Database System

What is Data?

In simple words data can be facts related to any object in consideration.

For example your name, age, height, weight, etc are some data related to you.

A picture , image , file , pdf etc can also be considered data.

What is a Database?

Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy. Let's discuss few examples.

An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.

Your electricity service provider is obviously using a database to manage billing , client related issues, to handle fault data, etc.

Let's also consider the facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more.

We can provide countless number of examples for usage of databases .

What is a Database Management System (DBMS)?

Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data systematically.

It also helps to control access to the database.

Database Management Systems are not a new concept and as such had been first implemented in 1960s.

Charles Bachmen's Integrated Data Store (IDS) is said to be the first DBMS in history.

With time database technologies evolved a lot while usage and expected functionalities of databases have been increased immensely.

Application and Uses of Database Management System (DBMS)

Due the evolution of Database management system, companies are getting more from their work because they can keep records of everything. Also it makes them faster to search information and records about any people or product that makes them more effective in work. So here we are sharing some of the applications and **uses of database management system (DBMS)**.

Railway Reservation System

Database is required to keep record of ticket booking, train's departure and arrival status. Also if trains get late then people get to know it through database update.

Library Management System

There are thousands of books in the library so it is very difficult to keep record of all the books in a copy or register. So DBMS used to maintain all the information relate to book issue dates, name of the book, author and availability of the book.

Banking

We make thousands of transactions through banks daily and we can do this without going to the bank. So how banking has become so easy that by sitting at home we can send or get money through banks. That is all possible just because of DBMS that manages all the bank transactions.

Universities and colleges

Examinations are done online today and universities and colleges maintain all these records through DBMS. Student's registrations details, results, courses and grades all the information are stored in database.

Credit card transactions

For purchase of credit cards and all the other transactions are made possible only by DBMS. A credit card holder knows the importance of their information that all are secured through DBMS.

Social Media Sites

We all are on social media websites to share our views and connect with our friends. Daily millions of users signed up for these social media accounts like facebook, twitter, pinterest and Google plus. But how all the information of users are stored and how we become able to connect to other people, yes this all because DBMS.

Telecommunications

Any telecommunication company cannot even think about their business without DBMS. DBMS is must for these companies to store the call details and monthly post paid bills.

Finance

Those days have gone far when information related to money was stored in registers and files. Today the time has totally changed because there are lots f thing to do with finance like storing sales, holding information and finance statement management etc.

Military

Military keeps records of millions of soldiers and it has millions of files that should be keep secured and safe. As DBMS provides a big security assurance to the military information so it is widely used in militaries. One can easily search for all the information about anyone within seconds with the help of DBMS.

Online Shopping

Online shopping has become a big trend of these days. No one wants to go to shops and waste his time. Everyone wants to shop from home. So all these products are added and sold only with the help of DBMS. Purchase information, invoice bills and payment, all of these are done with the help of DBMS.

Human Resource Management

Big firms have many workers working under them. Human resource management department keeps records of each employee's salary, tax and work through DBMS.

Manufacturing

Manufacturing companies make products and sales them on the daily basis. To keep records of all the details about the products like quantity, bills, purchase, supply chain management, DBMS is used.

Airline Reservation system

Same as railway reservation system, airline also needs DBMS to keep records of flights arrival, departure and delay status.

So in short, one can say the DBMS is used everywhere around us and we cannot rely without DBMS.

Purpose of Database System

Database management systems were developed to handle the following difficulties of typical file-processing systems supported by conventional operating systems.

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation – multiple files and formats
- Integrity problems
- Atomicity of updates
- Concurrent access by multiple users
- Security problems

- **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. Moreover, the same information may be duplicated in several places (files). For example, if a student has a double major (say,

music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same datamayno longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

- **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. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all students. The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

- **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.

- **Integrity problems:** The data values stored in the database must satisfy certain types of **consistency constraints**. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a

department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compoundedwhen constraints involve several data items from different files.

• **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. Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A butwas not credited to the balance of department B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

• **Concurrent-access anomalies:** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.

• **Security problems:** Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

View of Data

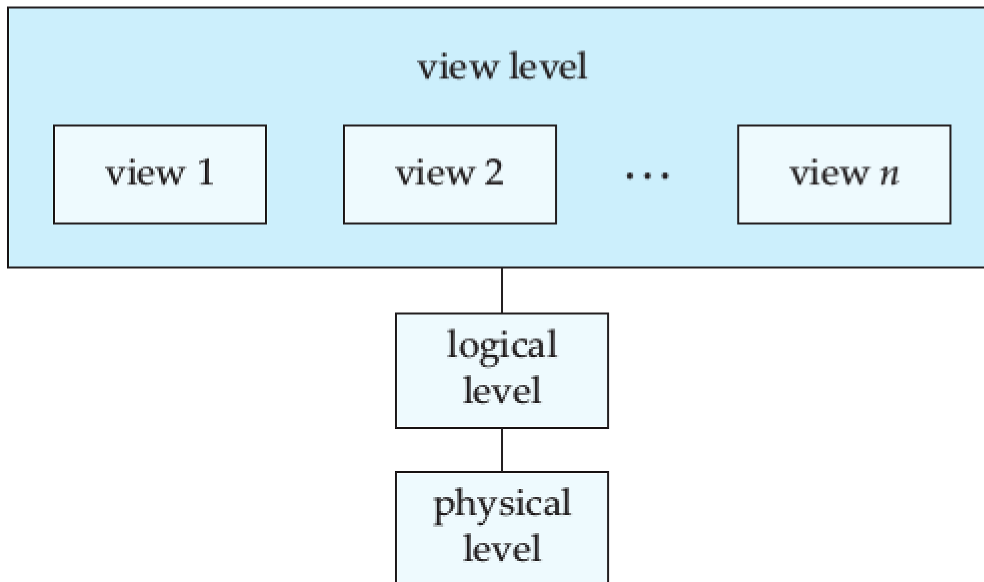


Figure 1.1 The three levels of data abstraction.

- **Physical level**: describes how a record (e.g., *customer*) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

type *customer* = **record**

name : string; *street* : string; *city* : integer;

end;

- **View level:** application programs hide details of data types. Views can also hide information (e.g. salary) for security purposes.

Database Languages

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)

Data Definition Language (DDL)

- Specification notation for defining the database schema

- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
- *Data storage and definition* language – special type of DDL in which the storage structure and access methods used by the database system are specified

Some of the common Data Definition Language commands are:

- **CREATE**
- **ALTER**
- **DROP**

1. **CREATE- Data Definition language(DDL)**

The main use of the create command is to build a new table and it comes with a predefined syntax. It creates a component in a relational database management system.

2. **ALTER- Data Definition language(DDL)**

An existing database object can be modified by the ALTER statement. Using this command, the users can add up some additional column and drop existing columns.

3. **Drop- Data Definition language(DDL)**

By the use of this command, the users can delete an index, table or view. A component from a relational database management system can be removed by a DROP statement in SQL. There are many systems that allow the DROP and some other Data Definition Language commands for occurring inside a transaction and then it can be rolled back.

Data Manipulation Language (DML)

A data manipulation language (DML) is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting

data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.

- **Two classes of languages**
  - Procedural – user specifies what data is required and how to get those data
  - Nonprocedural – user specifies what data is required without specifying how to get those data

DML resembles simple English language and enhances efficient user interaction with the system. The functional capability of DML is organized in manipulation commands like SELECT, UPDATE, INSERT INTO and DELETE FROM, as described below:

- **SELECT:** This command is used to retrieve rows from a table. The syntax is SELECT [column name(s)] from [table name] where [conditions]. SELECT is the most widely used DML command in SQL.
- **UPDATE**: This command modifies data of one or more records. An update command syntax is UPDATE [table name] SET [column name = value] where [condition].
- **INSERT:** This command adds one or more records to a database table. The insert command syntax is INSERT INTO [table name] [column(s)] VALUES [value(s)].

DELETE: This command removes one or more records from a table according to specified conditions. Delete command syntax is DELETE FROM [table name] where [condition].

Relational database

A relational database is a collection of information that organizes data points with defined relationships for easy access. In the relational database model, the data structures -- including data tables, indexes and views -- remain separate from the physical storage, allowing administrators to edit the physical data storage without affecting the logical data structure.

Database Design

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

The physical data design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

Object Based and Semi-Structured Databases

Object Based

Object DATABASE OR Object oriented database management system is a database in which the information is represented in form of object as used in object-oriented programming. It is different from rational database. This type of database is used when there is complex data or/and multiple data relationships. It have a many-to-many object relationship. It should not be used when there are few join tables and there are large volume of simple transaction data.

It works well with the following application:

--> Multimedia Application.
--> CAS Application

Features of Object Oriented Database:

- It support transactions.
- It supply querying in bulk data.
- Concurrent Access
- Security

Semi-Structured Databases

In **Semi-Structured Database** the data are in the form of structured data that does not conform with the formal structure of data models associated with rational databases or other form of data. Therefore, it is also known as self-describing structure.

Types of Semi-Structured Database:

- XML semi-structured database
- JSON (JavaScript Object Notation)semi-structured database

Advantages of Semi-Structured Database

- It can show the information of data source that is not constrained by schema.
- It is used to view structured data as semi-structured data.
- The data transfer format may be portable

Transaction Management

- A *transaction* is a collection of operations that performsa single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g. power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Storage Management

- 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 thesystem.
- The storage manager is responsible for the following tasks:
  - interaction with the file manager
  - efficient storing, retrieving, and updating of data

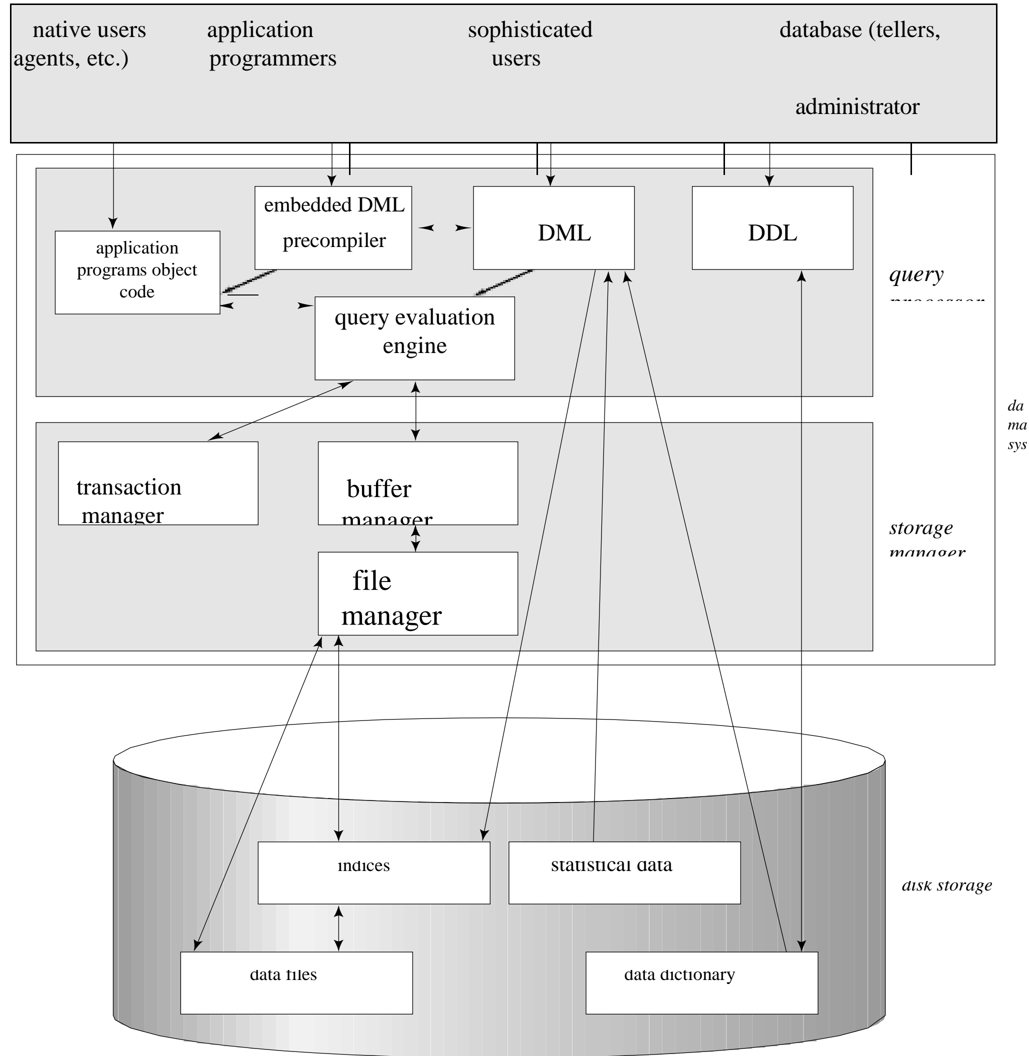
Database Users

- Users are differentiated by the way they expect to interact with the system
- Application programmers – interact with system through DML calls
- Sophisticated users – form requests in a database query language

- Specialized users – write specialized database applications that do not fit into the traditional data processing framework

- Naive users – invoke one of the permanent application programs that have been written previously

Database Architecture



Database Administrator

Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.

- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

Data Mining and Analysis

Data analysis and data mining are a subset of business intelligence (BI), which also incorporates data warehousing, database management systems, and Online Analytical Processing (OLAP).

The technologies are frequently used in customer relationship management (CRM) to analyze patterns and query customer databases. Large quantities of data are searched and analyzed to discover useful patterns or relationships, which are then used to predict future behavior.

Some estimates indicate that the amount of new information doubles every three years. To deal with the mountains of data, the information is stored in a repository of data gathered from various sources, including corporate databases, summarized information from internal systems, and data from external sources. Properly designed and implemented, and regularly updated, these repositories, called data warehouses, allow managers at all levels to extract and examine information about their company, such as its products, operations, and customers' buying habits.

With a central repository to keep the massive amounts of data, organizations need tools that can help them extract the most useful information from the data. A data warehouse can bring together data in a single format, supplemented by metadata through use of a set of input mechanisms known as extraction, transformation, and loading (ETL) tools. These and other BI tools enable organizations to quickly make knowledgeable business decisions based on good information analysis from the data.

Analysis of the data includes simple query and reporting functions, statistical analysis, more complex multidimensional analysis, and data mining (also known as knowledge discovery in databases, or KDD). Online analytical processing (OLAP) is most often associated with multidimensional analysis, which requires powerful data manipulation and computational capabilities.

With the increasing data being produced each year, BI has become a hot topic. The increasing focus on BI has caused a number of large organizations have begun to increase their presence in the space, leading to a consolidation around some of the largest software vendors in the world. Among the notable purchases in the BI market were Oracle's purchase of Hyperion Solutions; Open Text's acquisition of Hummingbird; IBM's buy of Cognos; and SAP's acquisition of Business Objects.

Data mining can be defined as the process of extracting data, analyzing it from many dimensions or perspectives, then producing a summary of the information in a useful form that identifies relationships within the data. There are two types of data mining: descriptive, which gives information about existing data; and predictive, which makes forecasts based on the data.

History of Database Systems

Information processing drives the growth of computers, as it has from the earliest days of commercial computers. In fact, automation of data processing tasks predates computers. Punched cards, invented by Herman Hollerith, were used at the very beginning of the twentieth century to record U.S. census data, and mechanical systemswere used to process the cards and tabulate results. Punched cards were later widely used as a means of entering data into computers.

Techniques for data storage and processing have evolved over the years:

• 1950s and early 1960s:

Magnetic tapes were developed for data storage. Data processing tasks such as payroll were automated, with data stored on tapes.

• Late 1960s and 1970s:

Widespread use of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access

to data. The position of data on disk was immaterial, since any location on disk could be accessed in just tens of milliseconds.

A landmark paper by Codd [1970] defined the relational model and nonprocedural ways of querying data in the relational model, and relational databaseswere born.

1980s:

Although academically interesting, the relational model was not used in practice initially, because of its perceived performance disadvantages; relational databases could notmatch the performance of existing network and hierarchical databases. That changed with System R, a groundbreaking project at IBM Research that developed techniques for the construction of an efficient relational database system. Excellent overviews of System R are provided by Astrahan et al. [1976] and Chamberlin et al. [1981]. The fully functional System R prototype led to IBM's first relational database product, SQL/DS.

The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases.

• Early 1990s:

The SQL language was designed primarily for decision support applications, which are query-intensive, yet the mainstay of databases in the 1980s was transaction-processing applications, which are update-intensive. Decision support and querying re-emerged as a major application area for databases. Tools for analyzing large amounts of data saw large growths in usage.

• 1990s:

The major event of the 1990s was the explosive growth of the World WideWeb. Databaseswere deployedmuchmore extensively than ever before. Database systems now had to support very high transaction-processing rates, as well as very high reliability and 24 × 7 availability.

• 2000s:

The first half of the 2000s saw the emerging of XML and the associated query language XQuery as a new database technology. Although XML is widely used for data exchange, as well as for storing certain complex data types, relational databases still form the core of a vast majority of large-scale database applications. In this time periodwe have also witnessed the growth in "autonomic-computing/auto-admin" techniques for minimizing system administration effort.

Unit – II

Structure of Relational Database

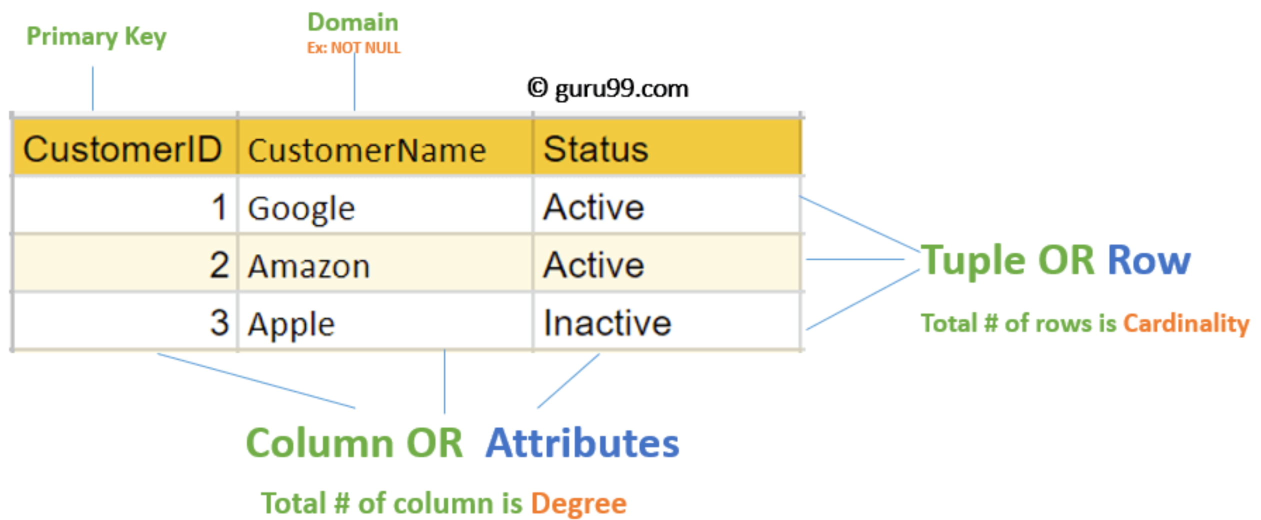
Relational Model

RELATIONAL MODEL (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain.



Relational Integrity constraints

Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain constraints
2. Key constraints
3. Referential integrity constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName

CHECK (value not NULL)
```

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Referential integrity constraints

Referential integrity constraints is base on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300

Operations in Relational Model

Four basic update operations performed on relational database model are

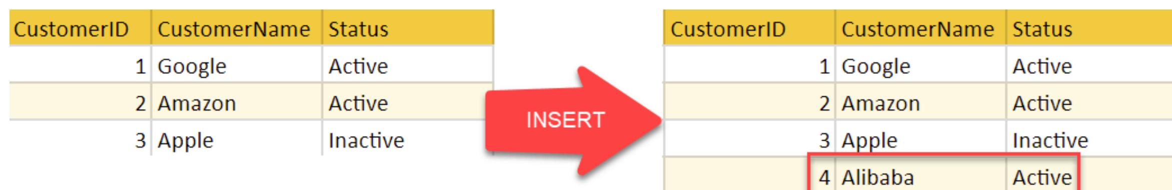
Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

Fundamental Relational Algebra Operations

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

The fundamental operations of relational algebra are as follows −

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

We will discuss all these operations in the following sections.

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol:)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

1. **Select Operation (σ)**

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

Notation − $\sigma_p(r)$

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

For example −

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

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

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

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

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

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

2. **Projection(π)**

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active

3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

3. **Union operation (υ)**

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

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

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A			Table B	
column 1	**column 2**		**column 1**	**column 2**
1	1		1	1
1	2		1	3

A ∪ B gives

Table A ∪ B	
column 1	**column 2**
1	1
1	2
1	3

4. Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A-B

Table A - B	
column 1	**column 2**
1	2

5. Cartesian product(X)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

Example – Cartesian product

$\sigma_{\text{column 2} = \text{'1'}}$ (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)

column 1	column 2
1	1
1	1

6. Rename Operation (ρ)

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

Notation − ρ_x (E)

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

Extended Operators in Relational Algebra

Extended operators are those operators which can be derived from basic operators.There are mainly three types of extended operators in Relational Algebra:

- **Join**
- **Intersection**
- **Divide**

The relations used to understand extended operators are STUDENT, STUDENT_SPORTS, ALL_SPORTS and EMPLOYEE which are shown in Table 1, Table 2, Table 3 and Table 4 respectively.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Table 1

STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

Table 2

ALL_SPORTS

SPORTS
Badminton
Cricket

Table 3

EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

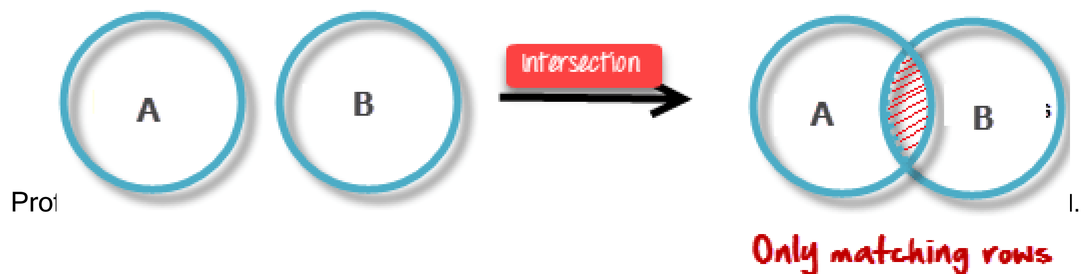
Table 4

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Intersection operator when applied on two relations as R1∩R2 will give a relation with tuples which are in R1 as well as R2. Syntax:

Relation1 ∩ Relation2

Find a person who is student as well as employee- **STUDENT ∩ EMPLOYEE**

In terms of basic operators (union and minus) :

STUDENT ∩ EMPLOYEE = STUDENT + EMPLOYEE - (STUDENT U EMPLOYEE)

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18

Conditional Join(⋈c): Conditional Join is used when you want to join two or more relation based on some conditions. Example: Select students whose ROLL_NO is greater than EMP_NO of employees

STUDENT ⋈c STUDENT.ROLL_NO>EMPLOYEE.EMP_NOEMPLOYEE

In terms of basic operators (cross product and selection) :

σ (STUDENT.ROLL_NO>EMPLOYEE.EMP_NO)(STUDENT×EMPLOYEE)

RESULT:

ROLL _NO	NAME	ADDRESS	PHONE	AGE	EMP _NO	NAME	ADDRESS	PHONE	AGE
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	9455123451	18
3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	9455123451	18

Equijoin(⋈): Equijoin is a **special case of conditional join** where only equality condition holds between a pair of attributes. As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.

Example:Select students whose ROLL_NO is equal to EMP_NO of employees

STUDENT⋈STUDENT.ROLL_NO=EMPLOYEE.EMP_NOEMPLOYEE

In terms of basic operators (cross product, selection and projection) :

∏(STUDENT.ROLL_NO, STUDENT.NAME, STUDENT.ADDRESS, STUDENT.PHONE, STUDENT.AGE EMPLOYEE.NAME, EMPLOYEE.ADDRESS, EMPLOYEE.PHONE, EMPLOYEE>AGE)(σ (STUDENT.ROLL_NO=EMPLOYEE.EMP_NO) (STUDENT×EMPLOYEE))

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	SURESH	DELHI	9156768971	18

Natural Join(⋈): It is a special case of equijoin in which equality condition hold on all attributes which have same name in relations R and S (relations on which join operation is applied). While applying natural join on two relations, there is no need to write equality condition explicitly. Natural Join will also return the similar attributes only once as their value will be same in resulting relation. Example: Select students whose ROLL_NO is equal to ROLL_NO of STUDENT_SPORTS as:

STUDENT⋈STUDENT_SPORTS

In terms of basic operators (cross product, selection and projection) :

∏(STUDENT.ROLL_NO, STUDENT.NAME, STUDENT.ADDRESS, STUDENT.PHONE, STUDENT.AGE STUDENT_SPORTS.SPORTS)(σ (STUDENT.ROLL_NO=STUDENT_SPORTS.ROLL_NO) (STUDENT×STUDENT_SPORTS))

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	SPORTS
1	RAM	DELHI	9455123451	18	Badminton
2	RAMESH	GURGAON	9652431543	18	Cricket
2	RAMESH	GURGAON	9652431543	18	Badminton
4	SURESH	DELHI	9156768971	18	Badminton

Natural Join is by default inner join because the tuples which does not satisfy the conditions of join does not appear in result set. e.g.; The tuple having ROLL_NO 3 in STUDENT does not match with any tuple in STUDENT_SPORTS, so it has not been a part of result set.

Left Outer Join(⟕): When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Left Outer Joins gives all tuples of R in the result set. The tuples of R which do not satisfy join condition will have values as NULL for attributes of S.

Example:Select students whose ROLL_NO is greater than EMP_NO of employees and details of other students as well

STUDENT⟕STUDENT.ROLL_NO>EMPLOYEE.EMP_NOEMPLOYEE

RESULT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	EMP_NO	NAME	ADDRESS	PHONE	AGE
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	9455123451	18
3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	9455123451	18
1	RAM	DELHI	9455123451	18	NULL	NULL	NULL	NULL	NUL

Right Outer Join(⟖): When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Right Outer Joins gives all tuples of S in the result set. The tuples of S which do not satisfy join condition will have values as NULL for attributes of R.

Example: Select students whose ROLL_NO is greater than EMP_NO of employees and details of other Employees as well

STUDENT⟖STUDENT.ROLL_NO>EMPLOYEE.EMP_NOEMPLOYEE

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	EMP_NO	NAME	ADDRESS	PHONE	AGE
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	945512345	18

3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	945512345	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	945512345	18
NULL	NULL	NULL	NULL	NUL	5	NARES	HISAR	978291819	22
NULL	NULL	NULL	NULL	NUL	6	SWETA	RANCHI	985261762	21
NULL	NULL	NULL	NULL	NUL	4	SURESH	DELHI	915676897	18

Full Outer Join(⟗): When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Full Outer Joins gives all tuples of S and all tuples of R in the result set. The tuples of S which do not satisfy join condition will have values as NULL for attributes of R and vice versa.

Example:Select students whose ROLL_NO is greater than EMP_NO of employees and details of other Employees as well and other Students as well

STUDENT⟗$_{STUDENT.ROLL_NO>EMPLOYEE.EMP_NO}$EMPLOYEE

RESULT:

ROLL _NO	NAME	ADDRESS	PHONE	AGE	EMP_ NO	NAME	ADDRE SS	PHONE	AG E
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	9455123451	18
3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	9455123451	18
NULL	NULL	NULL	NULL	NUL	5	NARESH	HISAR	9782918192	22
NULL	NULL	NULL	NULL	NUL	6	SWETA	RANCHI	9852617621	21
NULL	NULL	NULL	NULL	NUL	4	SURESH	DELHI	9156768971	18
1	RAM	DELHI	9455123451	18	NULL	NULL	NULL	NULL	NU

Division Operator (÷): Division operator A÷B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)

- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

Consider the relation STUDENT_SPORTS and ALL_SPORTS given in Table 2 and Table 3 above.

To apply division operator as

STUDENT_SPORTS÷ ALL_SPORTS

- The operation is valid as attributes in ALL_SPORTS is a proper subset of attributes in STUDENT_SPORTS.
- The attributes in resulting relation will have attributes {ROLL_NO,SPORTS}-{SPORTS}=ROLL_NO
- The tuples in resulting relation will have those ROLL_NO which are associated with all B's tuple {Badminton, Cricket}. ROLL_NO 1 and 4 are associated to Badminton only. ROLL_NO 2 is associated to all tuples of B. So the resulting relation will be:

ROLL_NO
2

UNIT – III

SQL - Overview

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?

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. 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.

Also, they are using different dialects, such as −

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

A Brief History of SQL

- **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** − Structured Query Language appeared.
- **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

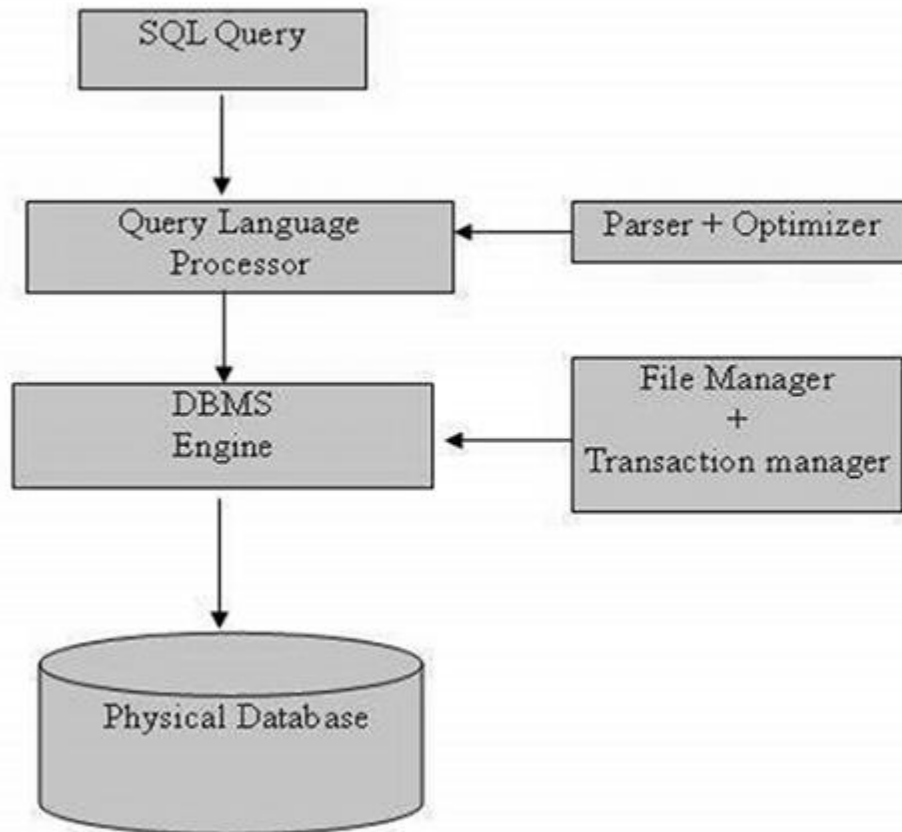
There are various components included in this process.

These components are −

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

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 −



Basic Structure of SQL Query

Basic Query Structure

SELECT field1 [,"field2",etc]
FROM table
[WHERE "condition"]
[GROUP BY "field"]
[ORDER BY "field"]

[ ] = optional

The **SELECT** statement is used to query the database and retrieve the fields that you specify. You can select as many fields (column names) as you want, or use the asterisk symbol "*" to select all fields.

The **FROM** statement specifies the table names that will be queried to retrieve the desired data.

The **WHERE** clause (optional) specifies which data values or rows will be

returned or displayed, based on the criteria you specify.

The **GROUP BY** clause (optional) organizes data into groups.

The **ORDER BY** clause (optional) sorts the data by the field specified.

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
Like	String comparison test. See Below

LIKE condition

The **LIKE** operator can be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "%" can be used as a wild card to match any possible character that might appear before or after the characters specified. For example:

SELECT name
FROM members
WHERE name LIKE 'Mar%'

Will select all names starting with "Mar" such as "Mark, Mary and Margaret"

SELECT name
FROM members
WHERE name LIKE '%ll%'

Selects all names containing the double ll combination such as "Jill, Milly and William"

ALL and DISTINCT

ALL and **DISTINCT** are keywords used to select either ALL (default) or the "distinct" (unique) records in your data base. Using the DISTINCT keyword will not display any duplicate records in the field(s) specified. ALL will display "all" of the specified fields including all of the duplicates. The ALL keyword is the default if nothing is specified.

SELECT DISTINCT firstname
FROM members

This statement will return all of the unique firstnames in the name table.

GROUP BY clause

The **GROUP BY** clause will gather all of the rows together that contain data in the field(s) and will allow aggregate functions to be performed on the one or more columns.

SELECT max(age), city, name, address
FROM members
GROUP BY city

This query will select the maximum age for the members in each unique city. Basically, the age for the person who is oldest in each city will be displayed. Their name, address and city will be returned.

ORDER BY clause

ORDER BY is an optional clause which will allow you to display the results of your query in a sorted order -- either ascending (ASC - Default) or descending (DESC) based on the fields that you specify to order by. If you would like to order based on multiple columns, you must seperate the columns with commas.

SELECT name, city, age

FROM members

ORDER by city, age DESC

SET Operations in SQL

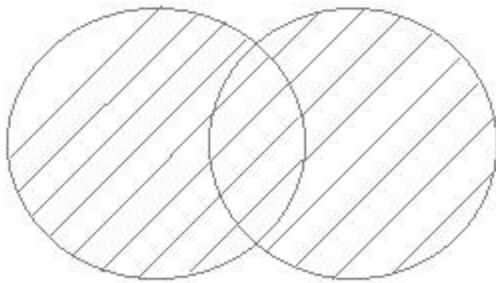
SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

In this tutorial, we will cover 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.



Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

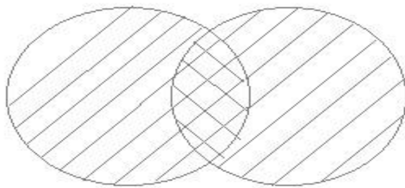
```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.



Example of Union All

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Union All query will be like,

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

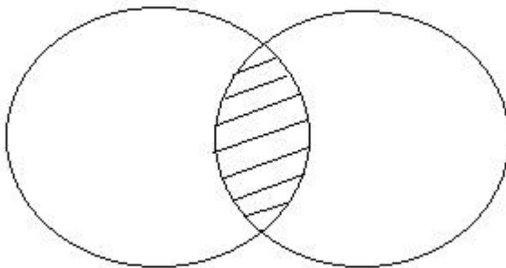
The resultset table will look like,

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.



Example of Intersect

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Intersect query will be,

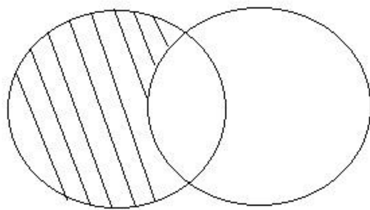
```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like

ID	NAME
2	adam

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



Example of Minus

The **First** table,

ID	NAME

1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

1) Count()
2) Sum()
3) Avg()
4) Min()
5) Max()

```
mysql> create table employee(eno int, name varchar(25), salary
int); Query OK, 0 rows affected (0.05 sec)
mysql> select * from employee;
```

```
+.................+........+.........+
| eno | name | salary |
+.................+........+.........+
| 121 | kayal | 200000 |
|  221 | kavi  | 300000 |
|  321 | abi   | 400000 |
+.................+........+.........+
3 rows in set (0.00 sec)
mysql> select min(salary) from employee;
+---------+
| min(salary) |
+---------+
|              200000  |
+---------+
1 row in set (0.03 sec)

mysql> select max(salary) from employee;
+---------+
| max(salary) |
+---------+
|              400000 |
+---------+
1 row in set (0.00 sec)

mysql> select avg(salary) from employee;
+---------+
| avg(salary) |
+---------+
| 300000 |
+---------+
1 row in set (0.00 sec)
```

```
mysql> select count(*) from employee;
+----------+
| count(*) |
+----------+
|        3    |
+----------+
1 row in set (0.00 sec)

mysql> select sum(salary) from employee;
+----------+
| sum(kil) |
+----------+
|  900000  |
+----------+
1 row in set (0.00 sec)
```

SQL NULL Values

What is a NULL Value?

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

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

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

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	Ana Trujillo	Avda. de la Constitución	México	05021	Mexico
3	Antonio Moreno Taquería	Antonio	Mataderos 2312	México	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina	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 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;

Nested Sub Queries and Complex Queries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

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.

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 −

- Subqueries must be enclosed within parentheses.
- 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.
- 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.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

  Example

  mysql> create database azhar;

- Query OK, 1 row affected (0.10 sec)
- mysql> use azhar;
- Database changed

- mysql> create table customer(Cusid int,CustomerName varchar(25),City varchar(20),Country varchar(20));
- Query OK, 0 rows affected (0.13 sec)
- mysql> create table supplier(Supplierid int,SupplierName varchar(25),City varchar(20),Country varchar(20));
- Query OK, 0 rows affected (0.05 sec)
- mysql> insert into customer values(101,'Mohamed','Perambalur','India');
- Query OK, 1 row affected (0.00 sec)
- mysql> insert into customer values(102,'Archana','Thuraiyur','India');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> insert into customer values(103,'Gayathri','Perambalur','India');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> insert into customer values(103,'Ramesh','Trichy','US');
- Query OK, 1 row affected (0.00 sec)
- mysql> insert into supplier values(101,'Umar','Chennai','India');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> insert into supplier values(102,'Azhar','Perambalur','India');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> insert into supplier values(103,'Moni','Thuraiyur','India');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> insert into supplier values(104,'Libi','Maxico','US');
- Query OK, 1 row affected (0.00 sec)
- mysql> select * from customer;
- +-------+--------------+------------+---------+
- | Cusid | CustomerName | City | Country |
- +-------+--------------+------------+---------+

- | 101 | Mohamed | Perambalur | India |
- | 102 | Archana | Thuraiyur | India |
- | 103 | Gayathri | Perambalur | India |
- | 104 | Ramesh | Trichy | US |
- +-------+--------------+------------+---------+
- 4 rows in set (0.00 sec)
- mysql> select * from supplier;
- +------------+--------------+------------+---------+
- | Supplierid | SupplierName | City | Country |
- +------------+--------------+------------+---------+
- | 101 | Umar | Chennai | India |
- | 102 | Azhar | Perambalur | India |
- | 103 | Moni | Thuraiyur | India |
- | 104 | Libi | Maxico | US |
- +------------+--------------+------------+---------+
- 4 rows in set (0.00 sec)
-
- mysql> select * from customer where **exists**(select * from supplier where customer.city=supplier.city);
- +-------+--------------+------------+---------+
- | Cusid | CustomerName | City | Country |
- +-------+--------------+------------+---------+
- | 101 | Mohamed | Perambalur | India |
- | 102 | Archana | Thuraiyur | India |
- | 103 | Gayathri | Perambalur | India |
- +-------+--------------+------------+---------+
- 3 rows in set (0.12 sec)
-
- mysql> select * from customer where exists(select * from supplier where customer.CustomerName=supplier.SupplierName);
- Empty set (0.00 sec)

-
- mysql> select * from supplier where exists(select * from customer where customer.city=supplier.city);
- +------------+--------------+------------+---------+
- | Supplierid | SupplierName | City | Country |
- +------------+--------------+------------+---------+
- | 102 | Azhar | Perambalur | India |
- | 103 | Moni | Thuraiyur | India |
- +------------+--------------+------------+---------+
- 2 rows in set (0.00 sec)
-
- mysql> select * from customer where not exists(select * from supplier where customer.city=supplier.city);
- +-------+--------------+--------+---------+
- | Cusid | CustomerName | City | Country |
- +-------+--------------+--------+---------+
- | 104 | Ramesh | Trichy | US |
- +-------+--------------+--------+---------+
- 1 row in set (0.00 sec)
-
- mysql> select * from supplier where not exists(select * from customer where customer.city=supplier.city);
- +------------+--------------+---------+---------+
- | Supplierid | SupplierName | City | Country |
- +------------+--------------+---------+---------+
- | 101 | Umar | Chennai | India |
- | 104 | Libi | Maxico | US |
- +------------+--------------+---------+---------+
- 2 rows in set (0.00 sec)
-
- mysql> select * from customer;

- +-------+--------------+------------+---------+
- | Cusid | CustomerName | City | Country |
- +-------+--------------+------------+---------+
- | 101 | Mohamed | Perambalur | India |
- | 102 | Archana | Thuraiyur | India |
- | 103 | Gayathri | Perambalur | India |
- | 104 | Ramesh | Trichy | US |
- +-------+--------------+------------+---------+
- 4 rows in set (0.00 sec)
-
- mysql> select DISTINCT City from customer;
- +------------+
- | City |
- +------------+
- | Perambalur |
- | Thuraiyur |
- | Trichy |
- +------------+
- 3 rows in set (0.00 sec)
-
- mysql> select DISTINCT Country from customer;
- +---------+
- | Country |
- +---------+
- | India |
- | US |
- +---------+
- 2 rows in set (0.00 sec)

Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

CREATE VIEW *view_name* AS

SELECT *column1*, *column2*, ...

FROM *table_name*

WHERE *condition*;

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

mysql> create database azhar;

Query OK, 1 row affected (0.10 sec)

mysql> use azhar;

Database changed

mysql> create table customer(Cusid int,CustomerName varchar(25),City varchar(20),Country varchar(20));

Query OK, 0 rows affected (0.13 sec)

mysql> insert into customer values(101,'Mohamed','Perambalur','India');

Query OK, 1 row affected (0.00 sec)

mysql> insert into customer values(102,'Archana','Thuraiyur','India');

Query OK, 1 row affected (0.00 sec)

mysql> insert into customer values(103,'Gayathri','Perambalur','India');

Query OK, 1 row affected (0.00 sec)

mysql> insert into customer values(103,'Ramesh','Trichy','US');

Query OK, 1 row affected (0.00 sec)

mysql> select * from customer;

```
+-------+--------------+------------+---------+
| Cusid | CustomerName | City       | Country |
+-------+--------------+------------+---------+
```

```
| 101 | Mohamed      | Perambalur | India   |
| 102 | Archana      | Thuraiyur  | India   |
| 103 | Gayathri     | Perambalur | India   |
| 104 | Ramesh       | Trichy     | US      |
+-------+--------------+------------+---------+
4 rows in set (0.00 sec)
```

```
mysql> create view status as select Cusid,CustomerName from customer where country='India';
Query OK, 0 rows affected (0.07 sec)

myql> select * from status;
+-------+--------------+
| Cusid | CustomerName |
+-------+--------------+
|   101 | Mohamed      |
|   102 | Archana      |
|   103 | Gayathri     |
+-------+--------------+
3 rows in set (0.03 sec)
mysql> create view details as select Cusid,CustomerName,City from customer where
City='Perambalur';
Query OK, 0 rows affected (0.06 sec)
mysql> select * from details;
+-------+--------------+------------+
| Cusid | CustomerName | City       |
+-------+--------------+------------+
|   101 | Mohamed      | Perambalur |
|   103 | Gayathri     | Perambalur |
+-------+--------------+------------+
2 rows in set (0.00 sec)
```

Modification of the Database

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

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 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

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

=>

After

pno	descr	color
P1	Widget	Blue
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

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

=>

After

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

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

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

=>

After

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

sno	name	city
S1	Pierre	Paris
S2	John	London

=>

After

sno	name	city
S1	Pierre	Paris
S2	John	London

S3	Mario	Rome

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.

DELETE Examples

DELETE FROM sp WHERE pno = 'P1'

Before

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

=>

After

sno	pno	qty
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

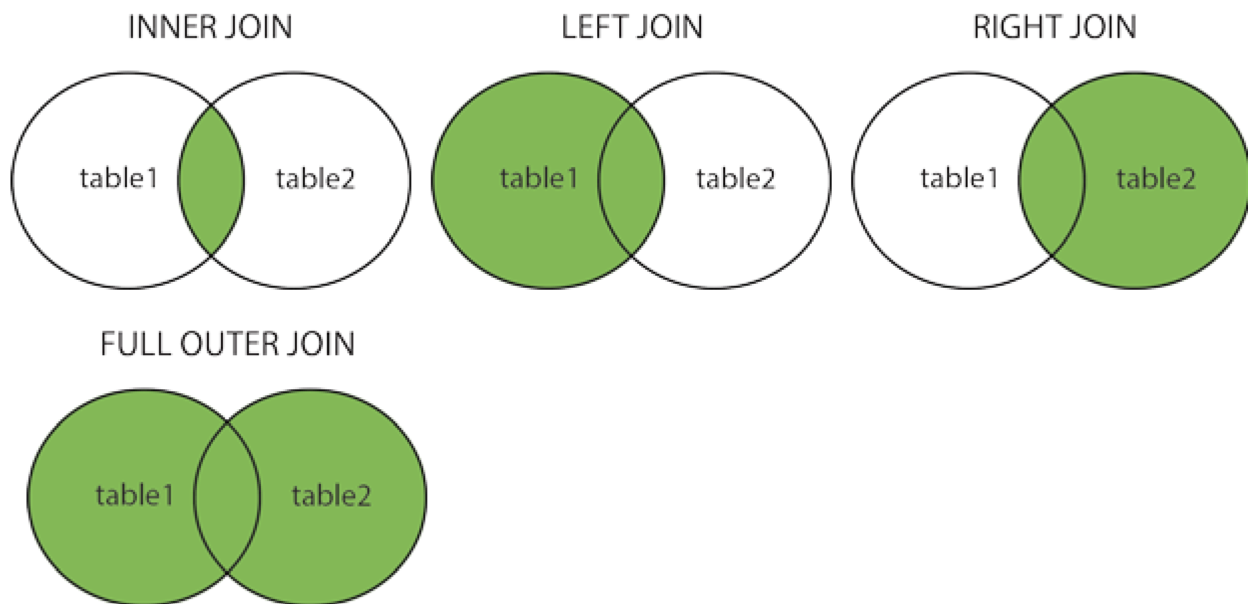
Joined Relations

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



mysql> create table one(id int,name varchar(15),city varchar(15));

Query OK, 0 rows affected (0.13 sec)

mysql> create table two(id int,name varchar(15),city varchar(15));

Query OK, 0 rows affected (0.03 sec)

mysql> insert into one values(101,'Ram','Trichy');

Query OK, 1 row affected (0.00 sec)

mysql> insert into one values(102,'Aanand','Perambalur');

Query OK, 1 row affected (0.00 sec)

mysql> insert into one values(103,'Rajkumar','Ariyalur');

Query OK, 1 row affected (0.00 sec)

mysql> insert into one values(104,'Azhar','Ariyalur');

Query OK, 1 row affected (0.00 sec)

```
mysql> insert into two values(101,'Ranjani','Perambalur');
Query OK, 1 row affected (0.00 sec)
mysql> insert into two values(102,'Gayathri','Perambalur');
Query OK, 1 row affected (0.00 sec)
mysql> insert into one values(103,'Archana','Trichy');
Query OK, 1 row affected (0.00 sec)
mysql> insert into two values(103,'Rajkumar','Trichy');
Query OK, 1 row affected (0.00 sec)
mysql> insert into two values(104,'Azhar','Trichy');
Query OK, 1 row affected (0.00 sec) mysql> select * from one;
mysql> insert into two values(106,'Ubaid','Trichy');
Query OK, 1 row affected (0.00 sec)
+------+----------+------------+
| id   | name     | city       |
+------+----------+------------+
|  101 | Ram      | Trichy     |
|  102 | Aanand   | Perambalur |
|  103 | Rajkumar | Ariyalur   |
|  104 | Azhar    | Ariyalur   |
|  105 | Archana  | Trichy     |
+------+----------+------------+
5 rows in set (0.00 sec)

mysql> select * from two;
+------+----------+------------+
| id   | name     | city       |
+------+----------+------------+
|  101 | Ranjani  | Perambalur |
|  102 | Gayathri | Perambalur |
|  103 | Rajkumar | Trichy     |
|  104 | Azhar    | Trichy     |
```

```
| 106 | Ubaid    | Trichy     |
+------+----------+------------+
5 rows in set (0.00 sec)
mysql> select * from one inner join two;
+------+----------+------------+------+----------+------------+
| id   | name     | city       | id   | name     | city       |
+------+----------+------------+------+----------+------------+
| 101 | Ram      | Trichy     | 101 | Ranjani  | Perambalur |
| 101 | Ram      | Trichy     | 102 | Gayathri | Perambalur |
| 101 | Ram      | Trichy     | 103 | Rajkumar | Trichy     |
| 101 | Ram      | Trichy     | 104 | Azhar    | Trichy     |
| 102 | Aanand   | Perambalur | 101 | Ranjani  | Perambalur |
| 102 | Aanand   | Perambalur | 102 | Gayathri | Perambalur |
| 102 | Aanand   | Perambalur | 103 | Rajkumar | Trichy     |
| 102 | Aanand   | Perambalur | 104 | Azhar    | Trichy     |
| 103 | Rajkumar | Ariyalur   | 101 | Ranjani  | Perambalur |
| 103 | Rajkumar | Ariyalur   | 102 | Gayathri | Perambalur |
| 103 | Rajkumar | Ariyalur   | 103 | Rajkumar | Trichy     |
| 103 | Rajkumar | Ariyalur   | 104 | Azhar    | Trichy     |
| 104 | Azhar    | Ariyalur   | 101 | Ranjani  | Perambalur |
| 104 | Azhar    | Ariyalur   | 102 | Gayathri | Perambalur |
| 104 | Azhar    | Ariyalur   | 103 | Rajkumar | Trichy     |
| 104 | Azhar    | Ariyalur   | 104 | Azhar    | Trichy     |
| 105 | Archana  | Trichy     | 101 | Ranjani  | Perambalur |
| 105 | Archana  | Trichy     | 102 | Gayathri | Perambalur |
| 105 | Archana  | Trichy     | 103 | Rajkumar | Trichy     |
| 105 | Archana  | Trichy     | 104 | Azhar    | Trichy     |
+------+----------+------------+------+----------+------------+
20 rows in set (0.00 sec)
mysql> select * from one left join two on one.id=two.id;
+------+----------+------------+------+----------+------------+
```

```
| id   | name     | city       | id   | name     | city       |
+------+----------+------------+------+----------+------------+
|  101 | Ram      | Trichy     |  101 | Ranjani  | Perambalur |
|  102 | Aanand   | Perambalur |  102 | Gayathri | Perambalur |
|  103 | Rajkumar | Ariyalur   |  103 | Rajkumar | Trichy     |
|  104 | Azhar    | Ariyalur   |  104 | Azhar    | Trichy     |
|  105 | Archana  | Trichy     | NULL | NULL     | NULL       |
+------+----------+------------+------+----------+------------+
5 rows in set (0.00 sec)
mysql> select * from one right join two on one.id=two.id;
+------+----------+------------+------+----------+------------+
| id   | name     | city       | id   | name     | city       |
+------+----------+------------+------+----------+------------+
|  101 | Ram      | Trichy     |  101 | Ranjani  | Perambalur |
|  102 | Aanand   | Perambalur |  102 | Gayathri | Perambalur |
|  103 | Rajkumar | Ariyalur   |  103 | Rajkumar | Trichy     |
|  104 | Azhar    | Ariyalur   |  104 | Azhar    | Trichy     |
| NULL | NULL     | NULL       |  106 | Ubaid    | Trichy     |
+------+----------+------------+------+----------+------------+
5 rows in set (0.00 sec)
mysql> select * from one join two on one.id=two.id;
+------+----------+------------+------+----------+------------+
| id   | name     | city       | id   | name     | city       |
+------+----------+------------+------+----------+------------+
|  101 | Ram      | Trichy     |  101 | Ranjani  | Perambalur |
|  102 | Aanand   | Perambalur |  102 | Gayathri | Perambalur |
|  103 | Rajkumar | Ariyalur   |  103 | Rajkumar | Trichy     |
|  104 | Azhar    | Ariyalur   |  104 | Azhar    | Trichy     |
+------+----------+------------+------+----------+------------+
4 rows in set (0.00 sec)
```

SQL Data Types and Schemas

Built-in Data Types

date: Dates, containing a (4 digit) year, month and date Example: date '2005-7-27'

time: Time of day, in hours, minutes and seconds.ν Example: time '09:00:30' time '09:00:30.75'

timestamp: date plus time of dayν Example: timestamp '2005-7-27 09:00:30.75'

interval: period of timeν Example: interval '1' dayλ Subtracting a date/time/timestamp value from another gives anλ interval value Interval values can be added to date/time/timestamp values

User Defined Types

create type construct in SQL creates user-defined type create type Dollars as numeric (12,2) final

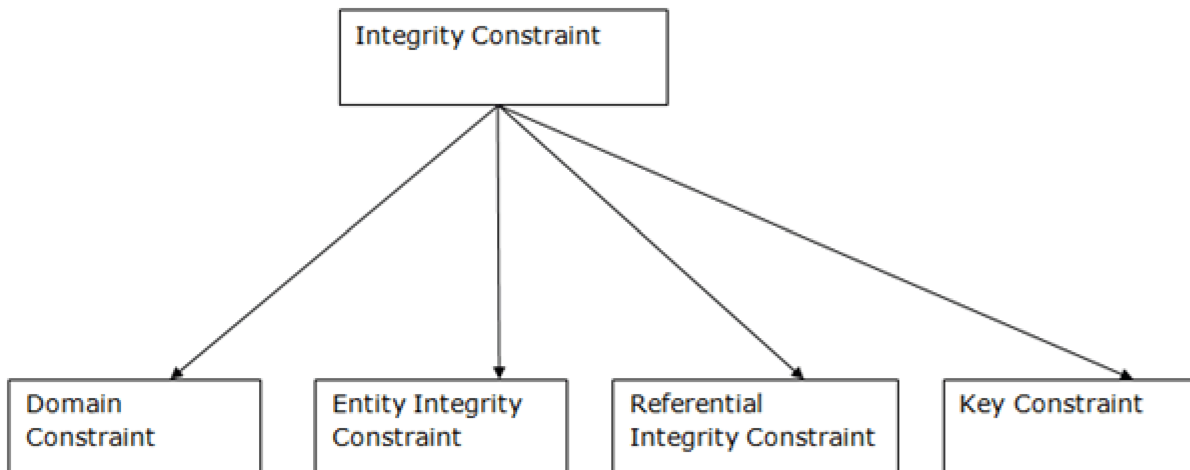
create domain construct in SQL-92 creates user-defined domainν types create domain person_name char(20) not null

Types and domains are similar. Domains can have constraints, suchν as not null, specified on them.

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Although many DBMSs and case tools use this as a way of instilling referential integrity, it overly complicates the process unnecessarily.

- The occurrence of limited set values to domain data represent what is called the permitted value set for the domain. It represents metadata for that domain. Here are some examples of domain constraints:
- *Valid value sets.* These are valid translation values for a particular data item. These include code tables, translation tables, and existence check tables. For example, CT might be a valid value for state code 21 in a valid state code table.
- *Valid range table.* These are valid ranges for a particular data item. These can be numeric/alphanumeric range edit tables or reasonability range tables. An example of this would be state code must be a value between 01 and 52.
- *Algorithmic derived data.* This is data that is derivable by computational activity, such as adding, subtracting, multiplying or dividing a data item. An example of this would be review date = hire date + 180.

- *Translation.* These are in effect valid value set tables that are not used for validation but as a print translation table that allows processing to be completed on the codified data and translated only when it has to be presented to the outside world, such as on a transaction screen or on a print.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

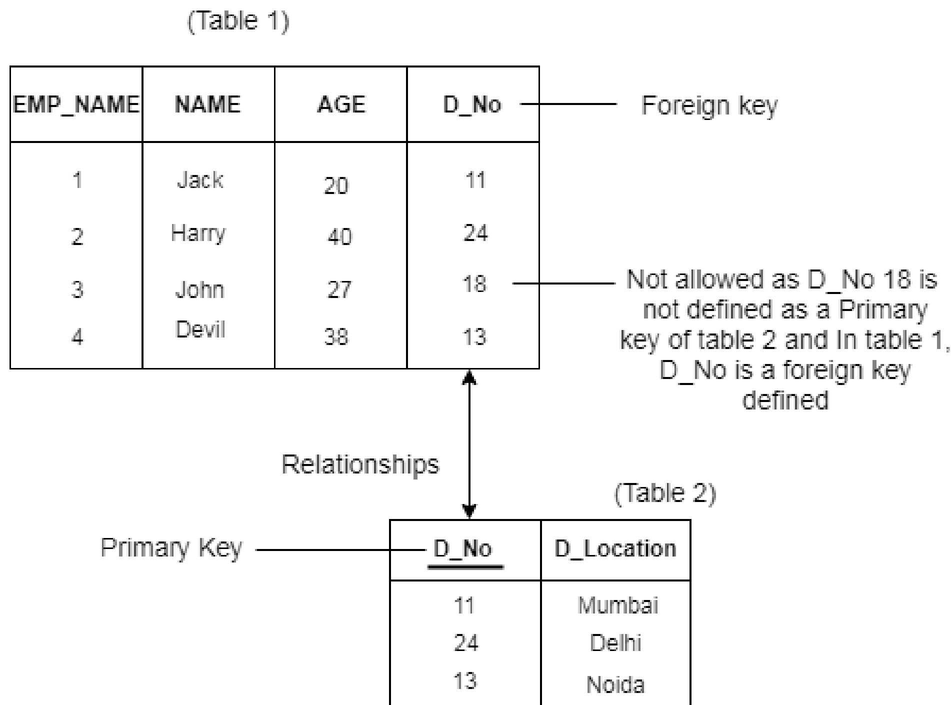
Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.

- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

CREATE TABLE *table_name* (
column1 datatype constraint,
column2 datatype constraint,
column3 datatype constraint,
....
);

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. 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.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

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

Authorization in Sql

When you create database objects, you must explicitly grant permissions to make them accessible to users. Every securable object has permissions that can be granted to a principal using permission statements.

Role-Based Permissions

Granting permissions to roles rather than to users simplifies security administration. Permission sets that are assigned to roles are inherited by all members of the role. It is easier to add or remove users from a role than it is to recreate separate permission sets for individual users. Roles can be nested; however, too many levels of nesting can degrade performance. You can also add users to fixed database roles to simplify assigning permissions.

You can grant permissions at the schema level. Users automatically inherit permissions on all new objects created in the schema; you do not need to grant permissions as new objects are created.

Permission Statements

The three Transact-SQL permission statements are described in the following table.

TABLE 1

Permission Statement	Description
GRANT	Grants a permission.
REVOKE	Revokes a permission. This is the default state of a new object. A permission revoked from a user or role can still be inherited from other groups or roles to which the principal is assigned.
DENY	DENY revokes a permission so that it cannot be inherited. DENY takes precedence over all permissions, except DENY does not apply to object owners or members of sysadmin. If you DENY permissions on an object to the public role it is denied to all users and roles except for object owners and sysadmin members.

- The GRANT statement can assign permissions to a group or role that can be inherited by database users. However, the DENY statement takes precedence over all other permission statements. Therefore, a user who has been denied a permission cannot inherit it from another role.

Embedded SQL

SQL stands for Structured Query Language, it provides as a declarative query language. However, a general-purpose programming language requires to get access to the database because

- SQL is not as powerful as any of the general purpose language available today.
- There are many declarative actions such as interacting with the user sending the result to a GUI or printing a report which we cannot do using SQL.
- There are many queries that we can express in C, Pascal, Cobol and many more but we cannot express in SQL.

What is Embedded SQL?

This is a method for combining data manipulation capabilities of SQL and computing power of any programming language. Then embedded statements are in line with the program source code of the host language. The code of embedded SQL is parsed by a preprocessor which is also embedded and is replaced by the host language called for the code library it is then compiled via the compiler of the host.

Two steps which define by SQL standards community, they are –

Module language defining which is formalization and then an embedded SQL standard derived from the module language. Most popular hosting language is C, it is called for example Pro*C in Oracle and Sybase database management systems and ECPG in the PostgreSQL database management system.

Need for Embedded SQL in DBMS

When you embed SQL with another language. The language that is embedded is known as host language and the SQL standard which defines the embedding of SQL is known as embedded SQL.

- The result of a query is made available to the program which is embedded as one tuple or record at a time
- For identification of this, we request to the preprocessor via EXEC SQL statement: EXEC SQL embedded SQL statement END-EXEC

- Its statements are declare cursor, fetch and open statements.
- It can execute the update, insert a delete statement

Systems that Support Embedded SQL

i. Altibase

- C/C++

An embedded SQL precompiler is given by Altibase Corp. for its DBMS server.

- IBM DB2

The version 9 IBM DB2 for **Linux**, UNIX and Windows support it for C, C++, **Java**, COBOL, FORTRAN, and REXX although they don't support FORTRAN and REXX.

ii. Microsoft SQL Server

- C/C++

Embedded SQL for C has been discontinued by Microsoft SQL Server 2008 although earlier versions of the product support it.

- Mimer SQL

They support it, namely Linux, OpenVMS, and Windows

- C/C++

Linux, OpenVMS, and Windows support this.

- COBOL

OpenVMS support embedded SQL for COBOL

- Fortran

OpenVMS support embedded SQL for Fortran.

iii. Oracle Database

- Ada

Pro*Ada was officially desupported by Oracle in version 7.3. Starting with Oracle8, Pro*Ada was replaced by SQL*Module but appears to have not been updated since. It supports the Ada83 language standard for Ada.

- C/C++

Pro*C became Pro*C/C++ with Oracle8. Oracle Database 11g supports Pro*C/C++ .

- COBOL

Oracle Database 11g supports Pro*COBOL version.

- Fortran

Pro*FORTRAN is no longer updated as of Oracle8 although Oracle will continue to issue patch releases as bugs are reported and corrected.

- Pascal

Pro*Pascal was not released with Oracle8.

- PL/I

Pro*PL/I has been removed from the Oracle Documentation Library.

Do you know about SQL Comment?

iv. PostgreSQL

- C/C++

ECPG is part of PostgreSQL since the arrival of version 6.3.

- COBOL

Cobol-IT is presently giving a COBOL precompiler for PostgreSQL.

v. Raima Database Manager (RDM)

The 14.0 version supports it for C/C++ and PL/SQL.

vi. SAP Sybase

The 15.7 version supports embedding of SQL for C and COBOL as making it a part of the Software Developer Kit Sybase.

SAP Sybase SQL Anywhere supports it for C and C++ too as it is a part of the SQL Anywhere database management system SQL Anywhere.

4. Embedding of SQL Through Domain-Specific Languages

LINQ-to-SQL- It embeds SQL-like language into .NET languages.

JPA- It embeds **SQL**-like language via Criteria API into Java.

jOOQ- It embeds SQL-like language to Java.

So, this was all in Embedded SQL. Hope you liked our explanation.

Unit - IV

Relational Calculus

Relational calculus is a non procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result where as relational algebra gives the method to get the result. It informs the system what to do with the relation, but does not inform how to perform it.

For example, steps involved in listing all the students who attend 'Database' Course in relational algebra would be

- SELECT the tuples from COURSE relation with COURSE_NAME = 'DATABASE'
- PROJECT the COURSE_ID from above result
- SELECT the tuples from STUDENT relation with COUSE_ID resulted above.

In the case of relational calculus, it is described as below:

Get all the details of the students such that each student have course as 'Database'.

See the difference between relational algebra and relational calculus here. From the first one, we are clear on how to query and which relations to be queried. But the second tells what needs to be done to get the students with 'database' course. But it does tell us how we need to proceed to achieve this. Relational calculus is just the explanative way of telling the query.

There are two types of relational calculus – Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).

Tuple Relational Calculus (TRC) in DBMS

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Calculus, a query is expressed as

```
{t| P(t)}
```

where t = resulting tuples,
P(t) = known as Predicate and these are the conditions that are used to fetch t

Thus, it generates set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).
It also uses quantifiers:
∃ t ∈ r (Q(t)) = "there exists" a tuple in t in relation r such that predicate Q(t) is true.
∀ t ∈ r (Q(t)) = Q(t) is true "for all" tuples in relation r.

Example:

Table-1: Customer

CUSTOMER NAME	STREET	CITY
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table-2: Branch

BRANCH NAME	BRANCH CITY
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-3: Account

ACCOUNT NUMBER	BRANCH NAME	BALANCE
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table-4: Loan

LOAN NUMBER	BRANCH NAME	AMOUNT
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table-5: Borrower

CUSTOMER NAME	LOAN NUMBER
Saurabh	L33
Mehak	L49
Ria	L98

Table-6: Depositor

CUSTOMER NAME	ACCOUNT NUMBER
Saurabh	1111
Mehak	1113
Sumiti	1114

Queries-1: Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

Resulting relation:

LOAN NUMBER	BRANCH NAME	AMOUNT
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, t[amount] is known as tupple variable.

Queries-2: Find the loan number for each loan of an amount greater or equal to 10000.

{t| ∃ s ∈ loan(t[loan number] = s[loan number]

∧ s[amount]>=10000)}

Resulting relation:

LOAN NUMBER
L33
L35
L98

Queries-3: Find the names of all customers who have a loan and an account at the bank.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name])

∧ ∃ u ∈ depositor(t[customer-name] = u[customer-name])}

Resulting relation:

CUSTOMER NAME
Saurabh
Mehak

Queries-4: Find the names of all customers having a loan at the “ABC” branch.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name]

∧ ∃ u ∈ loan(u[branch-name] = “ABC” ∧ u[loan-number] = s[loan-number]))}

Resulting relation:

CUSTOMER NAME
Saurabh

Domain Relational Calculus in DBMS

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

$\{ < x_1, x_2, x_3, ..., x_n > | P (x_1, x_2, x_3, ..., x_n) \}$

where, $< x_1, x_2, x_3, \ldots, x_n >$ represents resulting domains variables and $P (x_1, x_2, x_3, \ldots, x_n)$ represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:

Table-1: Customer

CUSTOMER NAME	STREET	CITY
Debomit	Kadamtala	Alipurduar
Sayantan	Udaypur	Balurghat
Soumya	Nutanchati	Bankura
Ritu	Juhu	Mumbai

Table-2: Loan

LOAN NUMBER	BRANCH NAME	AMOUNT
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

CUSTOMER NAME	LOAN NUMBER
Ritu	L01
Debomit	L08
Soumya	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{<l, b, a> \mid <l, b, a> \in loan \land (a \geq 100)\}$

Resulting relation:

LOAN NUMBER	BRANCH NAME	AMOUNT
L01	Main	200
L03	Main	150
L10	Sub	90

Query-2: Find the loan number for each loan of an amount greater or equal to 150.

$\{<l> \mid \exists\, b, a\ (<l, b, a> \in loan \land (a \geq 150)\}$

Resulting relation:

LOAN NUMBER
L01
L03

Query-3: Find the names of all customers having a loan at the "Main" branch and find the loan amount .

$\{<c, a> \mid \exists\, l\ (<c, l> \in borrower \land \exists\, b\ (<l, b, a> \in loan \land (b = \text{"Main"})))\}$

Resulting relation:

CUSTOMER NAME	AMOUNT

CUSTOMER NAME	AMOUNT
Ritu	200
Debomit	60
Soumya	150

Note:
The domain variables those will be in resulting relation must appear before | within $\prec$ and $\succ$ and all the domain variables must appear in which order they are in original relation or table.

Query-by- Example

QBE Stands for "Query By Example." QBE is a feature included with various database applications that provides a user-friendly method of running database queries. Typically without QBE, a user must write input commands using correct SQL (Structured Query Language) syntax. This is a standard language that nearly all database programs support. However, if the syntax is slightly incorrect the query may return the wrong results or may not run at all.

The Query By Example feature provides a simple interface for a user to enter queries. Instead of writing an entire SQL command, the user can just fill in blanks or select items to define the query she wants to perform. For example, a user may want to select an entry from a table called "Table1" with an ID of 123. Using SQL, the user would need to input the command, "SELECT * FROM Table1 WHERE ID = 123". The QBE interface may allow the user to just click on Table1, type in "123" in the ID field and click "Search."

QBE is offered with most database programs, though the interface is often different between applications. For example, Microsoft Access has a QBE interface known as "Query Design View" that is completely graphical. The phpMyAdmin application used with MySQL, offers a Web-based interface where users can select a query operator and fill in blanks with search terms. Whatever QBE implementation is provided with a program, the purpose is the same – to make it easier to run database queries and to avoid the frustrations of SQL errors.

Query by example (QBE) is a query method implemented in most database systems, most notably for relational databases. QBE was created by Moshe Zloof at IBM in the 1970s in parallel to SQL's development. It is a graphical query language where users can input commands into a table like conditions and example elements. It's a common feature in most database programs.

Query by example is a query language used in relational databases that allows users to search for information in tables and fields by providing a simple user interface where the user will be able to input an example of the data that he or she wants to access. The principle of QBE is that it is merely an abstraction between the user and the real query that the database system will receive. In the background, the user's query is transformed into a database manipulation language form such as SQL, and it is this SQL statement that will be executed in the background.

QBE, like SQL, was developed at IBM and QBE is an IBM trademark, but a number of other companies sell QBE-like interfaces, including Paradox. Some systems, such as Microsoft Access,

offer partial support for form-based queries and reflect the influence of QBE. Often a QBE-like interface is offered in addition to SQL, with QBE serving as a more intuitive user-interface for simpler queries and the full power of SQL available for more complex queries. An appreciation of the features of QBE offers insight into the more general, and widely used, paradigm of tabular query interfaces for relational databases.

Overview of the Design Process

What is Database Design?

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

The physical data design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

Why Database Design is Important ?

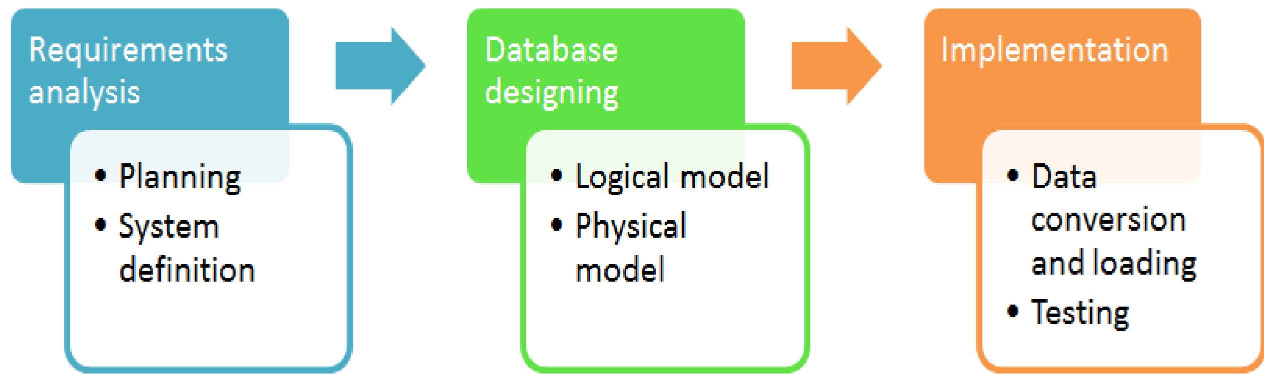
It helps produce database systems

1. That meet the requirements of the users
2. Have high performance.

Database designing is crucial to **high performance** database system.

Note , the genius of a database is in its design . Data operations using SQL is relatively simple

Database development life cycle



The database development life cycle has a number of stages that are followed when developing database systems.

The steps in the development life cycle do not necessary have to be followed religiously in a sequential manner.

On small database systems, the database system development life cycle is usually very simple and does not involve a lot of steps.

In order to fully appreciate the above diagram, let's look at the individual components listed in each step.

Requirements analysis

- **Planning** - This stages concerns with planning of entire Database Development Life Cycle It takes into consideration the Information Systems strategy of the organization.
- **System definition** - This stage defines the scope and boundaries of the proposed database system.

Database designing

- **Logical model** - This stage is concerned with developing a database model based on requirements. The entire design is on paper without any physical implementations or specific DBMS considerations.
- **Physical model** - This stage implements the logical model of the database taking into account the DBMS and physical implementation factors.

Implementation

- **Data conversion and loading** - this stage is concerned with importing and converting data from the old system into the new database.
- **Testing** - this stage is concerned with the identification of errors in the newly implemented system .It checks the database against requirement specifications.

The Entity-Relationship Model

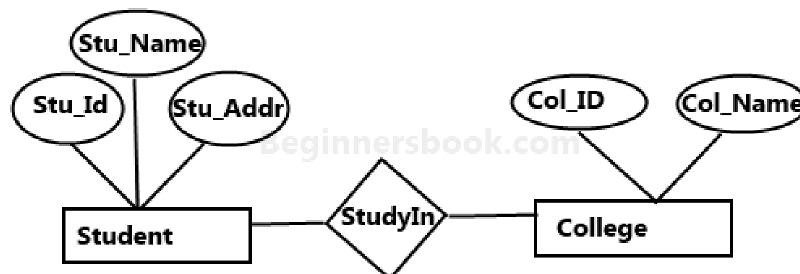
An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

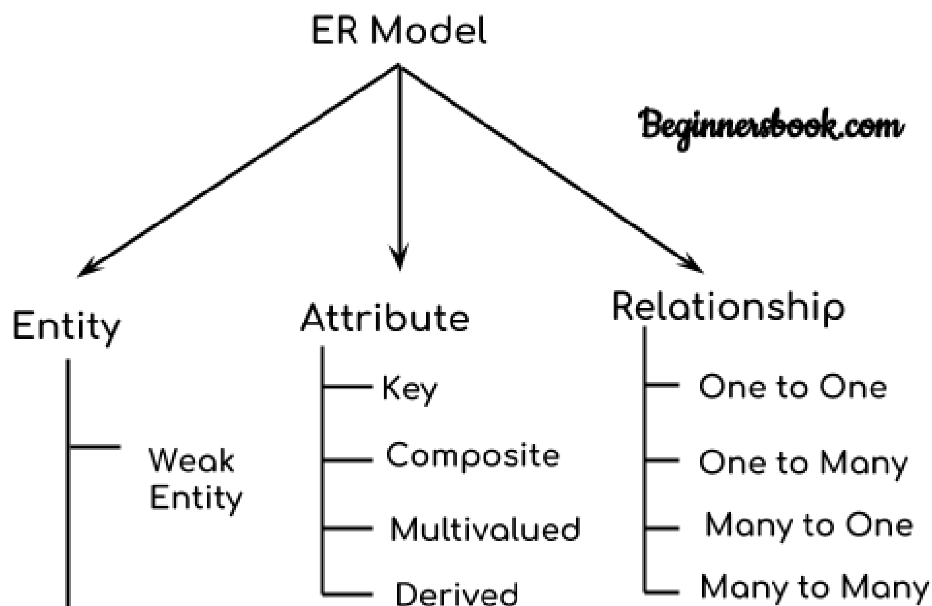


Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram

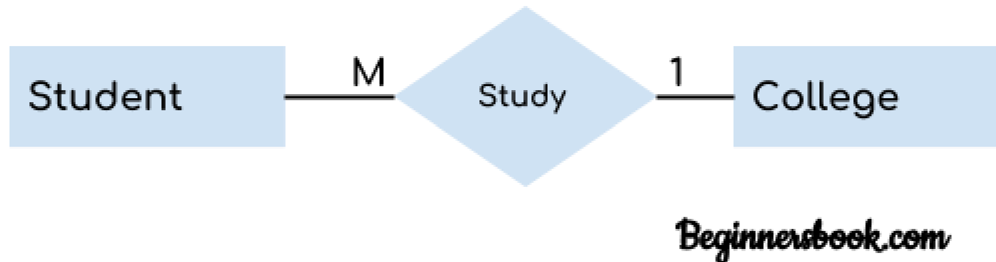


As shown in the above diagram, an ER diagram has three main components:

1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram. For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.



Weak Entity:
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

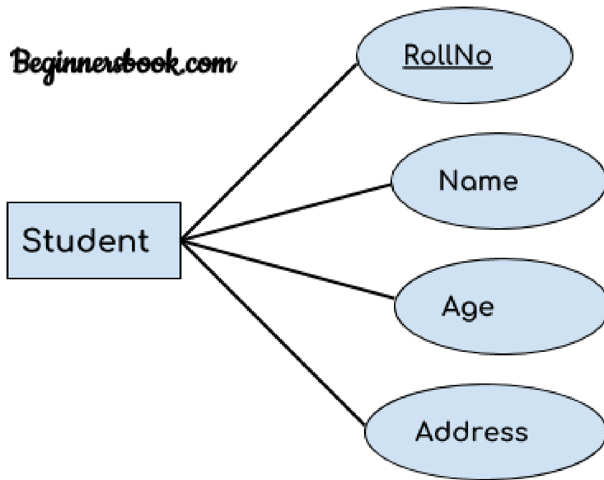
2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

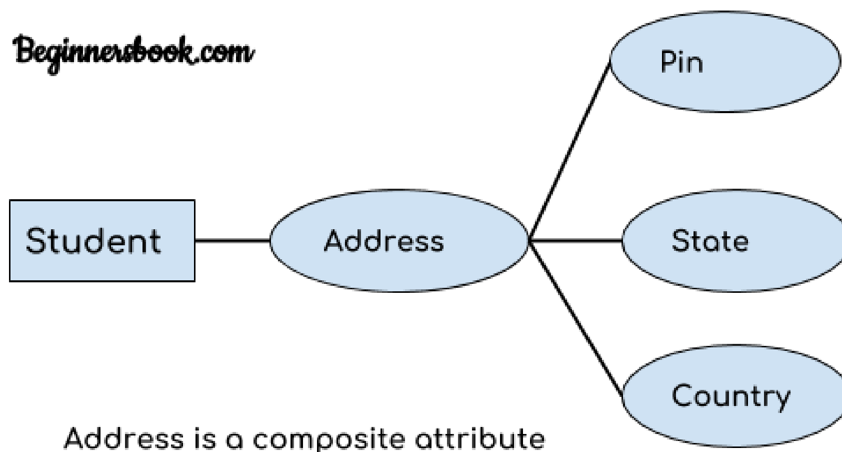
1. Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.



2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



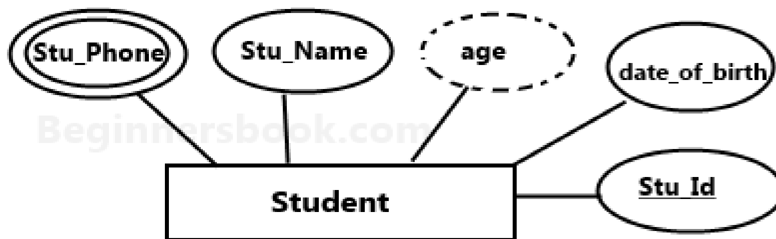
3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:

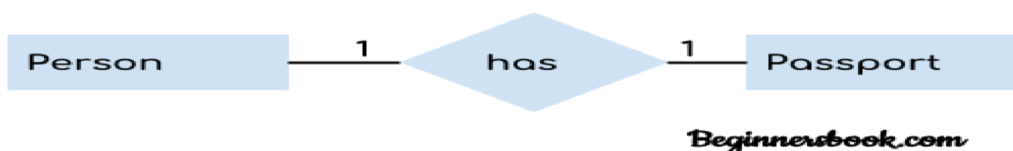


3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

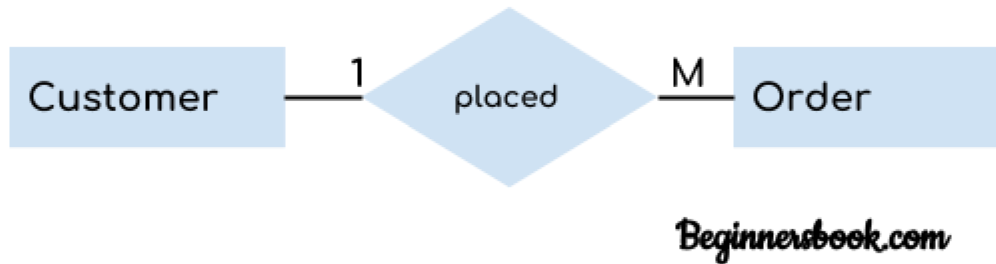
1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.



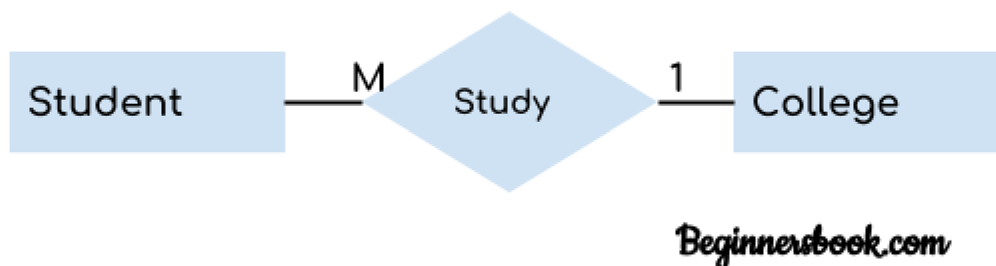
2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.



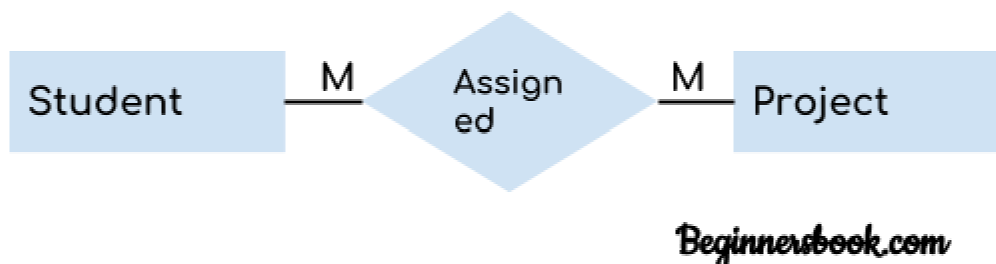
3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



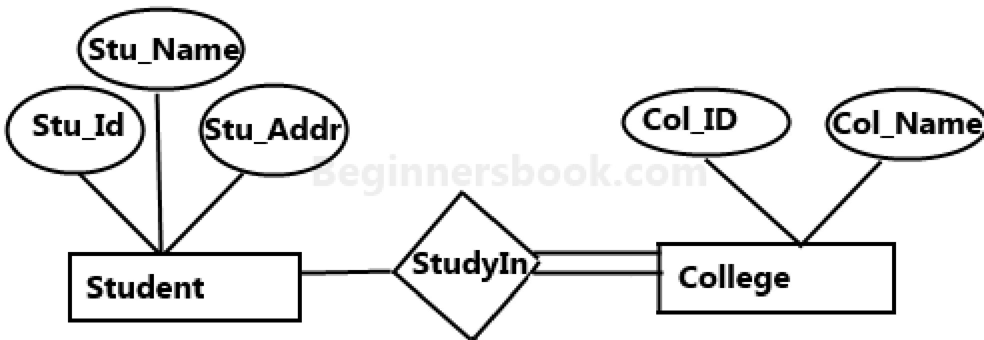
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

E-R Model

ENTITY RELATIONAL (ER) MODEL is a high-level conceptual data model diagram. ER modeling helps you to analyze data requirements systematically to produce a well-designed database. The Entity-Relation model represents real-world entities and the relationship between them. It is considered a best practice to complete ER modeling before implementing your database.

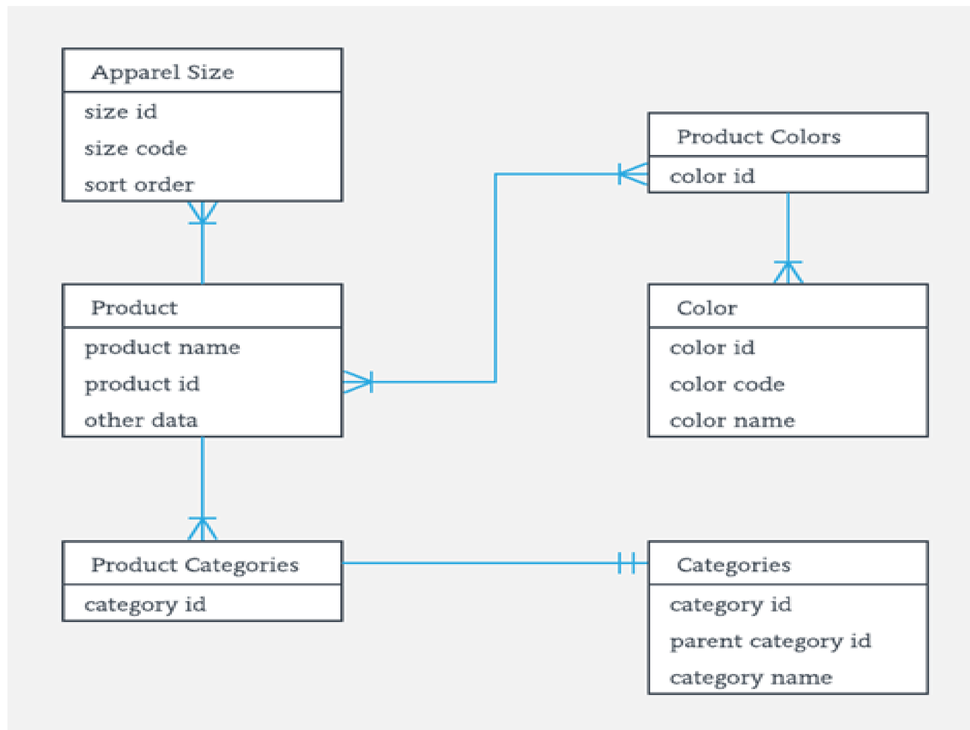
ER modeling helps you to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing your database.

History of ER models

ER diagrams are a visual tool which is helpful to represent the ER model. It was proposed by Peter Chen in 1971 to create a uniform convention which can be used for relational database and network. He aimed to use an ER model as a conceptual modeling approach.

What is ER Diagrams?

ENTITY-RELATIONSHIP DIAGRAM (ERD) displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.



Sample ER Diagram

Why use ER Diagrams?

Here, are prime reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD is allowed you to communicate with the logical structure of the database to users

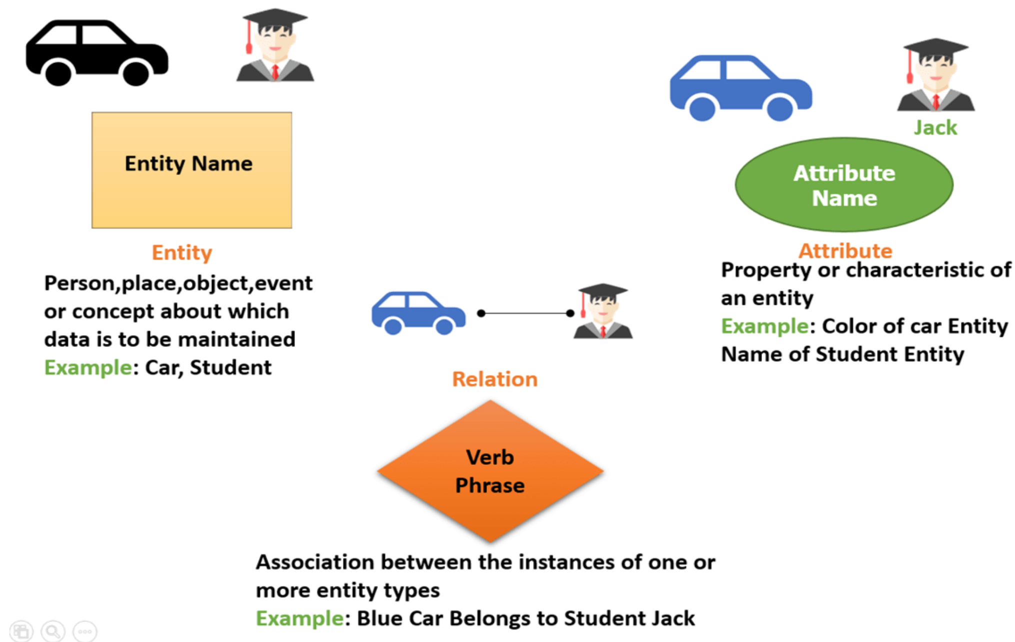
Components of the ER Diagram

This model is based on three basic concepts:

- Entities
- Attributes
- Relationships

Example

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.



Facts about ER Diagram Model:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

WHAT IS ENTITY?

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Examples of entities:

- **Person:** Employee, Student, Patient
- **Place:** Store, Building
- **Object:** Machine, product, and Car

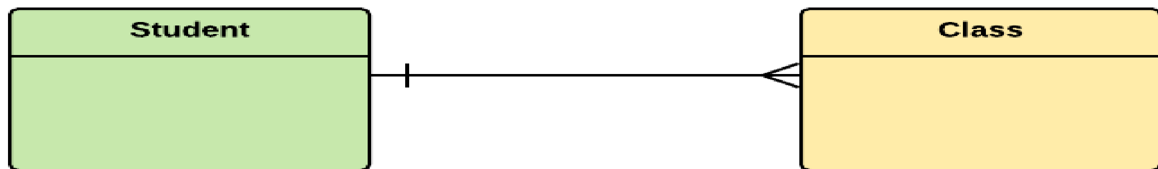
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course

Notation of an Entity

Entity set:

Student

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.

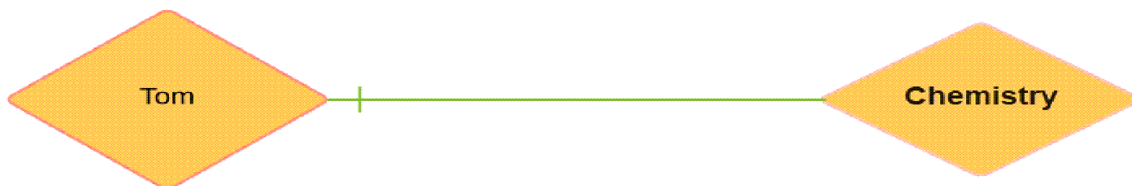


Example of Entities:

A university may have some departments. All these departments employ various lecturers and offer several programs. Some courses make up each program. Students register in a particular program and enroll in various courses. A lecturer from the specific department takes each course, and each lecturer teaches a various group of students.

Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.



Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

Strong Entity Set	**Weak Entity Set**
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.

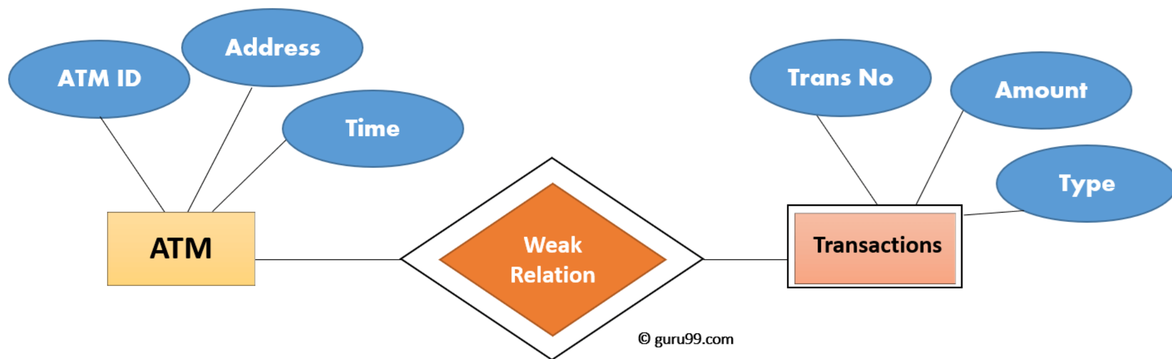
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

For example:

- You are attending this lecture
- I am giving the lecture
- Just loke entities, we can classify relationships according to relationship-types:
- A student attends a lecture
- A lecturer is giving a lecture.

Weak Entities

weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.



In aboe example, "Trans No" is a discriminator within a group of transactions in an ATM.

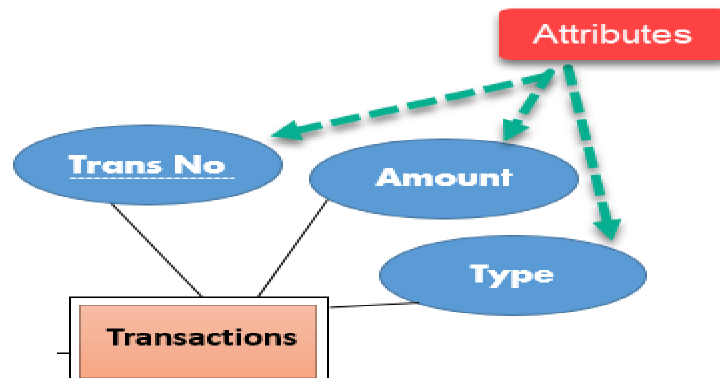
Let's learn more about a weak entity by comparing it with a Strong Entity

Attributes

It is a single-valued property of either an entity-type or a relationship-type.

For example, a lecture might have attributes: time, date, duration, place, etc.

An attribute is represented by an Ellipse



Types of Attributes	Description
Simple attribute	Simple attributes can't be divided any further. For example, a student's contact number. It is also called an atomic value.
Composite attribute	It is possible to break down composite attribute. For example, a student's full name may be further divided into first name, second name, and last name.
Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age

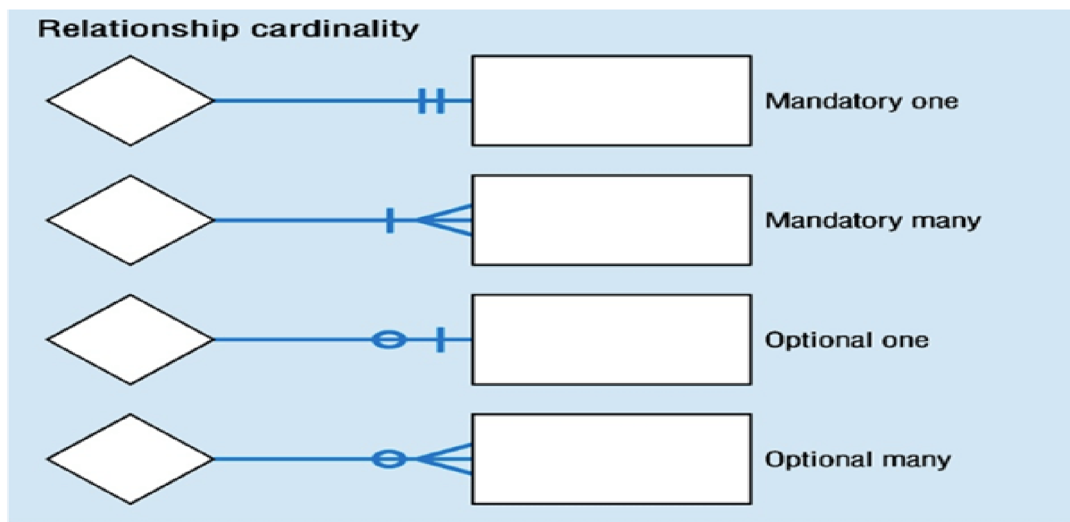
	should not be stored directly. Instead, it should be derived from the DOB of that employee.
Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

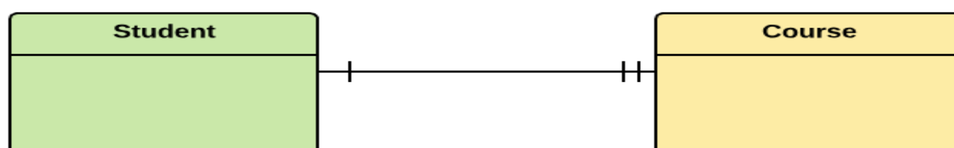
- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships
- Many-to-Many Relationships



1. One-to-one:

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

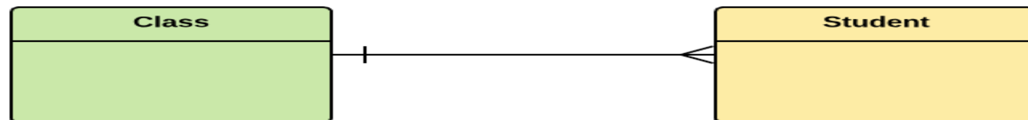
Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.



2. One-to-many:

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

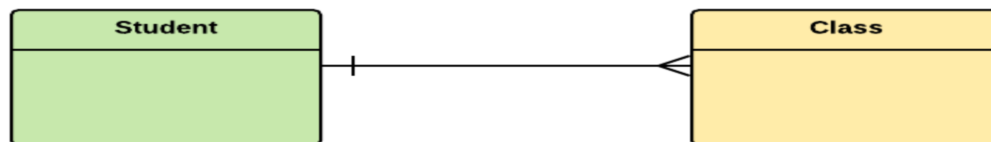
For example, one class is consisting of multiple students.



3. Many to One

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

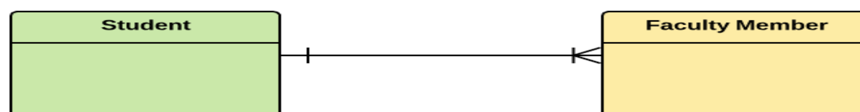
For example, many students belong to the same class.



4. Many to Many:

One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.



ER- Diagram Notations

ER- Diagram is a visual representation of data that describe how data is related to each other.

- **Rectangles:** This symbol represent entity types
- **Ellipses :** Symbol represent attributes

- **Diamonds:** This symbol represents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes

Steps to Create an ERD

Following are the steps to create an ERD.

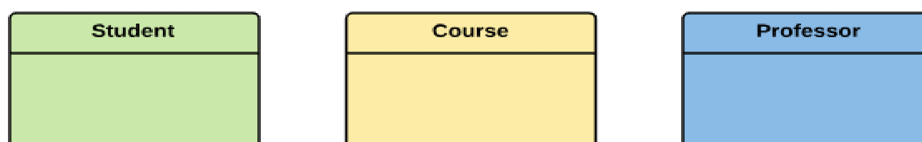
Let's study them with an example:

In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

Step 1) Entity Identification

We have three entities

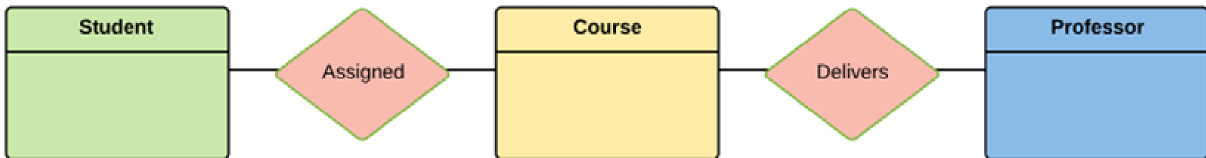
- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

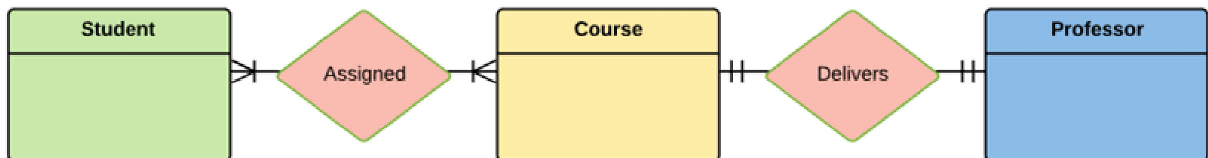
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



Step 4) Identify Attributes

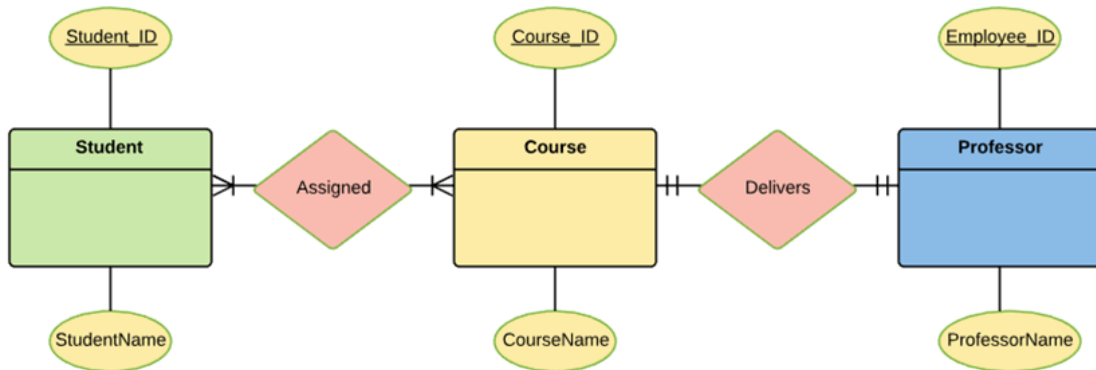
You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName

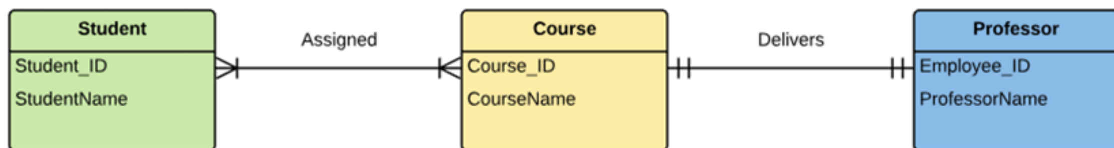
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD

A more modern representation of ERD Diagram



Best Practices for Developing Effective ER Diagrams

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other
- You should use colors to highlight important portions of the ER diagram

Summary

- The ER model is a high-level data model diagram
- ER diagrams are a visual tool which is helpful to represent the ER model
- Entity relationship diagram displays the relationships of entity set stored in a database
- ER diagrams help you to define terms related to entity relationship modeling
- ER model is based on three basic concepts: Entities, Attributes & Relationships
- An entity can be place, person, object, event or a concept, which stores data in the database
- Relationship is nothing but an association among two or more entities
- A weak entity is a type of entity which doesn't have its key attribute
- It is a single-valued property of either an entity-type or a relationship-type
- It helps you to defines the numerical attributes of the relationship between two entities or entity sets
- ER- Diagram is a visual representation of data that describe how data is related to each other
- While Drawing ER diagram you need to make sure all your entities and relationships are properly labeled.

Unit – V

Relational Database Design

Relational database was proposed by Edgar Codd (of IBM Research) around 1969. It has since become the dominant database model for commercial applications (in comparison with other database models such as hierarchical, network and object models). Today, there are many commercial *Relational Database Management System* (RDBMS), such as Oracle, IBM DB2 and Microsoft SQL Server. There are also many free and open-source RDBMS, such as MySQL, mSQL (mini-SQL) and the embedded JavaDB (Apache Derby).

A relational database organizes data in *tables* (or *relations*). A table is made up of rows and columns. A row is also called a *record* (or *tuple*). A column is also called a *field* (or *attribute*). A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data.

A language called SQL (Structured Query Language) was developed to work with relational databases.

Database Design Objective

A well-designed database shall:

- Eliminate Data Redundancy: the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- Ensure Data Integrity and Accuracy:
- [TODO] more

Database Design Process

Database design is more art than science, as you have to make many decisions. Databases are usually customized to suit a particular application. No two customized applications are alike, and hence, no two database are alike. Guidelines (usually in terms of what not to do instead of what to do) are provided in making these design decision, but the choices ultimately rest on the you - the designer.

Step 1: Define the Purpose of the Database (Requirement Analysis)

Gather the requirements and define the objective of your database, e.g. ...

Drafting out the sample input forms, queries and reports, often helps.

Step 2: Gather Data, Organize in tables and Specify the Primary Keys

Once you have decided on the purpose of the database, gather the data that are needed to be stored in the database. Divide the data into subject-based tables. Choose one column (or a few columns) as the so-called *primary key*, which uniquely identify the each of the rows.

Primary Key

In the relational model, a table cannot contain duplicate rows, because that would create ambiguities in retrieval. To ensure uniqueness, each table should have a column (or a set of columns), called *primary key*, that uniquely identifies every records of the table. For example, an unique number `customerID` can be used as the primary key for the `Customers` table; `productCode` for `Products` table; `isbn` for `Books` table. A primary key is called a *simple key* if it is a single column; it is called a *composite key* if it is made up of several columns.

Most RDBMSs build an index on the primary key to facilitate fast search and retrieval.

The primary key is also used to reference other tables (to be elaborated later).

You have to decide which column(s) is to be used for primary key. The decision may not be straight forward but the primary key shall have these properties:

- The values of primary key shall be unique (i.e., no duplicate value). For example, customerName may not be appropriate to be used as the primary key for the Customers table, as there could be two customers with the same name.
- The primary key shall always have a value. In other words, it shall not contain NULL.

Consider the followings in choose the primary key:

- The primary key shall be simple and familiar, e.g., employeeID for employees table and isbn for books table.
- The value of the primary key should not change. Primary key is used to reference other tables. If you change its value, you have to change all its references; otherwise, the references will be lost. For example, phoneNumber may not be appropriate to be used as primary key for table Customers, because it might change.
- Primary key often uses integer (or number) type. But it could also be other types, such as texts. However, it is best to use numeric column as primary key for efficiency.
- Primary key could take an arbitrary number. Most RDBMSs support so-called *auto-increment* (or AutoNumber type) for integer primary key, where (current maximum value + 1) is assigned to the new record. This arbitrary number is *fact-less*, as it contains no factual information. Unlike factual information such as phone number, fact-less number is ideal for primary key, as it does not change.
- Primary key is usually a single column (e.g., customerID or productCode). But it could also make up of several columns. You should use as few columns as possible.

Let's illustrate with an example: a table customers contains columns lastName, firstName, phoneNumber, address, city, state, zipCode. The candidates for primary key are name=(lastName, firstName), phoneNumber, Address1=(address, city, state), Address1=(address, zipCode). Name may not be unique. Phone number and address may change. Hence, it is better to create a fact-less auto-increment number, say customerID, as the primary key.

Step 3: Create Relationships among Tables

A database consisting of independent and unrelated tables serves little purpose (you may consider to use a spreadsheet instead). The power of relational database lies in the relationship that can be defined between tables. The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

1. one-to-many
2. many-to-many
3. one-to-one

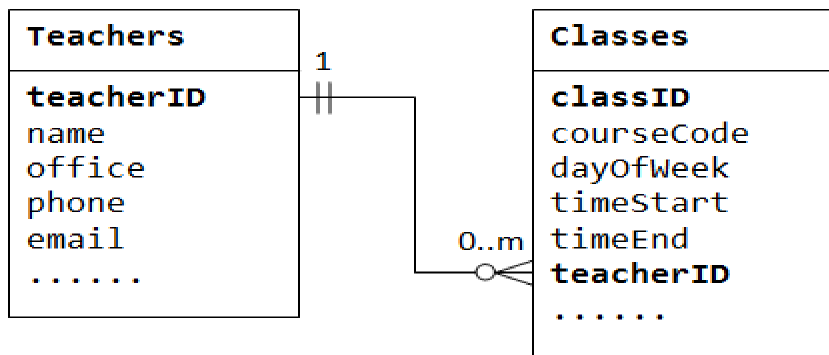
One-to-Many

In a "class roster" database, a teacher may teach zero or more classes, while a class is taught by one (and only one) teacher. In a "company" database, a manager manages zero or more employees, while an employee is managed by one (and only one) manager. In a "product sales" database, a

customer may place many orders; while an order is placed by one particular customer. This kind of relationship is known as *one-to-many*.

One-to-many relationship cannot be represented in a single table. For example, in a "class roster" database, we may begin with a table called `Teachers`, which stores information about teachers (such as `name`, `office`, `phone` and `email`). To store the classes taught by each teacher, we could create columns `class1`, `class2`, `class3`, but faces a problem immediately on how many columns to create. On the other hand, if we begin with a table called `Classes`, which stores information about a class (`courseCode`, `dayOfWeek`, `timeStart` and `timeEnd`); we could create additional columns to store information about the (one) teacher (such as `name`, `office`, `phone` and `email`). However, since a teacher may teach many classes, its data would be duplicated in many rows in table `Classes`.

To support a one-to-many relationship, we need to design two tables: a table `Classes` to store information about the classes with `classID` as the primary key; and a table `Teachers` to store information about teachers with `teacherID` as the primary key. We can then create the one-to-many relationship by storing the primary key of the table `Teacher` (i.e., `teacherID`) (the "one"-end or the *parent table*) in the table `classes` (the "many"-end or the *child table*), as illustrated below.



The column `teacherID` in the child table `Classes` is known as the *foreign key*. A foreign key of a child table is a primary key of a parent table, used to reference the parent table.

Take note that for every value in the parent table, there could be zero, one, or more rows in the child table. For every value in the child table, there is one and only one row in the parent table.

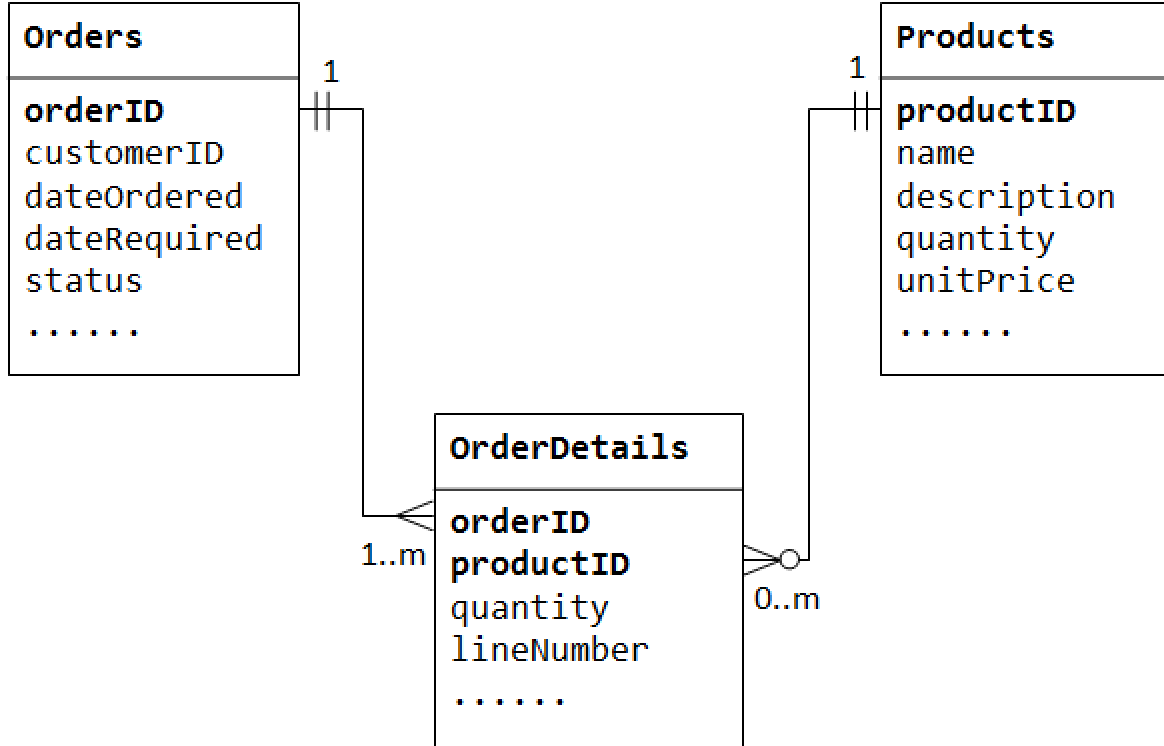
Many-to-Many

In a "product sales" database, a customer's order may contain one or more products; and a product can appear in many orders. In a "bookstore" database, a book is written by one or more authors; while an author may write zero or more books. This kind of relationship is known as *many-to-many*.

Let's illustrate with a "product sales" database. We begin with two tables: `Products` and `Orders`. The table `products` contains information about the products (such as `name`, `description` and `quantityInStock`) with `productID` as its primary key. The table `orders` contains customer's orders (`customerID`, `dateOrdered`, `dateRequired` and `status`). Again, we cannot store the items ordered inside the `Orders` table, as we do not know how many columns to reserve for the items. We also cannot store the order information in the `Products` table.

To support many-to-many relationship, we need to create a third table (known as a *junction table*), say `OrderDetails` (or `OrderLines`), where each row represents an item of a particular order. For the `OrderDetails` table, the primary key consists of two columns: `orderID` and `productID`, that uniquely identify each row. The columns `orderID` and `productID` in `OrderDetails` table are used

to reference Orders and Products tables, hence, they are also the foreign keys in the OrderDetails table.



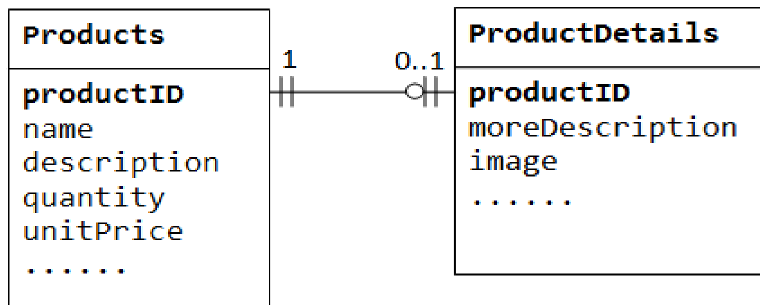
The many-to-many relationship is, in fact, implemented as two one-to-many relationships, with the introduction of the junction table.

1. An order has many items in OrderDetails. An OrderDetails item belongs to one particular order.
2. A product may appears in many OrderDetails. Each OrderDetails item specified one product.

One-to-One

In a "product sales" database, a product may have optional supplementary information such as image, moreDescription and comment. Keeping them inside the Products table results in many empty spaces (in those records without these optional data). Furthermore, these large data may degrade the performance of the database.
Instead, we can create another table (say ProductDetails, ProductLines or ProductExtras) to store the optional data. A record will only be created for those products with optional data. The two tables, Products and ProductDetails, exhibit a *one-to-one relationship*. That is, for every row in the parent table, there is at most one row (possibly zero) in the child table. The same column productID should be used as the primary key for both tables.

Some databases limit the number of columns that can be created inside a table. You could use a one-to-one relationship to split the data into two tables. One-to-one relationship is also useful for storing certain sensitive data in a secure table, while the non-sensitive ones in the main table.



Column Data Types

You need to choose an appropriate data type for each column. Commonly data types include: integers, floating-point numbers, string (or text), date/time, binary, collection (such as enumeration and set).

Normalization

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

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

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

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

Atomic Domains and First Normal Form

First normal form (**1NF**) is a property of a relation in a relational database. A relation is in first normal form if and only if the domain of each attribute contains only atomic (indivisible) values, and the value of each attribute contains only a single value from that domain.[1] The first definition of the term, in a 1971 conference paper by Edgar Codd, defined a relation to be in first normal form when none of its domains have any sets as elements.

First normal form is an essential property of a relation in a relational database. Database normalization is the process of representing a database in terms of relations in standard normal forms, where first normal is a minimal requirement.

First normal form enforces these criteria:[*citation needed*]

- Eliminate repeating groups[*clarification needed*] in individual tables
- Create a separate table for each set of related data[*definition needed*]
- Identify each set of related data with a primary key

Atomicity

Edgar F. Codd's definition of 1NF makes reference to the concept of 'atomicity'. Codd states that the "values in the domains on which each relation is defined are required to be atomic with respect to the DBMS." Codd defines an atomic value as one that "cannot be decomposed into smaller pieces by the DBMS (excluding certain special functions)" meaning a column should not be divided into parts with more than one kind of data in it such that what one part means to the DBMS depends on another part of the same column.

Rules for First Normal Form

The first normal form expects you to follow a few simple rules while designing your database, and they are:

Rule 1: Single Valued Attributes

Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

Rule 2: Attribute Domain should not change

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

For example: If you have a column `dob` to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

Rule 3: Unique name for Attributes/Columns

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.

If one or more columns have same name, then the DBMS system will be left confused.

Rule 4: Order doesn't matters

This rule says that the order in which you store the data in your table doesn't matter.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It

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

creates a table that looks like this:

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

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

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

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

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

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

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

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}

Non prime attribute: teacher_age

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

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

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

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

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

Third Normal form (3NF)

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

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

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

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

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

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

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

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

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

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

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

employee table:

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

employee_zip table:

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

Boyce Codd normal form (BCNF)

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

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

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

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

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

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

emp_id	**emp_nationality**
1001	Austrian
1002	American

emp_dept table:

emp_dept	**dept_type**	**dept_no_of_emp**
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

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

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

- Trivial functional dependency
- non-trivial functional dependency
- Multivalued dependency
- Transitive dependency

Trivial functional dependency

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

Symbolically: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

For example: Consider a table with two columns Student_id and Student_Name.

{Student_Id, Student_Name} -> Student_Id is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

Also, Student_Id -> Student_Id & Student_Name -> Student_Name are trivial dependencies too.

Non trivial functional dependency

If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:

An employee table with three attributes: emp_id, emp_name, emp_address.

The following functional dependencies are non-trivial:

emp_id -> emp_name (emp_name is not a subset of emp_id)

emp_id -> emp_address (emp_address is not a subset of emp_id)

On the other hand, the following dependencies are trivial:
{emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
Refer: trivial functional dependency.

Completely non trivial FD:

If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

Multivalued dependency

Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.

For example: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

bike_model	manuf_year	color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

Here columns manuf_year and color are independent of each other and dependent on bike_model. In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:

bike_model ->> manuf_year

bike_model ->> color

Transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.

X -> Z is a transitive dependency if the following three functional dependencies hold true:

- X->Y
- Y does not ->X
- Y->Z

Note: A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Example: Let’s take an example to understand it better:

Book	Author	Author_age
Game of Thrones	George R. R. Martin	66

Harry Potter	J. K. Rowling	49
Dying of the Light	George R. R. Martin	66

{Book} ->{Author} (if we know the book, we knows the author name)

{Author} does not ->{Book}

{Author} -> {Author_age}

Therefore as per the rule of **transitive dependency**: {Book} -> {Author_age} should hold, that makes sense because if we know the book name we can know the author's age.