Database Systems

Roll No	**:**
Name	**:**
Dept	**:**

PG DEPARTMENT OF COMPUTER SCIENCE

BHARATH COLLEGE OF SCIENCE AND MANAGEMENT,TNJ-5

CONTENT

SUBJECT CODE: 16SCCCS4

DATABASE SYSTEMS

Objective :

To provide the basic concepts of the Database Systems including Data Models, Storage

Structure, Normalization and SQL

Unit I

Introduction: Database-System Applications- Purpose of Database Systems - View of Data - Database Languages - Relational Databases - Database Design -Data Storage and Querying Transaction Management -Data Mining and Analysis - Database Architecture - Database Users and Administrators - History of Database Systems.

Unit II

Relational Model: Structure of Relational Databases -Database Schema - Keys - Schema

Diagrams - Relational Query Languages - Relational Operations Fundamental Relational Algebra Operations Additional Relational-Algebra Operations- Extended Relational-Algebra

Operations - Null Values - Modification of the Database.

Unit III

SQL:Overview of the SQL Query - Language - SQL Data Definition - Basic Structure of SQL Queries - Additional Basic Operations - Set Operations - Null Values Aggregate Functions - Nested Subqueries - Modification of the Database -Join Expressions - Views - Transactions - Integrity Constraints - SQL Data Types and Schemas - Authorization

Unit IV

Relational Languages: The Tuple Relational Calculus - The Domain Relational Calculus Database Design and the E-R Model: Overview of the Design Process - The Entity Relationship Model - Reduction to Relational Schemas - Entity-Relationship Design Issues - Extended E-R Features - Alternative Notations for Modeling Data - Other Aspects of Database Design

Unit V

Relational Database Design: Features of Good Relational Designs - Atomic Domains and First Normal Form - Decomposition Using Functional Dependencies - Functional-Dependency Theory - Decomposition Using Functional Dependencies - Decomposition Using Multivalued Dependencies-More Normal Forms - Database-Design Process

Text Book:

1. Database System Concepts, Sixth edition, Abraham Silberschatz, Henry F. Korth, S.Sudarshan, McGraw-Hill-2010.

Reference Books:

1 Database Systems: Models, Languages, Design and Application, Ramez Elmasri, Pearson Education 2014

Unit I

Introduction: Database-System Applications- Purpose of Database Systems - View of Data - Database Languages - Relational Databases - Database Design -Data Storage and Querying Transaction Management -Data Mining and Analysis - Database Architecture - Database Users and Administrators - History of Database Systems.

1. Introduction to Database Management Systems:

As the name suggests, the database management system consists of two parts. They are:

1. ***Database and***
2. ***Management System***

What is a Database?

To find out what ***database is, we have to start*** from data, which is the basic building block ***of any DBMS.***

1. **Data**: Facts, figures, ***statistics etc. having no particular meaning*** (e.g. 1, ABC, 19 etc).
2. **Record**: Collection of ***related data items, e.g. in*** the above example the three data items had no meaning. ***But if we organize them in the following way, then they collectively represent meaningful information.***
3. Table or Relation: ***Collection of related records***.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

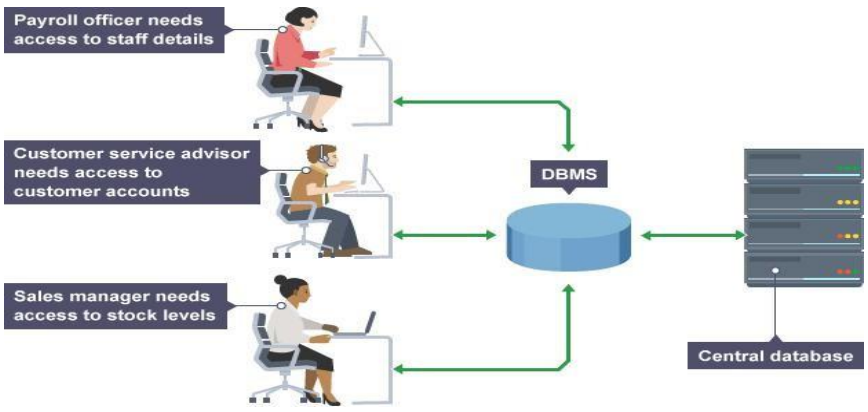


Figure 1.1: Employees are accessing Data through DBMS

1.1 What is Management System?

1. A **database-management system** (DBMS) is a ***collection of interrelated data and a set of programs to access those data.***
2. This is a collection of related data ***with an implicit meaning and*** hence is a database.
3. The ***collection of data, usually referred*** to as the **database**, contains information relevant to an enterprise.
4. The primary goal of a ***DBMS is to provide a way to store and*** retrieve database information that is both ***convenient and efficient.***
5. By **data,** we mean known ***facts that can be recorded*** and that have implicit ***meaning.***

1.2 Database System Applications:

1.2.1 Enterprise Information

1. **Sales:** For customer, product, ***and purchase information***.
2. **Accounting:** For payments, receipts, ***account balances, assets and other accounting*** information.
3. **Human resources**: For information ***about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.***
4. **Manufacturing:** For management of the ***supply chain and for tracking*** production of items in factories, inventories of items in ***warehouses and stores, and orders*** for items.
5. **Online retailers:** For sales data noted ***above plus online order tracking***, generation of recommendation lists, ***and maintenance of online*** product evaluations.

1.2.2 Banking Finance

1. **Banking:** For customer information, ***accounts, loans, and banking*** transactions.
2. **Credit card transactions:** For purchases on credit ***cards and generation of monthly*** statements.
3. **Finance:** For storing information about ***holdings, sales, and purchases of*** financial

instruments such as stocks and bonds; also ***for storing real-time market data to enable*** online trading ***by customers and automated trading by the firm***.

1.2.3 Others

1. **Universities:** For student information, ***course registrations, and grades*** (in addition to standard enterprise information ***such as human resources and accounting***).
2. **Airlines:** For reservations and ***schedule information. Airlines*** were among the first to use databases in a ***geographically distributed manner.***
3. **Telecommunication:** For keeping ***records of calls made, generating*** monthly bills, maintaining balances on prepaid ***calling cards, and storing information*** about the ***communication networks.***

1.3 Purpose of Database Systems

Database systems arose in response to early methods of computerized management of commercial data.

1. **Data redundancy and inconsistency:** Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages
2. **Difficulty in accessing data :** Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area.
3. **Data isolation :** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity problems :** The data values stored in the database must satisfy certain types of consistency constraints.
5. **Atomicity problems :** A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.
6. **Concurrent-access anomalies :** To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.
7. **Security problems:** Not every user of the database system should be able to access all the data.

1.3.1 Advantages of Database Systems :

1. ***Controlling of Redundancy***
2. ***Improved Data Sharing***
3. ***Data Integrity***
4. ***Security***
5. ***Data Consistency***
6. ***Efficient Data Access***
7. ***Enforcements of Standards***
8. ***Data Independence***
9. ***Reduced Application Development and Maintenance***

1.3.2Disadvantages of Database Systems :

1) It is bit complex. Since it ***supports multiple functionality*** to give the user the best, the underlying software ***has become complex. The designers and developers*** should have thorough knowledge about the ***software to get the most out of*** it.

2) Because of its ***complexity and functionality***, it uses large amount of memory. It also needs large ***memory to run efficiently***.

3) DBMS system ***works on the centralized system, i.e.; all the*** users from all over the world access this ***database. Hence any failure of the DBMS, will impact*** all the users.

4) DBMS is generalized ***software, i.e.; it is written work*** on the entire systems rather specific one***. Hence some of the application will run slow.***

1.4 View of Database Systems:

1. A database system is a ***collection of interrelated data and a set of programs*** that allow users to ***access and modify these data.***
2. A major purpose of a database system is to provide users with an ***abstract view*** of the data.
3. There is an 3 layers for viewing the data from the database they are as follows :
   - ***Physical level (or Internal View / Schema)***
   - ***Logical level (or Conceptual View / Schema)***
   - ***View level (or External View / Schema)***

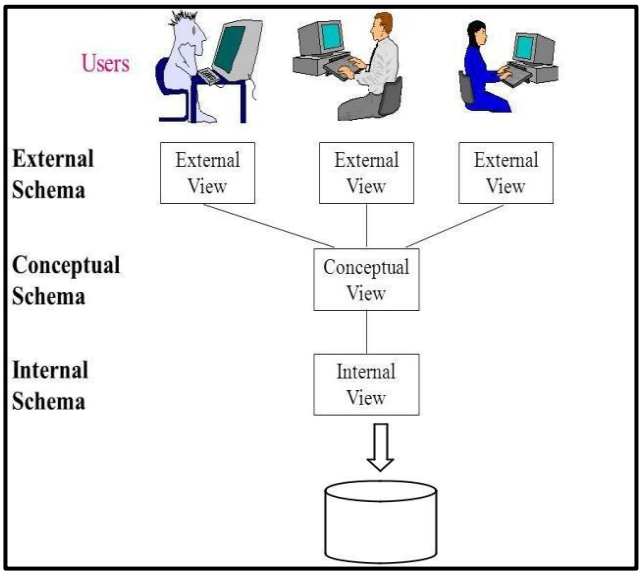


Fig 1.2 : Levels of Abstraction in DBMS

1. **Physical level (or Internal View / Schema)**:
    1. The ***lowest level*** of ***abstraction describes how the data are actually*** stored.
    2. The physical level describes ***complex low-level data structures in detail***.

3. **Logical level (or Conceptual View / Schema)**:
    1. The next-higher level of abstraction describes ***what*** data are stored in the ***database, and what relationships exist among those data***.
    2. The logical level thus describes the entire database ***in terms of a small number of relatively simple structures.*** Although implementation of the simple structures at the logical level ***may involve complex physical-level structures***, the user of the logical level does not need to be aware of this complexity. This is referred to as ***physical data independence***.

3. **View level (or External View / Schema):**
    1. The highest level of ***abstraction describes only part of the entire database***.
    2. Even though the logical ***level uses simpler structures***, complexity remains because of the ***variety of information stored in a large database***

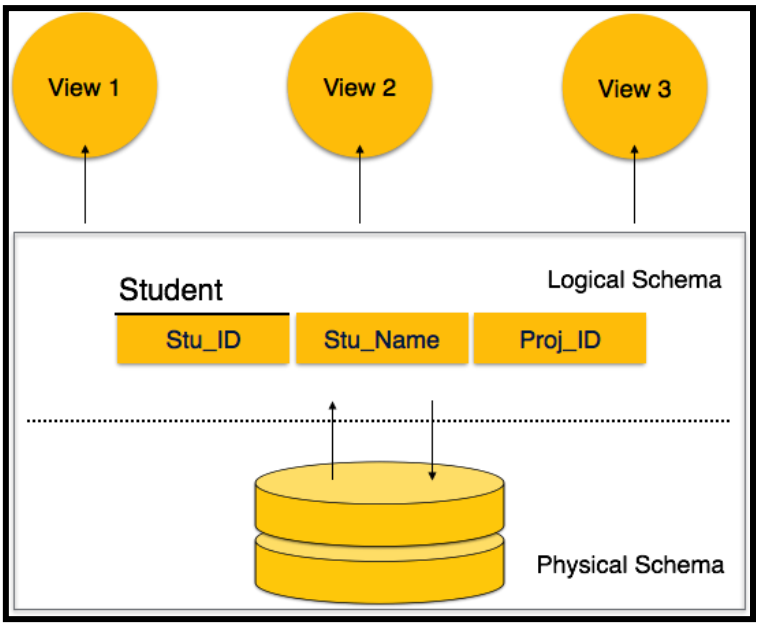
1.5 Instances and Schemas:

1.5.1 Schemas

1. The overall design ***of a database is called schema.***

2. A database ***schema is the skeleton structure of*** the database. It represents the ***logical view of the entire database.***

3. A schema ***contains schema objects like table, foreign*** key, primary key, views,***columns, data types, stored procedure,*** etc.
4. A database schema ***can be represented by using the visual*** diagram.
   That ***diagram shows the database objects and relationship with each other***.
5. A database ***schema is designed by the database*** designers to help programmers whose ***software will interact with the database. The process of database*** creation is ***called data modeling.***



A database schema can be divided broadly into two categories −

- **Physical Database Schema** − This ***schema pertains to the actual storage*** of data and its form of storage like files, indices, ***etc. It defines how the data will*** be stored in a secondary storage.
- **Logical Database Schema** − This ***schema defines all*** the logical constraints that need to be applied on the data ***stored. It defines tables, views, and integrity constraints***.

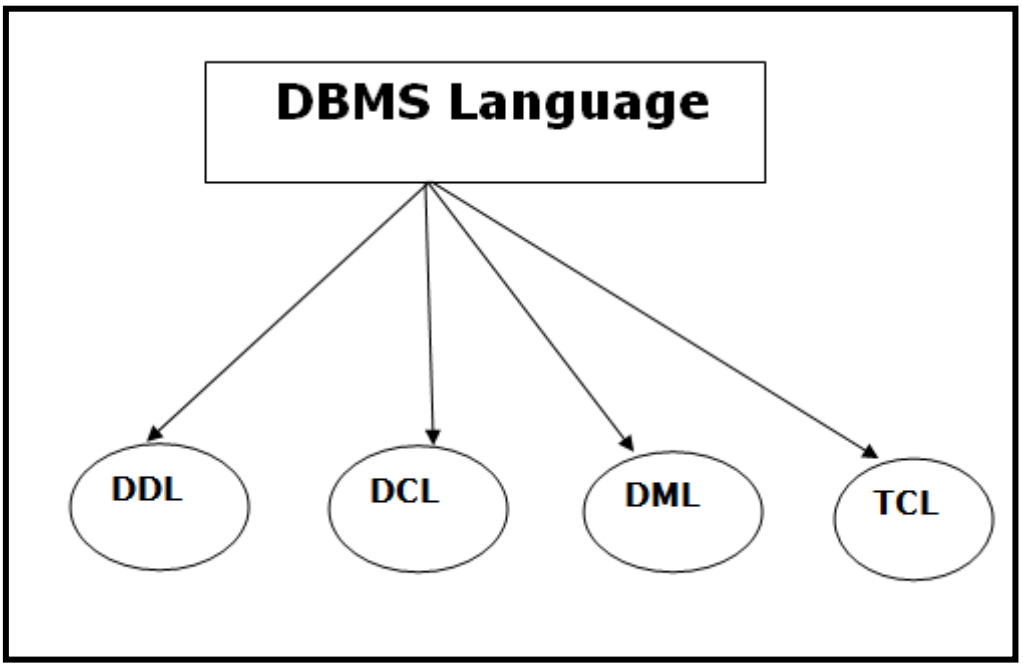
1.5.2 Instances

1. The data which is stored in the ***database at a particular moment of time is called an instance of the database.***
2. Database schema is ***the skeleton of database***.

3. A database schema does not ***contain any data or information***.
4. A database instance is a state of ***operational database with*** data at any given time. It contains a snapshot of the ***database. Database instances tend to change with time.***

1.6 Database Languages:

A DBMS has appropriate languages and interfaces to express database queries and updates. Database languages can be used to read, store and update the data in the database.



1.6.1 Data Definition Language (DDL) :

1. It is used to ***define database structure or pattern***.
2. It is used to ***create schema, tables, indexes, constraints, etc. in the database.***
3. Using the DDL ***statements, you can create the skeleton of the database***.
4. Data definition ***language is used to store the information of*** metadata like the number of tables and schemas, their names, ***indexes, columns in each table, constraints, etc***.

Here are **some tasks** that come under DDL:

1. **Create:** It is used to create ***objects in the database***.
2. **Alter:** It is used to alter the ***structure of the database***.
3. **Drop:** It is used to delete objects ***from the database.***
4. **Truncate**: It is used to remove all ***records from a table.***

5. **Rename:** It is used to ***rename an object.***
6. **Comment:** It is used to ***comment on the data dictionary.***

These commands are used to update the database schema that's why they come under Data definition language.

1.6.2 Data Manipulation Language (DML):

It is used for ***accessing and manipulating data in a database. It handles user requests***.

Here are some tasks that come under DML:

1. **Select:** It is used to ***retrieve data from a database***.
2. **Insert:** It is used to ***insert data into a table.***
3. **Update:** It is used to update ***existing data within a table***.
4. **Delete:** It is used to delete ***all records from a table***.
5. **Merge:** It performs ***UPSERT operation, i.e., insert or*** update operations.
6. **Call:** It is used to call a ***structured query language or a Java subprogram.***
7. **Explain Plan:** It has the ***parameter of explaining data.***
8. **Lock Table:** It ***controls concurrency***.

1.6.3 Data Control Language (DCL):

1. It is used to ***retrieve the stored or saved data.***
2. The DCL ***execution is transactional.***
3. It also has ***rollback parameters.***

Here are **some tasks that** come under DCL:

- **Grant:** It is used to give user ***access privileges to a database***.
- **Revoke:** It is used to take back ***permissions from the user.***

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

1.6.4 Transaction Control Language

1. TCL is used to run the changes ***made by the DML statement***.
2. TCL can be ***grouped into a logical transaction.***

Here are **some tasks that** come under TCL:

- **Commit:** It is used to save the ***transaction on the database.***
- **Rollback:** It is used to restore the ***database to original since the last Commit***.

1.7 Data Models:

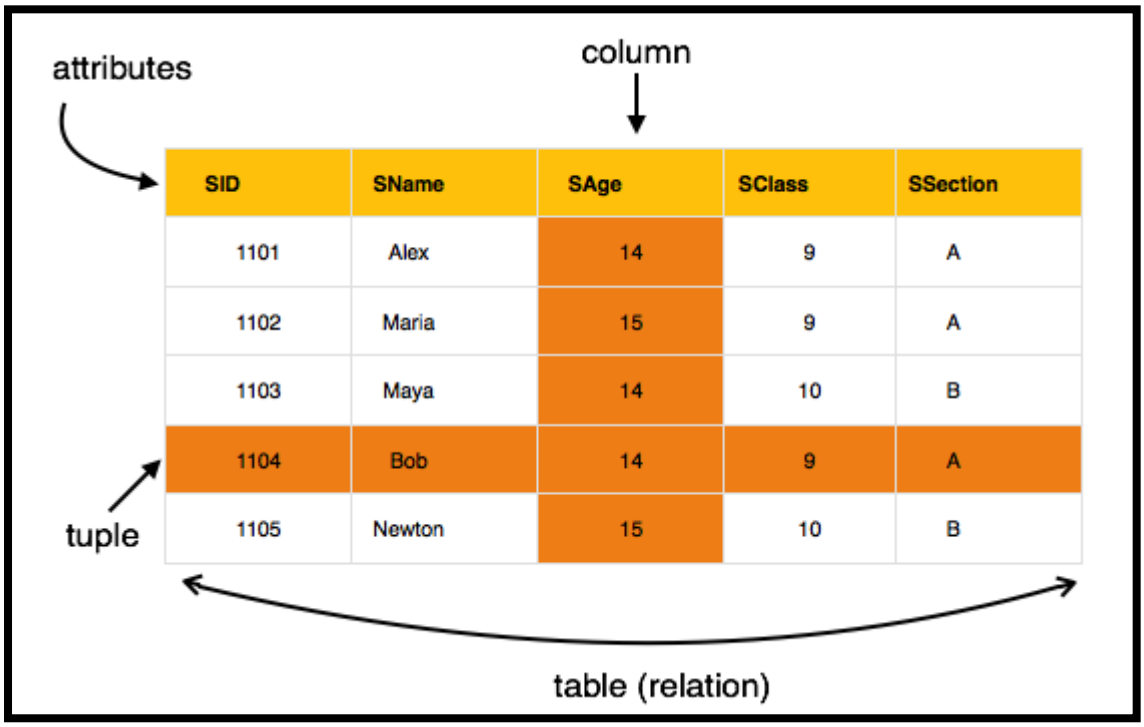
1. Data models define how the ***logical structure of a database is modeled***.
2. Data Models are fundamental entities to ***introduce abstraction in a DBMS***.
3. Data models ***define how data is connected*** to each other and how they are processed and ***stored inside the system.***
4. There are 2 model available
    a. ***ER Model***
    b. ***Relational Model***

1.7.1 ER model

1. ER model stands for an ***Entity-Relationship model.***
2. It is a high-***level data model.***
3. This model is used to define the data ***elements and relationship for a specified*** system.
4. It develops a ***conceptual design for the database.***
5. It also develops a ***very simple and easy to design view*** of data.
6. In ER modeling, the database ***structure is portrayed as a diagram called*** an entity-***relationship diagram.***

1.7.2 Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an n-ary



1.8 Database Design

1.8.1 Entity-Relationship Model

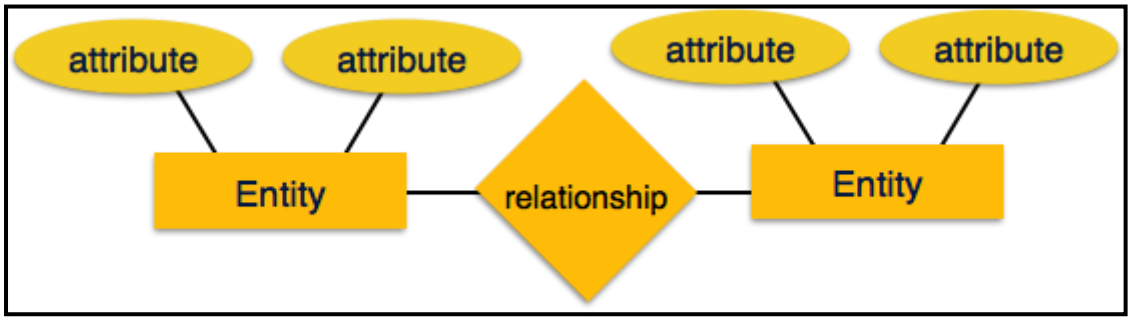
1. Entity-Relationship (ER) Model is based on the ***notion of real-world entities and relationships among them.***
2. While formulating real-***world scenario into the database*** model, the ER Model creates ***entity set, relationship set, general attributes and constraints***.

ER Model is best used for the conceptual design of a database.

ER Model is based on −

- **Entities** and their *attributes.*
- **Relationships** among entities.

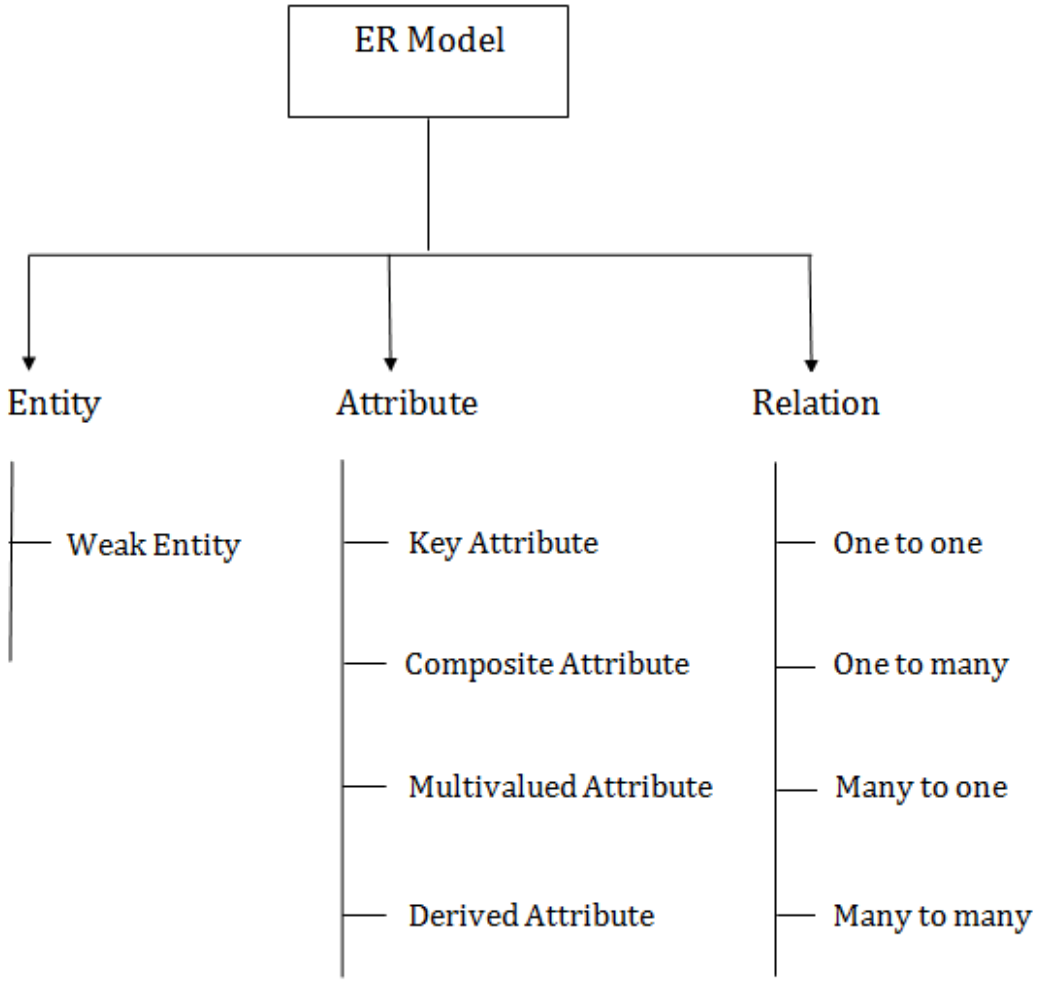
These concepts are explained below.



What is ER Modeling?

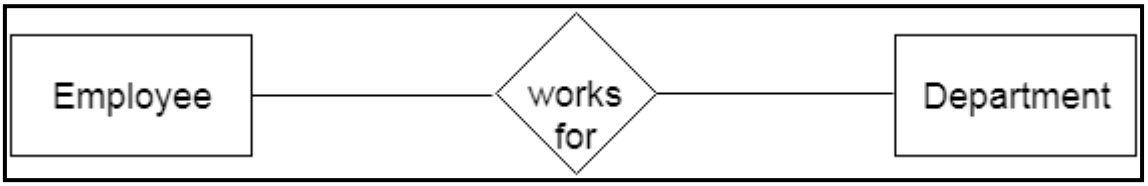
A graphical technique for understanding and organizing the data independent of the actual database implementation

Component of ER Diagram:



1. Entity:

1. An entity may be any ***object, class, person or place. In the ER diagram, an*** entity can be ***represented as rectangles.***
2. Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity



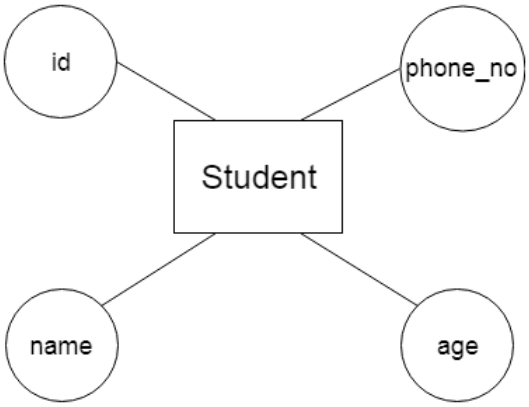
a. Weak Entity

1. An entity that depends on ***another entity called a weak entity***.
2. The weak entity doesn't ***contain any key attribute of its own.*** The weak entity is represented by ***a double rectangle.***

2. Attribute

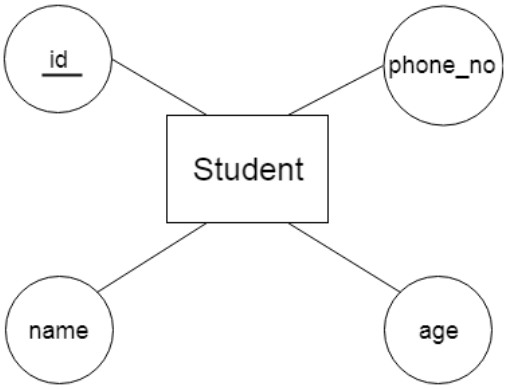
The attribute is used to ***describe the property*** of an entity. Eclipse is used ***to represent an attribute.***

For example, id, age, contact number, name, etc. can be attributes of a student.



a. Key Attribute

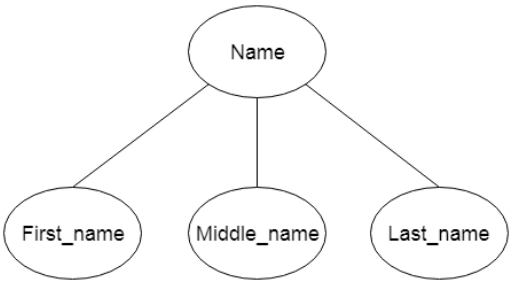
The key attribute is used to ***represent the main characteristics of an entity***. It represents a primary key. The key attribute is ***represented by an ellipse with the text underlined***.



b. Composite Attribute

An attribute that composed ***of many other attributes*** is known as a composite attribute. The ***composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.***

c. Multivalued Attribute

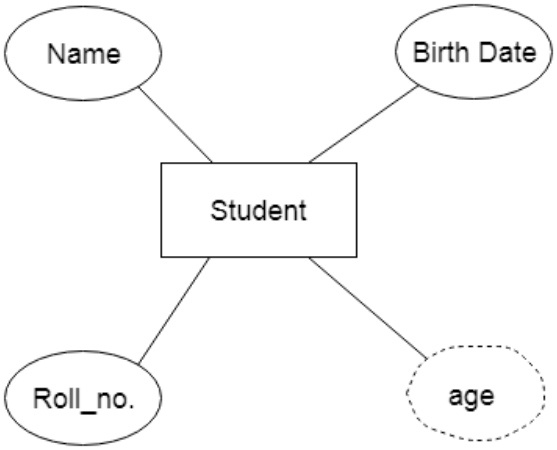


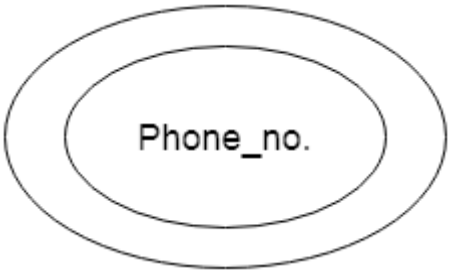
An ***attribute can have more than*** one value. These attributes are ***known as a multivalued attribute***. The d***ouble oval is used to represent multivalued*** attribute.**For example,** a student can have more than one phone number.

d. Derived Attribute

An attribute that ***can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.***

For example, A person's age changes over time and can be derived from another attribute like Date of birth.





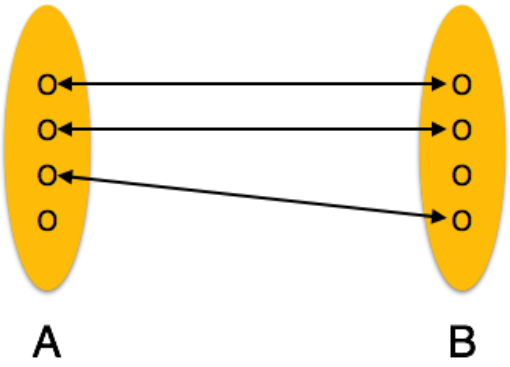
Relationship − The logical association among ***entities is called relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Mapping cardinalities −

- ***one to one***
- ***one to many***
- ***many to one***
- ***many to many***

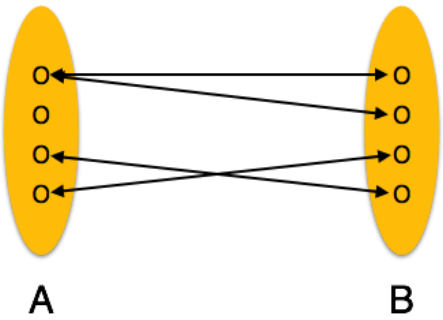
Cardinality defines the ***number of entities in one entity*** set, which can be associated with the number of entities of ***other set via relationship set.***

1. **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



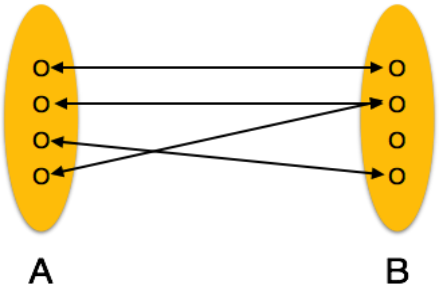
One-to-one

2. **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



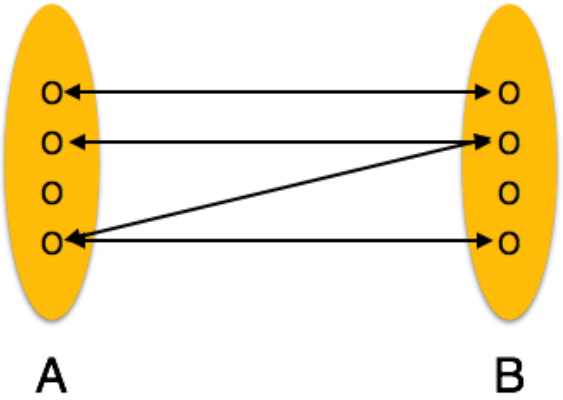
One-to-Many

3. **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



Many -to- one

4. **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



Many -to- Many

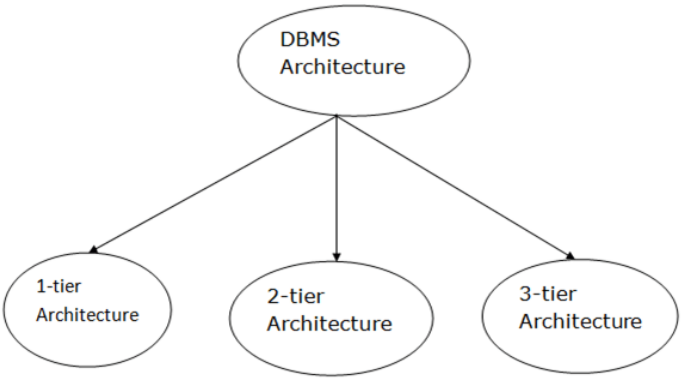
OBJECT-BASED DATA MODEL.

1. Object-oriented programming (especially in Java, C++, or C#) has become the dominant ***software-development methodology.***
2. This led to the development of an object- ***oriented data model that can be seen as*** extending the E-R model with notions of encapsulation, methods (functions), and ***object identity.***

SEMI-STRUCTURED DATA MODEL.

1. The semi-structured ***data model permits the specification of*** data where individual data items of the same type may have different sets of attributes.
2. This is in contrast to the data models ***mentioned earlier, where every*** data item of a particular type must have the same set of attributes.
3. The ***Extensible Markup Language (XML) is widely used to represent semi-*** structured data.

1.9DATABASE ARCHITECTURE



- The DBMS design depends upon its architecture.
- The basic client/server ***architecture is used to deal with a large number of PCs***, web servers, database servers and ***other components that are connected with networks***.
- The ***client/server architecture consists of many*** PCs and a workstation which are connected via the network.
- DBMS ***architecture depends upon how users are connected*** to the database to get their ***request done***

Types of DBMS Architecture

1-Tier Architecture

- In this architecture, the database is ***directly available to the user. It means*** the user can directly sit on uses it.
- Any changes done here will ***directly be done on the database itself. It doesn't provide a handy tool for end users.***
- The 1-Tier architecture is used ***for development of the local application***, where programmers can directly ***communicate with*** the database for the quick response.
- The following user role this tier the are as follows:

  - ✓ ***Naïve Users (Tellers, Agents & Web Users)***
  - ✓ ***Application Programmers***
  - ✓ ***Sophisticated Users (Analysts)***
  - ✓ ***Database Administrators***

2-Tier Architecture

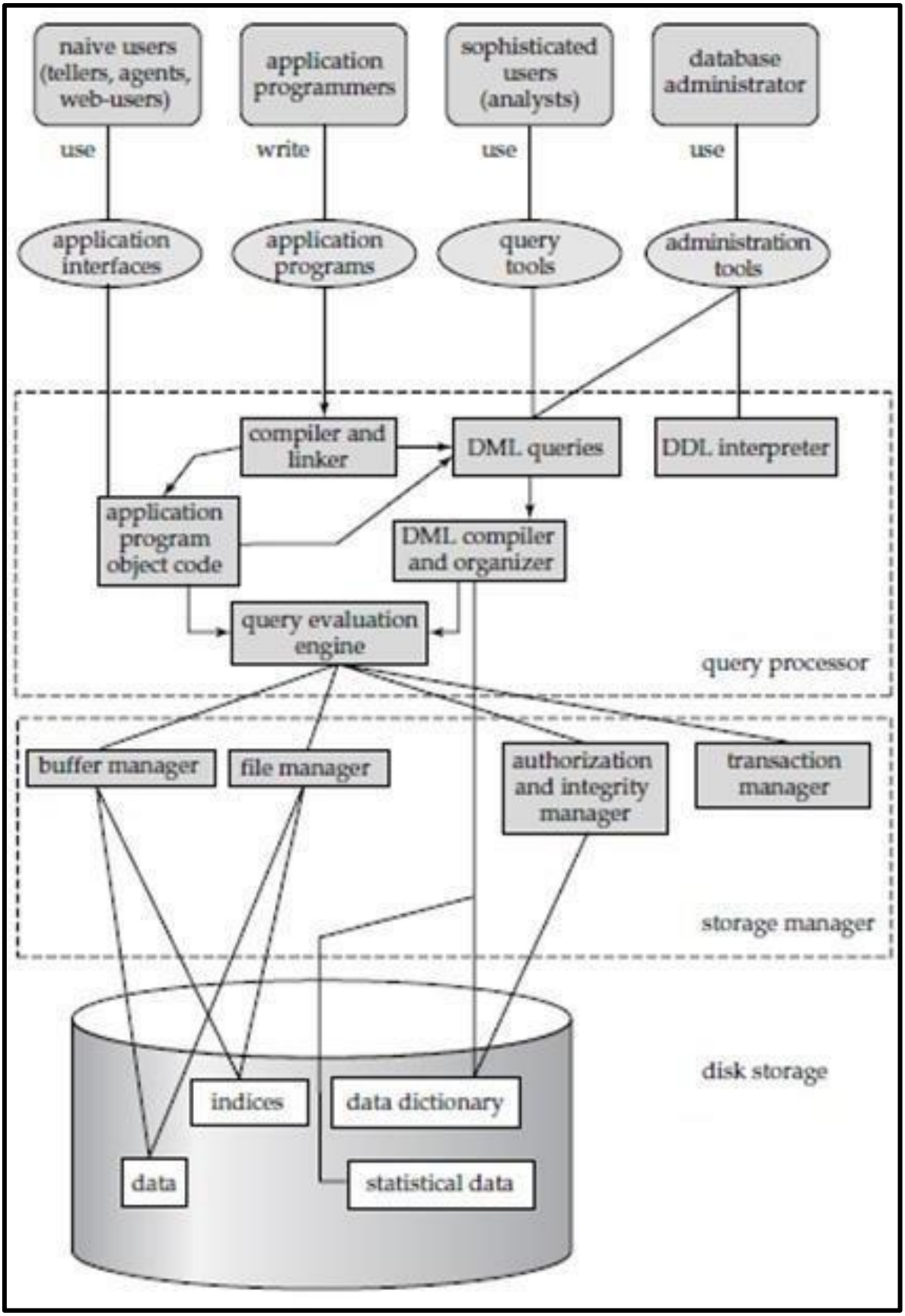
- The 2-Tier architecture ***is same as basic client-server***. In the two-tier architecture, applications on the client end ***can directly communicate with the database at the server side. For this interaction, API's like:*** *ODBC, JDBC* ***are used.***
- The user interfaces and ***application programs are run on the*** client-side.
- The server side ***is responsible to provide the functionalities like: query processing and transaction management.***
- To communicate ***with the DBMS, client-side application*** establishes a connection with the server side.
- The Query Processor have the following :

  - ✓ ***DDL Interpreter***
  - ✓ ***DML Compiler***
  - ✓ ***Query Evaluation Engine***

3-Tier Architecture

- The 3-Tier ***architecture contains another layer between the client and server.*** In this architecture, client can't directly communicate with the server.
- The application ***on the client-end interacts with an application server which further communicates with the database system.***
- End user has no idea ***about the existence of the database beyond*** the application server. ***The database also has no idea about any other user beyond the application.***
- ***The 3-Tier architecture is used in case of large web application.***
- This Tier is responsible for the Storage and it have Storage Manager that have the following functions

  - ✓ ***Authorization and Integrity Manager***
  - ✓ ***Transaction Manager***
  - ✓ ***File Manager***
  - ✓ ***Buffer Manager***


naive users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

DATA DICTIONARY

1. We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships.
2. The DBMS data dictionary provides the DBMS with its self describing characteristic. In effect, the data dictionary resembles and X-ray of the company's entire data set, and is a crucial element in the data administration function.

For example, the data dictionary typically stores descriptions of all:

1. Data elements that are ***define in all tables of all databases***. Specifically the data dictionary stores ***the name, datatypes, display formats, internal storage formats***, and ***validation rules. The data dictionary*** tells where an element is used, by whom it is used and so on.
2. Tables define in all ***databases. For example, the data*** dictionary is likely to store the name of the table ***creator, the date of creation access authorizations,*** the number of columns, and so on.
3. Indexes ***define for each database tables. For each index the DBMS stores*** at least the index name the attributes used, the location, specific index characteristics and the creation date.
4. Define ***databases: who created each database, the date of creation where*** the database is located, who the
5. ***DBA is and so on.***
6. End users and ***The Administrators of the data base***
7. Programs that ***access the database including screen formats, report formats application formats, SQL queries and so on.***
8. Access authoriz***ation for all users of all databases***.
9. Relationships ***among data elements which elements are involved: whether the relationship are mandatory or optional, the connectivity and*** cardinality and so on.

Database Users and User Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

NAIVE USERS

1. Are unsophisticated ***users who interact with the system*** by invoking one of the application ***programs that have been written previously.***
2. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*.

APPLICATION PROGRAMMERS

1. Are computer ***professionals who write application programs***. Application programmers can ***choose from many tools to develop user interfaces.***
2. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program.

SOPHISTICATED USERS

1. Interact with the ***system without writing programs. Instead***, they form their requests in a database query language.
2. They submit each ***such query to a query processor, whose function*** is to break down DML statements into instructions that the storage manager understands. Analysts who submit ***queries to explore data in the database*** fall in this category.

ONLINE ANALYTICAL PROCESSING (OLAP)

1. Tools simplify ***analysts' tasks by letting them view summaries*** of data in different ways.
2. For instance, ***an analyst can see total sales*** by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total ***sales of each product in each region***).

QUERY PROCESSOR:

The query processor components include

DDL interpreter, which interprets ***DDL statements and records the definitions in the data dictionary.***

DML compiler, which translates ***DML statements in a query language*** into an evaluation plan ***consisting of low-level instructions that the query evaluation*** engine understands.

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

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

STORAGE MANAGER:

A ***storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted*** to the system. The storage manager is responsible for the interaction with the file manager.

The storage manager components include:

1. **Authorization and integrity manager**, which tests for ***the satisfaction of*** integrity constraints and checks ***the authority of users to*** access data.
2. **Transaction manager**, which ***ensures that the database remains in a*** consistent (correct) state ***despite system failures, and that concurrent transaction executions*** proceed without conflicting.
3. **File manager**, which ***manages the allocation of space on disk storage and the data structures used to represent information stored on disk.***
4. **Buffer manager**, which is responsible for ***fetching data from disk storage*** into main memory, and ***deciding what data to cache in main memory***. The buffer manager is ***a critical part of the database system, since*** it enables the database to handle data sizes that ***are much larger than the size of main memory.***

TRANSACTION MANAGER:

A ***transaction is a collection of operations that performs a single*** logical function in a database application. ***Each transaction is a unit of both atomicity and*** consistency. Thus, we require that ***transactions do not violate any database-consistency constraints.***

1.10 SUPERKEY

A **superkey** is a set of ***one or more attributes that, taken collectively***, allow us to ***identify uniquely a tuple in the relation***. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey. The *name* attribute of *instructor*, on the other hand, is not a superkey, because several instructors might have the same name.

A superkey may ***contain extraneous attributes. For example, the combination*** of *ID* and *name* is a superkey for the relation *instructor*. If *K* is a superkey, then so is any superset of *K*. We are often interested in superkeys for ***which no proper subset is a superkey. Such minimal superkeys are called candidate keys***.

1.11 HISTORY OF DATABASE SYSTEMS.

1960S

1. Computerized database started in the ***1960s, when the use*** of computers became a more cost-effective option for private organizations.
2. ***There were two popular data models in this decade: a network model called CODASYL and a hierarchical model called IMS.***
3. One database system that proved to be a commercial success was the SABRE system that was used by IBM to help American Airlines manage its reservations data.

1970 TO 1972

1. E.F. ***Codd published an important paper to propose the*** use of a relational database model, and his ideas changed ***the way people thought about databases***.
2. In his model, the ***database's schema, or logical organization***, is disconnected from physical information storage, and this became the standard principle ***for database systems.***

1970S

1. Two major ***relational database system prototypes were created*** between the years 1974 and ***1977, and they were the Ingres, which was developed*** at UBC, and System R, ***created at IBM San Jose.***
2. Ingres used a query language known as ***QUEL, and it led to the creation of*** systems such as Ingres ***Corp., MS SQL Server, Sybase, Wang's PACE, and Britton-Lee.*** On

the other hand, ***System R used the SEQUEL query language***, and it contributed to the development of SQL/DS, DB2, ***Allbase, Oracle, and Non-Stop*** SQL.

3. It was also in this ***decade that Relational Database Management*** System, or RDBMS, ***became a recognized term.***

1976

1. A new database ***model called Entity-Relationship, or ER, was proposed*** by P. Chen this year.
2. This model made it ***possible for designers to focus on data application,*** instead of logical table structure.

1980S

1. Structured Query Language***, or SQL, became the standard query language.***

2. Relational database systems became a commercial success as the ***rapid increase in computer sales boosted t***he database market, and this caused a major decline in the popularity of network and hierarchical database models.
3. DB2 became the ***flagship database product for IBM, and*** the introduction of the IBM PC resulted in the establishments of many new database companies and the development of ***products such as PARADOX, RBASE 5000, RIM, Dbase*** III and IV, OS/2 Database Manager, and Watcom SQL.

EARLY 1990S

1. After a database ***industry shakeout, most of the surviving companies sold*** complex database products at high prices.
2. Around this time, new client tools for applic***ation development were released, and these included the Oracle*** Developer, PowerBuilder, VB, and others.

3. A number of tools for personal productivity, such as ODBC and Excel/Access, were also developed. ***Prototypes for Object Database Management Systems***, or ODBMS, were created ***in the early 1990s.***

MID 1990S

1. The advent of the Internet led to ***exponential growth of the database*** industry.
2. Average desktop ***users began to use client-server database systems to access*** computer systems that contained legacy data.

LATE 1990S

1. Increased investment in ***online businesses resulted in a rise*** in demand for Internet database connectors, such ***as Front Page, Active Server Pages, Java Servelets,*** Dream Weaver, ***ColdFusion, Enterprise Java Beans, and Oracle Developer 2000.***
2. ***The use of cgi, gcc, MySQL, Apache, and*** other systems ***brought open source solution to the Internet. With the increased u***se of point-of-sale technology, online transaction processing and online analytic processing began to come of age.

2000S

1. Although the Internet ***industry experienced a decline in the*** early 2000s, database applications ***continue to grow.***
2. New ***interactive applications were developed for PDAs, point-of-sale transactions, and consolidation of vendors. Presently, the three leading database*** companies in the western world ***are Microsoft, IBM, and Oracle***.

TODAY

1. Today, databases are ***everywhere and are used to enhance our day***-to-day life. From personal cloud storage to predicting ***the weather, many of the services we utilize today*** are possible due to databases.
2. Presently, there ***are many new players in the non-relational database*** space offering specific solutions. Some of the current relational databases include giants such as ***Oracle, MySQL, and DB2.***
3. We're also seeing ***new trends emerging that focus on making powerful*** technology accessible to everyone.
4. Quick Base is an ***online database platform built on a relational database,*** which gives users of any skill ***level the ability to create custom*** applications using the power of a relational database, but with ***the simplicity of a point-and-click user interface***.

1.12 DATA MINING

There is a huge amount of data available in the Information Industry. This data is of no use until it is converted into useful information. It is necessary to analyze this huge amount of data and extract useful information from it.

WHAT IS DATA MINING?

Data Mining is defined as ***extracting information from huge sets of data. In*** other words, we can say that data mining ***is the procedure of mining knowledge from data. The information*** or knowledge extracted so can be used for any of the following applications −

- ***Market Analysis***
- ***Fraud Detection***
- ***Customer Retention***
- ***Production Control***
- ***Science Exploration***

DATA MINING APPLICATIONS

Data mining is highly useful in the following domains −

- ***Market Analysis and Management***
- ***Corporate Analysis & Risk Management***
- ***Fraud Detection***

Apart from these, data mining can also be used in the areas of production control, customer retention, science exploration, sports, astrology, and Internet Web Surf-Aid

MARKET ANALYSIS AND MANAGEMENT

Listed below are the various fields of market where data mining is used −

- **Customer Profiling** − Data mining ***helps determine what kind of people*** buy what kind of products.
- **Identifying Customer Requirements** − Data ***mining helps in identifying*** the best products for different customers. It ***uses prediction to find the factors that may attract*** new customers.
- **Cross Market Analysis** − Data mining ***performs Association/correlations*** between product sales.
- **Target Marketing** − Data ***mining helps to find clusters of model customers*** who share the same ***characteristics such as interests, spending habits, income***, etc.

- **Determining Customer purchasing pattern** − Data mining ***helps in determining*** customer ***purchasing pattern.***
- **Providing Summary *Information*** − ***Data mining provides us various*** multidimensional summary reports.

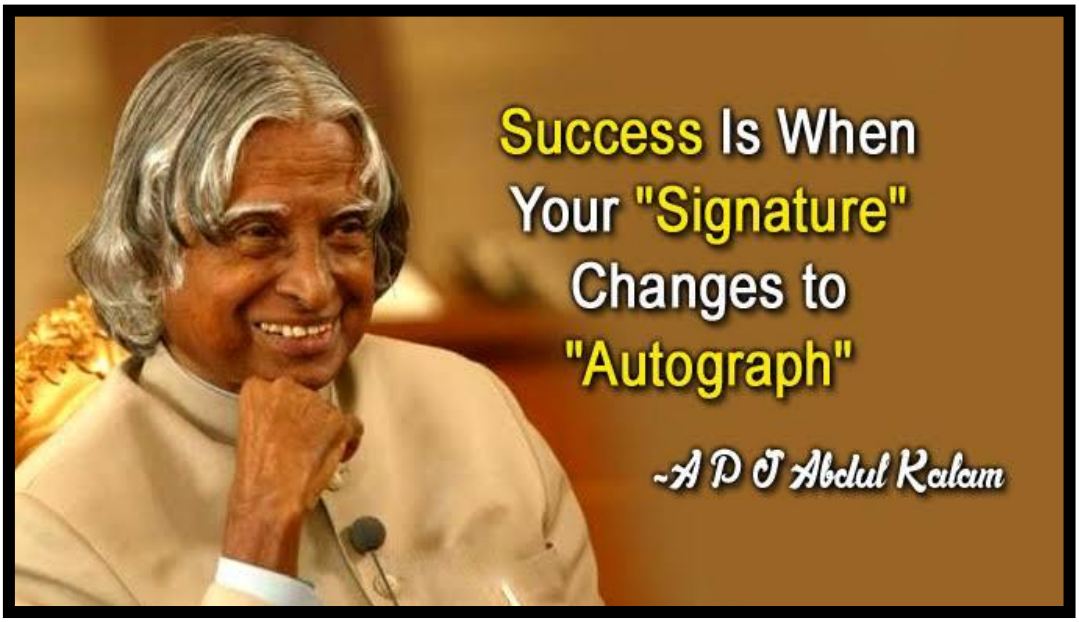
CORPORATE ANALYSIS AND RISK MANAGEMENT

Data mining is used in the following fields of the Corporate Sector −

- **Finance Planning and Asset Evaluation** − It involves ***cash flow analysis*** and prediction, ***contingent claim analysis to evaluate*** assets.
- **Resource Planning** − It ***involves summarizing and comparing*** the resources and spending.
- **Competition** − It involves ***monitoring competitors and market directions***.

FRAUD DETECTION

1. Data mining is also used in the fields of credit ***card services and telecommunication*** to detect frauds.
2. In fraud telephone calls, it ***helps to find the destination of the call, duration*** of the call, time of the day or ***week, etc. It also analyzes the patterns that deviate from expected norms.***



Unit II

Relational Model: Structure of Relational Databases -Database Schema - Keys – Schema Diagrams - Relational Query Languages - Relational Operations Fundamental Relational Algebra Operations Additional Relational-Algebra Operations- Extended Relational-Algebra Operations - Null Values - Modification of the Database.

2.1 Relational Databases

1. The relational model represents the ***database as a collection of relations***. A relation is nothing ***but a table of values***.
2. Every row in the ***table represents a collection*** of related data values.
3. These rows in the ***table denote a real-world entity or relationship***.

2.2 Structure of Relational Databases

1. A relational database ***consists of a collection of tables, each having a unique name***.
2. A row in a table represents a ***relationship among a set of values***.
3. Thus a table ***represents a collection of relationships***.

In Relational Data base model ***records are stored into tables***. Relational data model is easier to understand than the hierarchal ***data models and network data*** models. Relational data model provides a logical view of the data ***and its relationship among*** other data.

Characteristics of a Table:

1. A table is ***composed of rows and columns***.
2. Each row of the table represents ***one entity (tuple) in the entity set***.
3. Each column represents an ***attribute and each column has distinct*** name.
4. Each cell ***represents a single value***.
5. All values in a column must ***have same data format***.
6. Each column has a ***specified range of values which is called*** domain.
7. The order of the rows and columns is ***immaterial to the DBMS***.
8. Each table must have an attribute or group of ***attributes that uniquely identified*** each ***row.***

The following Student table shows above characteristics.

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS
321452	Bowser	William	C	12-Feb-1975	42	So
324257	Smithson	Anne	K	15-Nov-1981	81	Jr
324258	Brewer	Juliette		23-Aug-1969	36	So
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr
324273	Smith	John	D	30-Dec-1958	102	Sr
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr
324299	Smith	John	B	30-Nov-1986	15	Fr

2.3Relation Schema:

The relation schema describes the ***column heads for the table***. The ***schema specifies*** the relation's name, the name of ***each field (column, attribute) and the*** 'domain' of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

Example:

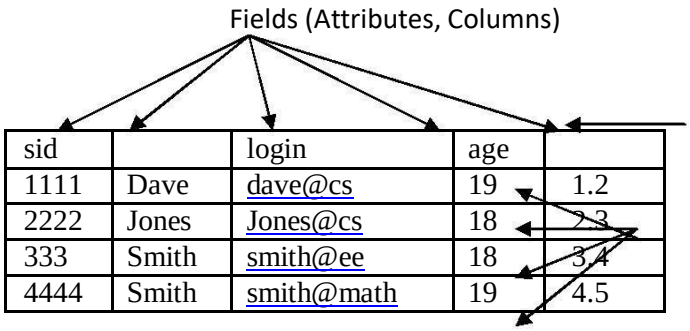
Student information in a university database to illustrate the parts of a relation schema. Students (Sid: string, name: string, login: string, age: integer, gross: real)

This says that the field named 'sid' has a domain named 'string'.The set of values associated with domain 'string' is the set of all character strings.

Relation Instance:

1. This is a table specifying ***the information***.
2. An instance of a relation is a ***set of 'tuples', also called*** 'records', in which each tuple has the same number of ***fields as the relation schemas***.
3. A relation instance can be ***thought of as a table in which each tuple is a row and all rows have the same number of fields.***
4. The relation instance is also called as 'relation'.***Each relation is defined to be a set of unique tuples or rows.***

Example:



sid		login	age	
1111	Dave	dave@cs	19	1.2
2222	Jones	Jones@cs	18	2.3
333	Smith	smith@ee	18	3.4
4444	Smith	smith@math	19	4.5

Field names

Tuples (Records, Rows)

This example is an instance of the students relation, which consists 4 tuples and 5 fields. No two rows are identical.

2.4KEYS

Key is an attribute or ***collection of attributes that uniquely identifies an entity among entity set.***

For example, the roll_number of a student makes him/her identifiable among students.

1. **Super Key** − A set of ***attributes (one or more) that collectively identifies an entity in an entity set.***

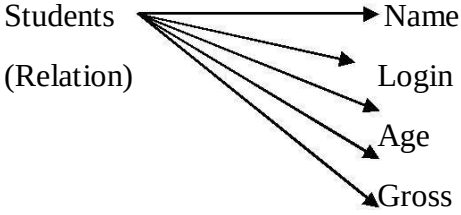
2. **Candidate Key** − A minimal super ***key is called a candidate key. An entity set may have more than one candidate key.***

3. **Primary Key** − A primary key is one of ***the candidate keys chosen by the database designer to uniquely identify the entity set.***

2.4.1 Super Key

The set of fields that ***contains a key is called as a 'super ke***y'. The set of ***1 or more attributes that allows us to identify uniquely an entity in t***he entity set . A ***super key specifies*** a uniqueness ***constraint that no 2 distinct tuples*** can have the same value. Every relation has at least 1 default ***super key as the set of all attributes.***

Example:



One of the super key = {Sid, Name, Login,Gross}

2.4.2 CANDIDATE KEY:

1. A set of fields that uniquely ***identifies a tuple according to*** a key ***constraint is calle***d as a '***Candidate Key' for the relation.***
2. This is also called as a 'key'.

From the definition of candidate key, we have,

Two distinct tuples in a ***legal instance cannot have identical values*** in all the fields of a key.i.e, in any legal instance, the values in the key fields ***uniquely identify a tuple*** in the instance.The values in ***the key fields uniquely identify a tuple in the instance..No subset of the set of fields in key is a unique identifier for a tuple***,

i.e., the set of fields {sid, name} is not a key for Students. A relation schema may have more than key.

Example: In the above Students relation, the 'sid' field is a candidate key.

{sid}.

The value of a ***key attribute can be used to identify uniquely each*** tuple in the relation.

'A set of attributes constituting a key' is ***a property of the relation*** schema. A key is determined from the meaning of attributes.

Every relation is ***guaranteed to have a key. Since a relation is a set of tuples***, the set of all fields is always a super key.

2.4.3 PRIMARY KEY:

1. This is also a candidate key, whose values ***are used to identify tuples in the*** relation. It is common to designate one of ***the candidate keys as a primary key of the relation***.
2. The attributes that ***form the primary key of a relation schema are*** underlined.
3. It is used to denote a ***candidate key that is chosen by the database*** designer as the principal means of ***identifying entities with an entity set.***

Example:

'Sid' of Students relation.

2.5 RELATIONAL QUERY LANGUAGES

Relational query ***languages use relational algebra to break the*** user requests and instruct the ***DBMS to execute the requests***. It is the language by which user communicates with the database. These relational query languages can ***be procedural or non-procedural.***

PROCEDURAL QUERY LANGUAGE

A procedural query ***language will have set of queries instructing the DBMS*** to ***perform various transactions in the sequence to meet the user request***. For example, *get_CGPA* procedure will have various queries to get the marks of student in each subject, calculate the total marks, and then decide the CGPA based on his total marks.

NON-PROCEDURAL QUERY LANGUAGE

Non-procedural queries will ***have single query on one or more tables to get result from the database***. For example, get the name and address of the student with particular ID will have single query on STUDENT table.

2.6RELATIONAL ALGEBRA

Relational algebra is a ***procedural query language***. It takes one or more relations / tables and performs the ***operation and produce the result. This result is also considered as a new table or relation.***

Relational algebra will have ***operators to indicate the operations. This*** algebra can be applied on single relation – called ***unary*** or can be applied on two tables – ***called*** *binary.*

2.6.1 SELECT (Σ)

1. **Select (σ) –** This is a unary relational operation.
2. This operation pulls the horizontal subset (subset of rows) of the relation that satisfies the conditions.
3. This can use operators like <, >, <=, >=, = and != to filter the data from the relation. It can also use logical AND, OR and NOT operators to combine the various filtering conditions. This operation can be represented as below:

2.6.2 $\sigma_p(r)$

1. Where σ is the symbol for select ***operation, r represents the relation/table, and p is the*** logical ***formula or the filtering conditions to get the subset. Let us see an example as below:***

$\sigma_{STD\_NAME = "James"}$ (STUDENT)

What does above relation algebra do? It selects the record/tuple from the STUDENT table with Student name as 'James'

$\sigma_{dept\_id = 20 \text{ AND } salary>=10000}$ (EMPLOYEE) – Selects the records from EMPLOYEE table with department ID = 20 and employees whose salary is more than 10000.

2.6.3 PROJECT (∏)

1. **Project (∏) –** This is a ***unary operator and is similar to select*** operation above. It creates the subset of relation based ***on the conditions*** specified.
2. Here, it selects only ***selected columns/attributes from*** the relation- vertical subset of relation. The select operation above ***creates subset of relation but for*** all the attributes in the relation. ***It is denoted as below:***

$\prod_{a1, a2, a3}$ (r)

Where ∏ is the operator for projection, r is the relation and a1, a2, a3 are the attributes of the relations which will be shown in the resultant subset.

$\prod_{std\_name, address, course}$ (STUDENT) – This will select all the records from STUDENT table but only selected columns – std_name, address and course. Suppose we have to select only these 3 columns for particular student then we have to combine both project and select operations.

$\prod_{STD\_ID, address, course}$ ($\sigma_{STD\_NAME = "James"}$(STUDENT)) – this selects the record for 'James' and displays only std_ID, address and his course columns. Here we can see two unary operators are combined, and it has two operations performing. First it selects the tuple from STUDENT table for 'James'. The resultant subset of STUDENT is also considered as intermediary relation. But it is temporary and exists till the end of this operation. It then filters the 3 columns from this temporary relation.

2.6.4 RENAME (P)

1. **Rename (ρ) –** This is a unary ***operator used to rename the tables and columns of a relation.***
2. When we perform self ***join operation, we have to differentiate*** two same tables. In such case rename operator on tables comes into picture.
3. When we join two or more tables and if ***those tables have same column*** names, then it is always better to ***rename the columns to differentiate them. This occurs when we perform Cartesian product operation.***

$\rho_R(E)$

Where ρ is the rename operator, E is the existing relation name, and R is the new relation name.

ρ STUDENT (STD_TABLE) – Renames STD_TABLE table to STUDENT

Let us see another example to rename the columns of the table. If the STUDENT table has ID, NAME and ADDRESS columns and if they have to be renamed to STD_ID, STD_NAME, STD_ADDRESS, then we have to write as follows.

$\rho_{STD\_ID,\ STD\_NAME,\ STD\_ADDRESS}(STUDENT)$ – It will rename the columns in the order the names appear in the table

2.6.5 CARTESIAN PRODUCT (X)

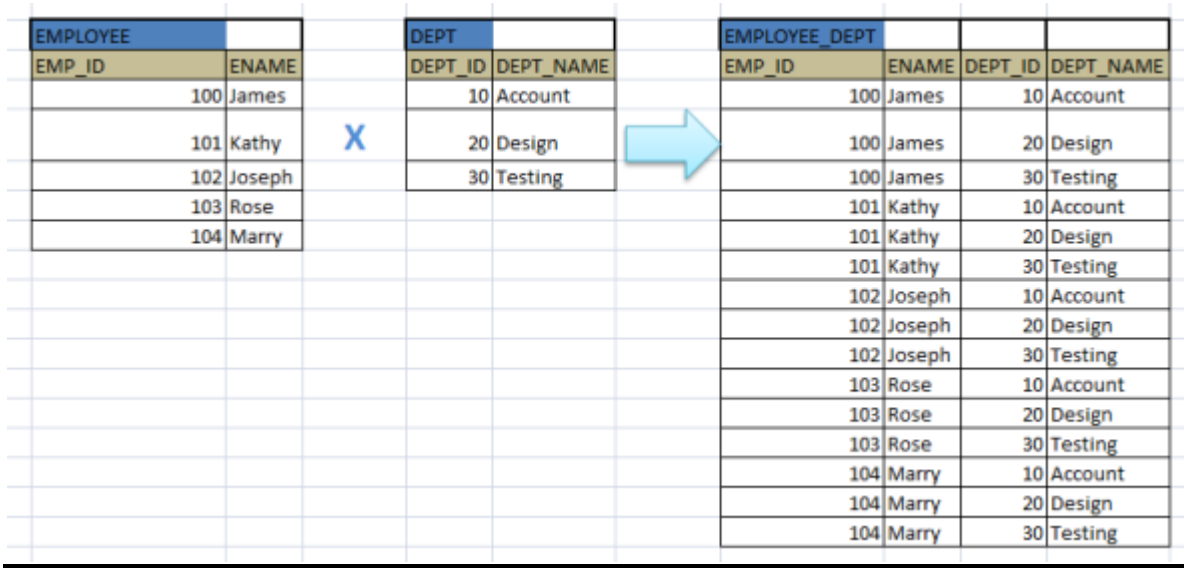
1. **Cartesian product (X): –** This is a ***binary operator. It combines the tuples of two relations into one relation.***

RXS

Where R and S are two relations and X is the operator. If relation R has m tuples and relation S has n tuples, then the resultant relation will have **mn** tuples. For example, if we perform cartesian product on EMPLOYEE (5 tuples) and DEPT relations (3 tuples), then we will have new tuple with 15 tuples.

EMPLOYEE X DEPT

This operator will simply create a pair between the tuples of each table. i.e.; each employee in the EMPLOYEE table will be mapped with each department in DEPT table. Below diagram depicts the result of cartesian product.



EMPLOYEE

EMP_ID	ENAME
100	James
101	Kathy
102	Joseph
103	Rose
104	Marry

X

DEPT

DEPT_ID	DEPT_NAME
10	Account
20	Design
30	Testing

EMPLOYEE_DEPT

EMP_ID	ENAME	DEPT_ID	DEPT_NAME
100	James	10	Account
100	James	20	Design
100	James	30	Testing
101	Kathy	10	Account
101	Kathy	20	Design
101	Kathy	30	Testing
102	Joseph	10	Account
102	Joseph	20	Design
102	Joseph	30	Testing
103	Rose	10	Account
103	Rose	20	Design
103	Rose	30	Testing
104	Marry	10	Account
104	Marry	20	Design
104	Marry	30	Testing

2.6.6. Union (U)

Union (U) – It is a binary operator, ***which combines the tuples of two relations. It is denoted by*** **R U S**

Where R and S are the relations and U is the operator.

DESIGN_EMPLOYEE U TESTING_EMPLOYEE

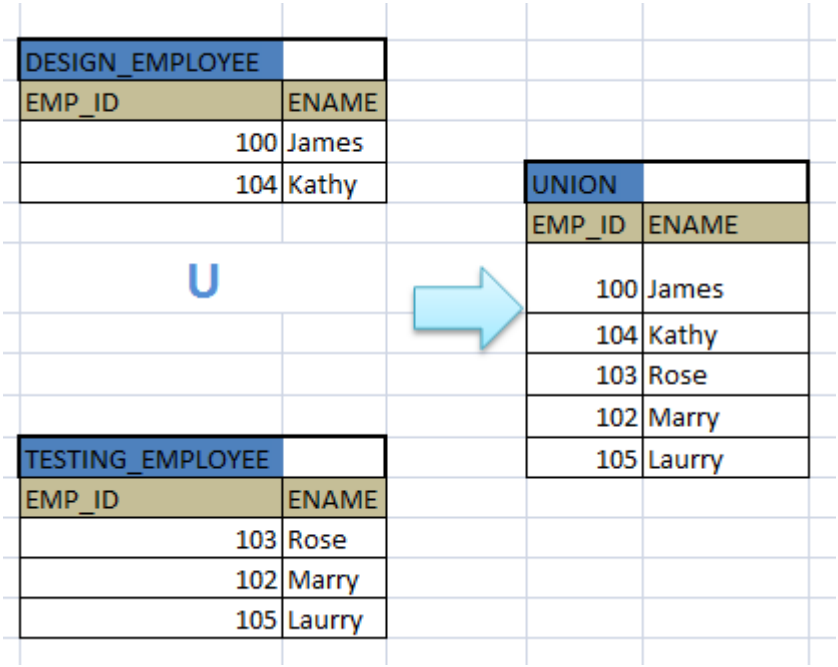
Where DESIGN_EMPLOYEE and TESTING_EMPLOYEE are two relations.

It is different from cartesian product in:

- Cartesian product combines the attributes of two relations into one relation whereas Union combines the tuples of two relations into one relation.
- In Union, both relations should have same number of columns. Suppose we have to list the employees who are working for design and testing department. Then we will do the union on employee table. Since it is union on same table it has same number of attributes. Cartesian product does not concentrate on number of attribute or rows. It blindly combines the attributes.

- In Union, both relations should have same types of attributes in same order. In the above example, since union is on employee relation, it has same type of attribute in the same order.

It need not have same number of tuples in both the relation. If there is a duplicate tuples as a result of union, then it keeps only one tuple. If a tuple is present in any one relation, then it keeps that tuple in the new relation. In the above example, number of employees in design department need not be same as employees in testing department. Below diagram shows the same. We can observe that it combines the table data in the order they appear in the table.



We would not able to join both these tables if the order of columns or the number of columns were different.

2.6.7 SET-DIFFERENCE (-)

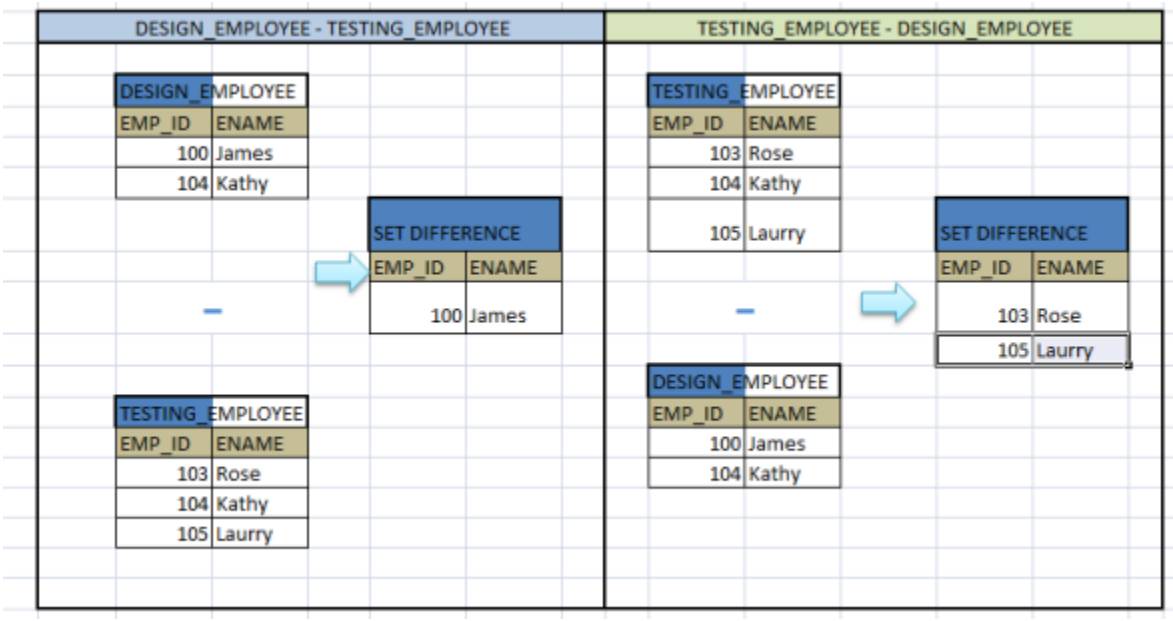
Set-difference (-) – This is a ***binary operator. This operator creates a new relation with tuples that are in one relation but not in other relation. It is denoted by '-'symbol.***

R – S

Where R and S are the relations.

Suppose we want to retrieve the employees who are working in Design department but not in testing.

DESIGN_EMPLOYEE –TESTING_EMPLOYEE



There are additional relational operations based on the above fundamental operations. Some of them are:

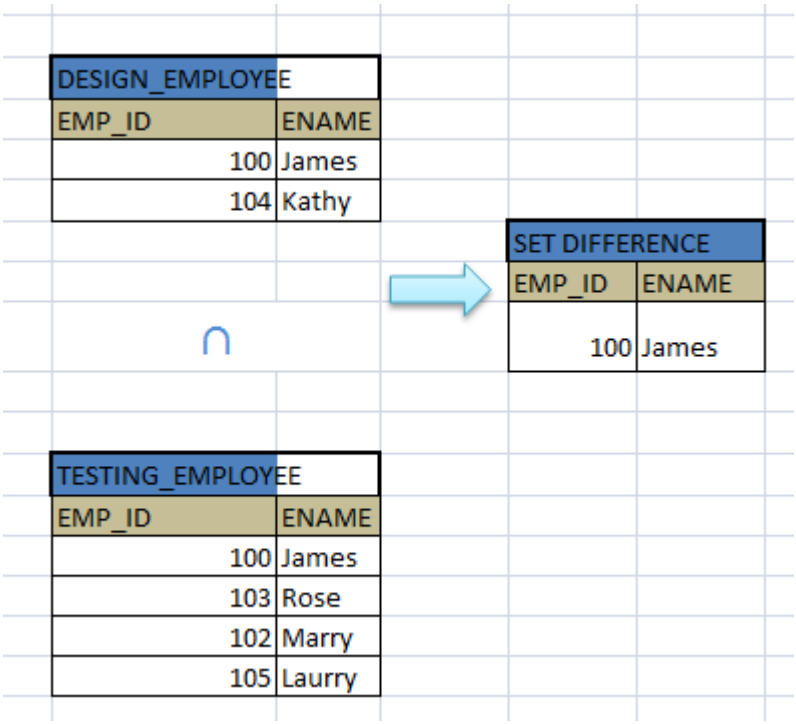
2.6.7 SET INTERSECTION

1. **Set Intersection –** This operation ***is a binary operation. It results in a relation with tuples that are in both the relations. It is denoted by*** **'∩ '.**

   R∩S

Where R and S are the relations. It picks all the tuples that are present in both R and S, and results it in a new relation.

Suppose we have to find the employees who are working in both design and testing department. If we have tuples as in above example, the new result relation will not have any tuples. Suppose we have tuples like below and see the new relation after set difference.



DESIGN_EMPLOYEE

EMP_ID	ENAME
100	James
104	Kathy

∩

TESTING_EMPLOYEE

EMP_ID	ENAME
100	James
103	Rose
102	Marry
105	Laurry

SET DIFFERENCE

EMP_ID	ENAME
100	James

This set intersection can also be written as a combination of set difference operations.

R ∩ S → R-(R-S)

i.e.; it evaluates R-S to get the tuples which are present only in R and then it gets the record which are present only in R but not in new resultant relation of R-S.

In above example of employees,

DESIGN_EMPLOYEE – (DESIGN_EMPLOYEE – TESTING_EMPLOYEE)

It first filters only those employees who are only design employees – (104, Kathy). This result is then used to find the difference with design employee. This will find those employees who are design employees but not in new result – (100, James). Thus it gives the result tuple which is both designer and tester. We can see here fundamental relational operator is used twice to get set intersection. Hence this operation is not fundamental operation.

2.6.8 ASSIGNMENT

1. **Assignment –** As the name indicates, the assignment operator '←' is used to assign the result of a relational operation to temporary relational variable. This is useful when there is multiple steps in relational operation and handling everything in one single expression is difficult. Assigning the results into temporary relation and using this temporary relation in next operation makes task simple and easy.

T ← S – denotes relation S is assigned to temporary relation T

A relational operation $\prod_{a1, a2}(\sigma_p(E))$ with selection and projection can be divided as below.

$T \leftarrow \sigma_p(E)$

$S \leftarrow \prod_{a1, a2}(T)$

Our example above in projection for getting STD_ID, ADDRESS and COURSE for the Student 'James' can be re-written as below.

$\prod_{STD\_ID, address, course}(\sigma_{STD\_NAME = "James"}(STUDENT))$

↓

$T \leftarrow \sigma_{STD\_NAME = "James"}(STUDENT)$

$S \leftarrow \prod_{STD\_ID, address, course}(T)$

2.6.9 NATURAL JOIN

1. **Natural join –** As we have seen above, cartesian product simply combines the attributes of two relations into one. But the new relation will not have correct tuples. It has only combinations of tuples.
2. In order to get the correct tuples, we have to use selection operation on the cartesian product result. This set of operations – cartesian product followed by selection – is combined into one relation called natural join. It is denoted by ∞

R∞S

Suppose we want to select the employees who are working for department 10. Then we will perform the cartesian product on the EMPLOYEES and DEPT and find the DEPT_ID in both relations matching to 10. The same is done with natural join as

$\sigma_{EMPLOYEE.DEPT\_ID = DEPT>DEPT\_ID\ AND\ EMPLOYEE.DEPT\_ID = 10}$**(EMPLOYEE X DEPT)**

Same can be written using natural join as **EMPLOYEE ∞ DEPT**

EMPLOYEE_DEPT				
EMP_ID	ENAME	DEPT_ID	DEPT_ID	DEPT_NAME
100	James	10	10	Account
100	James	10	20	Design
100	James	10	30	Testing
101	Kathy	20	10	Account
101	Kathy	20	20	Design
101	Kathy	20	30	Testing
102	Joseph	30	10	Account
102	Joseph	30	20	Design
102	Joseph	30	30	Testing
103	Rose	10	10	Account
103	Rose	10	20	Design
103	Rose	10	30	Testing
104	Marry	20	10	Account
104	Marry	20	20	Design
104	Marry	20	30	Testing

EMPLOYEE_DEPT				
EMP_ID	ENAME	DEPT_ID	DEPT_ID	DEPT_NAME
100	James	10	10	Account
100	James	10	20	Design
100	James	10	30	Testing
101	Kathy	20	10	Account
101	Kathy	20	20	Design
101	Kathy	20	30	Testing
102	Joseph	30	10	Account
102	Joseph	30	20	Design
102	Joseph	30	30	Testing
103	Rose	10	10	Account
103	Rose	10	20	Design
103	Rose	10	30	Testing
104	Marry	20	10	Account
104	Marry	20	20	Design
104	Marry	20	30	Testing

From the above example, we see that only the matching data from both the relations are retained in the final relation. Suppose we want to retain all the information from first relation and the corresponding information from the second relation irrespective of if it exists or not. ***There are three types of outer joins***

2.6.10 LEFT OUTER JOIN

1. **Left outer join –** In this ***operation, all the tuples in the left hand side relation*** is ***retained.***
2. All matching attribute in the ***right hand relation is displayed with values and the ones which do not have value are shown as NULL.***

Below example of left outer join on DEPT and EMPLOYEE table combines the matching combination of DEPT_ID = 10 with values. But DEPT_ID = 30 does not have any employees yet. Hence it displays NULL for those employees. Thus this outer join makes more meaningful to combining two relations than a cartesian product.

DEPT	
DEPT_ID	DEPT_NAME
10	Account
20	Design
30	Testing

Left Outer Join

EMPLOYEE		
EMP_ID	ENAME	DEPT_ID
100	James	10
101	Kathy	20
103	Rose	10
104	Marry	20

DEPT_ID	DEPT_NAME	EMP_ID	ENAME	DEPT_ID
10	Account	100	James	10
10	Account	103	Rose	10
20	Design	101	Kathy	20
20	Design	104	Marry	20
30	Testing			

2.6.11 RIGHT OUTER JOIN

1. **Right outer join –** This is opposite ***of left outer join. Here all the attributes*** of right hand side is retained and it matching ***attribute in left hand relation is found and displayed.***
2. If no matching is found then null is ***displayed. Same above example is re-written to understand this as below:***

EMPLOYEE		
EMP_ID	ENAME	DEPT_ID
100	James	10
101	Kathy	20
103	Rose	10
104	Marry	20

Right Outer Join

DEPT	
DEPT_ID	DEPT_NAME
10	Account
20	Design
30	Testing

EMP_ID	ENAME	DEPT_ID	DEPT_ID	DEPT_NAME
100	James	10	10	Account
101	Kathy	20	20	Design
103	Rose	10	10	Account
104	Marry	20	20	Design
			30	Testing

Notice the order and column difference in both the cases.

2.6.12 FULL OUTER JOIN

1. **Full outer join –** This is the ***combination of both left and right outer join. It displays all the attributes from both the relation.***
2. If the matching attribute exists ***in other relation, then that will be displayed, else those attributes are shown as null.***

EMPLOYEE		
EMP_ID	ENAME	DEPT_ID
100	James	10
101	Kathy	20
103	Rose	10
104	Marry	20
105	Alex	40

Full Outer Join

DEPT	
DEPT_ID	DEPT_NAME
10	Account
20	Design
30	Testing

EMP_ID	ENAME	DEPT_ID	DEPT_ID	DEPT_NAME
100	James	10	10	Account
101	Kathy	20	20	Design
103	Rose	10	10	Account
104	Marry	20	20	Design
105	Alex	40		
			30	Testing

2.7 SQL NULL VALUES

1. A field with ***a NULL value is a field with no value.***

2. If a field in a table is optional, it is ***possible to insert a new record or update a record without adding a value to this fi***eld.

3. Then, the field will be ***saved with a NULL value.***

How to Test for NULL Values?

It is not possible to test for NULL values with comparison ***operators, such as =, <, or <>.***

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

SELECT *column_names*
FROM *table_name*
WHERE *column_name* IS NULL;

IS NOT NULL Syntax

SELECT *column_names*
FROM *table_name*
WHERE *column_name* IS NOT NULL;

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

Example

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all customers with a value in the "Address" field:

Example

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

2.8 MODIFICATION OF DATABASE

SQL Modification Statements

The SQL Modification Statements make changes to database data in tables and columns. There are 3 modification statements:

- ***INSERT Statement -- add rows to tables***
- ***UPDATE Statement -- modify columns in table rows***
- ***DELETE Statement -- remove rows from tables***

2.8.1 INSERT STATEMENT

The INSERT Statement adds one or ***more rows to a table. It has two formats***:

INSERT INTO table-1 [(column-list)] VALUES (value-list)

and,

INSERT INTO table-1 [(column-list)] (query-specification)

The first ***form inserts a single row into table-1*** and explicitly specifies the column values for the row. The second form uses the result of ***query-specification*** to insert one or more rows into ***table-1.*** The result rows from the query are the rows added to the insert table. Note: the query cannot r***eference table-1.***

Both forms have an optional ***column-list specification***. Only the columns listed will be assigned values. Unlisted ***columns are set to null, so unlisted columns must allow nulls***. The values ***from the VALUES Clause (first form) or the columns from the query-specification rows*** (second form) are assigned to the corresponding column in *column-list* in order.

If the ***optional column-list is missing, the default column list is substituted. The default column list contains all columns in table-1 in the order they were declared in CREATE TABLE, or CREATE VIEW.***

VALUES Clause

The VALUES Clause in the INSERT Statement provides a set of values to place in the columns of a new row. It has the following general format:

VALUES (value-1 [, value-2] ...)

value-1 and *value-2* are Literal Values or Scalar Expressions involving literals. They can also specify NULL.

The values list in the VALUES clause must match the explicit or implicit column list for INSERT in degree (number of items). They must also match the data type of corresponding column or be convertible to that data type.

INSERT Examples

INSERT INTO p (pno, color) VALUES ('P4', 'Brown')

Before **After**

pno	descr	color
P1	Widget	Blue

=>

pno	descr	color
P1	Widget	Blue

P2	Widget	Red
P3	Dongle	Green

P2	Widget	Red
P3	Dongle	Green
P4	*NULL*	*Brown*

INSERT INTO sp

SELECT s.sno, p.pno, 500

FROM s, p

WHERE p.color='Green' AND s.city='London'

Before **After**

sno	**pno**	**qty**
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

=>

sno	**pno**	**qty**
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200
S2	*P3*	*500*

2.8.2 UPDATE STATEMENT

The UPDATE statement modifies columns ***in selected table rows. It has the following general format:***

UPDATE table-1 SET set-list [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to update. If it is missing, all rows are in *table-1* are updated.

The *set-list* contains assignments of new values for selected columns. See SET Clause.

The SET Clause expressions and WHERE Clause predicate can contain subqueries, but the subqueries cannot reference *table-1*. This prevents situations where results are dependent on the order of processing.

SET Clause

The SET Clause in the ***UPDATE Statement updates (assigns new value to) columns in the selected table rows. It has the following general*** format:

SET column-1 = value-1 [, column-2 = value-2] ...

column-1 and *column-2* are columns in the Update table. *value-1* and *value-2* are expressions that can reference columns from the update table. They also can be the keyword -- NULL, to set the column to *null*.

Since the assignment expressions can reference columns from the current row, the expressions are evaluated first. After the values of all Set expressions have been computed, they are then assigned to the referenced columns. This avoids results dependent on the order of processing.

UPDATE Examples

UPDATE sp SET qty = qty + 20

Before **After**

sno	**pno**	**qty**
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

=>

sno	**pno**	**qty**
S1	P1	NULL
S2	P1	*220*
S3	P1	*1020*
S3	P2	*220*

UPDATE s

SET name = 'Tony', city = 'Milan'

WHERE sno = 'S3'

Before **After**

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

=>

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	*Tony*	*Milan*

2.8.3 DELETE STATEMENT

The DELETE Statement removes ***selected rows from a table. It has the following general format:***

DELETE FROM table-1 [WHERE predicate]

The optional WHERE Clause has the ***same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which*** table rows to delete. If it is missing, all rows are in *table-1* are removed.

The WHERE Clause predicate can contain subqueries, but the subqueries cannot reference *table-1*. This prevents situations where results are dependent on the order of processing.

DELETE Examples

DELETE FROM sp WHERE pno = 'P1'

Before **After**

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000

=>

sno	pno	qty
S3	P2	200

S3	P2	200

DELETE FROM p WHERE pno NOT IN (SELECT pno FROM sp)

Before

pno	descr	color
P1	Widget	Blue
P2	Widget	Red
P3	Dongle	Green

=>

After

pno	descr	color
P1	Widget	Blue
P2	Widget	Red

Unit III

SQL:Overview of the SQL Query - Language - SQL Data Definition - Basic Structure of SQL Queries - Additional Basic Operations - Set Operations - Null Values Aggregate Functions - Nested Subqueries - Modification of the Database -Join Expressions - Views - Transactions - Integrity Constraints - SQL Data Types and Schemas - Authorization

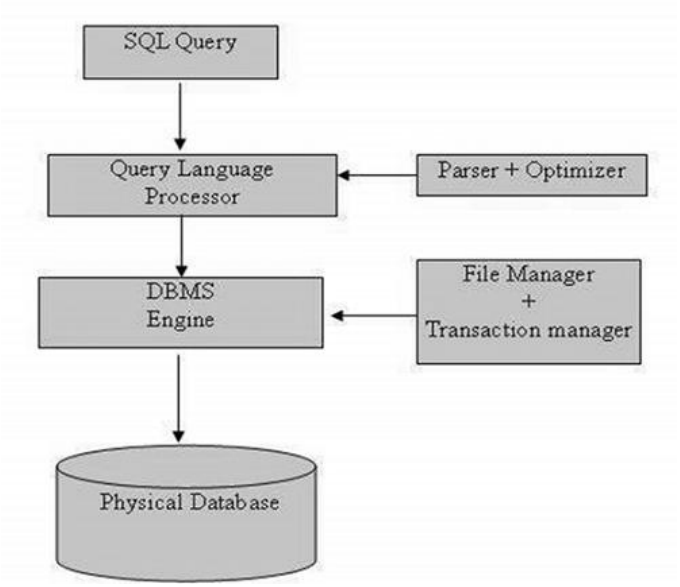
3.1 OVERVIEW OF THE SQL QUERY

SQL is a language to ***operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL*** is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is SQL?

1. SQL is Structured Query Language, ***which is a computer language for storing, manipulating and retrieving data stored in a relational database.*** SQL is the standard language for Relational Database System.
2. All the ***Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use*** SQL as their standard database language.A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a ***simple diagram showing the SQL Architecture*** −



3.2 SQL COMMANDS

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature −

3.2.1 DDL - DATA DEFINITION LANGUAGE

Sr.No.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

3.2.2 DML - DATA MANIPULATION LANGUAGE

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

3.3.3 DCL - DATA CONTROL LANGUAGE

Sr.No.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

3.3. BASIC STRUCTURE OF SQL QUERIES

SQL is based on set and relational operations with certain modifications and enhancements

• A ***typical SQL query has the form:***

select A1, A2, ..., An

from r1, r2, ..., rm

where P

– A is represent attributes

– r is represent relations

– P is a predicate.

• This query is equivalent to the relational algebra expression:

ΠA1, A2, ..., An(σP (r1 × r2 × ... × rm))

• The result of an SQL query is a relation.

3.3.1 THE SELECT CLAUSE

1. The select clause corresponds to ***the projection operation of the relational algebra***.
2. It is used to list the attributes ***desired in the result of a query.***

• Find the names of all branches in the loan relation

select branch-name from loan

In the "pure" relational algebra syntax, this query would be:

Πbranch-name (loan)

• An asterisk in the select clause denotes "all attributes"

select ∗from loan

1. SQL allows duplicates in ***relations as well as in query results.***

2. To force the elimination of ***duplicates, insert the keyword distinct after select.***

Find the names of all branches in the loan relation, and remove duplicates

select distinct branch-name from loan

• The keyword all specifies that duplicates not be removed.

select all branch-name from loan

• The select clause can contain arithmetic ***expressions involving the operators, +, −, ∗, and /,*** and operating on constants or attributes of tuples.

• The query:

select branch-name, loan-number, amount ∗ 100 from loan

would return a relation which is the same as ***the loan relation, except that the*** attribute amount is multiplied by 100

3.3.2 The where Clause

• The where clause corresponds to the selection ***predicate of the relational algebra. It consists*** of a predicate involving attributes of ***the relations that appear in the from clause.***

• Find all loan numbers for loans made at the ***Perryridge branch with loan amounts greater than $1200.***

select loan-number from loan

where branch-name = "Perryridge" and amount > 1200

• SQL uses the logical connectives and, or, and ***not. It allows the use of arithmetic expressions as operands to the comparison op***erators.

SQL includes a between comparison ***operator in order to simplify where clauses*** that specify that a value be less than or ***equal to some value and greater than or equal to some other value***.

• Find the loan number of those loans with loan amounts between ***$90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)***

select loan-number from loan

where amount between 90000 and 100000

3.3.3The from Clause

• The from clause corresponds to the Cartesian ***product operation of the relational algebra.*** It lists the relations to be scanned in ***the evaluation of the expression.***

• Find the Cartesian ***product borrower × loan***

select ∗from borrower, loan

• Find the name and loan number of all customers having a loan ***at the Perryridge branch.***

select distinct customer-name, borrower.loan-number from borrower, loan

where borrower.loan-number = loan.loan-number and branch-name = "Perryridge"

3.3.4 The Rename Operation

• The SQL mechanism for renaming relations and ***attributes is accomplished through*** the as clause: ***old-name as new-name***

• Find the name and loan number of all customers ***having a loan at the Perryridge branch;*** replace the ***column name loan-number with the name loan-id.***

select distinct customer-name, borrower.loan-number as loan-id from borrower, loan

where borrower.loan-number = loan.loan-number and branch-name = "Perryridge"

3.3.5 Tuple Variables

• Tuple variables are defined in the from clause ***via the use of the as clause.***

• Find the customer names and their loan ***numbers for all customers having a loan at some branch.***

select distinct customer-name, T.loan-number from borrower as T, loan as S where T.loan-number = S.loan-number

• Find the names of all branches that have greater assets than some branch located in Brooklyn.

select distinct T.branch-name from branch as T, branch as S where T.assets > S.assets and S.branch-city = "Brooklyn"

3.3.6 Set Operations

• The set operations union, intersect, ***and except operate on relations and correspond*** to the relational algebra ***operations ∪, ∩, and −.***

• Each of the above operations ***automatically eliminates duplicates***; to retain all duplicates use the corresponding multiset versions union all, ***intersect all and except all.Suppose a tuple*** occurs m times in r ***and n times in s, then, it occurs***:

– m + n ***times in r union all s***

– min(m, n) times in r intersect all s

– max(0, m − n) times in r except all s

• Find all customers who have a loan, an account, or both:

(select customer-name from depositor)

union

(select customer-name from borrower)

• Find all customers who have both a loan and an account.

(select customer-name from depositor)

intersect

(select customer-name from borrower)

• Find all customers who have an account but no loan.

(select customer-name from depositor)

except

(select customer-name from borrower

3.4 NULL VALUE

1. It is possible for tuples to have a ***null value, denoted by null, for some of their attributes; null signifies an unknown value or that a value does not exist***.
2. The result of any arithmetic expression involving ***null is null.• Roughly speaking***, all comparisons involving null ***return false.***

More precisely,

– Any comparison with null returns unknown

– (true or unknown) = true, (***false or unknown) = unknown (unknown or unknown) = unknown,***

(true and unknown) = unknown, (false and unknown) = false,(unknown and unknown) = unknown

– Result of where clause predicate is ***treated as false if it evaluates to unknown***

– "P is unknown" evaluates to ***true if predicate P evaluates to unknown***

Find all loan numbers which appear in the ***loan relation with null values for amount.***

select loan-number from loan

where amount is null

• Total all loan amounts select ***sum (amount) from loan***

Above statement ignores null amounts; result is null ***if there is no non-null amount***.

• All aggregate operations ***except count(*) ignore tuples with null values on the aggregated attributes***

3.5 SUB QUERY

1. A ***Subquery or Inner query or a Nested query is a query*** within another SQL query and ***embedded within the WHERE clause.***
2. A subquery is used to return ***data that will be used in the main query as a*** condition to further restrict ***the data to be retrieved.***
3. Subqueries can be used with the ***SELECT, INSERT, UPDATE, and*** DELETE statements along with the ***operators like =, <, >, >=, <=, IN, BETWEEN, etc.***

There are a few rules that subqueries must follow −

1. Subqueries ***must be enclosed within parentheses.***
2. A subquery can have only one column ***in the SELECT clause,*** unless multiple columns are in the main query for the ***subquery to compare its selected columns***.
3. An ORDER BY command ***cannot be used in a subquery, although the*** main query can use an ORDER BY. The GROUP ***BY command can be used to perform*** the same function as the ***ORDER BY in a subquery***.
4. Subqueries that return more ***than one row can only be used*** with multiple value operators such ***as the IN operator***.
5. The SELECT list ***cannot include any references to values*** that evaluate to a BLOB, ***ARRAY, CLOB, or NCLOB.***
6. A subquery cannot ***be immediately enclosed in a set function.***
7. The BETWEEN operator ***cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.***

3.6 SQL MODIFICATION STATEMENTS

The SQL Modification Statements make changes to database data in tables and columns. There are 3 modification statements:

- ***INSERT Statement -- add rows to tables***
- ***UPDATE Statement -- modify columns in table rows***
- ***DELETE Statement -- remove rows from tables***

3.6.1 INSERT Statement

The INSERT Statement adds one or more rows to a table. It has two formats:

INSERT INTO table-1 [(column-list)] VALUES (value-list)

and,

INSERT INTO table-1 [(column-list)] (query-specification)

The first form inserts a single row into ***table-1 and explicitly specifies*** the column values for the row. The second form uses the ***result of query-specification*** to insert one or more rows into table-1. The result rows from the ***query are the rows added to*** the insert table. Note: the query cannot reference table-1.

Both forms have an optional column-***list specification. Only the*** columns listed will be assigned values. Unlisted ***columns are set to null, so unlisted columns must*** allow nulls. The values from the VALUES Clause (first form) or the columns from the ***query-specification rows (second form) are ass***igned to the corresponding column in column-list in order.

If the optional column-list is ***missing, the default column list is substituted. The*** default column list contains ***all columns in table-1 in the order they were declared*** in CREATE ***TABLE, or CREATE VIEW.***

3.6.2 UPDATE STATEMENT

The UPDATE statement modifies columns in selected table rows. It has the following general format:

UPDATE table-1 SET set-list [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE ***clause chooses which table rows to update***. If it is missing, all rows are in table-1 are updated.

The set-list contains assignments of ***new values for selected columns. See SET Clause.***

The SET Clause expressions and ***WHERE Clause predicate can contain subqueries***, but the ***subqueries cannot reference table***-1. This prevents situations where results are dependent on the order of processing.

3.6.3 SET CLAUSE

The SET Clause in the UPDATE ***Statement updates (assigns new value to) columns in*** the selected table rows. ***It has the following general format:***

SET column-1 = value-1 [, column-2 = value-2] ...

column-1 and column-2 are ***columns in the Update table. value-1 and value***-2 are expressions that can reference columns from the ***update table. They also can be the keyword*** -- NULL, to set the column to null.

Since the assignment expressions ***can reference columns from the current*** row, the expressions are evaluated first. ***After the values of all Set expressions have been computed***, they are then assigned to ***the referenced columns. This avoids results dependent on the order of processing.***

UPDATE Examples

UPDATE sp SET qty = qty + 20

Before After

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

=>

sno	pno	qty
S1	P1	NULL
S2	P1	220
S3	P1	1020
S3	P2	220

UPDATE s

SET name = 'Tony', city = 'Milan'

WHERE sno = 'S3'

Before After

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

=>

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Tony	Milan

DELETE Statement

The DELETE Statement removes selected rows from a table. It has the following general format:

DELETE FROM table-1 [WHERE predicate]

The optional WHERE Clause has the same format as in the SELECT Statement. See WHERE Clause. The WHERE clause chooses which table rows to delete. If it is missing, all rows are in table-1 are removed.

The WHERE Clause predicate can contain subqueries, but the subqueries cannot reference table-1. This prevents situations where results are dependent on the order of processing.

3.7 JOIN EXPRESSIONS :

1. Sql join is used to ***fetch data from two or more tables, which is joined to appear as single set of data.***
2. It is used for combining column ***from two or more tables by using values common*** to both tables.
3. Join keyword is used in ***sql queries for joining two or more tables. minimum*** required condition for joining ***table, is (n-1) where n, is number of tables***.
4. A table can also join to itself, which ***is known as, self join.***

Types of join

Following are the types of join that we can use in sql:

- inner
- outer
- left
- right

3.7.1 INNER JOIN OR EQUI JOIN

This is a simple join in ***which the result is based on matched data as per the equality condition specified*** in the sql query.

inner join syntax is,

select column-name-list from

table-name1 inner join table-name2

where table-name1.column-name = table-name2.column-name;

example of inner join

consider a class table,

id	name
1	abhi
2	adam
3	alex
4	anu

and the class_info table,

id	address
1	delhi
2	mumbai
3	chennai

inner join query will be,

select * from class inner join class_info where class.id = class_info.id;

the result set table will look like,

id	name	id	address
1	abhi	1	delhi
2	adam	2	mumbai
3	alex	3	chennai

3.7.2 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.***

The syntax for Natural Join is,

SELECT * FROM

table-name1 NATURAL JOIN table-name2;

Example of Natural JOIN

Here is the class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

SELECT * from class NATURAL JOIN class_info;

The result set table will look like,

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

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.

3.7.3 OUTER JOIN

Outer join is based on both ***matched and unmatched data. outer joins subdivide further*** into,

- ***left outer join***
- ***right outer join***
- ***full outer join***

3.7.3.1 LEFT OUTER JOIN

The left outer join returns a ***resultset table with the matched data from the*** two tables and then the ***remaining rows of the left table and null from the right table's columns***.

syntax for left outer join is,

select column-name-list from

table-name1 left outer join table-name2

on table-name1.column-name = table-name2.column-name;

to specify a condition, we use the on keyword with outer join.

left outer join syntax for oracle is,

select column-name-list from

table-name1, table-name2 on table-name1.column-name = table-name2.column-name(+);

example of left outer join here is the class table,

id	name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

id	address
1	delhi
2	mumbai

3 chennai

7 noida

8 panipat

left outer join query will be,

select * from class left outer join class_info on (class.id = class_info.id);

the resultset table will look like,

id	name	id	address
1	abhi	1	delhi
2	adam	2	mumbai
3	alex	3	chennai
4	anu	null	null
5	ashish	null	null

3.7.3.2 RIGHT OUTER JOIN

The right outer join returns a ***resultset table with the matched data from the two tables being joined, then the remaining rows of the right ta***ble and null for the remaining left table's columns.

syntax for right outer join is,

select column-name-list from

table-name1 right outer join table-name2

on table-name1.column-name = table-name2.column-name;

right outer join syntax for oracle is,

select column-name-list from

table-name1, table-name2

on table-name1.column-name(+) = table-name2.column-name;

example of right outer join once again the class table,

id	name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

id	address
1	delhi
2	mumbai
3	chennai
7	noida
8	panipat

right outer join query will be,

select * from class right outer join class_info on (class.id = class_info.id);

the resultant table will look like,

id	name	id	address
1	abhi	1	delhi
2	adam	2	mumbai
3	alex	3	chennai
null	null	7	noida
null	null	8	panipat

3.7.3.3 FULL OUTER JOIN

The full outer join returns ***a resultset table with the matched data of two table then remaining rows of both left table and then the right tab***le.

syntax of full outer join is,

select column-name-list from

table-name1 full outer join table-name2

on table-name1.column-name = table-name2.column-name;

example of full outer join is,

the class table,

id	name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

id	address
1	delhi
2	mumbai
3	chennai
7	noida
8	panipat

full outer join query will be like,

select * from class full outer join class_info on (class.id = class_info.id);

the resultset table will look like,

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

3.8 VIEWS IN SQL

1. A view is nothing more than a sql ***statement that is stored in the database*** with an associated name.
2. A view is actually a ***composition of a table in the form of a predefined sql query***.
3. A view can contain all ***rows of a table or select rows from a table***.
4. A view can be created ***from one or many tables which depends on the written*** sql query to create a view.
5. views, which are a type of virtual ***tables allow users to do the following*** −
   - Structure data in a way ***that users or classes of users*** find natural or intuitive.
   - Restrict access to the data ***in such a way that a user can see and (***sometimes) modify exactly what ***they need and no more***.
   - Summarize data from ***various tables which can be used to generate reports.***

3.8.1 CREATING VIEWS

Database views are created using the ***CREATE VIEW statement***. Views can be created from a single table, ***multiple tables or another view. To create a view***, a user must have the appropriate system ***privilege according to the specific implementation***.

The basic CREATE VIEW syntax is as follows −

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];

Consider the CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

SQL > CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age

FROM CUSTOMERS;

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

SQL > SELECT * FROM CUSTOMERS_VIEW;

This would produce the following result.

+----------+-----+

| name | age |

+----------+-----+

| Ramesh | 32 |

| Khilan | 25 |

| kaushik | 23 |

| Chaitali | 25 |

| Hardik | 27 |

| Komal | 22 |

| Muffy | 24 |

+----------+-----+

THE WITH CHECK OPTION

The WITH CHECK OPTION is a ***CREATE VIEW statement option***. The purpose of the WITH CHECK ***OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.***

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age

FROM CUSTOMERS

WHERE age IS NOT NULL

WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

UPDATING A VIEW

A view can be updated under certain conditions which are given below −

1. The ***SELECT clause may not contain the keyword*** DISTINCT.
2. The SELECT ***clause may not contain summary functions***.
3. The SELECT ***clause may not contain set functions***.
4. The ***SELECT clause may not contain set operators***.
5. The ***SELECT clause may not contain an ORDER*** BY clause.
6. The FROM ***clause may not contain multiple tables.***
7. The ***WHERE clause may not contain subqueries***.
8. The query may not ***contain GROUP BY or HAVING.***
9. Calculated ***columns may not be updated.***

All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

SQL > UPDATE CUSTOMERS_VIEW

SET AGE = 35

WHERE name = 'Ramesh';

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

INSERTING ROWS INTO A VIEW

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the ***INSERT command.***

Here, we cannot insert rows in the ***CUSTOMERS_VIEW*** because we have not included all the ***NOT NULL columns in this view,*** otherwise you can insert rows in a view in a similar way as you insert them in a table.

DELETING ROWS INTO A VIEW

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

SQL > DELETE FROM CUSTOMERS_VIEW

WHERE age = 22;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

DROPPING VIEWS

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

DROP VIEW view_name;

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

DROP VIEW CUSTOMERS_VIEW;

3.9 TRANSACTIONS

1. A transaction is a unit of work ***that is performed against a database***.
2. Transactions are units or ***sequences of work accomplished in a logical order***, whether in a manual fashion by a user or ***automatically by some sort of a database*** program.
3. A transaction is the ***propagation of one or more changes to the database***.
4. For example, if you are ***creating a record or updating a record or deleting a*** record from the table, then you ***are performing a transaction on that table.***

PROPERTIES OF TRANSACTIONS

Transactions have the following four standard properties, usually referred to by the acronym ACID

- **ATOMICITY** − ensures that all operations within ***the work unit are completed*** successfully. Otherwise, the ***transaction is aborted at the point of failure and*** all the previous operations are rolled ***back to their former state.***
- **CONSISTENCY** − ensures that the database ***properly changes states upon*** a successfully ***committed transaction***.
- **ISOLATION** − enables ***transactions to operate independently of and transparent to each other.***
- **DURABILITY** − ensures that the result or ***effect of a committed transaction persists*** in case of ***a system failure.***

TRANSACTION CONTROL

The following commands are used to control transactions.

- **COMMIT** − to save ***the changes***.
- **ROLLBACK** − to ***roll back the changes.***
- **SAVEPOINT** − creates points within the groups ***of transactions in which to ROLLBACK.***
- **SET TRANSACTION** − Places a ***name on a transaction.***

3.10 INTEGRITY CONSTRAINTS

1. Sql constraints are used to ***specify rules for the data in a table***.
2. Integrity constraints are ***used to ensure accuracy and consistency*** of the data in a ***relational database. Data integrity is*** handled in a relational database through the ***concept of referential integrity***.
3. Constraints are used to ***limit the type of data that can go into a*** table.
4. This ensures the accuracy and ***reliability of the data in the table***. if there is any violation between the ***constraint and the data action, the action is aborted***.
5. Constraints can be column level ***or table level. column level constraints*** apply to a column, and table level ***constraints apply to the whole table***.

6. The following ***constraints are commonly used in sql:***
   - **NOT NULL** - ensures that a ***column cannot have a null value***
   - **UNIQUE** - ensures that all values in a ***column are different***
   - **PRIMARY KEY** - a combination of a not ***null and unique. uniquely identifies each*** row in a table
   - **FOREIGN KEY** - uniquely identifies a row***/record in another table***
   - **CHECK** - ensures that all values in a column ***satisfies a specific condition***
   - **DEFAULT** - sets a default ***value for a column when no value is specified***
   - **INDEX** - used to create and ***retrieve data from the database very quickly***

3.11 SQL DATA TYPES AND SCHEMAS

1. A Schema in SQL is a collection of ***database objects associated with a database***.
2. The ***username of a database is called a Schema owner (owner*** of logically grouped structures of data).
3. Schema always belong to a ***single database whereas a database can have single or multiple schemas.***
4. Also, it is also very similar to separate ***namespaces or containers, which stores database objects.***
5. It includes various database objects ***including your tables, views, procedures, index, etc.***

Syntax to create SQL:

CREATE SCHEMA [schema_name] [AUTHORIZATION owner_name]

[DEFAULT CHARACTER SET char_set_name]

[PATH schema_name[, ...]]

[ANSI CREATE statements [...]]

[ANSI GRANT statements [...]];

3.12 AUTHORIZATION

We can access the DB2 Database and its functionality within the DB2 database system, which is managed by the DB2 Database manager. Authorization is a process managed by the DB2 Database manager. The manager obtains information about the current authenticated user, that indicates which database operation the user can perform or access.

Here are different ways of permissions available for authorization:

1. **Primary permission:** Grants the authorization ID directly.
2. **Secondary permission:** Grants to the groups and roles if the user is a member
3. **Public permission:** Grants to all users publicly.
4. **Context-sensitive permission:** Grants to the trusted context role.

Authorization can be given to users based on the categories below:

System-level authorization

1. ***System administrator [SYSADM]***
2. ***System Control [SYSCTRL]***
3. ***System maintenance [SYSMAINT]***
4. ***System monitor [SYSMON]***

Authorities provide of control over instance-level functionality. Authority provide to group privileges, to control maintenance and authority operations. For instance, database and database objects.

Database-level authorization

1. ***Security Administrator [SECADM]***
2. ***Database Administrator [DBADM]***
3. ***Access Control [ACCESSCTRL]***
4. ***Data access [DATAACCESS]***
5. ***SQL administrator. [SQLADM]***
6. ***Workload management administrator [WLMADM]***
7. ***Explain [EXPLAIN]***

Authorities provide controls within the database. Other authorities for database include with LDAD and CONNECT.

Object-Level Authorization: Object-Level authorization ***involves verifying privileges when an operation is performed on an object.***

Content-based Authorization: User can have read and ***write access to individual rows and columns on a particular table using Label-based access Control [LBAC***].

Unit IV

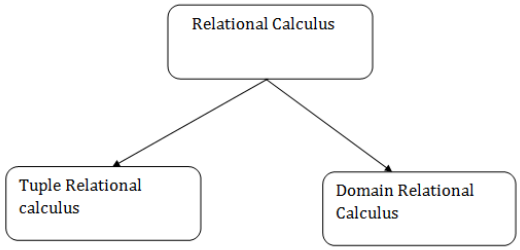
Relational Languages: The Tuple Relational Calculus - The Domain Relational Calculus Database Design and the E-R Model: Overview of the Design Process - The Entity Relationship Model - Reduction to Relational Schemas - Entity-Relationship Design Issues - Extended E-R Features - Alternative Notations for Modeling Data - Other Aspects of Database Design

4.1 RELATIONAL CALCULUS

Relational calculus is a non-***procedural query language. In the non-procedural query language, the user is concerned with the details of how to*** obtain the end results.

The relational calculus tells what to do but never explains how to do.

Types of Relational calculus:



4.1.1. Tuple Relational Calculus (TRC)

The tuple relational calculus is specified to ***select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.***

The result of the relation can have one or more tuples.

Notation:

{T | P (T)} or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

{ T.name | Author(T) AND T.article = 'database' }

OUTPUT: This query selects the tuples ***from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article*** on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

4.1.2. Domain Relational Calculus (DRC)

The second form of relation is known as ***Domain relational calculus. In domain relational calculus, filtering variable uses the domain of*** attributes.

Domain relational calculus uses the same operators as tuple calculus. ***It uses logical connectives ∧ (and), ∨ (or) and ¬ (not).***

It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

Notation:

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1, a2 are attributes

P stands for formula built by inner attributes

For example:

{< article, page, subject > | ∈javatpoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

4.2 ENTITY-RELATIONSHIP DESIGN ISSUES

Use of entity sets vs. attributes :

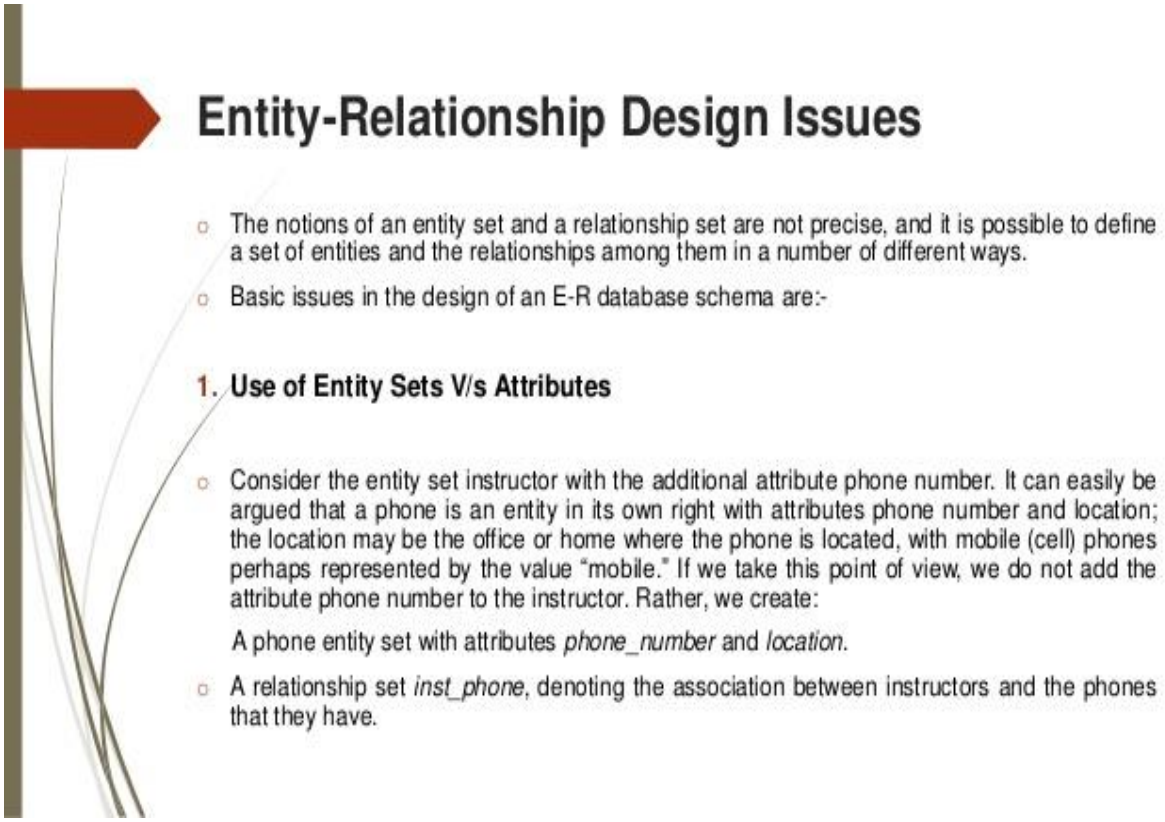
Choice mainly depends on the structure ***of the enterprise being modeled, and on the semantics associated with the attribute in qu***estion.

Use of entity sets vs. relationship sets :

Possible guideline is to designate a ***relationship set to describe an action that occurs between entities***

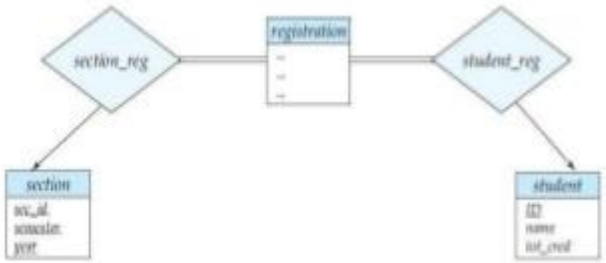
Binary versus n-ary relationship sets :

Although it is possible to ***replace a nonbinary (n-ary, for n > 2) relationship set by a number of distinct binary relationship sets, a n-ary relationship*** set shows more clearly that several entities participate in a single relationship.



2. Use of Entity Sets V/s Relationship Sets

- It is not always clear whether an object is best expressed by an entity set or a relationship set. An alternative is to imagine that there is a course-registration record for each course that each student takes.
- Then, we have an entity set to represent the course-registration record. Let us call that entity set *registration*. Each *registration* entity is related to exactly one student and to exactly one section, so we have two relationship sets, one to relate course registration records to students and one to relate course-registration records to sections.



3. Binary V/s n-ary Relationship Sets

- Relationships in databases are often binary. Some relationships that appear to be non binary could actually be better represented by several binary relationships.
- For instance, one could create a ternary relationship *parent*, relating a child to his/her mother and father. However, such a relationship could also be represented by two binary relationships, *mother* and *father*, relating a child to his/her mother and father separately.
- Using the two relationships *mother* and *father* provides us a record of a child's mother, even if we are not aware of the father's identity; a null value would be required if the ternary relationship *parent* is used. Using binary relationship sets is preferable in this case.
- In fact, it is always possible to replace a non binary (*n*-ary, for $n > 2$) relationship set by a number of distinct binary relationship sets. For simplicity, consider the abstract ternary ($n = 3$) relationship set *R*, relating entity sets *A*, *B*, and *C*.

a. *RA*, relating *E* and *A*.
b. *RB*, relating *E* and *B*.
c. *RC*, relating *E* and *C*.

Unit V

Relational Database Design: Features of Good Relational Designs - Atomic Domains and First Normal Form - Decomposition Using Functional Dependencies - Functional-Dependency Theory - Decomposition Using Functional Dependencies - Decomposition Using Multivalued Dependencies-More Normal Forms - Database-Design Process

5.1 DESIGNING GOOD RELATIONAL DATABASES:

Databases have a reputation for being difficult to construct and hard to maintain. The power of modern database software makes it possible to create a database with a few mouse-clicks. The databases created this way, however, are typically the databases that are hard to maintain and difficult to work with because they are designed poorly. Modern software makes it easy to construct a database, but doesn't help much with the design aspect of database creation.

Database design has nothing to do with using computers. It has everything to do with research and planning. The design process should be completely independent of software choices. The basic elements of the design process are:

1. Defining the problem or objective
2. Researching the current database
3. Designing the data structures
4. Constructing database relationships
5. Implementing rules and constraints
6. Creating database views and reports
7. Implementing the design

Notice that implementing the database design in software is the final step. All of the preceding steps are completely independent of any software or other implementation concerns.

Defining the problem or objective.

The most important step in database design is the first one: defining the problem the database will address or the objective of the database. It is important however, to draw a distinction between:

- How the database will be used and
- What information needs to be stored in it.

Researching the current database.

In most database design situations, there is some sort of database already in existence. That database may be Post-it notes, paper order forms, a spreadsheet of sales data, a word processor file of names and addresses, or a full-fledged digital database (possibly in an outdated software package or older legacy system).

Designing the data structures.

A database is essentially a collection of data tables, so the next step in the design process is to identify and describe those data structures. Each table in a database should represent some distinct subject or physical object, so it seems reasonable to simply analyse the subjects or physical objects relevant to the purpose of the database, then arrive at a list of tables.

Once the tables have been determined and fields have been assigned to each, the next step is to develop the specifications for each field. The perfect field should be atomic: It should be unique in all tables in the database (unless it is used as a key) and contain a single value, and it should not be possible to break it into smaller components.

Constructing database relationships.

Once the data structures are in place, the next step is to establish the relationships between the databases. First you must ensure that each table has a unique key that can identify the individual records in each table.

Implementing rules and constraints.

In this step, the fields in the database are still fairly amorphous. Defining the fields as text or numeric and getting a rough feel for the types of data that the client needs to store has narrowed them down, but there is room for further refinement.

Creating database views and reports.

Now that the data design is essentially complete, the penultimate step is to create the specifications that help turn the data into useful information in the form of a report or view of the data. *Views* are simply collections of the data available in the database combined and made accessible in one place.

Implementing the design in software.

All of the work to this point has been accomplished without explicitly worrying about the details of the program being used to produce the database. In fact, the design should only exist as diagrams and notes on paper

5.2 NORMALIZATION

1. Normalization is the process of ***organizing the data in the database***.
2. Normalization is used to minimize the ***redundancy from a relation or set of relations***.
3. It is also used to eliminate the ***undesirable characteristics like insertion, update and deletion anomalies.***
4. Normalization divides the larger table into the ***smaller table and links them using*** relationship.
5. The normal form is used ***to reduce redundancy from the database table***.

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

5.2 .1 ATOMIC DOMAINS AND FIRST NORMAL FORM

1. First Normal Form is ***defined in the definition of relations (tables) itself.***
2. This rule defines that all the ***attributes in a relation must have*** atomic domains. The values in an atomic ***domain are indivisible units.***
3. First normal form is an ***essential property of a relation in a relational database.***
4. If tables in a database are not ***even in the 1st Normal Form***, it ***is considered as bad database design.***

- A relation will be 1NF ***if it contains an atomic value.***
- It states that an attribute of a table ***cannot hold multiple values. It must hold only single-valued attribute.***
- First normal form disallows the ***multi-valued attribute, composite attribute, and their combinations.***

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

5.2.2 SECOND NORMAL FORM (2NF)

- In the 2NF, ***relational must be in 1NF.***
- In the second normal form, all non-***key attributes are fully functional dependent on the primary key***

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

5.2.3. THIRD NORMAL FORM (3NF)

- A relation will be in 3NF if it is in ***2NF and not contain any transitive partial dependency.***
- 3NF is used to reduce the data duplication. It is also ***used to achieve the data integrity.***
- If there is no transitive ***dependency for non-prime attributes, then the relation must be*** in third ***normal form.***

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

5.2.3 BOYCE CODD NORMAL FORM (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232

D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

5.2.4 FOURTH NORMAL FORM (4NF)

- A relation will be in 4NF if it is in Boyce ***Codd normal form and has no multi-valued dependency.***
- For a dependency A → B, if for a single ***value of A, multiple values of B exists, then the relation will be a multi-valued dependency.***

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket

59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing**and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

5.2.5 FIFTH NORMAL FORM (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1

Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

5.3 RELATIONAL DECOMPOSITION

1. When a relation in the ***relational model is not in appropriate normal form then the decomposition of a relation is required.***
2. In a database, it breaks the table into ***multiple tables. If the relation has*** no proper decomposition, then it may lead ***to problems like loss of information***.
3. Decomposition is used to ***eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy***

Types of Decomposition

1. ***Lossless Decomposition***
2. ***Dependency Preserving***

Lossless Decomposition

- If the information is not lost ***from the relation that is decomposed, then the decomposition will be lossless.***
- The lossless decomposition guarantees that ***the join of relations will result in the same relation as it was decomposed.***
- The relation is said to be lossless ***decomposition if natural joins of all the decomposition give the original relation.***

DEPENDENCY PRESERVING

- It is an important constraint of ***the database.***
- In the dependency preservation, at least one ***decomposed table must satisfy every dependency.***
- If a relation R is decomposed into relation R1 and ***R2, then the dependencies*** of R either must be a part of ***R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.***
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is ***decomposed into R1(ABC) and*** R2(AD) which is dependency preserving ***because FD A->BC is a part of relation R1(ABC***).

5.4 FUNCTIONAL DEPENDENCY

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.

1. 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:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency

5.4.1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

5.4.2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB

5.5 MULTIVALUED DEPENDENCY

- Multivalued dependency occurs ***when two attributes in a table are independent of each*** other but, ***both depend on a third attribute***.
- A multivalued ***dependency consists of at least*** two attributes that are dependent on a third attribute ***that's why it always requires at least three*** attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".