COUNT Function

The COUNT function counts the number of rows in a table, or the number of distinct values for a given column. It can operate, therefore, at the column or row level. The syntax differs for each. To count the number of rows in the EMP table, issue this SQL statement:

```
SELECT COUNT (*)
FROM   DSN8810.EMP;
```

It does not matter what values are stored in the rows being counted. DB2 will simply count the number of rows and return the result. To count the number of distinct departments represented in the EMP table, issue the following

```
SELECT COUNT (DISTINCT WORKDEPT)
FROM   DSN8810.EMP;
```

The keyword DISTINCT is not considered an argument of the function. It simply specifies an operation to be performed before the function is applied. When DISTINCT is coded, duplicate values are eliminated.
If DISTINCT is not specified, then ALL is implicitly specified. ALL also can be explicitly specified in the COUNT function. When ALL is specified, duplicate values are not eliminated.

Note: The argument of the COUNT function can be of any built-in data type other than a large object: CLOB, DBCLOB, or BLOB. Character string arguments can be no longer 255 bytes and graphic string arguments can be no longer than 127 bytes. The result of the COUNT function cannot be null. COUNT always returns an INTEGER value greater than or equal to zero.

The COUNT_BIG Function

The COUNT_BIG function is similar to the COUNT function. It counts the number of rows in a table, or the number of distinct values for a given column. However, the COUNT_BIG function returns a result of data type DECIMAL(31,0), whereas COUNT can return a result only as large as the largest DB2 integer value, namely +2,147,483,647.

The COUNT_BIG function works the same as the COUNT function, except it returns a decimal value. Therefore, the example SQL for COUNT is applicable to COUNT_BIG. Simply substitute COUNT_BIG for COUNT. For example, the following statement counts the number of rows in the EMP table (returning a decimal value, instead of an integer):

```
SELECT  COUNT_BIG(*)
FROM   DSN8810.EMP;
```

NOTE: The COUNT_BIG function has the same restrictions as the COUNT function. The argument of the COUNT_BIG function can be of any built-in data type other than a large object: CLOB, DBCLOB, or BLOB. Character string arguments can be no longer than 255 bytes and graphic string arguments can be no longer than 127 bytes. The result of the COUNT_BIG function cannot be null. COUNT_BIG returns a decimal value greater than or equal to zero.

The MAX Function:

The MAX function returns the largest value in the specified column or expression. The following SQL statement determines the project with the latest end date:

```
SELECT MAX(ACENDATE)
FROM   DSN8810.PROJACT;
```

NOTE: The result of the MAX function is of the same data type as the column or expression on which it operates. The argument of the MAX function can be of any built-in data type other than a large object: CLOB, DBCLOB, or BLOB. Character string arguments can be no longer than 255 bytes and graphic string arguments can be no longer than 127 bytes.

A somewhat more complicated example using MAX is shown below. It returns the largest salary paid to a man in department D01:

```
SELECT MAX(SALARY)
FROM  DSN8810.EMP
WHERE  WORKDEPT = 'D01'
AND   SEX = 'M';
```

The MIN Function

The MIN function returns the smallest value in the specified column or expression. To retrieve the smallest bonus given to any employee, issue this SQL statement:

```
SELECT  MIN(BONUS)
FROM   DSN8810.EMP;
```

NOTE: The result of the MIN function is of the same data type as the column or expression on which it operates. The argument of the MIN function can be of any built-in data type other than a large object: CLOB, DBCLOB, or BLOB. Character string arguments can be no longer than 255 bytes and graphic string arguments can be no longer than 127 bytes.

The STDDEV Function

The STDDEV function returns the standard deviation of a set of numbers. The standard deviation is calculated at the square root of the variance. For example:

```
SELECT STDDEV(SALARY)
FROM  DSN8810.EMP
WHERE WORKDEPT = 'D01';
```

NOTE: The argument of the STDDEV function can be any built-in numeric data type. The resulting standard deviation is a double precision floating-point number.

The SUM Function:

The accumulated total of all values in the specified column or expression are returned by the SUM column function. For example, the following SQL statement calculates the total yearly monetary output for the corporation:

```
SELECT SUM(SALARY+COMM+BONUS)
FROM   DSN8810.EMP;
```

This SQL statement adds each employee's salary, commission, and bonus. It then aggregates these results into a single value representing the total amount of compensation paid to all employees.

NOTE: The argument of the SUM function can be any built-in numeric data type. The resulting sum must be within the range of acceptable values for the data type. For example, the sum of an INTEGER column must be within the range –2,147,483,648 to +2,147,483,647. This is because the data type of the result is the same as the data type of the argument values, except:

- The sum of SMALLINT values returns an INTEGER result.
- The sum of single precision floating point values returns a double precision floating-point result.

The VARIANCE Function

The VARIANCE function returns the variance of a set of numbers. The result is the biased variance of the set of numbers. The variance is calculated as follows:

```
VARIANCE = SUM(X**2)/COUNT(X) - (SUM(X)/COUNT(X))**2
```

NOTE: The argument of the VARIANCE function can be any built-in numeric data type. The resulting variance is a double precision floating-point number. For brevity and ease of coding, VARIANCE can be shortened to VAR.

Row Functions:

A row function can be used only in contexts that are specifically described for the function.

UNPACK

The UNPACK function returns a row of values that are derived from unpacking the input binary string. It is used to unpack a string that was encoded according to the PACK function.

```
>>-UNPACK-- (--expression--) ------------------------------------><
```

The schema is SYSIBM.

expression

An expression that returns the string value to be unpacked. The *expression* must be a binary string that is not a BLOB and that is not null. The format of the binary string must match the one that is produced by the PACK function.

The UNPACK function can only be specified in the SELECT list.

The result of the function is a row of fields corresponding to the data elements that were encoded in the input packed string. The result is not null.

Example: Assume that a user-defined function named myUDF returns a VARBINARY result. The body of the function includes the following invocation of the PACK function to pack some data into a binary string:

```
 SET :udf_result = PACK(CCSID 1208, 'Alina', DATE('1977-08-01'),
                DOUBLE(0.5));
```

The following SELECT statement unpacks the result of the myUDF function and returns a row of individual column values:

```
 SELECT UNPACK(myUDF(C1)).* AS(Name VARCHAR(40) CCSID UNICODE,
                     DOB DATE,
                     Score DOUBLE)
FROM T1;
```

The use of ".*" indicates that the result of the UNPACK function should be flattened into a list of result column values. When the UNPACK function is used in a select clause, an AS clause is specified to provide the names and data types for the resulting values.

UNIT V
DB2 Storage: Backup & Recovery

DB2 Backup & Recovery: DB2 Logging, DB2 Backup: Taking Backup from Control Center, Table space backup, Online and Offline Backups, Incremental and Delta backup, Database recovery using Control Center.
DB2 Utilities on Linux / UNIX: Import Utility, Export Utility, Load Utility, db2move Utility
UNLOAD, LOAD, COPY, RECOVERY, REORG, RUNSTATS, STOSPACE
Basic Concepts of OLAP & Data Warehousing, Data Migration (DB2/Oracle/MS SQL/Sybase)

<u>DB2 Backup & Recovery Concepts</u>:

Recovery Scenarios:

To prevent the loss of your data, you need to have a recovery strategy, ensure that it works, and consistently practice and follow it. The following are some recovery scenarios you should consider.

- **System outage:** A power failure, hardware failure, or software failure can cause your database to be in an inconsistent state.
- **Transaction failure:** Users may inadvertently corrupt your database by modifying it with incorrect data or deleting useful data.
- **Media failure**: If your disk drive becomes unusable, you may lose all or part of your data.
- **Disaster**: The facility where your database server is located may be damaged by fire, flooding, power loss, or other catastrophe.

Unit of Work (Transaction):

A **unit of work** (UOW), also known as a **transaction,** consists of one or more SQL statements that end with a **COMMIT** or **ROLLBACK** statement. All of the statements inside this UOW are treated as a complete entity, which ensures data and transactional consistency.

Types of Recovery:

There are three types of recovery in DB2:

- Crash recovery
- Version recovery
- Roll forward recovery

1. **Crash Recovery - Crash** recovery protects a database from being left in an inconsistent state following an abnormal termination. An example of an abnormal termination is a power failure. Using the banking example above, if a power failure occurred after the update statements, but prior to the COMMIT statement, the next time DB2 is restarted and the database accessed, DB2 will ROLLBACK the UPDATE statements. Note that statements are rolled back in the reverse order that **they were performed originally.** This

ensures that the data is consistent, and that the person still has the $100 in his or her savings account.

2. **Version Recovery** - Version recovery allows you to restore a snapshot of the database taken at a point in time using the BACKUP DATABASE command.

   The restored database will be in the same state it was in when the BACKUP command completed. If further activity was performed against the database after this backup was taken, those updates are lost. For example, assume you back up a database and then create two tables, *table1* and *table2*. If you restore the database using the backup image, your restored database will not contain these two tables.

3. **Roll Forward Recovery** - Roll forward recovery extends version recovery by using full database and table space backups in conjunction with the database log files. A backup must be restored first as a baseline, and then the logs are reapplied to this backup image. Therefore, all committed changes you made *after* you backed up the database can be applied to the restored database.

DB2 Transaction Logs:

DB2 uses transaction logs to record all changes to your database so that they can be rolled back if you issue the ROLLBACK command, reapplied or rolled back in the event that you need to restore a database backup, or during crash recovery.

Log File States:

The state of a log is determined by whether the transactions that are recorded in it have been committed and whether or not they have been externalized to disk. There are three log file states: active, online archive, and offline archive.

1. **Active Logs**

   A log is considered **active** if any of the following applies:

   - It contains transactions that have not yet been committed or rolled back.
   - It contains transactions that have been committed but whose changes have not yet been written to the database disk (externalized).
   - It contains transactions that have been rolled back but whose changes have been written to the database disk (externalized).

2. **Online Archive Logs**

   Online archive logs are files that contain only committed, externalized transactions. In other words, they are logs that are no longer active, and therefore no longer needed for crash recovery.

   Online archive logs still reside in the active log directory. This is why they are called "online." The term *online archive logs* may sound complicated, but all it means is that inactive logs reside in the active log directory.

3. Offline Archive Logs

File systems and disk drives have limited space. If all of the online archive logs are kept in the active log directory, this directory will soon be filled up, causing a log disk full condition. Therefore, the online archive logs should be moved out of the active log directory as soon as possible. You can do this manually, or DB2 can invoke a program or procedure to do this for you. Once this has been done, these logs become **offline archive logs.**

Database Logging:

Database logging is an important part of your highly available database solution design because database logs make it possible to recover from a failure, and they make it possible to synchronize primary and secondary databases. All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

Logging Methods:

DB2 supports three logging methods: circular logging, archival logging, and infinite active logging.

1. Circular Logging

Circular logging is the default logging mode for DB2. As the name suggests, in this method the logs are reused in a circular mode. For example, if you have three primary logs, DB2 uses them in this order: Log #1, Log #2, Log #3, Log #1, Log #2….

Note that in the above sequence Log #1 and Log #2 are reused. When a log file is reused, its previous contents are completely overwritten. Therefore, a log can be reused if and only if the transactions it contains have already been committed or rolled back and externalized to the database disk. In other words, the log must not be an active log. This ensures DB2 will have the necessary logs for crash recovery if needed. Figure 14.3 shows how circular logging works.

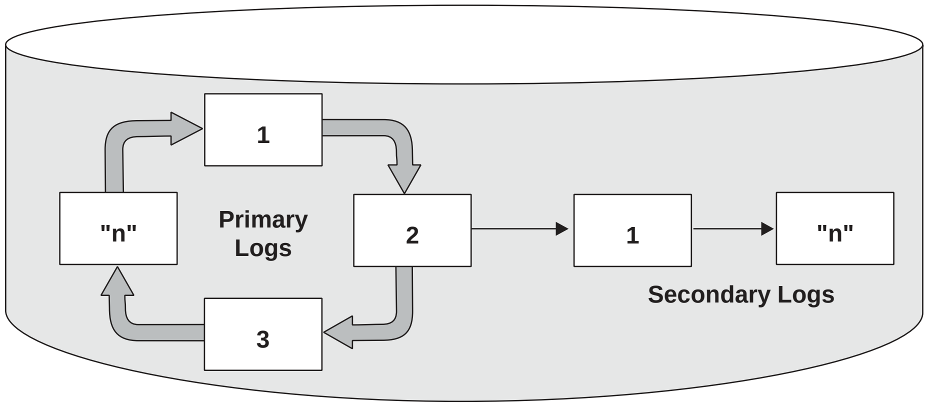


Figure 14.3 Circular logging

Although the ability to recover from a crash is assured, you cannot reapply the transactions that were in these logs, because they have been overwritten. Therefore, circular logging only supports crash recovery and version recovery, not roll forward recovery.

2. Archival Logging

Archival logging keeps the log files even after they contain committed and externalized data. To enable archival logging, you can change the value of the LOGARCHMETH1 database configuration parameter. We will discuss the possible values for the LOGARCHMETH1 parameter later in this section.

With archival logging, roll forward recovery is supported. The contents of inactive logs are saved rather than overwritten; therefore, they can be reapplied during roll forward recovery. Depending on the value set in LOGARCHMETH1, you can have the log files copied or saved to various locations. When the log is needed during roll forward recovery, DB2 retrieves it from that location and restores it into the active log directory.

With archival logging, if you have three primary logs in a database, DB2 will allocate them in this order: Use Log #1, use Log #2, use Log #3, archive Log #1 (when no longer active), create and use Log #4, archive Log #2, create and use Log #5…. Notice that the log number increases as new logs are required. Figure 14.4 shows how archival logging works.

How DB2 archives and retrieves a log file depends on the value set in the LOGARCHMETH1 database parameter. The possible values are OFF, LOGRETAIN, USEREXIT, TSM, and VENDOR and are discussed in detail in Table 14.2.

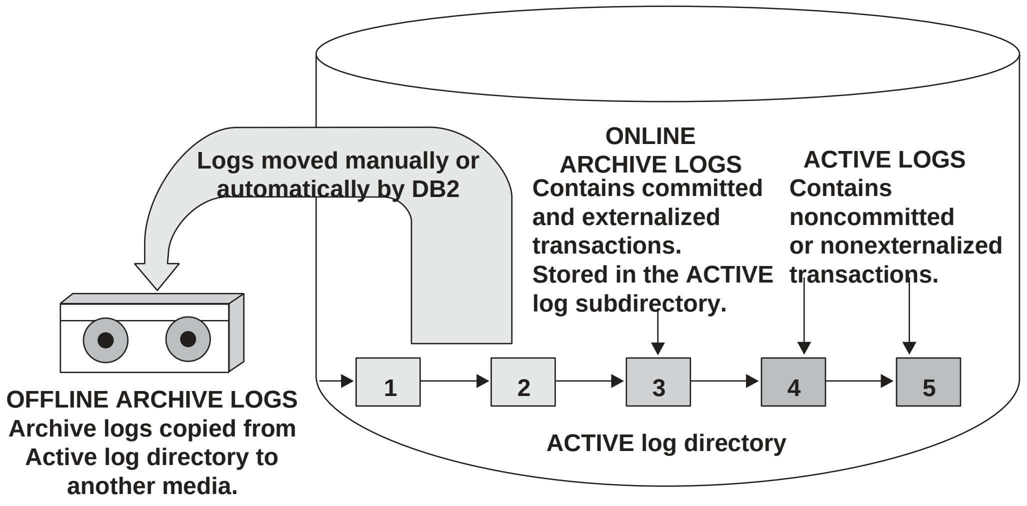

Logs moved manually or automatically by DB2
ONLINE ARCHIVE LOGS
Contains committed and externalized transactions. Stored in the ACTIVE log subdirectory.
ACTIVE LOGS
Contains noncommitted or nonexternalized transactions.
1
2
3
4
5
ACTIVE log directory
OFFLINE ARCHIVE LOGS
Archive logs copied from Active log directory to another media.

Figure 14.4 Archival logging

Table 14.2 Additional Logging-Related Database Configuration Parameters

LOGARCHMETH1 Values	Description/Usage
OFF	Archival logging is disabled and circular logging is used.
LOGRETAIN	The log files will be retained in the active log directory.

(continues)

Table 14.2 Additional Logging-Related Database Configuration Parameters *(Continued)*

LOGARCHMETH1 Values	Description/Usage
USEREXIT	The archive and retrieval of the logs are performed automatically by a user-supplied user exit program called **db2uext2.**
DISK:directory	With this setting, archival logging uses a similar algorithm as in USEREXIT. The only difference is instead of calling the user exit program, DB2 will automatically archive the logs from the active log directory to the specified *directory*. During recovery, DB2 will automatically retrieve these logs back to the active log directory.
TSM:[management class name]	With this setting archival logging uses a similar algorithm as in USEREXIT. The only difference is that the logs will be archived on the local Tivoli Storage Manger (TSM) server. The management class name parameter is optional. If not specified, the default management class is used.
VENDOR:*library*	With this setting, archival logging uses a similar algorithm as in USEREXIT. The only difference is that logs are archived using the specified vendor library.

You can optionally configure the LOGARCHMETH2 parameter. This parameter specifies the secondary archive log method, and can be set using the same values as for LOGARCHMETH1. If set, logs will be archived to both this destination and the destination specified by the LOGARCHMETH1 parameter.

In addition to LOGARCHMETH1 and LOGARCHMETH2, Table 14.3 lists a number of other logging-related database parameters.

3. Infinite Active Logging

Infinite active logging is built on top of archival logging. With circular logging and archival logging, log space can potentially be filled with active logs if you have very long running transactions. If you have long-running transactions you can use infinite active logging so that you do not run out of log, or log disk, space.

To enable infinite active logging:

- Archive logging must be enabled with one of the automatic archival methods; that is, LOGARCHMETH1 must be set to one of USEREXIT, DISK, TSM, or VENDOR.
- Set the LOGSECOND database configuration parameter to –1.

When archival logging is enabled, a log is marked for archival as soon as it becomes full. However, DB2 leaves the log in the log directory until it becomes inactive for performance reasons, and then renames the file for reuse. With infinite logging, DB2 still archives the log as soon as it is full, but it does not wait for it to become inactive before it renames the file for reuse. This guarantees that the active log directory will never fill up, because any logs can be reused once they are filled and archived. Note that the use of infinite active logging can prolong crash recovery times as active logs may need to be retrieved from the archive site.

4. Log Mirroring:

Even with all the protection provided by DB2 logging, there is still a concern of someone accidentally deleting an active log file, or a disk crash that causes data corruption in your database. Mirroring the log files helps protect your database from these potential disasters. Log mirroring allows you to specify a secondary path for the database to manage copies of the active logs. DB2 will attempt to write the log buffer to the log files in both places. When one log path is damaged for whatever reason, DB2 can continue to read and write to the other log path, keeping the database up and running.

To enable log mirroring, set the MIRRORLOGPATH database configuration parameter to a valid drive, path, or device.

Database Backup:

A **database backup** is a complete copy of your database objects. In addition to the data, a backup copy contains information about the table spaces, containers, the system catalog, database configuration file, the log control file, and the recovery history file. Note that a backup does *not* contain the Database Manager Configuration file or the values of registry variables.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to perform a backup.

To perform an offline backup of the sample database and store the backup copy in the directory d:\mybackups, use the following command on Windows.

The d:\mybackups directory must be created before the backup can be performed. To perform an offline backup of the sample database and store the backup copy in two separate directories, use the following command for Linux/UNIX shown in Figure 14.6:

```
BACKUP DATABASE sample        (1) TO
/db2backup/dir1, /db2backup/dir2   (2)
WITH 4 BUFFERS  (3) BUFFER 4096              (4)
PARALLELISM 2                       (5)
```

Figure 14.6 Backup Database command example

where:

(1) Indicates the name (or alias) of the database to back up.
(2) Specifies the location(s) where you want to store the backup file. DB2 will writeto both locations in parallel.
(3) Indicates how many buffers from memory can be used during the backup operation. Using more than one buffer can improve performance.

(4) Indicates the size of each buffer in 4KB pages.
(5) Specifies how many media reader and writer threads are used to take the backup.

Table Space Backup:

In a database where only some of your table spaces change considerably, you may opt not to back up the entire database but only specific table spaces. To perform a table space backup, you can use the following syntax:

```
BACKUP DATABASE sample
TABLESPACE (syscatspace, userspace1, userspace2)
ONLINE
TO /db2tbsp/backup1, /db2tbsp/backup2
```

The keyword TABLESPACE indicates this is a table space backup, not a full database backup. You can also see from the example that you can include as many table spaces as desired in the backup. Temporary table spaces cannot be backed up using a table space level backup.

You should back up related table spaces together. For example, if using DMS table spaces where one table space is used for the table data, another one for the indexes, and another one for LOBs, you should back up all of these table spaces at the same time so that you have consistent information. This is also true for table spaces containing tables defined with referential constraints between them.

Incremental Backups:

As database sizes continue to grow, the time and resources required to back up and recover these databases also grows substantially. Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous.

To address this issue, DB2 provides incremental backups. An **incremental backup** is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database metadata (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

There are two kinds of incremental backups.

- In incremental **cumulative** backups, DB2 backs up all of the data that has changed since the last full database backup.
- In **delta** backups, DB2 backs up only the data that has changed since the last successful full, cumulative, or delta backup.

Figure 14.7 illustrates these concepts.

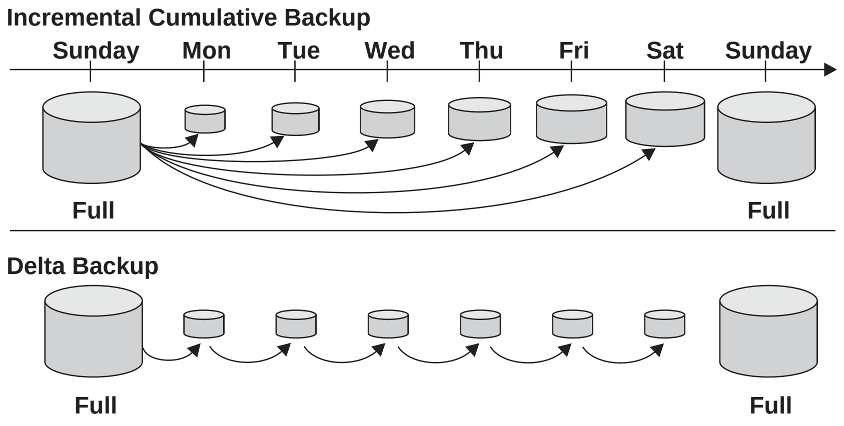


Figure 14.7 Incremental and delta backups

For incremental backups, if there was a crash after the incremental backup on Friday, you would restore the first Sunday's full backup, followed by the incremental backup taken on Friday.

For delta backups, if there was a crash after the delta backup on Friday, you would restore the first Sunday's full backup, followed by each of the delta backups taken on Monday through Friday inclusive.

To enable incremental and delta backups, the TRACKMOD database configuration parameter must be set to YES. This allows DB2 to track database modifications so that the backup utility can detect which database pages must be included in the backup image. After setting this parameter to YES, you must take a full database backup to have a baseline against which incremental backups can be taken.

To perform a cumulative incremental backup on the SAMPLE database to directory /dev/rdir1, issue:

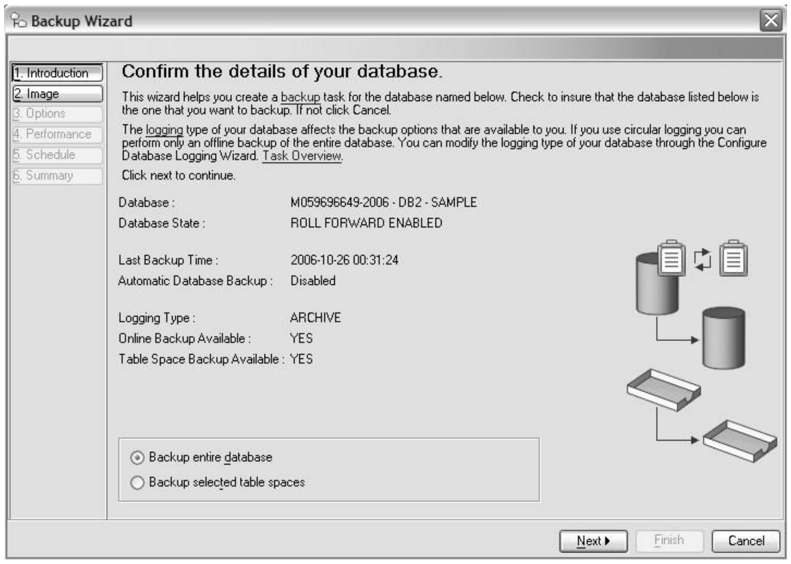
```
BACKUP DB sample
INCREMENTAL TO /dev/rdir1
```

To perform a delta backup on the SAMPLE database to the directory /dev/rdir1, issue:

```
BACKUP DB sample
INCREMENTAL DELTA TO /dev/rdir1
```

Backing Up a Database with the Control Center:

You can use the Backup Wizard to perform backups. From the Control Center, expand your database folder, right-click on the database name you wish to back up and select **Backup.** The database Backup Wizard appears. Figure 14.8 shows that you can choose to perform either a database-level backup or a table space-level backup. From here, the Backup Wizard will guide you through the options.



Confirm the details of your database.

This wizard helps you create a backup task for the database named below. Check to insure that the database listed below is the one that you want to backup. If not click Cancel.

The logging type of your database affects the backup options that are available to you. If you use circular logging you can perform only an offline backup of the entire database. You can modify the logging type of your database through the Configure Database Logging Wizard. Task Overview.

Click next to continue.

Database :	M059696649-2006 - DB2 - SAMPLE
Database State :	ROLL FORWARD ENABLED
Last Backup Time :	2006-10-26 00:31:24
Automatic Database Backup :	Disabled
Logging Type :	ARCHIVE
Online Backup Available :	YES
Table Space Backup Available :	YES

[✓ Backup entire database] Backup selected table spaces

Figure 14.8 Incremental and delta backups

The Backup Files

The backup images are stored as files. The name of the backup file contains the following parts:

- Database alias
- Type of backup (0=Full database, 3=Table space, 4=Copy from LOAD)
- Instance name
- Database partition (always NODE0000 for a single-partition database)
- Catalog partition number (always CATN0000 for a single-partition database)
- Timestamp of the backup
- The image sequence number

V9 The naming convention applies to all platforms. In Figure 14.9, you can see the full filename of each backup image.

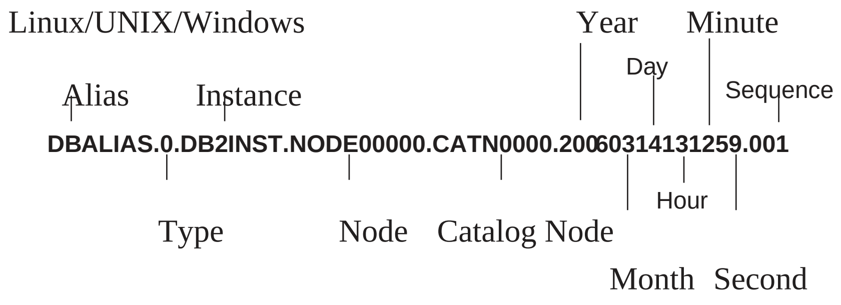
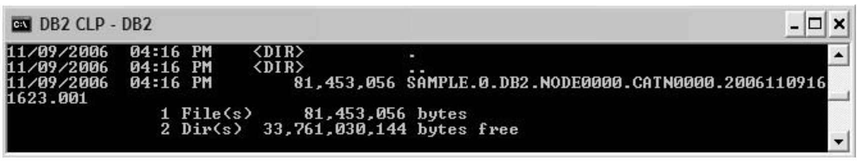


Figure 14.9 Backup file name hierarchy

For example, the command (on a Windows server):

BACKUP DATABASE sample to D:\tmp produces

the backup image shown in Figure 14.10.



```
11/09/2006  04:16 PM    <DIR>          .
11/09/2006  04:16 PM    <DIR>          ..
11/09/2006  04:16 PM        81,453,056 SAMPLE.0.DB2.NODE0000.CATN0000.2006110916
1623.001
               1 File(s)     81,453,056 bytes
               2 Dir(s)  33,761,030,144 bytes free
```

Figure 14.10 Backup file on Windows

The backup image can be found in the directory specified in the **BACKUP DATABASE** command, or the directory where the command is issued from.

Database Recovery:

You can restore a backup image using the **RESTORE DATABASE** command. You can choose to recover everything in the image, or just an individual table space, or multiple table spaces.

You can restore a database backup image and create a new database or you can restore over top of an existing database. You need SYSADM, SYSCTRL, or SYSMAINT authority to restore into an existing database, and SYSADM or SYSCTRL authority restore to a new database.

Figure 14.11 shows the syntax diagram of the **RESTORE DATABASE** command.

To perform a restore of the sample database, you can use the syntax shown in Figure 14.12.

```
RESTORE DATABASE sample
(1) FROM C:\DBBACKUP
(2) TAKEN AT 20070428131259   (3)
WITHOUT ROLLING FORWARD   (4)
WITHOUT PROMPTING         (5)
```

Figure 14.12 Restore Database command example

where:

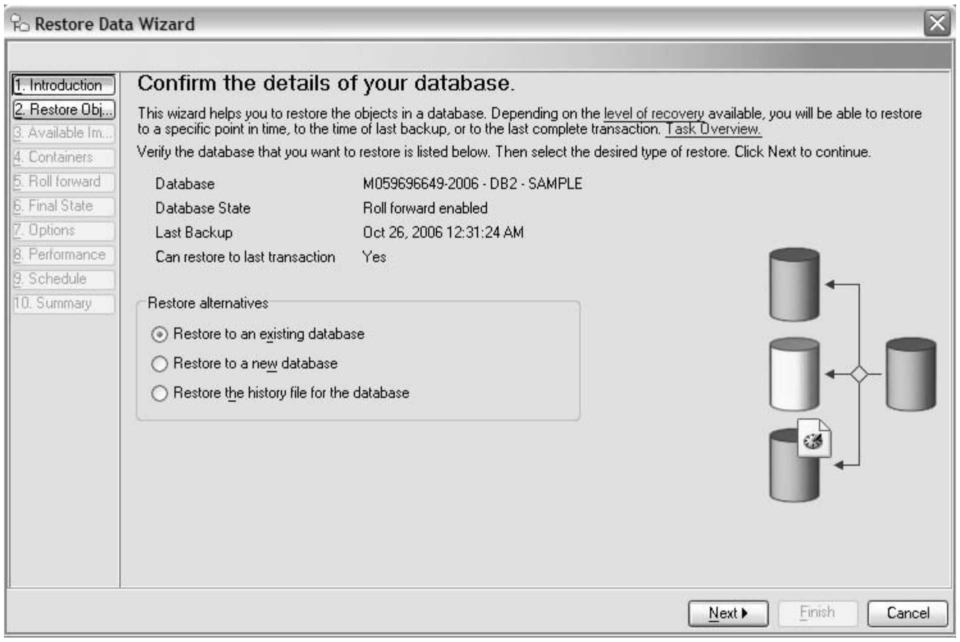
(1) Indicates the name of the database image to restore.
(2) Specifies the location where the input backup image is located.
(3) If there is more than one backup image in the directory, this option identifies thespecific backup based on the timestamp, which is part of the backup filename.
(4) If a database has archival logging enabled, a restore operation puts the databasein roll forward pending state, regardless of whether the backup was online or offline. If restoring from an offline backup, you can choose not to roll forward. This option tells DB2 not to place the database in roll forward pending state. When restoring from an online backup, the "without rolling forward" option cannot be used, as you must roll forward to at least the time that the backup completed.
(5) Specifies that the restore is to be performed unattended. Action that normallyrequires user intervention will return an error message. When using a

removable media device, such as tape or diskette, you will be prompted when the end of the device is reached, even if this option is specified.

Restoring a Database with the Control Center:

You can use the Restore Data Wizard to restore a database or a table space. In the Control Center, expand the database folder, right-click on the database you want to restore, and select **Restore.** The Restore Data Wizard is launched.

Figure 14.14 shows that you have the options to restore to an existing database, a new database, or only the history file. The Restore Data Wizard guides you through the restore command options. (We will discuss the history file in Section 14.7, Recovering a Dropped Table.)



Confirm the details of your database.

This wizard helps you to restore the objects in a database. Depending on the level of recovery available, you will be able to restore to a specific point in time, to the time of last backup, or to the last complete transaction. Task Overview.

Verify the database that you want to restore is listed below. Then select the desired type of restore. Click Next to continue.

Database	M059696649-2006 - DB2 - SAMPLE
Database State	Roll forward enabled
Last Backup	Oct 26, 2006 12:31:24 AM
Can restore to last transaction	Yes

Restore alternatives

- [✓ Restore to an existing database]
- Restore to a new database
- Restore the history file for the database

Figure 14.14 The Restore Data Wizard

Online and Offline Backups:

There are two types of backups we can perform using DB2 - offline and online. Offline backups require users and applications to be disconnected from the databases thereby requiring a certain down time. Online backups on the other hand, can be performed while users are still connected to databases. Trove currently supports full offline backups for DB2.

The method that you choose to back up your data determines whether IBM Spectrum Control remains online or offline during the backup process.

Advantages of an offline backup

The advantages of the offline backup method are:

- The offline backup method is the default method and it is easier than the online method to configure and to maintain.
- The circular type of logging that is used for offline backups is easier to configure and maintain than the type of logging that is used for online backups.

Disadvantages of an offline backup

The disadvantages of the offline backup method are:

- You must stop IBM Spectrum Control when you back up the data. So data is not collected and your storage resources are not being monitored during the back up process.
- You cannot collect performance data for the disk subsystems and SAN fabrics when data is being backed up.
- You might miss critical events, for example, failures within a SAN fabric, that occur during the backup process.

To minimize the loss of data for your storage resources and to ensure that you do not miss critical events, back up your data when your storage resources are not being used or when storage usage is low.

Advantages of an online backup

The advantages of the online backup method are:

- You continue to collect data and monitor your storage resources during the backup process because you do not have to stop IBM Spectrum Control.
- You continue to receive alerts and can respond quickly to critical events at any time of day.
- You continue to collect performance data for your disk subsystems and SAN fabrics.

Disadvantages of an online backup

The disadvantages of the online backup method are:

- The archive type of logging that is used with this type of backup is a more advanced method; it requires a good knowledge DB2® operation and administration.
- Software upgrades to IBM Spectrum Control that involve changes to the layout of the database might not complete successfully. In such cases, you can use circular logging to ensure that the software upgrade succeeds. You can switch back to archive logging after the software upgrade is installed.

<u>DB2 Utilities on Linux / UNIX:</u>

Moving data from one database server to another is a very common task in a production environment and in almost every phase of the development cycle. For example, a developer may want to export data from a production database and load it into lower environment tables for testing. In a production environment, a database administrator may want to export a few tables from production to a test database server to investigate a performance problem.

DB2 provides a number of utilities so that we can accomplish these tasks very easily. Different utilities • Different file formats used to move data

1. The EXPORT utility
2. The IMPORT utility
3. The LOAD utility
4. The DB2MOVE utility

THE DB2 EXPORT UTILITY:

The export utility extracts data from a table into a file. export to empdata.ixf of ixf select * from employee Using the optional messages clause, you can specify a filename where warning and error messages of the export operation are logged. If no message file accompanies the messages clause, the messages are written to standard output. Though optional, we highly recommend you use this clause so that all the messages generated by the utility are saved.

The export command also supports **SELECT** statements with joins, nested statements, and so on. Thus, if you want to export data from two tables, they can be joined as shown in the following example:

export to deptmgr.del of del messages deptmgr.out
select deptno, deptname, firstnme, lastname, salary
from employee, department
where empno = mgrno

File Type Modifiers Supported in the Export Utility:
The export utility exports data to a file using default file formats. For example, if you are exporting a table to a file in DEL format, the default column delimiter is a comma, and the default string delimiter is the double quote. What happens if the table data to be exported contains these delimiters as part of the data? The file exported may contain data that can be confused as a delimiter, making it impossible for an import or load operation to work correctly. To customize the delimited file format to use different delimiters other than the defaults, use the modified by clause.

Changing the Column Delimiter:

To use a column delimiter other than the comma, specify the **coldel** file type modifier in the modified by clause. The following example specifies to use a semicolon as the column modifier.

Note that there is no space between the keyword **coldel** and the semicolon.

export to deptmgr.del of del
modified by coldel; messages deptmgr.out
select deptno, deptname, firstnme, lastname, salary
from employee, department
where empno = mgrno

Changing the Character Delimiter:

You can enclose character strings with a different delimiter by using the keyword **chardel**.

```
export to deptmgr.del of del
modified by coldel; chardel’’ messages deptmgr.out
select deptno, deptname, firstnme, lastname, salary
 from employee, department
where empno = mgrno
```

Changing the Date Format:

You can also export data in a specific date format you prefer by using the timestampformat modifier.

```
export to deptmgr.del of del
modified by coldel; chardel’’ timestampformat=”yyyy.mm.dd hh:mm”
messages deptmgr.out
select deptno, deptname, firstnme, lastname, salary
from employee, department
where empno = mgrno
```

Exporting Large Objects:

DB2 supports the following types of large objects: character large objects (CLOBs), binary large objects (BLOBs), and double-byte character large objects (DBCLOBs). LOB values can be as large as 2GB for CLOBs and BLOBs and 1GB for DBCLOBs. Due to these sizes, the export utility by default extracts only the first 32KB of data of the LOB values in the export file. To extract the entire **LOB**, you must use the lobs to or lobfile clause or the lobsinfile modifier.

The **lobs** to clause specifies the directories in which the LOB files will be stored. If no lobs to clause is found, LOB data is written to the current working directory.

For example, the following export command generates three files. One file is the message file, mgrresume.out. Another file, mgrresume.del, is the data file, which contains all data columns for the rows except the LOB data. The third file, resume.001, is the file containing the LOB values for all rows.

```
export to mgrresume.del of del
lobs to c:\lobs
lobfile resume
modified by lobsinfile
messages mgrresume.out
select deptno, deptname, firstnme, lastname, resume
from employee a, emp_resume b
```

where a.empno = b.empno

THE DB2 IMPORT UTILITY:

The import utility inserts data from an input file into a table or a view. The utility performs inserts as if it was executing **INSERT** statements.

import from employee.ixf of ixf
messages employee.out
insert into employee

Import Modes:

Mode Description
INSERT Adds the imported data to the table without changing the existing table data. The target table must already exist.
INSERT_UPDATE Adds the imported data to the target table or updates existing rows with matching primary keys. The target table must already exist defined with primary keys.
CREATE Creates the table, index definitions, and row contents. The input file must use the IXF format because this is the only format that stores table and index definitions.
REPLACE Deletes all existing data from the table and inserts the imported data. The table definition and index definitions are not changed.
REPLACE_CREATE If the table exists, this option behaves like the replace option. If the table does not exist, this option behaves like the create option, which creates the table and index definitions and then inserts the row contents. This option requires the input file to be in IXF format.
The ***warningcount*** option indicates that the utility will stop after 10 warnings are received. If this option is not specified or is set to zero, the import operation will continue regardless of the number of warnings issued.

import from employee.del of del
messages empsalary.out
warningcount 10
replace into empsalary (salary, bonus, comm)

import from mgrresume.ixf of ixf
lobs from c:\lobs1, c:\lobs2, c:\lobs3
modified by lobsinfile commitcount 1000
messages mgrresume.out
create into newemployee in datats index in indexts long in lobts

THE DB2 LOAD UTILITY:

The load utility is another tool you can use to insert data into a table. Note that you cannot run the load utility against a view; the target must be a table that already exists.

The major difference between a load and an import is that a load is much faster. Unlike the import utility, data is not written to the database using normal insert operations. Instead, the load utility reads the input data, formats data pages, and writes directly to the database. Database changes are not logged and constraint validations (except unique constraint) are not performed during a load operation.

The Load Process

Basically, a complete load process consists of four phases.

1. **During the load phase**, the load utility scans the input file for any invalid data rows that do not comply with the table definition; for example, if a table column is defined as INTEGER but the input data is stored as "abcd". Invalid data will not be loaded into the table. The rejected rows and warnings will be written to a dump file specified by the dumpfile modifier. Valid data is then written into the table. At the same time, table statistics (if the statistics use profile option was specified) and index keys are also collected. If the savecount option is specified in the load command, points of consistency are recorded in the message file. Consistency points are established by the load utility. They are very useful when it comes to restarting the load operation. You can restart the load from the last successful consistency point.
2. **During the build phase**, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the statistics use profile option was specified).
3. **In the load phase**, the utility only rejects rows that do not comply with the column definitions. Rows that violated any unique constraint will be deleted in the delete phase. Note that only unique constraint violated rows are deleted. Other constraints are not checked during this phase or during any load phase. You have to manually check it after the load operation is complete.
4. **During the index copy phase**, index data is copied from a system temporary table space to the original table space. This will only occur if a system temporary table space was specified for index creation during a load operation with the read access option specified db2 "load from abcd.del of del modified by coldel MESSAGES MSG.TXT insert into schema_name.table_name NONRECOVERABLE"

   Tables are in load pending state or not
   db2 load query table RTOCM.UA_CONTACTHISTORY_ACCT_PROD

THE DB2MOVE IMPORT UTILITY:

The **db2move** utility retrieves a list of all user tables in a database from the system catalog. It then exports these tables in PC/IXF format, which is a version of an adaptation of the Integration Exchange Format (IXF) data interchange architecture.

The PC/IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another workstation platform and imported or loaded to a DB2 database on that platform. Files that **db2move**generates during an export operation are used as input files for the ensuing import or load operation (see Table 1). If a **db2move** operation is to succeed, the requesting user ID must have the correct authorization required by the underlying DB2 data movement utilities. A database connection is not required prior to invoking the **db2move** command; the utility does that for you.

The basic syntax of the **db2move** command is as follows:
Listing 1. The db2move command

```
db2move <database-name> <action> [<option> <value>]
```

You must specify the name of the database whose tables you want to move, and the action (export, import, or load) that is to be performed. You can then specify an option to define the scope of the operation. For example, you can limit the operation to certain tables (-tn), table spaces (-ts), table creators (-tc), or schema names (-sn). Specifying a subset of tables, table spaces, or table creators is valid with the export action only. If multiple values are specified, they must be separated by commas; no blanks are allowed between items in the list of values. The maximum number of items that can be specified is 10.
Alternatively, you can specify the -tf option with the name of a file that lists the tables to export. The fully qualified table names should be listed one per line. You can also specify:

-io ***import-option***
Specifies one of the modes under which the DB2 import utility can run. Valid options are: **CREATE, INSERT, INSERT_UPDATE, REPLACE, and REPLACE_CREATE**. The default is REPLACE_CREATE. For more information about these modes, see the DB2 product documentation.

-lo ***load-option***
Specifies one of the modes under which the DB2 load utility can run. Valid options are: **INSERT** and REPLACE. The default is **INSERT**. For more information about these modes, see the DB2 product documentation.

-l ***lobpaths***
Specifies the location in which LOB files are to be created or found. One or more absolute path names must be specified. If multiple paths are specified, they must be separated by commas; no blanks are allowed between values. The default is the current directory.

-u ***userid***
Specifies a user ID with which the utility can log on to a remote system.

-p ***password***
Specifies a password that authenticates the user; the utility requires a valid user ID and password to log on to a remote system.

UNLOAD:

The UNLOAD online utility copies data from one or more source objects to one or more BSAM sequential data sets in external formats. The output records that the UNLOAD utility writes are compatible as input to the LOAD utility. Therefore, you can use this output to reload the original table or different tables.

Although the function of the UNLOAD utility is often referred to as unloading data, the data is not deleted from the source object. The utility just makes a copy of the data. That copy includes the data only; it does not include all of the pages, such as the system pages and header pages, that are included in an image copy.

The source for UNLOAD can be Db2 table spaces or Db2 image copy data sets. The source cannot be a concurrent copy or a FlashCopy image copy.

You can unload rows from an entire table space or select specific partitions or tables to unload. You can also select columns by using the field specification list. If a table space is partitioned, you can unload all of the selected partitions into a single data set. Alternatively, you can unload each partition in parallel into physically distinct data sets.

UNLOAD must be run on the system where the definitions of the table space and the table exist.

Output

UNLOAD generates an unloaded table space or partition.

Authorization required

To execute this utility, you must use a privilege set that includes one of the following authorities:

- Ownership of the tables
- SELECT privilege on the tables
- DBADM authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on DSNDB04 or the implicitly created database is sufficient.
- DATAACCESS authority
- SYSADM authority
- SYSCTRL authority (catalog tables only)
- SQLADM authority (catalog tables only)
- System DBADM authority (catalog tables only)
- ACCESSCTRL authority (catalog tables only)
- SECADM authority (catalog tables only)

The UNLOAD utility operates in **these phases**:

1. UTILINIT initializes the environment.

2. UNLOAD unloads records to sequential data sets. One pass through the input data set is made. If UNLOAD is processing a table space or partition, Db2 takes internal commits. These commits provide commit points at which the utility can be restarted if the utility stops in this phase.
3. UTILTERM cleans up the environment.

LOAD:

Use the LOAD online utility to load one or more tables of a table space. The LOAD utility loads records into the tables and builds or extends any indexes that are defined on them.

If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data.

The loaded data is processed by any edit or validation routine that is associated with the table, and any field procedure that is associated with any column of the table. The LOAD utility ignores and does not enforce informational referential constraints.

To avoid the cost of running the RUNSTATS utility afterward, you can also specify the STATISTICS option collect inline statistics when you run the LOAD utility.

You can use the LOAD utility in conjunction with z/OS DFSMS data set encryption with the REPLACE option to encrypt or decrypt table spaces or indexes that use Db2-managed data sets. The LOAD utility accepts encrypted input data sets.

Output

LOAD DATA generates one or more of the following forms of output:

- A loaded table space or partition.
- A discard file of rejected records.
- A summary report of errors that were encountered during processing; this report is generated only if you specify ENFORCE CONSTRAINTS or if the LOAD utility involves unique indexes.
- The output can be encrypted if a key label is defined for the output data set.

Authorization required

To execute this utility, you must use a privilege set that includes one of the following authorizations:

- Ownership of the table
- LOAD privilege for the database
- STATS privilege for the database is required if STATISTICS keyword is specified
- DBADM or DBCTRL authority for the database. If the database is implicitly created, these privileges must be on the implicitly created database or on DSNDB04.

- DATAACCESS authority
- SYSCTRL or SYSADM authority

COPY:

The COPY online utility creates copies of certain objects. These copies, called image copies, can later be used for recovery.

COPY can create up to five image copies: two sequential image copies for the local site, two sequential image copies for the recovery site, and one FlashCopy image copy. These copies can be created for any of the following objects:

- Table space
- Table space partition
- Data set of a non-partitioned table space
- Index space
- Index space partition

The sequential image copies can be either full or incremental. A full image copy is a copy of all pages in a table space, partition, data set, or index space. An incremental image copy is a copy of the system pages and only those data pages that have been modified since the last use of the COPY utility.

The RECOVER utility uses image copies when recovering a table space or index space to the most recent time or to a previous point in time. Copies can also be used by the MERGECOPY, COPYTOCOPY, and UNLOAD utilities.

The COPY utility produces up to four sequential data sets that contain the image copy and optionally one Flash Copy image copy. COPY also adds rows in the SYSIBM.SYSCOPY catalog table that describe the image copy data sets that are available to the RECOVER utility. Your installation is responsible for ensuring that these data sets are available if the RECOVER utility requests them.

COPY resets COPY-pending status as follows:

- If the copy is a full image copy, Db2 resets any COPY-pending status for the copied table spaces.
- If you copy a single table space partition, Db2 resets any COPY-pending status only for the copied partition and not for the whole table space.
- If you copy a single piece of a multi-piece linear data set, Db2 does not reset any COPY-pending status.
- If you copy an index space or index, Db2 resets any informational COPY-pending (ICOPY) status.
- If you copy a NOT LOGGED table space, Db2 resets any informational COPY-pending (ICOPY) status

<u>RECOVER:</u>

The RECOVER utility recovers data to the current state or to a previous point in time by restoring a copy and then applying log records. The RECOVER utility can also recover data to a previous point in time by backing out committed work.

The largest unit of data recovery is the table space or index; the smallest is the page. You can recover a single object or a list of objects. The RECOVER utility recovers an entire table space, index, a partition or data set, pages within an error range, or a single page. You can recover data from sequential image copies of an object, a FlashCopy image copy of an object, a system-level backup, or the log. Point-in-time recovery with consistency automatically detects the uncommitted transactions that are running at the recover point in time and rolls back their changes on the recovered objects. After recovery, objects will be left in their transactionally consistent state.

You can use the RECOVER utility in conjunction with z/OS DFSMS data set encryption to encrypt or decrypt table spaces or indexes.

Output
Output from RECOVER consists of recovered data (a table space, index, partition or data set, error range, or page within a table space).

Authorization required
To run this utility, you must use a privilege set that includes one of the following authorities:

- RECOVERDB privilege for the database
- DBADM or DBCTRL authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on the implicitly created database or DSNDB04 is required.
- System DBADM authority
- DATAACCESS authority
- SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also run RECOVER, but only on a table space in the DSNDB01 or DSNDB06 database.

<u>RUNSTATS</u>:

The RUNSTATS online utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions. Db2 records these statistics in the Db2 catalog and uses them to select access paths to data during the bind process.

You can use these statistics to evaluate the database design and determine when table spaces or indexes must be reorganized. To obtain the updated statistics, you can query the catalog tables.

The two formats for the RUNSTATS utility are RUNSTATS TABLESPACE and RUNSTATS INDEX. RUNSTATS TABLESPACE gathers statistics on a table space and, optionally, on tables, indexes or columns; RUNSTATS INDEX gathers statistics only on indexes. RUNSTATS does not collect statistics for clone tables or index spaces.

RUNSTATS can collect statistics on any single column or set of columns.

RUNSTATS collects the following types of distribution statistics:

Frequency
The percentage of rows in the table that contain a value for a column or combination of values for a set of columns.

Cardinality
The number of distinct values in the column or set of columns.

Output

RUNSTATS updates the Db2 catalog with table space or index space statistics, prints a report, or both.

Authorization required

To execute this utility, you must use a privilege set that includes one of the following authorities:

- STATS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on the implicitly created database or DSNDB04 is required.
- System DBADM authority
- SQLADM authority
- SYSCTRL or SYSADM authority

STOSPACE:

The STOSPACE online utility updates Db2 catalog columns that indicate how much space is allocated for storage groups and related table spaces and indexes.

For user-defined spaces, STOSPACE does not record any statistics.

Output

The output from STOSPACE consists of updated values in the columns and tables in the following list. In each case, an amount of space is given in kilobytes (KB). If the value is too large to fit in the SPACE column, the SPACEF column is updated.

- SPACE in SYSIBM.SYSINDEXES shows the amount of space that is allocated to indexes. If the index is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSTABLESPACE shows the amount of space that is allocated to table spaces. If the table space is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSINDEXPART shows the amount of space that is allocated to index partitions. If the partition is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSTABLEPART shows the amount of space that is allocated to table partitions. If the partition is not defined using STOGROUP, or if STOSPACE has not been executed, the value is zero.
- SPACE in SYSIBM.SYSSTOGROUP shows the amount of space that is allocated to storage groups.
- STATSTIME in SYSIBM.SYSSTOGROUP shows the timestamp for the time at which STOSPACE was last executed.

Authorization required
To execute this utility, you must use a privilege set that includes one of the following authorities:

- STOSPACE privilege
- SYSCTRL or SYSADM authority

Execution phases of STOSPACE
The STOSPACE utility operates in these phases:

Phase	**Description**
UTILINIT	Performs initialization
STOSPACE	Gathers space information and updates catalog
UTILTERM	Performs cleanup

REORG INDEX:

The REORG INDEX online utility reorganizes an index space to improve access performance and reclaim fragmented space. You can specify the degree of access to your data during reorganization, and you can collect inline statistics by using the STATISTICS keyword.

You can determine when to run REORG INDEX by using the LEAFDISTLIMIT catalog query option. If you specify the REPORTONLY option, REORG INDEX produces a report that indicates whether a REORG is recommended; in this case, a REORG is not performed. These options are not available for indexes on the directory.

To avoid the cost of running the RUNSTATS utility afterward, you can also specify the STATISTICS option to collect inline statistics when you run the REORG INDEX utility.

You can use the REORG INDEX utility in conjunction with z/OS DFSMS data set encryption to encrypt or decrypt indexes. REORG INDEX always allocates new Db2-managed data sets unless the REUSE option is specified.

Output

The following list summarizes

- REORG INDEX output: REORG INDEX Reorganizes the entire index (all parts if partitioning).
- REORG INDEX PART n Reorganizes PART n of a partitioning index or of a data-partitioned secondary index

<u>REORG TABLESPACE:</u>

The REORG TABLESPACE online utility reorganizes a table space, partition, or range of partitions to reclaim fragmented space and improve access performance. You can also run REORG TABLESPACE to materialize pending definition changes.

You can use the DSNACCOX stored procedure to get recommendations on when to run REORG TABLESPACE.

To avoid the cost of running the RUNSTATS utility after running REORG TABLESPACE, you can request that REORG collect inline statistics by specifying the STATISTICS option.

Authorization required

To execute this utility on a user table space, you must use a privilege set that includes one of the following authorities:

- REORG privilege for the database
- DBADM or DBCTRL authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on the implicitly created database or DSNDB04 is required.
- SYSCTRL authority
- SYSADM authority
- DATAACCESS authority

As data is inserted, deleted, and updated in the database, the data might not be physically placed in a sequential order, which means that DB2 must perform additional read operations to access data. This usually requires more disk I/O operations, and we all know such operations are costly. To minimize I/O operations, you should consider physically reorganizing the table to the index so that related data are located close to each other.

Basic Concepts of OLAP:

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information. This chapter cover the types of OLAP, operations on OLAP, difference between OLAP, and statistical databases and OLTP.

Types of OLAP Servers

We have four types of OLAP servers −

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)
- Hybrid OLAP (HOLAP)
- Specialized SQL Servers

Relational OLAP - ROLAP servers are placed between relational back-end server and client front-end tools. To store and manage warehouse data, ROLAP uses relational or extended-relational DBMS.

ROLAP includes the following −

- Implementation of aggregation navigation logic.
- Optimization for each DBMS back end.
- Additional tools and services.

Multidimensional OLAP - MOLAP uses array-based multidimensional storage engines for multidimensional views of data. With multidimensional data stores, the storage utilization may be low if the data set is sparse. Therefore, many MOLAP server use two levels of data storage representation to handle dense and sparse data **sets.**

Hybrid OLAP - Hybrid OLAP is a combination of both ROLAP and MOLAP. It offers higher scalability of ROLAP and faster computation of MOLAP. HOLAP servers allows to store the large data volumes of detailed information. The aggregations are stored separately in MOLAP store.

Specialized SQL Servers - Specialized SQL servers provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

Data warehousing in DB2:

Data warehousing in Db2 is a suite of products that combines the strength of Db2 with a datawarehousing infrastructure from IBM®.

You can use data warehousing in Db2 to build a complete data warehousing solution that includes a highly scalable relational database, data access capabilities, and front-end analysis tools.

The following components are provided in data warehousing in Db2:

Data warehousing in Db2 application server

- Administration Console
  - SQL Warehousing (SQW) administration
  - Cubing Services administration
    - If you already have Cubing Services installed on your system, you can continue to use it when you upgrade your version of data warehousing in Db2 to the newest version. However, if Cubing Services is not already installed on your system, then it is not an option as you upgrade to the newest version of data warehousing in Db2. Instead, you can migrate to Cognos® Dynamic Cubes.
- IBM Data Server Client
- WebSphere® Application Server

Data warehousing in Db2 client

- Design Studio
  - SQL Warehousing (SQW) Tool
- Administration Console Command Line Client

- **<u>Components of data warehousing in Db2”:</u>**

The components of data warehousing in Db2 provide an integrated platform for warehouse administration and for the development of warehouse-based analytics.
The key components of data warehousing in Db2 are:

Data warehousing in Db2 Design Studio

The Design Studio provides a common design environment for creating physical data models, OLAP cubes, SQL data flows, and control flows. The Design Studio is built on the Eclipse Workbench, which is a development environment that you can customize,

Data warehousing in Db2 Administration Console

The data warehousing in Db2 Administration Console is a web application from which you can deploy and manage applications, control flows, database resources, and system resources. You can do the following types of tasks in the Administration Console:

Common configuration

Create and manage database and system resources, including driver definitions, log files, and notifications.

SQL warehousing

Run and monitor data warehousing applications and view deployment histories and execution statistics.

Mining

Run and monitor data mining applications.

Cognos Dynamic Cubes

You can use data warehousing in Db2 with Cognos Dynamic Cubes to leverage substantial in-memory data assets as well as aggregate awareness in order to achieve high performance interactive analysis & reporting over terabytes of warehouse data.

Cognos® 10 Business Intelligence For Reporting

IBM® Cognos 10 Business Intelligence For Reporting lets organizations implement a rich set of business intelligence capabilities. It is an ideal solution for companies that want to consolidate data marts, information silos, and business analytics to deliver a single version of the truth to all users. It allows easy creation of reports and quick analysis of data from the data warehouse.

IBM Optim™ Performance Manager Extended Edition

Optim Performance Manager Extended Edition is a follow on to Db2 Performance Expert. Optim Performance Manager Extended Edition helps optimize the performance and availability of mission critical databases and applications.

Intelligent Miner®

Data warehousing in Db2 includes the following mining features:

- Intelligent Miner Easy Mining
- Intelligent Miner Modeling
- Intelligent Miner Scoring

These features provide rapid enablement of data mining analysis in Data Warehousing, eCommerce, or traditional Online Transaction Processing (OLTP) application programs. While previously available as separately priced products, they are now available as an integrated part of data warehousing in Db2.

Text Analysis

With data warehousing in Db2, you can create business insight from unstructured information. You can extract information from text columns in your data warehouse then use the extracted information in reports, multidimensional analysis, or as input for data mining.

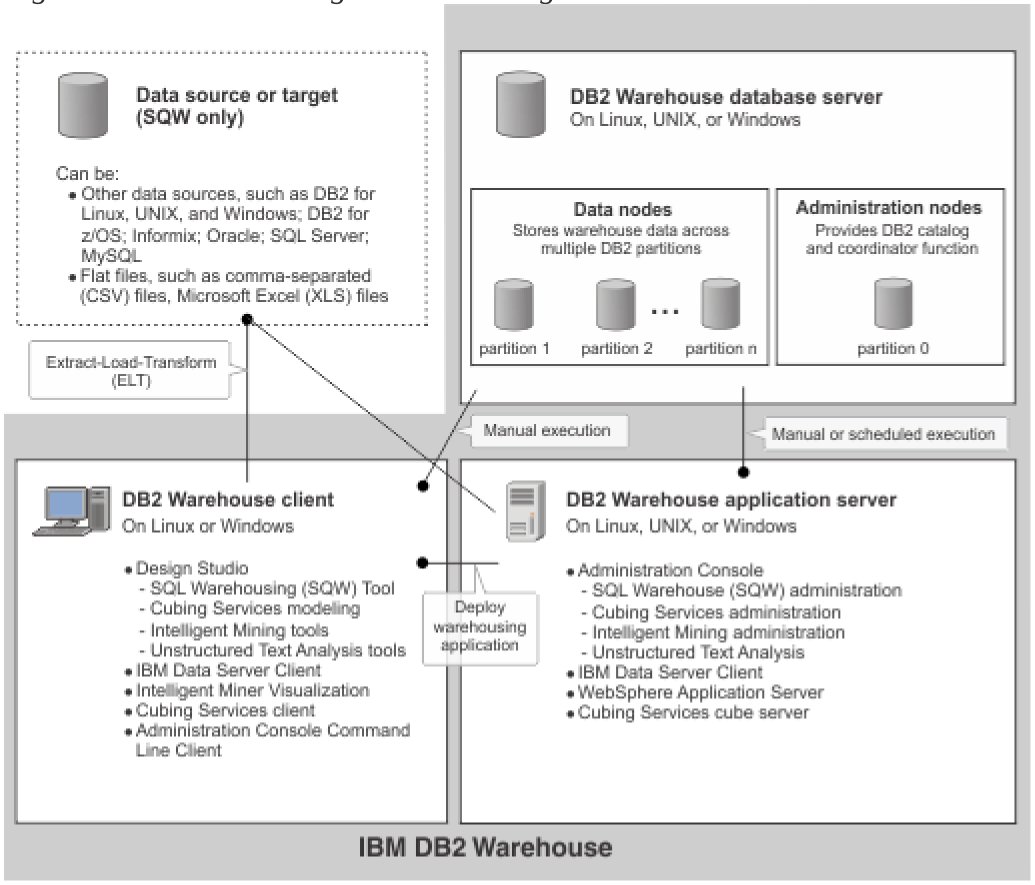
- **Installation architecture for data warehousing in Db2 on multiple computers**

Data warehousing in Db2 has a component-based architecture that consists of a data server component group, an application server component group, and a client component group. In a typical production environment, you install each of these component groups on different computers to create a complete warehousing solution.

Data warehousing in Db2 has a component-based architecture that consists of a data server component group, an application server component group, and a client component group. In a

typical production environment, you install each of these component groups on different computers to create a complete warehousing solution.
The following diagram illustrates the component architecture of the product and provides a basis for planning your installation across multiple computers.

Figure: Data warehousing in Db2 working with data sources and databases


Data source or target (SQW only)
Can be:
• Other data sources, such as DB2 for Linux, UNIX, and Windows; DB2 for z/OS; Informix; Oracle; SQL Server; MySQL
• Flat files, such as comma-separated (CSV) files, Microsoft Excel (XLS) files
DB2 Warehouse database server
On Linux, UNIX, or Windows
Data nodes
Stores warehouse data across multiple DB2 partitions
partition 1
partition 2
...
partition n
Administration nodes
Provides DB2 catalog and coordinator function
partition 0
Extract-Load-Transform (ELT)
Manual execution
Manual or scheduled execution
DB2 Warehouse client
On Linux or Windows
• Design Studio
- SQL Warehousing (SQW) Tool
- Cubing Services modeling
- Intelligent Mining tools
- Unstructured Text Analysis tools
• IBM Data Server Client
• Intelligent Miner Visualization
• Cubing Services client
• Administration Console Command Line Client
Deploy warehousing application
DB2 Warehouse application server
On Linux, UNIX, or Windows
• Administration Console
- SQL Warehouse (SQW) administration
- Cubing Services administration
- Intelligent Mining administration
- Unstructured Text Analysis
• IBM Data Server Client
• WebSphere Application Server
• Cubing Services cube server
IBM DB2 Warehouse

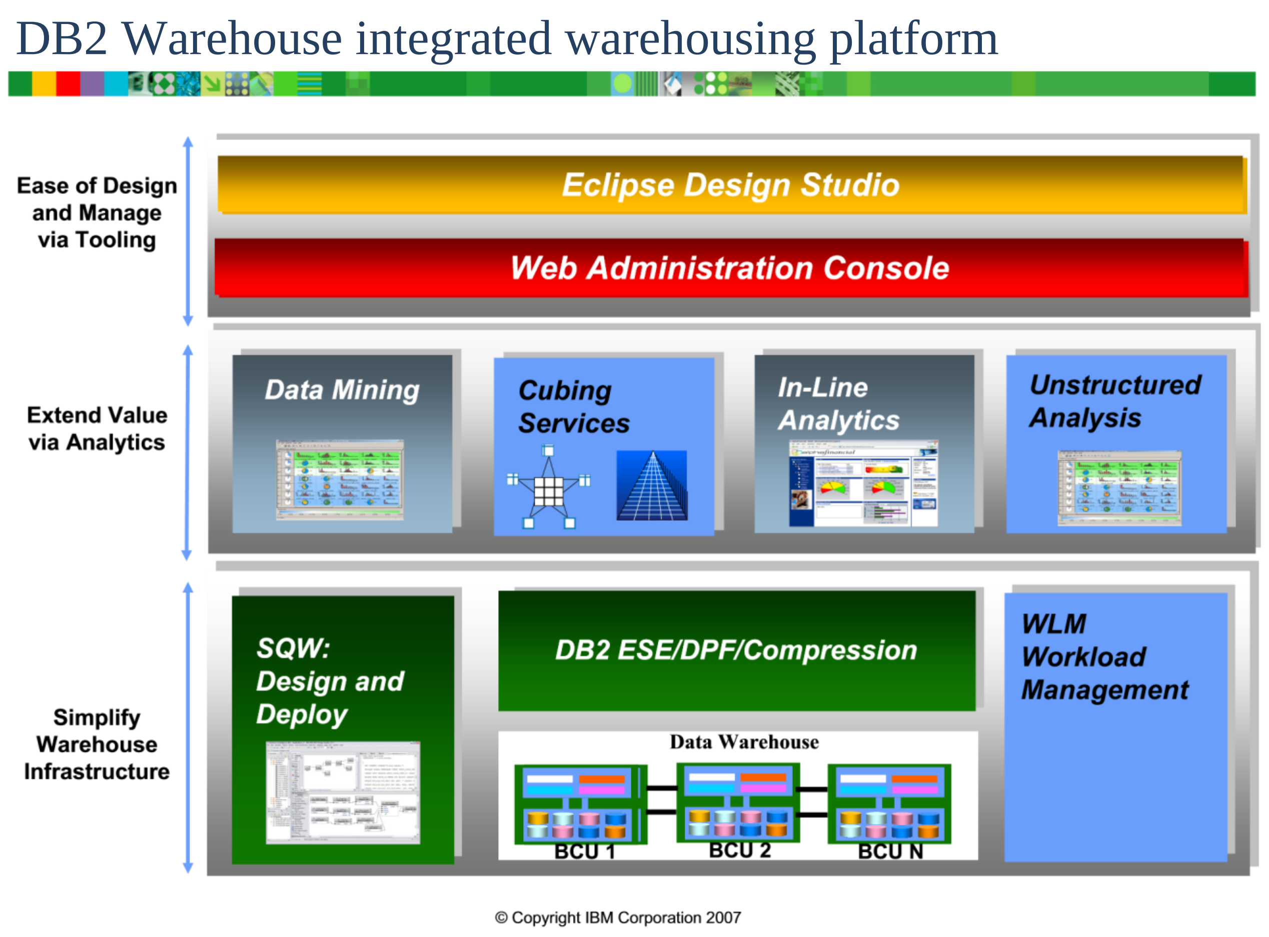

DB2 Warehouse integrated warehousing platform
Ease of Design and Manage via Tooling
Eclipse Design Studio
Web Administration Console
Extend Value via Analytics
Data Mining
Cubing Services
In-Line Analytics
Unstructured Analysis
Simplify Warehouse Infrastructure
SQW: Design and Deploy
DB2 ESE/DPF/Compression
Data Warehouse
BCU 1
BCU 2
BCU N
WLM Workload Management
© Copyright IBM Corporation 2007

Figure 7-9. DB2 Warehouse integrated warehousing platform

DB2 Data Warehouse is an integrated platform for warehouse-based analytics that runs on the DB2 LUW platforms. The Warehouse Tool component is the SQL Warehousing Tool (SQW) that is designed to move data already in DB2 into data marts. The mining component is DB2 intelligent Miner and includes a Miningblox component to facilitate reporting from a mining run via Alphablox. The OLAP component is Cubing Services, which is designed to be an MDX gateway and an OLAP accelerator for applications accessing a star schema stored in DB2. Alphablox is used as the in line analytics component.

The SQW, the data mining component, Cubing Services and the new Alphablox blox builder component all share the same Eclipse-base development environment and all, except Alphablox, utilize the same common Administration Console, a WebSphere Application Server application, for administration purposes.

A Rational Data Architect plug-in is used to do physical data modeling.

Data Migration (DB2 / Oracle / MS SQL / Sybase):

Data migration is the process of transferring data between data storage systems, data formats or computer systems. A data migration project is done for numerous reasons, which include replacing or upgrading servers or storage equipment, moving data to third-party cloud providers, website consolidation, infrastructure maintenance, application or database migration, software upgrades, company mergers or data center relocation.

Creating a data migration plan

A data migration project can be a challenge because administrators must maintain data integrity, time the project so there is minimal impact to the business and keep an eye on costs. Any problem that occurs during the migration will affect the business, so a data migration plan is key to ensuring minimal disruption and downtime to the active business processes.

Factors to consider during a data migration project include how long the migration will take; the amount of downtime required; and the risk to the business due to technical compatibility issues, data corruption, application performance issues, and missed data or data loss.

There are three broad categories of data movers.

1. Host-based software is best for application-specific migrations, such as platform upgrades, database replication and file copying.
2. Array-based software is primarily used to migrate data between like systems.
3. Network appliances migrate volumes, files or blocks of data depending on their configuration.

The following best practices should be used to protect data during a migration.

- Understand what data you are migrating, where it lives, what form it's in and the form it will take at its new destination.
- Extract, transform and deduplicate data before moving it.
- Implement data migration policies so data is moved in an orderly manner.
- Test and validate the migrated data to ensure it is accurate.
- Audit and document the entire data migration process.

Types of data migrations and their challenges

The data migration process is performed on three levels.

Storage migration is justified through technology refreshes, and the process is used as an optimal time to do data validation and reduction by identifying obsolete or corrupt data. The process involves moving blocks of storage and files from one storage system to another, whether it is on disk, tape or the cloud. There are numerous storage migration products and tools that help smooth the process. Storage migration also offers the chance to remediate any orphaned storageor inefficiencies.

Database migration is done when there is a need to change database vendors, upgrade the database software or move a database to the cloud. In this type of migration, the underlying data can change, which can affect the application layer when there is a change in protocol or data language. Data migrations in databases deal with modifying the data without changing the schema. Some key tasks include assessing the database size to determine how much storage is needed, testing applications and guaranteeing data confidentiality. Compatibility problems can occur during the migration process, so it is important to test the process first.

Application migration can occur when switching to another vendor application or platform. This process has its own inherent layers of complexity because applications interact with other applications, and each one has its own data model. Applications are not designed to be portable. Management tools, operating systems and virtual machine configurations can all differ from those in the environment where the application was developed or deployed. Successful application migration may require the use of middleware products to bridge technology gaps.

Cloud migration is a major technology trend, as the cloud provides on-demand flexibility, scalability and reduction of Capex for on-premises infrastructures. Public cloud providers offer a variety of services for storage, database and application migrations.

Data Migration from Oracle to DB2

Migration from Oracle Database to the IBM® DB2® on z/OS® database is not completely seamless and must be planned carefully. Administrators may face issues when migrating from Oracle to DB2 on z/OS due to locking differences between the two databases (see Table 1). However, these issues can be mitigated to a great extent.

One of the key locking behavior differences between Oracle and DB2 on z/OS is that Oracle does not hold any locks on a row while reading, and DB2 does. This difference can lead to a high probability of increased lock waits and issues such as deadlocks and timeouts in applications migrated from Oracle to DB2.

	Oracle	**DB2 on z/OS**
1	Read queries do not hold any locks on the rows unless explicitly requested by a FOR UPDATE clause.	By default, the read queries hold share-level locks.
2	Only row-level locking is possible implicitly. The entire table can be locked explicitly if needed.	Default locking is at page level, but locking also possible at row, table, tablespace, and LOB levels.
3	There is no concept of lock escalation.	Lock escalation can happen if the number of locks increases. Escalation results in promotion of row-level lock to table-level lock and page-level lock to tablespace-level lock, so that the number of locks is reduced.
4	Dirty read is not possible since there is no concept of Uncommitted read.	Uncommitted read is possible and is one of the ways to read rows without holding share-level locks.

Table 1. *Key locking differences between Oracle and DB2 on z/OS*

To handle locking issues, mitigation strategies are required at the database, application, and operational levels.

Data Migration from MySQL to DB2

MySQL-to-DB2 is a program to migrate MySQL databases to DB2 server. The program has high performance due to direct connection to data source and destination databases (it does not use ODBC or any other middleware software). Command line support allows to script, automate and schedule the conversion process.

Features

- All versions of IBM DB2 are supported
- All versions of Unix and Windows MySQL are supported
- All MySQL data types and attributes are supported
- Indexes and foreign keys are converted
- Option to convert individual tables
- Filter data to convert via SELECT-queries
- Option to merge MySQL data into an existing DB2 database
- Conversion settings can be stored into profile

- Command line support
- Easy-to-use wizard-style interface
- Full install/uninstall support
- Unlimited 24/7 support service
- 1-year subscription for updates

Limitations

- Does not convert views, stored procedures and triggers
- Demo version does not convert foreign keys
- Demo version converts only 50 records per table

Distributed Database in DB2:

A *distributed database* appears to a user as a single database but is, in fact, a set of databases stored on multiple computers. The data on several computers can be simultaneously accessed and modified using a network. Each database server in the distributed database is controlled by its local DBMS, and each cooperates to maintain the consistency of the global database. Figure shows the distributed database.

The database managers in a distributed relational database communicate and cooperate with each other in a way that allows a DB2® application program to use SQL to access data at any of the interconnected computer systems.

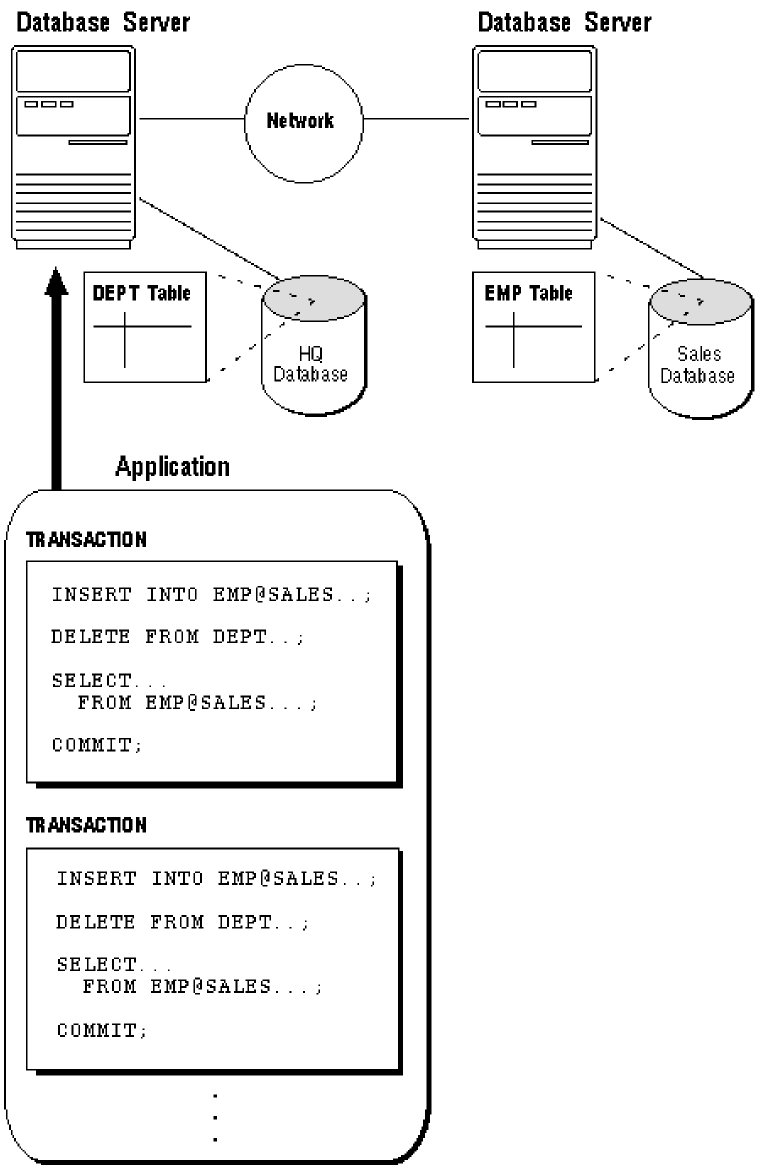
A distributed relational database consists of a set of tables and other objects that are spread across different, but interconnected, computer systems. Each computer system has a relational database manager, such as DB2, that manages the tables in its environment. The database managers communicate and cooperate with each other in a way that allows a DB2 application program to use SQL to access data at any of the computer systems. The DB2 subsystem where the application plan is bound is known as the local DB2 subsystem. Any database server other than the local DB2 subsystem is considered a remote database server, and access to its data is a distributed operation.

Distributed relational databases are built on formal requester-server protocols and functions. An application requester component supports the application end of a connection. It transforms an application's database request into communication protocols that are suitable for use in the distributed database network. These requests are received and processed by an application server component at the database server end of the connection. Working together, the application requester and application server handle the communication and location considerations so that the application is isolated from these considerations and can operate as if it were accessing a local database.

- **<u>Connections</u>**
  A *connection* is an association between an application process and a local or remote database server. Connections are managed by applications.

- **<u>Distributed unit of work</u>**
  The *distributed unit of work facility* provides for the remote preparation and execution of SQL statements.

- **<u>Remote unit of work</u>**
  The *remote unit of work facility* also provides for the remote preparation and execution of SQL statements, but in a much more restricted fashion than the distributed unit of work facility.



```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

```
INSERT INTO EMP@SALES..;

DELETE FROM DEPT..;

SELECT...
  FROM EMP@SALES...;

COMMIT;
```

******