Unit-1

Introduction to database

Topics

1) Database System Applications
2) Purpose of Database Systems
3) View of Data
4) Database Languages
5) Data Models
6) Database Users and Administrator

Database system application

1) Database :It is a collection of related data.
2) Data:It means a facts that can be recorded and have implicit meaning Example:Name,telephone numbers,address that can be stored in software like Msword,excel etc.
3) Database-management system(DBMS):It is a collection of interrelated data and set of programs to access those data.It's primary goal is to provide a way to store and retrieve database information in a convenient and efficient way.
4) DBMS must ensure the safety of the information stored,despite system crashes or attempts at unauthorised access.

Database System Applications

Database System Application

1.Enterprise Information

1) Sales:DBMS used to store customer,product and purchase information.
2) Accounting:DBMS used to store payments,receipts,account balances,assets and other accounting information.
3) Human Resources:stores information about employees,salaries,payroll taxes information for generation of pay checks.
4) Manufacturing:stores information of inventories of items in warehouse,stores,orders for items,informations for tracking production of items in factories,information for management of the supply chain etc.
5) Online retailers:stores information of order tracking,generation of recommendation lists and maintenance of online product evaluation.

2.Banking and Finance

Banking
1. Customer information
2. Accounts
3. Loans
4. Banking transaction

Credit card transaction
Information for:
1. Purchases on credit cards
2. Generation of monthly statements.

Finance
Information about:
1. Sales
2. Purchase of stock and bonds
3. Storing real-time market data for online trading
4. Holding etc.

3.Universities

1. Student information
2. Course registrations
3. Grades
4. Human Resources
5. Accounting

4.Airlines

For storing information of:

1. Reservations and schedule
2. Airlines were among the first to use database in a geographically distributed manner.

5.Telecommunications

1. For keeping records of call made
2. Generating monthly bill
3. Maintaining balances on prepaid calling cards
4. Storing information about the communication network.

Purpose of Database System

- Before DBMS were introduced ,to keep information on a computer **file-processing system** were used.
- In file processing system:

1. The system stores permanent records in various files.
2. It needs different application programs to extract records from and add records to,the appropriate files.

- It has number of major disadvantages:

1. Data redundancy and inconsistency
2. Difficulty in accessing data
3. Data isolation
4. Integrity problems
5. Atomicity problems
6. Concurrent-access anomalies
7. Security problems

1.Data redundancy and inconsistency
Data redundancy occurs when the same piece of data exists in multiple places(files).
Example:if a student has double major(Music and Maths),the students details(name,address,telephone number) may appear in both files of student records of students in Music and Maths department.
Data inconsistency occurs when various copies of same data may no longer agree.
Example:a changed student address may be reflected in the Music department record but not elsewhere in the system.

2.Difficulty in accessing a data
To carry out new task we need to write a new program.
Example:suppose university clerk needs to find out the names of students live within particular postal-code area.
But the designer of the original system did not anticipate this request.so there is no application program to meet it.so the clerk has two choices:
1.Either obtain the list of all students and extract needed information manually or
2.Ask a programmer to write necessary application program.
Hence it do not data to be retrieved in a convenient and efficient manner.

3.Data isolation
Because data are scattered in various files and in different formats.
Writing new application program to retrieve the appropriate data is difficult.

4.Integrity problems
The data value stored in database must satisfy certain type of consistency(integrity) constraints.
For example:bank balance should not go below 1000.Developers enforce these constraints by adding appropriate code in the various application programs.when new constraints are added it is difficult to change in all application programs for different files.

5.Atomicity problems
A computer system is subject to failure ,if failure occurs ,the data be restored to the consistent state that existed prior to the failure.
Example:if you want to transfer fund 1000 in account and if failure occurs,the fund transfer must be atomic-it must happen in its entirely or not at all.
It is difficult to ensure Atomicity in a file-processing system.

6.Concurrent- access anomalies
Simultaneous access of a data item should be handled carefully.
Example:if only one ticket is there and two customers are trying to book the ticket simultaneously the ticket should be allotted to any one customer.
It is difficult to handle in file-processing system due to the fact of data isolation ,redundancy etc.

7.Security problems
Not every user of the system should be able to access all the data.
Example:in an university a payroll personnel need to see only financial information not to academic records.But since application program are in a ad hoc manner ,enforcing such security constraints difficult.

View of Data

A main purpose of database system is to provide user abstract view of the data.

The developers hide internal irrelevant details from users and provides abstract view of data to users.This process of hiding irrelevant details from user is called Data Abstraction.

The developer hide the complexity from users through several levels of abstraction to simplify users interactions with the system.

The different levels of abstraction are:

1.Physical level

2.Logical level

3.View level

Levels of abstraction

1.Physical Level (Internal level)

This is the lowest level of data abstraction

It describes how data is actually stored in database.

You can get complex data structure details at this level.

2.Logical Level (Conceptual level)

This is middle level of data abstraction

It describes what data is stored in database and what relationships exist among those data.

Database administrators.who must decide what information to keep in the databases, use the logical level of abstraction.

3.View Level (External level)

It is the highest level of abstraction

It describes only part of the entire database i.e it describes the user interaction with database system via application programs that hide details of data types.

The system may provide many views for the same database.

External view	View 1 AC_Name Amount	View2 AC_No AC_Name Type
Logical/Conceptual view	AC_No AC_Name Type Amount	Numeric(15) Character(20) Character(10) Numeric(15)
Physical/Internal view	Stored-acc Account Type	Length=60 Type-bytes(15) Offset(10)

Instances and Schemas

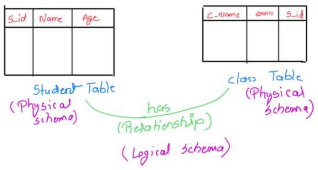
Instance of the database:The collection of information stored in the database at a particular moment is called an Instance of database.

Schema:Logical description of the database is known as schema of the database.

Schema are of two types:1.Physical Schema
2.Logical Schema

Physical Schema :It describes the database design at the physical level.
It is hidden beneath the logical schema and usually be changed without affecting the application program.

Logical Schema:It describes the database design at the logical level.
It is important as it effect on application program since programmer construct application by using logical schema.



A database may have several schema at the view level,sometimes called sub schemas.

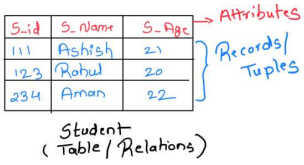
Physical data independence:If application program do not depend on the physical schema and thus need not to be rewritten if the physical schema changes.

Data Model

It is a collection of conceptual tools for describing data,data relationship ,data semantics and consistency constraints.
It helps to design a database at physical,logical and view levels.
It defines how data is connected to each other and how they are processed and stored inside the system.

Types of Data Model
1.Relational Model
2.Entity Relationship Model
3.Object based Data Model
4.Semi-structured Data Model
5.Others Model (Hirerachical Model,Network Model)

1.Relational Model
This model stores data in form of **table(Relations).**
Each table is a group of column and rows.
Column represents attributes of an **entity** (Example:student_id,name,age)
Row represents **record/tuples** .It defines a collection of attribute values.
Table-Relation
Tuple-rows
Attributes-column
Domain-set of permitted value
Degree-No. of columns in a relation



2.Entity Relationship Model

It describes the structure of a database with the help of a diagram,which is known as Entity -Relationship diagram.

It is a design or blueprint of a database that can later be implemented as a database.

It is based on the notion of real-world entities and relationship among them.

It is best used for conceptual design of a database.

It has following three components:

1.Entity:it is a real world entity like person ,place etc.

2.Relationship:the logical association between among entities.The relationship can be

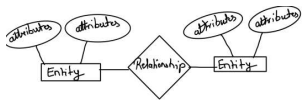
One to one

One to many

Many to one

Many to many

3.Attributes:are properties of entity like name,age ,address etc.



Advantages:

1.It makes the requirement simple and easily understandable by representing simple diagrams.

2.One can convert ER diagram into record based data model easily.

3.Easy to understand ER diagram

Disadvantage:

1.There is a great flexibility in the notation.It's all depends upon the designer ,how he draws it.

2.It is meant for high level designs.We cannot simplify for low level design like coding.

3.object based Data Model

They are of two types:

1.Object-oriented Data Model

2.Object-relational Data Model

1.Object-oriented Data Model

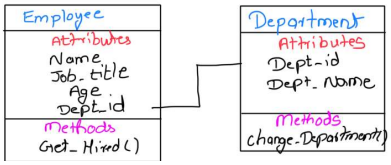
In this model data and their relationships are contained in a single structure which is referred as object .

In this ,real world problems are represented as objects with different attributes.

All objects have multiple relationships between them.

Two or more objects are connected through links.

Example:Here two objects are connected through a common attribute I.e Dept_id.



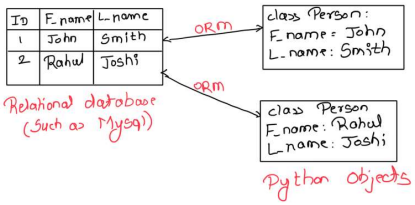
2.object -Relational Model

It is a combination of the object -oriented data model and relational data model.

It is created to handle complex data types such as audio ,video and image files that relational databases were not equipped to handle.

It uses object Relational Mapper a software or a framework which enables mapping of objects to relational tables and interaction between objects and tables.

Example:



Advantages:

1.Because of its inheritance property ,we can re-use the attributes and functionalities.

2.The information are encapsulated ,so there is no fear of being misused by other objects.

3.If we need any new feature we can easily add new class inherited from parent class and add new features.

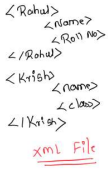
Disadvantages:

As it combines the features of both object-oriented data model and the Relational data model this data model can be rather complicated and hard to handle.

4.Semi-structured Data Model:

The semi-structured model is a database model where there is no separation between the data and the schema, and the amount of structure used depends on the purpose.

The XML/JSON is widely used to represent Semi -structured data.



Advantages:

1.It can represent the information of some data sources that cannot be constrained by schema.

2.It provides a flexible format for data exchange between different types of databases.

3.The schema can easily be changed.

Disadvantages:

1. Due to lack of a well-defined structure, it can not used by computer programs easilly
2. Interpreting the relationship between data is difficult as there is no separation of the schema and the data.

5.Others Model

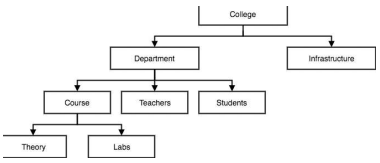
Some others Model are :
1.Hierarchical Data Model
2.Network Data Model

1 .Hierarchical Data Model

Here ,data is organised into a tree like structure with each record is having one parent record and many children record.

In hierarchical model, data have one to one or one-to-many relationship between two different types of data.

Pointers are used for linking records that tell which isa parent and which child record is.



Advantages:

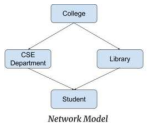
1.It is easy to understand.
2.Parent/child relationship promotes conceptual simplicity and data integrity.
3.It offers data security and this model was the first database model that offered data security.
4.There's also data integrity as it is based on the parent-child relationship

Disadvantages:

1.If a parent is deleted then the child automatically gets deleted.
2.Changes in structure require changes in all application programs
3.Implementation limitations
4.No data definition
5.Lack of standards

2.Network Data Model

It is an extension of the hierarchical model.
It is same as hierarchical model except that it has graph-like structure rather than a tree-based structure and are allowed to have more than one parent node.



Network Model

Advantages:

1.Because it has the many-many relationship, network database model can easily be accessed in any table record in the database
2.For more complex data, it is easier to use because of the multiple relationship founded among its data
3.Easier to navigate and search for information because of its flexibility

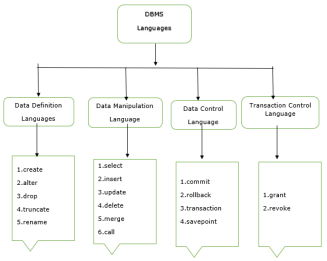
Disadvantages:

1.The network model is a very complex database model, so the user must be very familiar with the overall structure of the database.
2.Updating the database is a quite difficult and boring task. We need the help of the application programs that are being used to navigate the data.
3.Absence of structural independence.

Database Language

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

Types of Database Language



1. Data Definition Language

It is a language that allows the user to define the data and there relationship to other types of data.

It is used to create schema, tables, indexes, constraints, etc. in the database.

It is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc

DDL commands:

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

It is mainly used to create file ,database ,data dictionary and table within database.

2. Data Manipulation Language

DML stands for Data Manipulation Language.

It is used for accessing and manipulating data in a database.

It handles user requests.

DML Commands:

Select: It is used to retrieve data from a database.
Insert: It is used to insert data into a table.
Update: It is used to update existing data within a table.
Delete: It is used to delete all records from a table.
Merge: It performs UPSERT operation, i.e., insert or update operations.

3. Data control Language

DCL is used to access the stored data.

It is mainly used for revoke and grant the user access to a database.

The DCL commands are: Grant, Revoke.

4. Transaction Control Language (TCL)

TCL is a language which manages the transactions within the database.

It is used to execute the changes made by the data manipulation language statements.

The TCL commands are:

Commit, Rollback.

Database users and Administrator

People who work with a database can be categorised as database users or database administrators.

Database users are the ones who really use and take the benefits of the database.

There will be different types of users depending on their needs and way of accessing the database:

1. Application Programmers
2. Sophisticated Users
3. Specialized Users
4. Stand-alone Users
5. Native Users

1. Application Programmers – They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal, etc.

2. Sophisticated Users – They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database.

They directly interact with the database by means of a query language like SQL.

These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirements.

3. Specialized Users – These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.

4. Stand-alone Users – These users will have a stand-alone database for their personal use.

5. Native Users – these are the users who use the existing application to interact with the database.

For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

Database Administrator

It is a person or group incharge for implementing DBMS in an organisation.

The DBA job requires high degree of technical expertise .

DBA consist of a team of people rather than just one person.

Responsibilities of DBA:

1.DBA creates the original database schema by executing the set of DDL commands.

2.It plans storage structure and access strategy.

3.It carries out changes to the schema and physical organisation of data to improve performance.

4.DBA granting different types of authorisation for data access.

5.Provides support to users

6.It periodically backing up the database.

7.Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.

8.Monitoring the performance and responding to the changes in the requirement.

Unit -02

Database Design and E-R Model

Topics

1. Overview of the Design Process
2. The Entity-Relationship Model
3. Constraints
4. Entity Relationship Diagrams
5. Reduction to Relational Schemas
6. Schema Diagrams
7. Entity-Relationship Design Issues
8. Extended ER features.

Overview of Design Process

Design of the Database system involves:

1.Design of the Database Schema

2.design of the programs that access and update the data

3.design of a security scheme for controlling access to the data

The central role in the designing process involves the requirements of the user.

For designing a database, there are four phases, namely:

1.Initial Phase

2.Conceptual Design Phase

3.Logical Design Phase

4.Physical Design Phase

1.Initial Phase :

The Initial Phase of database design is to fully describe the data needs of the potential database users.

2.Conceptual Design Phase :

The designer chooses a **data model**, by applying the concepts of the data model that is chosen and translates these requirements into a conceptual schema of the database.

The **E-R model** is typically used to represent the conceptual design.

In this phase **focus is on describing the data and their relationship.**

A fully developed **conceptual schema indicates the functional needs of the user.**

Here, the users describe various operations that have to be performed on the data.

The operations can include updating, searching, retrieving and deleting data.

3.Logical Design Phase : The database designer plots the high-level conceptual schema onto the implementation of the data model of the database system that has to be used.

The implementation data model is also called the relational data model.

4.Physical Design Phase : Here, the designer uses the resulting system-specific database schema. In this phase, the physical features of the database are specified.

The following two issues are to be avoided In designing a database schema,

1.Redundancy

2.Incompleteness

The Entity-Relationship Model

To facilitate Database Design by allowing specification of an Enterprise Schema that represents the overall logical structure of a database is called as Entity-Relationship Model.

It is very useful in mapping the meaning and interactions of real-world enterprise onto a conceptual schema.

The E-R data model employs three basic concepts:

1.Entity sets

2.relationship sets

3.attributes

The E-R model also has an associated diagrammatic representation,the E-R diagram.

1.Entity set

Entity:An entity is a thing in a real-world with independent existence. An entity can exist independently and is distinguishable from other objects. It can be identified uniquely.

An entity can be of two types :

1.Tangible Entity : Entities that exist in the real world physically. Example: Person, car, etc.

2.Intangible Entity : Entities that exist only logically and have no physical existence. Example: Bank Account, etc.

An entity is represented by a set of attributes.

In a particular relation in RDBMS, a particular record is called an entity.

Entity Type: It refers to the category that a particular entity belongs to.

It is represented by the name of the table and its schema.

Example:A table named student in a university database.

Entity set:

An entity set is a collection or set of all entities of a particular entity type at any point in time. The type of all the entities should be the same.

Example:The collection of all the students from the student table at a particular instant of time is an example of an entity set.

Example

Consider a table student as follows :

Table Name : Student

Student_id	Student_name	Student_age	Student_gender
1	Avi	19	M
2	Ayush	22	M
3	Riya	23	F

Entity:Each row is an entity.

Example:1 Avi 19 M

Entity Type:Student

Entity Set:the records with student id 1, 2, 3, 4 are the entity set.

2.Attributes

Attributes : Attributes are descriptive properties possessed by each member of an entity set.

A set of attributes represents an entity.

For each attribute there is a set of permitted values,called the domain or value set of that attribute.Ex:the domain of attribute course_id might be a number of certain length.

Each entity can be described by a set of (attribute,data value) pairs.Ex:instructor entity may be described by the set {(id,76766),(name,Alice) ,(dept_name,computer)}

The attribute which uniquely identifies each entity in the entity set is called **key attribute.**Example:roll_number will be unique for each student.

An attribute can be characterised by the following attribute types:

1.simple and composite attribute

2.single-valued and multivalued attribute

3.Derived attribute

1.simple and composite attribute:

An attribute which contains a single value is called single/simple attribute.Ex:age

An attribute composed of many other attribute is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country

2.Multivalued Attribute –

An attribute consisting more than one value for a given entity. For example, Phone_No (can be more than one for a given student).

Upper and lower bounds may be placed on the number of values in a multivalued attributes.Ex:a university may limit the number of phone numbers recorded for a single instructor to two.

3.Derived Attribute –

An attribute which can be derived from other attributes of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB).

The value of a derived attribute is not stored but is computed when required.

An attribute takes a null value when an entity does not have a value for it.

The null value may indicate:

1.not applicable (it means value does not exist for the entity for ,example:one may have no middle name)

2.attribute value is unknown (the value exist ,but we do not have that information ,example:An address attribute have apartment number but we do not know what it is?)

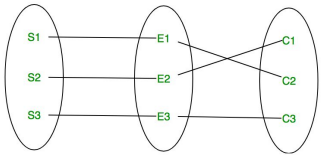
3.Relationship set:

A relationship is an association among several entities.

For example,'Enrolled in' is a relationship type that exists between entity type Student and Course.

A set of relationships of same type is known as relationship set.

The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



A relationship may also have an attributes called descriptive attributes.

Ex:consider a relationship "advisor" with entity sets "instructor" and "student".Attribute "date"can be associated with relationship to specify the date when an instructor became the advisor of a student.If date value is 11 July 2022,it means instructor became student's advisor from 11 July 2022.

Degree of a relationship set:

The number of different entity sets participating in a relationship set is called as degree of a relationship set.

When there are TWO entities set participating in a relation, the relationship is called as **binary relationship**.For example, Student is enrolled in Course.

n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

Recursive Relationship

When there is a relationship between two entities of the same type, it is known as a recursive relationship. This means that the relationship is between different instances of the same entity type.

Example:A student can be a class monitor and handle other students. Hence, this is a recursive relationship of entity student with itself. This is a 1 to many recursive relationship as one student is monitor of many students.

Constraints

Constraints are used for **modeling limitations on the relations** between entities.

There are two types of constraints on the Entity Relationship (ER) model −

1.Mapping cardinality or cardinality ratio.

2.Participation constraints.

3.Relational constraints

1.Mapping Cardinality

It is expressed as the number of entities to which another entity can be associated via a relationship set.

For the binary relationship set there are entity set A and B then the mapping cardinality can be one of the following −

One-to-one

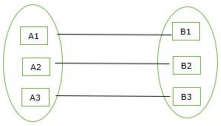
One-to-many

Many-to-one

Many-to-many

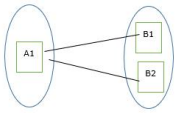
1.One-to-one relationship

An entity set A is associated with at most one entity in B and an entity in B is associated with at most one entity in A.



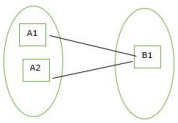
2.One-to-many relationship

An entity set A is associated with any number (zero or more) of entities in B with a possibility of zero and an entity in B is associated with at most one entity in A.



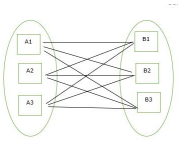
3.Many-to-one relationship

An entity set A is associated with at most one entity in B and an entity set in B can be associated with any number (zero or more) of entities in A.



4.Many-to-many relationship

An entity set A is associated with any number of entities in B with a possibility of zero and an entity in B is associated with any number of entities in A with a possibility of zero.



2.Participation constraint

The participation constraint specifies the number of instances of an entity can participate in a relationship set.

Participate constraints are two types as mentioned below −

1.Total participation

2.Partial Participation

1.Total participation

The participation of an entity set E in a relationship set R is **said to be total if every entity in E Participates in at least one relationship in R.**

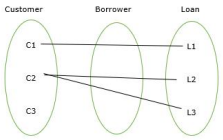
For Example − Participation of loan in the relationship borrower is total participation.

2.Partial Participation

Partial Participation

If only some of the entities in E participate in relationship R, then the participation of E in R is said to be partial participation.

For example − Participation of customers in the relationship borrower is partial participation.

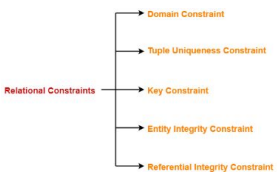


Here, the customer to Loan is partial participation and the loan to the customer is total participation.

3.Relational Constraints:

Relational constraints are the restrictions imposed on the database contents and operations.

They ensure the correctness of data in the database.



1. Domain constraints :

Every domain must contain atomic values(smallest indivisible units) it means composite and multi-valued attributes are not allowed.

We perform datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we cant give it values other then int datatype.

STU_ID	Name	Age
S001	Ram	20
S002	Sam	30
S003	Riya	A

Here, value 'A' is not allowed since only integer values can be taken by the age attribute.

2. Tuple Uniqueness Constraints :

it ensures that every tuple in the relation should be unique.

Stu_id	Name	Age
S001	Ram	20
S001	Ram	20
S002	Riya	19

It does not satisfy the tuple uniqueness constraint since here all the tuples are not unique

3. Key Constraint-

Key constraint specifies that in any relation-

1.All the values of primary key must be unique.

2,The value of primary key must not be null.

Stu_id	Name	Age
S001	Ram	20
S001	Sam	21
S002	Riya	19

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

4. Entity Integrity Constraint-

Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.

This is because the presence of null value in the primary key violates the uniqueness property.

Stu_id	Name	Age
S001	Ram	20
S001	Sam	21
	Riya	19

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

5. Referential Integrity Constraint-

This constraint is enforced when a foreign key references the primary key of a relation.

It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

Example-

Consider the following two relations- 'Student' and 'Department'.Here, relation 'Student' references the relation 'Department'.

Stu_id	Name	Dept_no
S001	Ram	D10
S002	Sam	D10
S003	Riya	D14

Name	Dept_no
Comp	D10
AI&DS	D10
MBATech	D12

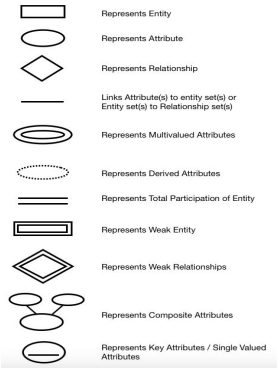
The relation 'Student' does not satisfy the referential integrity constraint.
This is because in relation 'Department', no value of primary key specifies department no. 14.

Entity Relationship Diagrams

An E-R diagram can express the overall logical structure of a database graphically.

E-R diagrams are used to represent E-R model in a database, which makes them easy to be converted into relations (tables).

It gives a standard solution of visualizing the data logically.



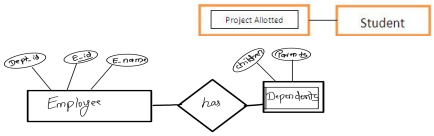
1. Entity:- Any real-world object can be represented as an entity about which data can be stored in a database. All the real world objects like a book, an organization, a product, a car, a person are the examples of an entity. Any living or non-living objects can be represented by an entity.

Entities can be characterized into two types:

Strong entity: A strong entity has a primary key attribute which uniquely identifies each entity. Symbol of strong entity is same as an entity.

Weak entity: A weak entity does not have a primary key attribute and depends on other entity via a foreign key attribute.And have no meaning in the diagram without their parent entity.

Example:



Total Participation:

It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set.

That is why, it is also called as mandatory participation.

Total participation is represented using a double line between the entity set and relationship set.

Here,

Double line between the entity set "Student" and relationship set "Enrolled in" signifies total participation.

It specifies that each student must be enrolled in at least one course.

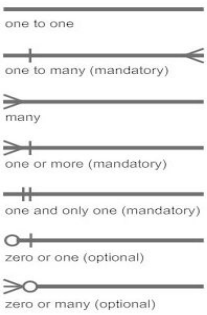
Connectivity of a relationship:- Connectivity of a relationship describes, how many instances of one entity type are linked to how many instances of another entity type. Various categories of connectivity of a relationship are;

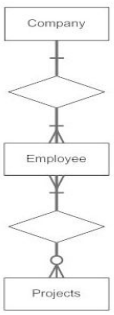
1.One to One (1:1) – "Student allotted a project" signifies a one-to-one relationship because only one instance of an entity is related with exactly one instance of another entity type.

2.One to Many (1:M) – "A department recruits faculty" is a one-to-many relationship because a department can recruit more than one faculty, but a faculty member is related to only one department.

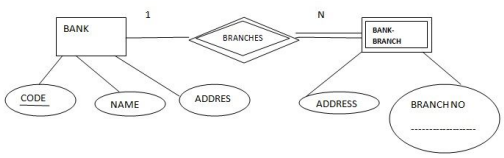
3.Many to One (M:1) – "Many houses are owned by a person" is a many-to-one relationship because a person can own many houses but a particular house is owned only a person.

4.Many to Many (M:N) – "Author writes books" is a many-to-many relationship because an author can write many books and a book can be written by many authors.





Which is the correct statement regarding the above diagram



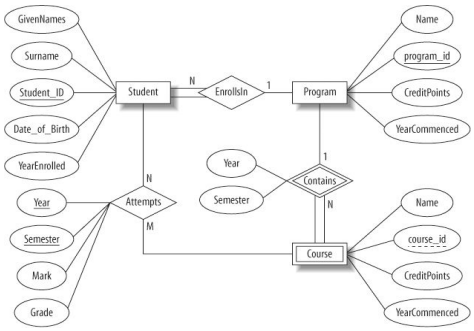
(a) Bank-Branch is the weak entity

(b) Branch No of the Bank-Branch entity is a partial Key

(c) The Participation of BANK in BRANCHES Relationship is partial

d) All of the above

University database design



Example:University Database design

Student is a strong entity, with an identifier, student_id, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).

Program is a strong entity, with the identifier program_id as the primary key used to distinguish between programs.

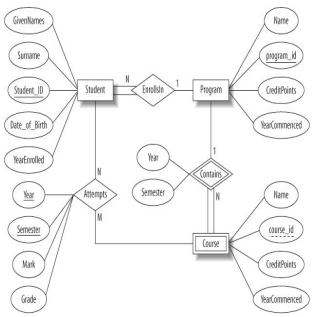
Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one EnrollsIn relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.

A Course has meaning only in the context of a Program, so it's a weak entity, with course_id as a weak key. This means that a Course is uniquely identified using its course_id and the program_id of its owning program.

As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.

Student and Course are related through the many-to-many Attempts relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.

When a student attempts a course, there are attributes to capture the Year and Semester, and the Mark and Grade.

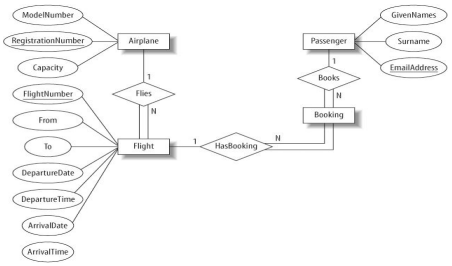


Draw an E-R diagram based on following requirement:

The flight database stores details about an airline's fleet, flights, and seat bookings.

Consider the following requirements list:

- The airline has one or more airplanes.
- An airplane has a model number, a unique registration number, and the capacity to take one or more passengers.
- An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time.
- Each flight is carried out by a single airplane.
- A passenger has given names, a surname, and a unique email address.
- A passenger can book a seat on a flight.



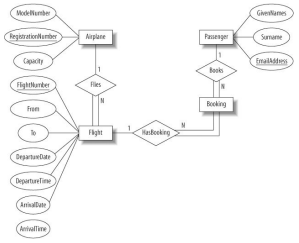
An Airplane is uniquely identified by its RegistrationNumber, so we use this as the primary key.

A Flight is uniquely identified by its FlightNumber, so we use the flight number as the primary key. The departure and destination airports are captured in the From and To attributes, and we have separate attributes for the departure and arrival date and time.

Because no two passengers will share an email address, we can use the EmailAddress as the primary key for the Passenger entity.

An airplane can be involved in any number of flights, while each flight uses exactly one airplane, so the Flies relationship between the Airplane and Flight relationships has cardinality 1:N; because a flight cannot exist without an airplane, the Flight entity participates totally in this relationship.

A passenger can book any number of flights, while a flight can be booked by any number of passengers. As discussed earlier in Intermediate Entities," we could specify an M:N Books relationship between the Passenger and Flight relationship, but considering the issue more carefully shows that there is a hidden entity here: the booking itself. We capture this by creating the intermediate entity Booking and 1:N relationships between it and the Passenger and Flight entities. Identifying such entities allows us to get a better picture of the requirements. Note that even if we didn't notice this hidden entity, it would come out as part of the ER-to-tables mapping process we'll describe next in Using the Entity Relationship Model."



Reduction to Relational Schema

We can represent a database that conforms to an E-R database schema by a collection of relational schema.

For each entity set and relationship set ,there is a unique relation schema to which we assign the name of the corresponding entity set or relationship set.

Reduction to relational schema defines:

1.how an E-R schema can be represented by relation schema

2.how constraints arising from the E-R design can be mapped to constraints on relation schemas.

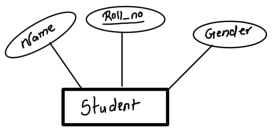
Reduction to Relational Schema

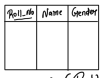
Rule 1:Strong Entity set with simple attributes

1.A strong entity set with only simple attributes will require only one table in relational model.

2. the attributes of the entity set will be the Attributes of the table.

3. the key attribute of the entity set will be the primary key of the table.





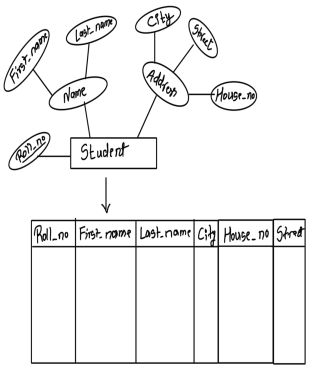
Schema: Student(Roll_no, Name, Gender)

Rule 2:Strong Entity set with composite Attribute

1.A strong entity with any number of composite attribute will require only one table in relational model.

2.While conversion ,simple attributes of the composite attributes are taken into account and not the composite attribute itself.

Schema:Student(Roll_no,First_name ,Last_name,city,House_no,Street)



Rule 3:Strong Entity set with multiple-valued Attributes

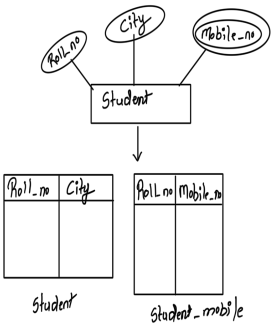
1.A strong entity set with any number of multi-valued attributes will require two tables in relational model.

2.One table will contain all the simple attributes with the primary key.

3.Other table will contain the primary key and all multi-valued attributes.

Schema:Student(Roll_no,City)

Student_mobile(Roll_no,Mobile_no)



Rule 4:Translating Relationship Set into a Table.

A relationship set will require one table in the relational model.

Attributes of the participating relation table are:

1.primary key attributes of the participating entity sets are declared as foreign keys to the respective relations.

2.it's own descriptive attributes if any.

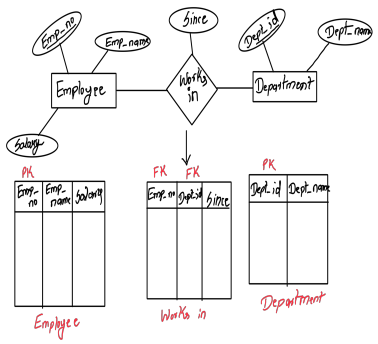
Set of non-descriptive attribute will be the primary key.

For given E-R diagram three tables will be required in relational model:

1.one table for the Entity set "Employee".

2.one table for the Entity set "**Department**".

3.one table for the relationship set "**works in**"



Schema: works_in(Emp_no,Dept_id,Since)

Rule 5:For Binary Relationships with Cardinality Ratios.

The following are the four possible cases:

1.Case-1-Binary relationship with cardinality ratio m:n

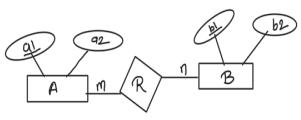
2.Case-2:Binary relationship with cardinality ratio 1:n

3.Case-3:Binary relationship with cardinality ratio m:1

4.Case-4:Binary relationship with cardinality ratio 1:1

Case 1-For Binary relationship with Cardinality Ratio m:n(Many-to-Many relationship

In this type of relationship three tables will be required.

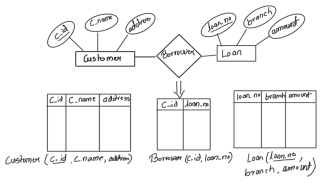


1.A(a1,a2)

2.R(a1,b1)

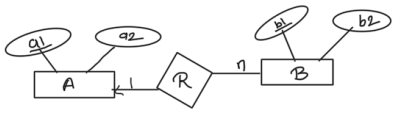
3.B(b1,b2)

Example



Case 2- For Binary Relationship with Cardinality Ratio 1:n

In one-to -Many Relationship ,two tables will be required.

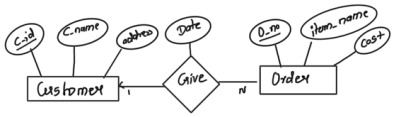


1.A(a1,a2)

2.BR(b1,b2,a1)

Here we combined table will be drawn for the entity set B and relationship set R.

Example:one customer give multiple orders



Date	C_id	O_no
6 aug	C1	O1
6 aug	C1	O2
6 aug	C2	O3

O_no	Item_ name	Cost
O1	Shoes	2500
O2	Shirt	1550
O3	Belt	500

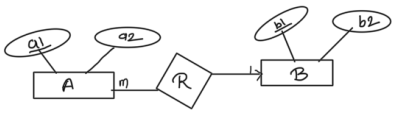
Reduce →

C_id	C_name	Address
C1	A	Delhi
C2	B	Mumbai
C3	C	NAVI mumbai

O_no	Item_ name	Cost	C_id	Date
01	Shoes	2500	C1	6 aug
02	Shirt	1550	C1	6 aug
03	Belt	500	C2	6 aug

Case 3-For Binary Relationship with Cardinality Ratio n:1

In Many-to-one Relationship ,two tables will be required.

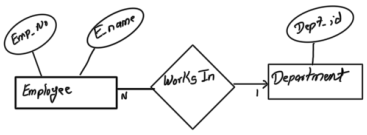


AR(a1, a2,b1)

B(b1,b2)

Here combined table will be drawn for the entity set A and relationship set R.

Example:Employee works in Department



Dept_id
1
2

Department

Emp_no	E_name
1	Rahul
2	Riya

Employee

Emp_no	Dept_id
1	1
2	1

worksIn

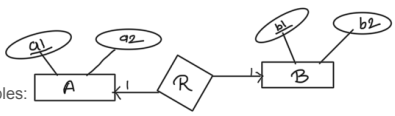
Reduce →

Emp_no	E_name	Dept_id
1	Rahul	1
2	Riya	1

Emp_works

Case 4-For Binary Relationship with Cardinality Ratio 1:1

In One-to-one Relationship ,two tables will be required.Either combine R with ‘A’ or ‘B’.



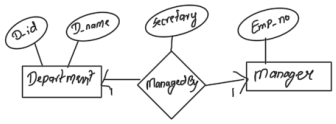
Tables:

1.AR(a1,a2,b1),B(b1,b2)

Or

2.A(a1,a2),BR(b1,b2,a1)

Example : Department managed by Manager



Here,secretary is assigned for a department that is managed by a manager.This attribute does not have existence without ‘ManagedBy’ relation.

D_id	D_name	Emp_no	secretary
1	Accounts	1	S1
2	Academic	2	S2

dep_man

Emp_no
1
2

Manager

Or

D_id	D_name
1	Accounts
2	Academic

Department

Emp_no	secretary	D_id
1	S1	1
2	S2	2

Emp_manag

Rule 6:For Binary relationship with both Cardinality constraints and participation constraints.

Cardinality constraints will be implemented as discussed in Rule-5.

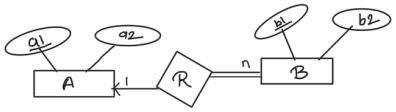
Because of the total participation constraint,foreign key acquires **NOT NULL** constraint.

Two cases:

Case-1:For Binary relationship with Cardinality constraint and total participation constraint from one side.

Case-2:For Binary relationship with Cardinality constraint and total participation constraint from both sides.

Case-1:For Binary relationship with Cardinality constraint and total participation constraint from one side.



Because Cardinality ratio =1:n,we will combine the entity set B and relationship set R.

Then,two table will be required:

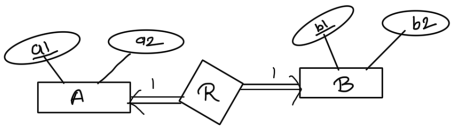
1.A(a1,a2)

2.BR(b1,b2,a1)

Because of total participation,foreign key a1 has acquired **NOT NULL** constraint,so it can not be NULL now.

Case-2:For Binary relationship with Cardinality constraint and total participation constraint from both sides.

If there is a key constraint from both the sides of an entity set with total participation,then that binary relationship is represented using only single table.

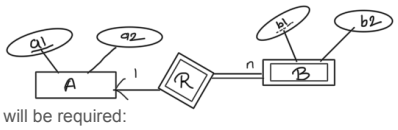


Here, only one table is required:

ARB(a1,a2,b1,b2)

Rule-7:For Binary Relationship with Weak Entity set.

Weak entity set always appears in association with identifying relationship with total participation constraint and there is always 1:n relationship from identifying entity set to weak entity set.

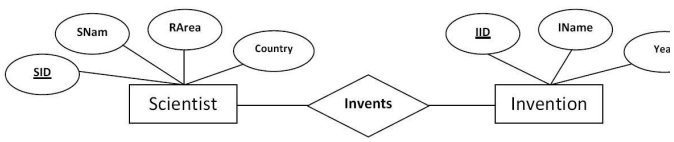


Here ,two tables will be required:

A(a1,a2)

BR(a1,b1,b2)

Convert/reduce the ER Diagram given in figure 1 below;

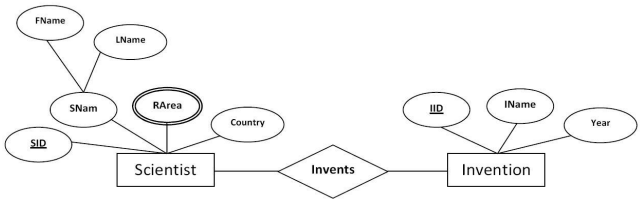


Final relation schemas of the given ER diagram are as follows;
Scientist (SID, SName, RArea, Country, IID)

Invention (IID, IName, Year)

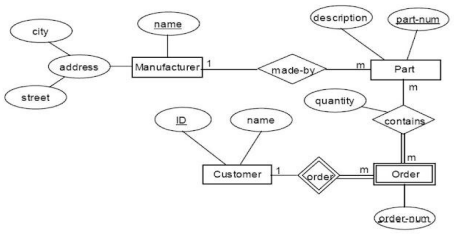
Invents (SID, IID)

Convert/reduce the ER Diagram given in figure 1 below;



Scientist (SID, FName, LName, Country)
Scientist_Area (SID, RArea)
Invention (IID, IName, Year)
Invents (SID, IID)

Convert/reduce the ER Diagram given in figure 1 below;



Manufacturer (name, street, city)
Part (part_num, description, name)
Customer (ID, Name)
Order (order_num, ID, part_num, Quantity)

E-R design issues

1.use of entity sets vs. attributes

2.use of entity sets vs. relationship sets

3.Binary vs. n-ary relationship sets

4.placement of relationship attribute.

Extended ER features

As the complexity of data increased in the late 1080's,it became more and more difficult to use the traditional E R model for database modelling.Hence some improvements or enhancement were made to the existing ER model to make it able to handle the complex application better.

Following new concept were added to the existing model:

1.Specialization

2.generalisation

3.Attribute inheritance

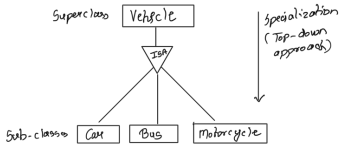
4.constraints on Generalization

5.Aggregation

1.Specialization

- Specialization is a process of identifying subsets of an entity that share some different characteristic.
- An entity is broken into sub-entities based on their characteristics.
- It is used to identify the subset of an entity set that shares some distinguish characteristics.
- It is a top-down approach,where high level entity is specialized into two or more lower level entities.
- It is depicted by triangle component labelled ISA.

Normally,the superclass is defined first,the subclass and it's related attributes are defined next,and relationship set are then added.

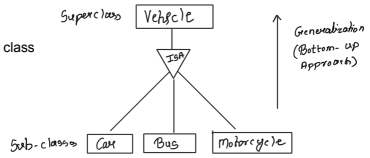


2.generalisation

It is the process of extracting common properties from a set of entities and creates a generalised entity from it.

It is bottom-up approach,in which two or more entities can be combined to form higher level entity if they have some attributes in common.

Here,three entities can be generalized into one super class named as Vehicle.



Reduction to Relational Schema

1.Representation of Generalization:

Create a schema for the higher-level entity set.

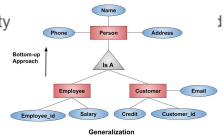
For each lower-level entity set,creates a schema that includes an attribute for each of the attributes of that entity set plus one for each attribute of the primary key of the higher-level entity set.

Example:entity Person contains only the common

attributes i.e. Name, Phone, and Address. Employee entity attribute like Employee_id and Salary.

the Customer entity type contains only specialized

attributes like Customer_id, Credit, and Email.



Constraints on Generalizations

There are three types of constraints:

1.Membership constraints

- 1.1 condition define
- 1.2 User defined

2.Disjointness constraints

- 2.1 disjoint
- 2.2 Overlapping

3.completeness constraints

- 3.1Total generalization or specialization
- 3.2 Partial generalization or specialization

1.Membership constraints

First one determines which entity can be a member of the low-level entity set.

Such kind of membership may be one of the following:

1.condition defined or attribute defined

2.user-defined

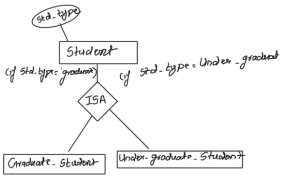
1.condition defined or attribute defined

In this lower-level entity sets, evaluation of membership is on basis whether an entity satisfies an explicit condition or not.

Example:All student entities are evaluated on the defining std_type attribute.

Only those entities that satisfy the condition std_type=graduate are allowed to belongs lower level entity Graduate_student.

Similarly,std_type =under_graduate are allowed to belongs to lower level entity under_graduate_student.



Example:condition defined

2.User- defined

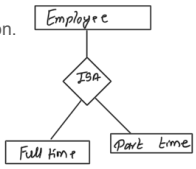
In this lower-level entity sets are not get constrained by a condition named membership; users of database assigns entities to a given entity set

Example:here ,employee is not assigned as full or

Part time on the basis of a explicit defining condition.

Instead the user in charge of this decision makes

Employee assign ass Full time or part time.



2.Disjointness constraints

It relates to whether or not entities belong to more than one lower-level entity set.

Following is one of the lower-level entity sets:

2.1Disjoint –

The requirement of this constraint is that an **entity should not belong to no more than one lower-level entity set.** For example, the entity of student entity satisfy only one condition for student type attribute i.e. Either an entity can be a graduate or an undergraduate student, but cannot be both at the same time.

2.2 Overlapping –

In this category of generalizations, within a single generalization, the **same entity may belong to more than one lower-level entity set**. For example, in the employee work-team assume that certain employees participate in more than one work team. Thus, it offers a given employee that he may appear in more than one of the team entity sets that are lower-level entity sets of employee. Thus, generalization is overlapping.

3.completeness constraints

It specifies whether or not an entity in the higher level entity set must belong to at least one of the lower level entity set within generalization

This constraint may be one of the following:

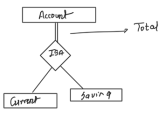
3.1Total generalization or specialization –

According to this constraint, each higher-level

entity must belong to a lower-level entity set.

Example:an account must be a saving account

Or current account.



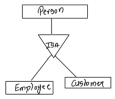
3.2 Partial generalization or specialization –

According to this constraint, some higher-level

entities may not belong to any lower-level entity set.

Example:consider person are just a visitors,so they do not belongs

to employee or customer



3.Attribute inheritance

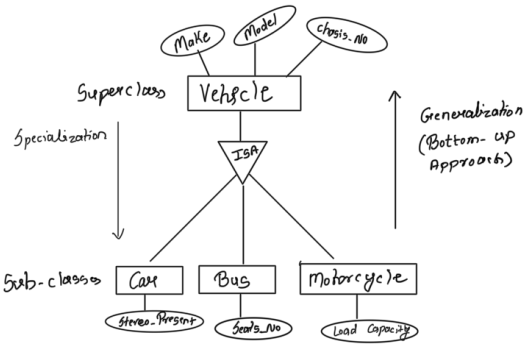
The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.

Example:consider relations Car and Bus inheriting attributes of vehicle.Thus,car is described by attributes of super-class vehicle as well as its own attributes.

This also extends to participation Inheritance in which relationships involving higher-level entity-sets are also inherited by lower-level entity sets.

This also extends to participation Inheritance in which relationships involving higher-level entity-sets are also inherited by lower-level entity sets.

A lower-level entity-set can participate in its own relationships-sets too.



Example:Attribute Inheritance

4.Aggregation

There is a one limitation with E-R model that it cannot express relationships among relationships. So aggregation is an abstraction through which relationship is treated as higher level entities.

It is a process when a relationship between two entities is considered as a single entity and again this single entity has a relationship with another entity.

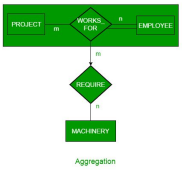
Example:

Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.

Representing aggregation via schema –

To represent aggregation, create a schema containing:

primary key of the aggregated relationship
primary key of the associated entity set
descriptive attribute, if exists.



Aggregation

Unit-03

Introduction to the Relational Model

Topics to be covered…

1.Structure of Relational Databases

2.Database Schema

3.Keys

4.Relational Algebra

5.Basic operators of Relational Algebra

6 Modification of Databases using Relational Algebra

7.Database Constraints

1.Structure of Relational Databases

A relational database is a database based on the relational model .

Current popular RDBMS include:

DB2 and informing Dynamic server —from IBM

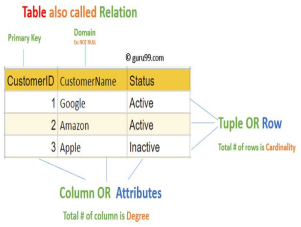
Oracle and Rdb from Oracle

SQL server and MS Access from Microsoft

In relational model, the data and relationships are represented by collection of inter-related tables , each of which is assigned a unique name.

Each table(relation) have:

- **Attribute**: It contains the name of a column in a particular table.Each attribute Ai must have a domain, dom(Ai)
- **Domain**: It contains a set of atomic values that an attribute can take.
- **Relational instance:** In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.
- **Relational schema:** A relational schema contains the name of the relation and name of all columns or attributes.
- **Relational key:** In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely
- **Null value:**it is a special value that signifies that the value is unknown or does not exist.
- **Degree**:The total number of columns or attributes in the relation.
- **Cardinality**:Total number of rows present in the Table.



2.Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database.

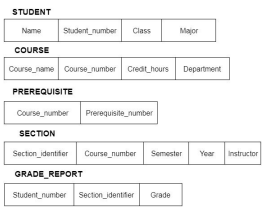
It defines how the data is organized and how the relations among them are associated.

It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them.

A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.

A database schema is designed by the database designers to help programmers whose software will interact with the database.



3.Keys

Keys play an important role in the relational database.

A key is an attribute or set of attributes that uniquely identifies any record(or tuple) from the table.

It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

Following are different types of keys:

1.super key

2. Candidate key

3.Primary key

4.Alternate key

5.Composite key

6.Foreign Key

1.Super key

A super key is a combination of all possible attributes that can uniquely identify the rows(or tuples) in the given relation.

Super key is a superset of a candidate key.

A super key may have additional attribute that are not needed for unique identity.

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

Super Keys:

1. {Emp_Id}
2. {Aadhar_No}
3. {Email_Id}
4. {Emp_Id, Aadhar_No}
5. {Aadhar_No, Email_Id}
6. {Emp_Id , Email_Id}
7. {Emp_Id, Aadhar_No, Email_Id}
8. {Emp_Id, Name}
9. {Emp_Id, Name, Dep_Id}
10. {Emp_Id,Name, Aadhar_No, Email_Id, Dept_Id}, etc....

2.Candidate key

It is an attribute or set of an attribute which can uniquely identify a tuple.

It is a minimal super key or **a super key with no redundant attributes,**because we select a candidate key from a set of super key such that selected candidate key is the minimum attribute required to uniquely identify the table.

It is defined as distinct set of attributes from which primary key can be selected.

It is not allowed to have NULL.

Candidate Keys

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

- **Candidate Keys**
  1. {Emp_Id}
  2. {Aadhar_No}
  3. {Email_Id}

3.Primary key

Primary key is a column of a table or a set of columns that helps to identify every tuple present in that table uniquely.

There can be only one primary Key in a table(relation).

The value of primary key can never be NULL.

The value of primary key must always be unique(not duplicate).

The value of primary key can never be changed I.e no updation is possible.

The value of primary key must be assigned when inserting a record.

Example:Emp_Id

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

4.Alternate key

All candidate keys except primary key is called Alternate key.

Primary Key Alternate Keys

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

- **Alternate Keys**
  1. {Aadhar_No}
  2. {Email_Id}

5.Composite key

A composite key is a set of two or more attributes that help identify each tuple in a table uniquely.

The attributes in the set may not be unique when considered separately.

However,when taken all together,they will ensure uniqueness.

Cust_Id	Order_Id	Product_Code	Product_Count
C01	001	P111	5
C02	012	P111	8
C02	012	P222	6
C01	001	P333	9

Composite Key:

{Cust_Id, Product_Code}

6.Foriegn key

It is a key used to link two tables together.

An attribute (or set of attributes) in one table that refers to the primary key in another table.

The purpose of the Foriegn key is to ensure referential integrity of the data.

It can take only those values which are present in the primary key of the referenced relation.

It may have a name other than that of a primary key.

It can take the NULL value.

There is no restriction on a foreign key to be unique.

In fact,foreign key is not unique most of the time.

Referenced relation may also be called as the master table or primary table or parent table.

Referencing relation may also be called as the foreign table or child table.

Foreign Keys

Employee Table (Referencing relation)

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

Foreign Key:
In Employee Table
1. Dept_Id

Primary Key

Department Table (Referenced relation)

Dept_Id	Dept_Name
1	Sales
2	Marketing
3	HR

Relational Query Language

Relational database system are expected be equipped with a query language that assist its users to query the database instances.

Query="Retrieval Program"

Two types of Query Language:

1.Procedural Query Language(Relational Algebra)

In Procedural query language,user instructs the system to perform a series of operations to produce the desired result.

User tells what's data to be retrieved from database and how to retrieve it.

2.Non-Procedural (Declarative)Query Language (relational calculus)

In this language ,user instructs the system to produce the desired result without telling the step by step process.

User tells what data to be retrieved from database,but doesn't tell how to retrieve it.

Relational Algebra

RELATIONAL ALGEBRA is a widely used procedural query language.

It collects instances of relations as input and gives occurrences of relations as output.

Relational algebra is a language for expressing relational database queries.

It uses various operations to perform queries.An operator can be either unitary or binary.

Types of operator in relational algebra are:

1.Basic /Fundamental operator

2.Additional/derived operator

Relational algebra operators works on one or more relations to define another relation without changing the original relations.

SQL Relational algebra query operations are performed recursively on a relation.

Relational Algebra Operations

Basic operators
1.Projection
2.Selection
3.Union
4.Set Difference
5.Cartesian product
6.Rename

Derived operators
1.Natural join
2.Set intersection
3.Division

1.Project operation

Project operation is done by Projection Operator which is represented by "pi"(∏).

It is used to retrieve certain attributes(columns) from the table.

It is also known as vertical partitioning as it separates the table vertically.

It is also a **unary operator**.

Notation : ∏ a (r)

Where ∏ is used to represent PROJECTION
r is used to represent RELATION
a is the attribute list

Duplicate rows are automatically eliminated from result.

The sql SELECT command corresponds to relational project

Student

roll_no	name	age
1	A	20
2	B	17
3	C	16
4	D	19
5	E	18
6	F	18

Query 1: Display (or project) the name of students in student table

Π_{name} (Student)

name
A
B
C
D
E
F

Student

roll_no	name	age
1	A	20
2	B	17
3	C	16
4	D	19
5	E	18
6	F	18

Query 2: Display the age of students in student table

Π_{age} (Student)

age
20
17
16
19
18

Note: By default, projection removes duplicate values

2.selection operation

Select operation is done by Selection Operator which is represented by "sigma"(σ).

It is used to retrieve tuples(rows) from the table where the given condition is satisfied.

It is a **unary operator** means it require only one operand.

Notation : $\sigma_p(R)$

Where σ is used to represent SELECTION
R is used to represent RELATION
p is the logic formula which may use logical connectors like **and, or,** and **not**. These terms may use relational operators like – =, ≠, ≥, < , >, ≤

The WHERE clause of a sql command corresponds to relational select .

Example:select tuples from student table whose age is greater than 17

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 1: Select student whose roll no. is 2

$\sigma_{roll\_no=2}$ (Student)

roll_no	name	age	address
2	B	17	Mumbai

Note: *In Selection operation, schema of resulting relation is identical to schema of input relation*

Student

roll_no	name	age	address
1	A	20	Bhopal
2	B	17	Mumbai
3	C	16	Mumbai
4	D	19	Delhi
5	E	18	Delhi

Query 4: Select students whose age is greater than 17 and who lives in Delhi

$\sigma_{age > 17 \wedge address = "Delhi"}$ (Student)

roll_no	name	age	address
4	D	19	Delhi
5	E	18	Delhi

3.Union operation

Union operation is done by Union Operator which is represented by "union"(U).

It selects all tuples from both relations but with the exception that for the union of two relations/tables both relations must have the same set (same number and same domain)of Attributes.

It is a **binary operator** as it requires two operands.

Notation: R U S
Where R is the first relation
S is the second relation

SQL: (select * from R)UNION (select * from S)

If relations don't have the same set of attributes, then the union of such relations will result in NULL.

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) U (Employee)

Roll_no	Name
1	A
2	B
3	C
4	D
8	G
9	H

Note: Union is commutative: A U B = B U A

4.Set Difference

Set Difference as its name indicates is the difference of two relations (R-S).

It is denoted by a "Hyphen"(-) and

it returns all the tuples(rows) which are in relation R but not in relation S.

For a set difference to be valid,the following conditions must hold:

1.two relations R and S both have same number of attributes.

2.Corresponding attribute(or column) have the same domain .

It is also a **binary operator**.

Notation : R - S
Where R is the first relation
S is the second relation

SQL :(select * from R)EXCEPT (select * from S)

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) - (Employee)

Roll_no	Name
1	A
3	C
4	D

5.Cartesian Product(x)

Cartesian product is denoted by the "X" symbol.

Let's say we have two relations R and S. Cartesian product will combine every tuple(row) from R with all the tuples from S.

Generally ,it is never a meaningful operation when it is performed alone.However,it becomes meang=ingful when it is followed by other operations.

It is also a **binary operator**.

Notation: R X S
Where R is the first relation
S is the second relation

R1

A	B
α	1
β	2

R2

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

R1 x R2

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Can build expressions using multiple operations

Example: $\sigma_{A=C}(R1 \times R2)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

R1 x R2

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

6.Rename operation

Rename operation is denoted by "Rho"(ρ).

As its name suggests it is used to rename the output relation.

Rename operator too is a **binary operator**.

Notation: ρ(R,S)

Where R is the new relation name
S is the old relation name

Derived operators

These include three operations:

1.Join Operations

2. Intersection operation

3. Division operation.

1.Join operation

Join Operations are binary operations that allow us to combine two or more relations.

It is a combination of a Cartesian product followed by a selection process.

Join= Cartesian Product+ Selection

A join operation pairs two tuples from different relations,if and only if a given join condition is satisfied.

Symbol:

Example

They are further classified into two types:

1. Inner Join. 2.Outer Join.

Inner join

An inner join includes only those tuples that satisfy the matching criteria ,while the rest of tuples are excluded.

There are three types of Inner join:

1.Theta join

2.Equi join

3.Natural join

2.Equi join

When a theta join uses only equivalence(=) condition,it becomes a Equi join.

It is a special case of theta (or condition) join where condition contains equalities(=).

Notation: $A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

S1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$S1 \bowtie_{S1.sid = R1.sid} R1$

S1.Sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

1.Theta join

It combines tuples from different relations provided they satisfy the theta (Θ) condition.

It is used when we want to join two or more relation based on some conditions.

The join condition is denoted by symbol Θ.

It uses all kinds of comparison operator like <,>,<=,>=,=

Notation: $R1 \bowtie_{\theta} R2$

Where **θ is a predicate/condition.**It can use any comparison operator.

$R1 \bowtie_{\theta} R2 = \sigma_{\theta}(R1 \times R2)$

S1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$S1 \bowtie_{S1.sid < R1.sid} R1$

S1.Sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.0	58	103	11/12/

3.Natural join

It can only be performed if there is at least one common attribute that exist between two relations.

The attributes must have same name and domain.

It does not use any comparison operator.

Notation:R1⋈R2

The result of the natural join is the set of all combinations of tuples in two relations A and B that are equal on their common attribute names.

It is by default inner join because the tuples which does not satisfy the conditions of join does not appear in result set.

loan

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

Loan ⋈ borrower

loan-number	branch-name	amount	customer-name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Outer join

Contains matching tuples that satisfy the matching condition,along with some or all tuples that do not satisfy the matching condition.

Contains all rows from either one or both relation.

Outer join-Natural Join + Extra information(from left table,right table or both table)

There are three types of Outer join:

1.Left Outer join

2.Right Outer join

3.Full Outer join

1.Left Outer

In Left outer join,all the tuples from the Left relation R1 are included in the resulting relation.

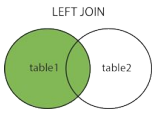
The tuples of R1 which do not satisfy join condition will have values as NULl for attributes of R2.

In short:

1. All records from left table
2. Only matching records from right table.

Symbol:⟕

Notation:R1⟕R2



Example: Left outer join

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟕ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	**NULL**
102	Electronics	Sara

2.Right Outer join

In Right outer join ,all the tuples from the right relation R2 are included in the resulting relation.

The tuples of R2 which do not satisfy join condition will have null values for attributes of R1.

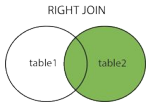
In short:

1.All records from right table

2.Only matching records from left table

Symbol:⟖

Notation:R1⟖R2



Example: Right outer join

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟖ HOD

CID	Course	Name
100	Database	Rohan
102	Electronics	Sara
104	**NULL**	Jiya

3.Full Outer join

In full outer join,all the tuples from both left relation R1 and right relation R2 are included in the resulting relation.

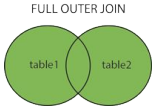
The tuples of both relations R1 and R2 which do not satisfy join condition ,their respective unmatched attributes are made NULL.

In short:

All records from all table

Symbol:⟗

Notation:R1⟗R2



Example: Full outer join

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟗ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	**NULL**
102	Electronics	Sara
104	**NULL**	Jiya

Set intersection

Suppose R and S are two relations.The set Intersection operation select all the tuples that are in both relations R and S.

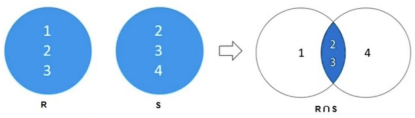
For a intersection to be valid ,the following condition must hold:

1.two relations R and S both have same number of attributes.

2.Corresponding attribute have the same domain

Notation:R ∩S

Sql: (select * from R) intersect (select * from S);



Example

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) ∩ (Employee)

Roll_no	Name
2	B

3.Division

Division Operator (÷)...

***Division operator* 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.***

Example 1:

A

x	y
a	1
b	2
a	2
d	4

÷

B

y
1
2

A÷B

x
a

Example 2: shows how it works

A

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

pno
P2

A/B1

sno
S1
S2
S3
S4

B2

pno
P2
P4

A/B2

sno
S1
S4

B3

pno
P1
P2
P4

A/B3

sno
S1

Unit-04

Structured Query Language

Topics to be covered….

1.Overview of the SQL Query Language

2.SQL Data Definition

3.SQL Constraints

4.Basic Structure of SQL Queries(queries on single relation and queries on multiple relation)

5.Additional Basic Operations(rename,string,attribute specification,ordering,where clause predicates)

6.DML operations

7.Set operations(union,intersect,except)

8. Aggregate Functions(Average,min,max,sum,count)

9.Nested Sub-queries

10. **Joins**

11. views.

1.Overview of the SQL Query Language

IBM developed the original version of SQL,originally called sequel.

Many products now support the sql language.

It is a standard relational database language.

It has several parts:

1.Data-definition language:It provides commands for defining relation schema,deleting relations and modifying relation schema.

2.Data -manipulation language:It provides ability to query information from the database and to insert tuples into,delete tuples from and modify tuples in database.

3.Integrity:It includes commands for specifying integrity constraints that the data stored in the database must satisfy.

4.View definition: it includes command for defining views.

5.Transaction Control:It includes command for specifying the beginning and ending of transaction.

6.Embedded SQL and dynamic SQL:it defines how sql statements can be embedded within general-purpose programming languages such as C,C ++ and java.

7.Authorization:It includes commands for specifying access rights to relations and views.

2.SQL Data Definition

The set of relations in a database must be specified to the system by means of a data-definition language (DDL).

It also allows specification of following information about each relation:

1.the schema for each relation

2.the types of values associated with each attribute

3.the integrity constraints

4.the set of indices to be maintained for each relation.

5.the security and authorization information for each relation.

6.the physical storage structure of each relation on disk.

2.1 Basic types

The SQL standard supports a variety of built-in types,including:

1.char(n):a fixed-length character string with user-specified length n.

2.varchar(n):a variable-length character string with user-specified maximum length n.

3.int:an integer

4.small int

5.Numeric(p,d):a fixed point number with user-specified precision.The number consist of p digits (plus a sign), and d of the p digits are to the right of the decimal point.Ex: numeric(3,1)=44.5,neither 444.5 or 0.32

6.real,double precision:floating-point and double-precision floating-point numbers with machine dependent precision.

7.float(n):a floating -point number,with precision of at least n digits.

In addition to basic data type,sql supports following data types:

1.date:a calendar date containing a (four-digit) year,month,and day of the month.

Ex: '2022-04-24'

2.time:The time of the day in hours,minutes and seconds.A variant ,time(p) can be used to specify the number of fractional digits for seconds.

Ex: '09:30:00'

3.timestamp:A combination of date and time .Time-zone information is also stored if with time zone is specified.

Ex:'2022-04-25 10:20:01'

Caste:it convert character string e to the types t,where t is one of date ,time or timestamp.The string must be in appropriate format.

Ex:CAST('30-APRIL-2015' AS DATE)

Extract:to extract individual fields of a date or time value.extract(field form d) where field can be one of year,month,day,hour,minute or second.

Ex:EXTRACT(MONTH FROM "2017-06-15")

Current_time:it returns current time (with time zone).Ex:SELECT CURRENT_TIME();

Current_date:it returns current date.Ex:SELECT CURRENT_DATE();

Current_timestamp:returns current local time with date.Ex:SELECT CURRENT_TIMESTAMP();

Large-object type:

Current generation database applications need to store attributes that can be large such as photograph or very large such as high-resolution medical image or video clip.SQL therefore provides following types of large object (lob) type:

1.BLOBs (Binary LOBs) used to store unstructured binary (also called “raw”) data, such as video clips.

2.CLOBs (Character LOBs) used to store large blocks of character data from the database character set.

Ex:CREATE TABLE Pictures (

ID NUMBER(12),

PicName VARCHAR2(20),

Picture BLOB

)

User-defined data type:User will create their own data type for attributes.

Syntax: create type type_name as data_type

Ex: create dollar as numeric(12,2)

2.2 Basic schema Definition

We define an sql relation by using the **create table** command.

The general form of create table command is:

CREATE TABLE table_name(

column1 datatype,

column2 datatype,

column3 datatype,

(Integrity-constraint $_1$),

…….,

(Inteegrity-constraint$_n$));

Here integrity constraints are optional constraints that restrict the set of allowed values for column.

3.SQL constraints

SQL supports following different integrity constraints:

1. NOT NULL Constraint − Ensures that a column cannot have NULL value.
2. DEFAULT Constraint − Provides a default value for a column when none is specified.
3. UNIQUE Constraint − Ensures that all values in a column are different.
4. PRIMARY Key − Uniquely identifies each row/record in a database table.
5. FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
6. CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.

1.NOT NULL constraint:

The NOT NULL constraint enforces a column to NOT accept NULL values. The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

create table person(p_id primary key, pname varchar(4) not null, mobile_no number(10), address varchar(100));

2.**Default:** The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

Example:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255) DEFAULT 'Sandnes'
);
insert into Persons(ID,LastName,FirstName,Age) values(1,'Bean','Mr.',55);
```

3.Unique constraint:

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it. Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

create table person(p_id number(4) primary key, pname varchar(40),mobile_no number(10) unique, address varchar(100));
Or

create table person(p_id number(4) primary key, pname varchar(40), mobile_no number(10), address varchar(100), unique(mobile_no));

4.Primary key constraint:

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values. A primary key column cannot contain NULL values. Most tables should have a primary key, and each table can have only ONE primary key.

create table Accounts(acc_no number(5) primary key, balance number(8,2))
or
create table Accounts(acc_no number(5), balance number(8,2) constraint primary key(acc_no));

5.FOREIGN KEY in one table points to a PRIMARY KEY in another table. Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table. The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table. The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

create table orders(o_id number(3) primary key, orderno number not null, constraint order_fk foreign key (p_id) references persons(p_id));

6.check(): The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

Example:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int check(Age>18));
```

4.Basic Structure of SQL Queries(queries on single relation and queries on multiple relation)

The fundamental structure of SQL queries includes three clauses(**in-built functions available to us in SQL**) that are select, from, and where clause.

- In the select clause, you have to specify the attributes that you want to see in the result relation
- In the from clause, you have to specify the list of relations that has to be accessed for evaluating the query.
- In the where clause involves a predicate that includes attributes of the relations that we have listed in the from clause.

Structure of the queries can be imposed on:

1.Single relation

2.multiple relation

1.Single relation

Here queries are applicable only on one relation.Following are some syntax use on single relations

1.Select column_name from Table_name

2.select distinct column_name from Table_name:Eliminates duplicate values.

3.The select clause may also contains arithmetic expressions involving the operators +,-,*,/ on constants or attributes of tuples.

Select id,name,salary*1.1 from employee

4.select column_name from Table_name where

Sql allows to use logical connectives like and,or,not in where clause.

2.Queries on Multiple Relations

A typical SQL query on multiple relations has the form:

Select A1,A2,A3…..An

From r1,r2,r3…rn

Where p;

The form clause itself defines a cartesian product of the relations listed in the clause.

Example:

select student.id,Employe.name from EMPLOYE,STUDENT where EMPLOYE.name=STUDENT.name;

5.Additional Basic operations

There are number of additional basic operations that are supported in sql are:

1.The Rename operation

2.String operation

3.Ordering the display of tuples

4.The Group by clause

5.The Having clause

1.The Rename operation

SQL '**AS**' is used to assign a new name temporarily to a table column or even a table.

Syntax:

SELECT Column_Name1 AS New_Column_Name, Column_Name2 As New_Column_Name FROM Table_Name;

Example:

SELECT day_of_order AS 'Date', Customer As 'Client', Product, Quantity FROM orders;

Assigning a temporary name to a table

SELECT Column_Name1, Column_Name2 FROM Table_Name as T1,Table_Name as T2;

Example:

SELECT s.Student_RollNo, s.Student_Name, s.Student_Gender, s.Student_PhoneNumber, s.Student_HomeTown FROM **students AS s** WHERE s.Student_RollNo = 3;

2.String operations

1.Concatenating:concatenates two strings or words and forms a new string in the result.

Syntax:SELECT CONCAT(Column_Name1, Column_Name2, ... column_NameN) AS Alias_Name FROM Table_Name;

Example: select CONCAT(city,pincode)as Address from student;

2.Extracting substring:SUBSTR() function returns a substring from the str starting at start_position with the substring_length length.

SELECT SUBSTR('Oracle Substring', 1, 6) SUBSTRING FROM dual;

3.Finding the length of string

returns the length of the given string. It shows the number of all characters and spaces from the sentence.

SYNTAX:SELECT LENGTH(Column_Name) AS Alias_Name FROM Table_Name;

Example: select LENGTH(city) from student;

4.**Converting string to uppercase and lower case**

LOWER:select LOWER(NAME)as Address from student;

UPPER:select UPPER(NAME)as Address from student;

5.PATTERN MATCHING

Pattern matching can be performed on strings,using the like operator.

We describe patterns by using two special characters:

1.Percent (%):It matches any substring.

2.Underscore(_):it matches any character.

Examples:

1.'Intro %':matches any string beginning with 'Intro'

2.'%Comp%: matches any string containing 'Comp' as substring

3.'_ _ _':matches any string of exactly three character.

4.'_ _ _%':matches any string of at least three character.

Exampe:select * from route_Header where destination like 'M%';

Attribute specification:

*-used to select all attributes from the table.

Select table_name1.* from table_name1,table_name2 where table_name1.column_name=table_name2.column_name;

Indicates that all attributes from table_name1 is selected.

Group By clause

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

SELECT column1, column2 FROM table_name WHERE [conditions]

GROUP BY column1, column2 ORDER BY column1, column2

Example:

select address,count(*) from EMP GROUP BY address order by address DESC

	ADDRESS	COUNT(*)
1	kota	2
2	Mumbai	1
3	Delhi	2
4	Bhopla	1
5	Ahmedabad	1

Employee

ID	SALARY	AGE	ADDRESS	NAME
1	2000	32	Ahmedabad	Ramesh
2	1500	25	Delhi	khilan
3	2000	23	kota	kaushik
4	2000	25	Delhi	chaitali
5	4500	27	Mumbai	hardik
6	8500	22	kota	komal
7	4500	23	Bhopla	Muffy

3.Ordering the display of tuples

The Order By clause defines in what order to return a data set retrieved with a SQL SELECT command.

Syntax:

SELECT "COLUMN_NAME"

FROM TABLE_NAME [WHERE CONDITIO]

ORDER BY COLUMN_NAME/COLUMN_NUMBER[ASC,DESC,NULLS FIRST,NULLS LAST]

Example:

1.select name from student order by name ASC;

2.select roll_no from student order by roll_no DESC;

3.select name from student order by address NULLS FIRST;

3.select name from student order by address NULLS LAST;

4.select name from student order by 1 ASC;

5.Select name from student order by 2 DESC;

2.select address,count(*) from EMP **where salary=2000** GROUP BY address order by address DESC ;

	ADDRESS	COUNT(*)
1	kota	1
2	Delhi	1
3	Ahmedabad	1

Having Clause

The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.

The **WHERE clause places conditions on the selected columns**, whereas **the HAVING clause places conditions on groups created by the GROUP BY clause**.

Syntax:

SELECT column1, column2

FROM table1, table2

WHERE [conditions]

GROUP BY column1, column2

HAVING [conditions]

ORDER BY column1, column2

ID	SALARY	AGE	ADDRESS	NAME
1	2000	32	Ahmedabad	Ramesh
2	1500	25	Delhi	khilan
3	2000	23	kota	kaushik
4	2000	25	Delhi	chaitali
5	4500	27	Mumbai	hardik
6	8500	22	kota	komal
7	4500	23	Bhopla	Muffy

Example:

select address,count(*) from EMP GROUP BY address having count(address)>=2;

	ADDRESS	COUNT(*)
1	Delhi	2
2	kota	2

6.DML operations

DML (Data Manipulation Language) statements are the element in the SQL language that is used for data retrieval and manipulation. Using these statements you can perform operations such as: adding new rows, updating and deleting existing rows, merging tables and so on.

1. SELECT: retrieve data from the database.
2. INSERT: insert into a table
3. UPDATE: updates existing data within a table
4. DELETE: delete all records from a database table without deleting the structure of the database.

7.SET OPERATION

SQL set operations are used for combining data from one or more tables

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

1.UNION:

In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.

The union operation eliminates the duplicate rows from its resultset.

SYNTAX:

SELECT column_name FROM table1

UNION

SELECT column_name FROM table2;

EXAMPLE:

SELECT * FROM STUDENT

UNION

SELECT * FROM EMPLOYE;

STUDENT

ID	NAME
1	A
2	B
3	C
4	D

EMPLOYEE

ID	NAME
2	B
8	G
9	H

OUTPUT

ID	NAME
1	A
2	B
3	C
4	D
5	G
6	H

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

SELECT column_name FROM table1

UNION ALL

SELECT column_name FROM table2;

EXAMPLE:

SELECT * FROM STUDENT

UNION ALL

SELECT * FROM EMPLOYE;

STUDENT

ID	NAME
1	A
2	B
3	C
4	D

EMPLOYE

ID	NAME
2	B
8	G
9	H

OUTPUT

ID	NAME
1	A
2	B
3	C
4	D
2	B
8	G
9	H

3.Intersect

returns the common rows from both the SELECT statements.

It has no duplicates and it arranges the data in ascending order by default.

SYNTAX:

SELECT column_name FROM table1

INTERSECT

SELECT column_name FROM table2;

EXAMPLE:

select * from STUDENT

INTERSECT

SELECT * FROM EMPLOYE;

STUDENT

ID	NAME
1	A
2	B
3	C
4	D

EMPLOYE

ID	NAME
2	B
8	G
9	H

OUTPUT

ID	NAME
2	B

4.MINUS/Except

- used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.
- EXCEPT is available in the PostgreSQL database and MINUS is available in mysql and Oracle.

SYNTAX:

SELECT column_name FROM table1

MINUS

SELECT column_name FROM table2;

EXAMPLE:

select * from STUDENT

MINUS

SELECT * FROM EMPLOYE;

STUDENT

ID	NAME
1	A
2	B
3	C
4	D

EMPLOYE

ID	NAME
2	B
8	G
9	H

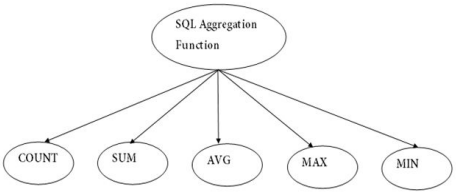
OUTPUT

	ID	NAME
1	8	G
2	9	H

8. Aggregate Functions

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

Types of SQL Aggregation Function



1.COUNT:

used to Count the number of rows in a database table.

SYNTAX:

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

EXAMPLE:

SELECT COUNT(DISTINCT name) FROM STUDENT;

	COUNT(DISTINCTNAME)
1	5

SELECT COUNT(*) FROM STUDENT;

	COUNT(*)
1	7

STUDENT

ID	NAME
2	B
8	G
9	H
1	A
3	C
4	A
5	A

2.SUM

used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()

or

SUM([ALL|DISTINCT] expression)

EXAMPLE:

SELECT SUM(DISTINCT id) FROM STUDENT;

	SUM(DISTINCTID)
1	32

SELECT SUM(id) FROM STUDENT;

	SUM(ID)
1	32

STUDENT

ID	NAME
2	B
8	G
9	H
1	A
3	C
4	A
5	A

3.AVG

used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

EXAMPLE:

SELECT AVG(id) FROM STUDENT;

	AVG(ID)
1	4.571428571428571428571428571428571428571

STUDENT

ID	NAME
2	B
8	G
9	H
1	A
3	C
4	A
5	A

4.MAX

used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

MAX([ALL|DISTINCT] expression)

EXAMPLE:

SELECT MAX(id) FROM STUDENT;

	MAX(ID)
1	9

STUDENT

ID	NAME
2	B
8	G
9	H
1	A
3	C
4	A
5	A

5.MIN

used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN()

or

MIN([ALL|DISTINCT] expression)

EXAMPLE

SELECT MIN(id) FROM STUDENT;

MIN(ID)
1

STUDENT

ID	NAME
2	B
8	G
9	H
1	A
3	C
4	A
5	A

9.Nested Sub-queries

In nested queries, a query is written inside a query. The result of inner query(subquery) is used in execution of outer query.

Subquery is appear within WHERE or HAVING clause of other query.

Outer query is called as **main query** and inner query which is written in main query is called **subquery**

Subquery in the WHERE clause: The result of the subquery is used to select some rows from main query.

Subquery in the HAVING clause: The result of the subquery is used to select some groups from main query.

Subqueries can be nested within other sub queries.

Syntax

SELECT column_name [, column_name]

FROM table1 [, table2]

WHERE column_name OPERATOR

(SELECT column_name [, column_name]

FROM table1 [, table2]

[WHERE])

- **The subquery (inner query) executes once before the main query (outer query) executes.**
- **The main query (outer query) use the subquery result.**

There are some guidelines to consider when using subqueries :

- A subquery must be **enclosed in parentheses.**
- A subquery must be placed on the **right side of the comparison operator.**
- Subqueries cannot manipulate their results internally, therefore **ORDER BY clause cannot be added into a subquery**. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
- Use single-row operators with single-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Types of Subqueries

Single Row Sub Query: Sub query which returns single row output. They mark the usage of single row comparison operators, when used in WHERE conditions.

Multiple row sub query: Sub query returning multiple row output. They make use of multiple row comparison operators like IN, ANY, ALL. There can be sub queries returning multiple columns also.

Correlated Sub Query: Correlated subqueries depend on data provided by the outer query.This type of subquery also includes subqueries that use the EXISTS operator to test the existence of data rows satisfying specified criteria.

Operators

1.Set Membership:IN, NOT IN, ANY, or ALL operators.

2. Set Comparison:< **some**, <= **some**, >= **some**, = **some**, and <> **some** comparisons.

< **all**, <= **all**, >= **all**, = **all**, and <> **all**

3. Test for Empty Relations:exists ,not exists,except

4. Test for the Absence of Duplicate Tuples:unique,not unique

There are mainly two types of nested queries:

1.Independent Nested Queries

2.Co-related Nested Queries

1.Set Membership

1.IN and NOT IN

This is multi row operator used to check that a given tuple in main memory satisfy at least one values returned by a subquery.

Inversely ,we can use NOT IN condition to check test value is not a member of result set of inner subquery.

Example:

Route_id	Route_no	Cate_code	Origin	Destination	Fare	Distance	Capacity
101	33	01	Madurai	Madras	35	250	50
102	25	02	Trichy	Madurai	40	150	50
103	15	03	Thanjavur	Madurai	59	140	50
104	36	04	Madras	Banglore	79	375	50
105	40	01	Banglore	Madras	80	375	50
106	38	02	Madras	Madurai	39	250	50
107	39	03	Hydrabad	Madras	50	430	50
108	41	04	Madras	Cochin	47	576	50

Which bus runs for minimum distance?

SELECT * FROM route_header

WHERE route_header.distance in

(select distance from route_header where route_header.distance>375);

ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	107	39	3 HYDRABAD	MADRAS	50	430	50
2	108	41	4 MADRAS	COCHIN	47	576	50

2. SELECT * FROM route_header
WHERE route_header.distance **not in**
(select distance from route_header where route_header.distance>250);

ANY and ALL

The operators ANY and ALL are always used in combination with one of the comparison operators.

The general syntax of both operators is

column_name operator [ANY | ALL] query

where operator stands for a comparison operator and query is an inner query.

The ANY operator evaluates to TRUE if the result of the corresponding inner query contains at least one row that satisfies the comparison. The keyword SOME is the synonym for ANY.

Route_id	Route_no	Cate_code	Origin	Destination	Fare	Distance	Capacity
101	33	01	Madurai	Madras	35	250	50
102	25	02	Trichy	Madurai	40	159	50
103	15	03	Thanjavur	Madurai	59	140	50
104	36	04	Madras	Banglore	79	375	50
105	40	01	Banglore	Madras	80	375	50
106	38	02	Madras	Madurai	39	250	50
107	39	03	Hydrabad	Madras	50	430	50
108	41	04	Madras	Cochin	47	576	50

SELECT * FROM route_header

WHERE distance>any (select distance from route_header where destination='MADRAS');

	ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	108	41	4	MADRAS	COCHIN	47	576	50
2	107	39	3	HYDRABAD	MADRAS	50	430	50
3	104	36	4	MADRAS	BANGLORE	79	375	50
4	105	40	1	BANGLORE	MADRAS	80	375	50

Works:

select distance from route_header where destination='MADRAS'

All values from route_header are compare with all distance values

	DISTANCE
1	250
2	375
3	430

The **ALL** operator compares a column value or literal value with the result of a subquery that returns a single-column values.

- A subquery used with the ALL operator, can only return a single column values.
- The ALL operator must be preceded by comparison operators like =, !=, >, >=, <, <=.
- The ALL operator uses AND with the result values of a subquery to compare a column of the outer query.
- The data type of the returned values from a subquery must be the same data type as the outer query expression.

Syntax:

expression <operator> ALL(subquery);

-- or

SELECT * FROM table_name

WHERE column_name <operator> ALL(subquery);

2.SELECT * FROM route_header

WHERE distance>all

(select distance from route_header where destination='MADRAS');

	ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	108	41	4	MADRAS	COCHIN	47	576	50

Works:

select distance from route_header where destination='MADRAS'

All values of route_header are compare with 430 only.

	DISTANCE
1	250
2	375
3	430

2. Set Comparison

1.Some (< **some**, <= **some**, >= **some**, = **some**, and <> **some** comparisons.

used to compare a value to each value in a list of results from a query and evaluate to true if the result of an inner query contains at least one row.

SELECT * FROM route_header

WHERE distance**<some**

(select distance from route_header where cat_code=4);

	ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	103	15	3	THANJAVUR	MADURAI	59	140	50
2	102	25	2	TRICHY	MADURAI	40	159	50
3	106	38	2	MADRAS	MADURAI	39	250	50
4	101	33	1	MADURAI	MADRAS	35	250	50
5	104	36	4	MADRAS	BANGLORE	79	375	50
6	105	40	1	BANGLORE	MADRAS	80	375	50
7	107	39	3	HYDRABAD	MADRAS	50	430	50

Works:

select distance from route_header where cat_code=4

ALL distance values from route_header compare with 375 and 576

	DISTANCE
1	375
2	576

3.Test for Empty Relations:exists ,not exists,except

1.exists

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

Syntax:

SELECT *column_name(s)*

FROM *table_name*

WHERE EXISTS

(SELECT *column_name* FROM *table_name* WHERE *condition*);

Example:route_header

SELECT * FROM route_header WHERE exists (select distance from route_header where cat_code=4);

Route_id	Route_no	Cate_code	Origin	Destination	Fare	Distance	Capacity
101	33	01	Madurai	Madras	35	250	50
102	25	02	Trichy	Madurai	40	159	50
103	15	03	Thanjavur	Madurai	59	140	50
104	36	04	Madras	Banglore	79	375	50
105	40	01	Banglore	Madras	80	375	50
106	38	02	Madras	Madurai	39	250	50
107	39	03	Hydrabad	Madras	50	430	50
108	41	04	Madras	Cochin	47	576	50

SELECT * FROM route_header WHERE exists (select distance from route_header where cat_code=5);

OutPut:NULL

not exists

It is used to restrict the number of rows returned by the subquery.

SELECT * FROM route_header

WHERE not exists

(select distance from route_header where cat_code=5);

	ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	103	15	3	THANJAVUR	MADURAI	59	140	50
2	102	25	2	TRICHY	MADURAI	40	159	50
3	106	38	2	MADRAS	MADURAI	39	250	50
4	101	33	1	MADURAI	MADRAS	35	250	50
5	104	36	4	MADRAS	BANGLORE	79	375	50
6	105	40	1	BANGLORE	MADRAS	80	375	50
7	107	39	3	HYDRABAD	MADRAS	50	430	50

SELECT * FROM route_header

WHERE not exists

(select distance from route_header where cat_code=4);

output:NULL

except/MINUS

SELECT * FROM route_header

MINUS

select * from route_header where route_header.distance>250;

	ROUTE_ID	ROUTE_NO	CAT_CODE	ORIGIN	DESTINATION	FARE	DISTANCE	CAPACITY
1	101	33	1	MADURAI	MADRAS	35	250	50
2	102	25	2	TRICHY	MADURAI	40	159	50
3	103	15	3	THANJAVUR	MADURAI	59	140	50
4	106	38	2	MADRAS	MADURAI	39	250	50

4. Test for the Absence of Duplicate Tuples:unique,not unique

The UNIQUE predicate evaluates to True only if no two rows returned by the subquery are identical. In other words, the UNIQUE predicate evaluates to True only if all the rows that its subquery returns are unique.

SELECT FirstName, LastName

FROM CUSTOMER

WHERE UNIQUE

(SELECT CustomerID FROM SALES

WHERE SALES.CustomerID = CUSTOMER.CustomerID);

Not unique:it evaluates to true if all the rows that its subquery returns are not unique.

10.Joins

JOINS are used to retrieve data from multiple tables. A JOIN is performed whenever two or more tables are joined in a SQL statement.

s1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55
58	rusty	10	35

r1

sid	bid	day
22	101	17-09-22
58	103	23-09-22
33	104	29-09-22

1.inner join :An inner join includes only those tuples that satisfy the matching criteria ,while the rest of tuples are excluded.

Syntax:

SELECT *column_name(s)*FROM *table1* INNER JOIN *table2*

ON *table1.column_name = table2.column_name*;
Or
SELECT * FROM *table1* INNER JOIN *table2*
ON *table1.column_name = table2.column_name*;

Example:
SELECT * FROM s1 INNER JOIN r1 ON s1.sid = r1.sid;

	SID	NAME	RATING	AGE	SID_1	BID	DAY
1	22	dustin	7	45	22	101	17-09-22
2	58	rusty	10	35	58	103	23-09-22

2.NATURAL JOIN: It can only be performed if there is at least one common attribute that exist between two relations.

Syntax:
SELECT * FROM TABLE1 NATURAL JOIN TABLE2

Example:
SELECT * FROM s1 NATURAL JOIN r1 ;

3.LEFT OUTER JOIN:returns all records from the left table (table1), and the matching records from the right table (table2).
Syntax:
SELECT * FROM TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME

Example:
SELECT * FROM s1 LEFT OUTER JOIN r1 ON s1.sid = r1.sid;

	SID	NAME	RATING	AGE	SID_1	BID	DAY
1	22	dustin	7	45	22	101	17-09-22
2	58	rusty	10	35	58	103	23-09-22
3	31	lubber	8	55	(null)	(null)	(null)

4.RIGHT OUTER JOIN:returns all records from the right table (table2), and the matching records from the left table (table1).

Syntax:

SELECT * FROM TABLE1 RIGHT OUTER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME

Example:

SELECT * FROM s1 RIGHT OUTER JOIN r1 ON s1.sid = r1.sid;

	SID	NAME	RATING	AGE	SID_1	BID	DAY
1	22	dustin	7	45	22	101	17-09-22
2	58	rusty	10	35	58	103	23-09-22
3	(null)	(null)	(null)	(null)	33	104	29-09-22

5.FULL OUTER JOIN:returns all records when there is a match in left (table1) or right (table2) table records.

SYNTAX:

SELECT * FROM TABLE1 FULL OUTER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME

EXAMPLE:

SELECT * FROM s1 FULL OUTER JOIN r1 ON s1.sid = r1.sid;

	SID	NAME	RATING	AGE	SID_1	BID	DAY
1	22	dustin	7	45	22	101	17-09-22
2	58	rusty	10	35	58	103	23-09-22
3	(null)	(null)	(null)	(null)	33	104	29-09-22
4	31	lubber	8	55	(null)	(null)	(null)

Unit-05

Relational Database Design

Topics to be covered….

1.Features of Good Relational Designs

2.Problems with bad design

3. Decomposition using concept of functional dependencies

4.Armstrong's axioms

5.Closure of functional dependency

6.Closure of attribute

7. Introduction to process of Normalization and denormalization,

8.Normal Forms- 1NF, 2NF, 3NF, BCNF, Denormalization

Features of Good Relational Designs

Till now we have discussed how to obtain relational schema ,now we are going to discuss how relation can be establish in relational database.

Once we design relational schema(using ERD),we need to have good relation among the schema so that we can fetch and use the data in an application environment.

We discuss features of good relational design by considering

1. large database schema

2.small database schema

1.larger schema

Instructor(id,name,salary,Dept_name)

Department(Dept_name,building,budget)

Inst_dept(id,name,salary,Dept_name,building,budget)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

For a good relational design
1. we should not have a **redundancy** in our relational schema.

2.if we want to **new department** in this schema,we can't do it because we need to add instructor information also.It makes **loss of information**(means we have department information to add but we can not put it unless we have instructor information.

Because of this two issues we need to decompose larger schema into two different parts.

Let's see how to decompose schema ,to decompose it we need to follow certain rules based on some constraints

functional dependency

The functional dependency :A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.

Dept_name—>building

Dept_name—->budget

Dept_name can uniquely identify the building and budget.

But dept_name is not used to uniquely identify id,name and salary of instructor.so for this attributes functional dependency wouldn't work .

Now we will decode this schema in our original schema having:

Instructor(id,name,salary,Dept_name)

Department(Dept_name,building,budget)

Disadvantage of Large Schemas

- **Update anomalies:**
  - **Modifying** the *budget* in one tuple but not all tuples leads to **inconsistency**.
  - Cannot **insert** a new department until the first instructor is hired
    - *ID is part of PK.*
  - **Deleting** the last instructor will lose the department information

But not all decompositions are good.suppose we decompose

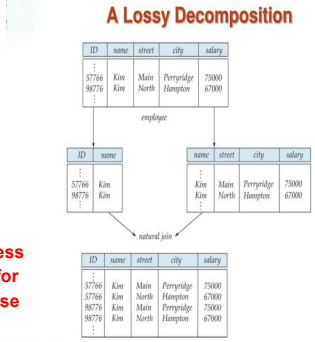
employee(id,name,city,salary)

employee(id,name)

employee(name,city,salary)

If we do natural join based on name:
1.it gives redundancy
2.makes data inconsistent
3.loss of information

Functional dependency and lossless decomposition are the key areas for good features of relational database design.



Decomposition using concept of functional dependencies

Functional dependencies

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.

It is denoted as X → Y, where X is a set of attributes that is capable of determining the value of Y.

The attribute set on the left side of the arrow, X is called Determinant,

while on the right side, Y is called the Dependent

Example:

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

valid functional dependencies:

1.roll_no → { name, dept_name, dept_building }
2.roll_no → dept_name , Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
3.dept_name → dept_building

Here are some invalid functional dependencies:

1.name → dept_name Students with the same name can have different dept_name, hence this is not a valid functional dependency.

2.dept_building → dept_name There can be multiple departments in the same ,hence dept_building → dept_name is an invalid functional dependency.

To check functional dependency we have following constraints:

For any attribute /set of attribute

x—–>y

If t1.x=t2.x and if t1.y=t2.y. Then we say functional dependency.

If t1.x=t2.x and if t1.y not equal t2.y ,then we say it's not functional dependency

R.No	Name	Marks	Dept	Course
1	A	78	CS	C1
2	B	60	EE	C1
3	A	78	CS	C2
4	B	60	EE	C3
5	C	80	IT	C3
6	D	80	EC	C3

R.No—>Name

Name—>R.No

R.No—>Marks

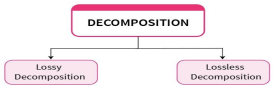
Dept—>Course

Course—>Dept

R.No,name—>Marks

Name—>Marks

If a relation is not properly decomposed, then it may lead to other problems like information loss, etc. There are two types of decomposition as shown below:



Rules for Decomposition
Whenever we decompose a relation, there are certain properties that must be satisfied to ensure no information is lost while decomposing the relations. These properties are:

1.Lossless Join Decomposition.
2.Dependency Preserving.

1.Lossless decomposition

1. Lossless decomposition

A lossless decomposition of a relation ensures that:

a) **No information is lost during decomposition.** This is why the term lossless is used in this decomposition as no information is lost.

b) If a relation R is divided into two relations R1 and R2 using lossless decomposition then the **natural join of R1 and R2 would return the original relation R.**

Rules of Lossless decomposition: For these rules, we are assuming that a relation R is divided into two relations R1 and R2.

Rule 1: Natural join or Union of R1 and R2 should return the original relation R.

R1 U R2 = R

Rule 2:The intersection of R1 and R2 should not be null. This is because there are some common attributes present in relation R1 and R2.

R1 ∩ R2 ≠ 0

Rule 3: The intersection of R1 and R2 is either a super key of R1 or R2, or both the relations R1 and R2.

R1 ∩ R2 = super key of R1 or R2 or both

Rule 4: Dependency preserving
The dependencies that exists in the original relation, exists after decomposition.

Let's say a relation R (A, B, C), where A is primary key is divided into two relations R1 (A, B) and R2 (C, A).

Let's check whether this decomposition is loss-less decomposition or not:

Rule 1:

R1 Natural join R2 = (A, B) Natural Join (C, A) = (A, B, C)

Union or R1 and R2 gives the original relations, thus first rule of lossless decomposition applies here.

Rule 2:

R1 ∩ R2 = (A, B) ∩ (C, A) = (A)

Result is not null so the second rule also applies here.

Rule 3:

R1 ∩ R2 = (A, B) ∩ (C, A) = (A)

Result is a super key of both the relations thus third rule also applies here.

Example of LossLess decomposition

StudentCourse Table

Student_Id	Student_Name	Course_Id	Course_Detail
S101	Chaitanya	C01	Maths
S102	Ajeet	C01	Maths
S103	Rahul	C02	Science
S104	Steve	C02	Science
S105	John	C03	English
S101	Chaitanya	C03	English
S102	Ajeet	C02	Science

The primary key of given relation is {Student_Id, Course_Id}

This table has **redundant data**.

Let's decompose this relation into two relations.

Student Table:

The primary key of this table is {Student_Id, Course_Id}

Student_Id	Student_Name	Course_Id
S101	Chaitanya	C01
S102	Ajeet	C01
S103	Rahul	C02
S104	Steve	C02
S105	John	C03
S101	Chaitanya	C03
S102	Ajeet	C02

Course Table:

The primary key of this table is {Course_Id}

Course_Id	Course_Detail
C01	Maths
C02	Science
C03	English

Rule 1:select * from student_tb natural join course_tb

	COURSE_ID	STUDENT_ID	STUDENT_NAME	COURSE_DETAIL
1	C01	S101	Chaitanya	Maths
2	C01	S102	Ajeet	Maths
3	C02	S103	Rahul	Science
4	C02	S104	Steve	Science
5	C03	S105	John	English
6	C03	S101	Chaitanya	English
7	C02	S102	Ajeet	Science

Rule 2:

select student_tb.course_id from student_tb

intersect

select course_tb.course_id from course_tb;

	COURSE_ID
1	C01
2	C02
3	C03

Rule 3:The result is a super key of the second relation R2 so the third rule also applies here.

Rule 4: Dependencies in original relation: Student_Id -> {Student_Name}

Course_Id -> {Course_Detail}

Lossy Decomposition

In **lossy decomposition**, the information is lost during decomposition.

Let's take the same example

Student_Id	Student_Name	Course_Id	Course_Detail
S101	Chaitanya	C01	Maths
S102	Ajeet	C01	Maths
S103	Rahul	C02	Science
S104	Steve	C02	Science
S105	John	C03	English
S101	Chaitanya	C03	English
S102	Ajeet	C02	Science

Now if we divide this relation like this:

Student Table:

The primary key of this table is {Student_Id}

Student_Id	Student_Name
S101	Chaitanya
S102	Ajeet
S103	Rahul
S104	Steve
S105	John

Course Table:

The primary key of this table is {Course_Id}

Course_Id	Course_Detail
C01	Maths
C02	Science
C03	English

This is a **lossy decomposition** as the intersection of Student and Course relation will return null so the second and third rule of lossless decomposition will fail here.

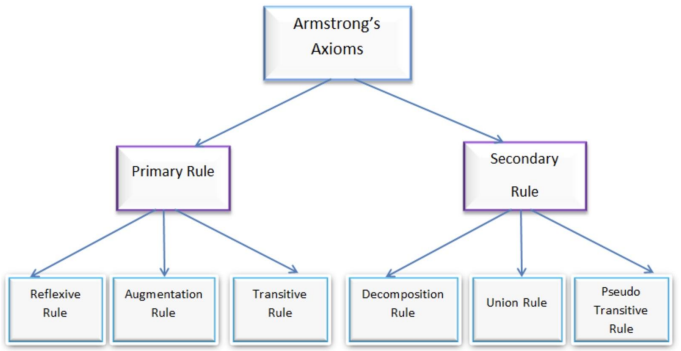
4.Armstrong's axioms

The term Armstrong axioms **refer to the sound and complete set of inference rules** or axioms, introduced by William W. Armstrong.

It is used to test the **logical implication of functional dependencies.**

If F is a set of functional dependencies then the **closure of F,** denoted as F^+ , **is the set of all functional dependencies logically implied by F.**

Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a **closure of functional dependencies**(the complete set of all possible attributes that can be functionally derived from given functional dependency).



1.Primary Rule:

1. Reflexive Rule
2. Augmentation Rule
3. Transitive Rule

1.Reflexive Rule

If A is a set of attributes and B is a subset of A, then A holds B. { A → B }

2.Augmentation

If A hold B and C is a set of attributes, then AC holds BC. {AC → BC}

It means that attribute in dependencies does not change the basic dependencies

3.Transitive

If A holds B and B holds C, then A holds C.

If {A → B} and {B → C}, then {A → C}

A holds B {A → B} means that A functionally determines B.

Example

Rno	Name	Marks	Dept	Course
1	A	78	CS	C1
2	B	60	EE	C1
3	A	78	CS	C2
4	B	60	EE	C3
5	C	80	IT	C3
6	D	80	EC	C2

1.Reflexive Rule
A={Rno,Name}
B={Name}
A—>B (True)

2.Augmentation
A={Rno},B={Marks},C={Name}
If A—>B,then AC—>BC (True)

Dept—>Course is not true,
But {Dept,Rno}—>{Course,Rno} (true)

3.Transitive
Rno—>Name,Name—>Dept
Then Rno—>Dept (True)

Secondary Rules

1.Union

If A holds B and A holds C, then A holds BC.

If{A → B} and {A → C}, then {A → BC}

2.Decomposition

If A holds BC and A holds B, then A holds C.

If{A → BC} and {A → B}, then {A → C}

3.Pseudo Transitivity

If A holds B and BC holds D, then AC holds D.

If{A → B} and {BC → D}, then {AC → D}

Example

Rno	Name	Marks	Dept	Course
1	A	78	CS	C1
2	B	60	EE	C1
3	A	78	CS	C2
4	B	60	EE	C3
5	C	80	IT	C3
6	D	80	EC	C2

Union
If Rno—>Name and Rno—>marks
Then Rno—>{Name ,marks}

Decomposition
If Rno—>{Name ,marks}
then
Rno—>Name and
Rno—>marks

Pseudo Transitivity

If Rno—>Name and {Name, Marks}—>Dept

Then { Rno,Marks}—>Dept

Closure of functional dependency

The Closure Of Functional Dependency means the **complete set of all possible attributes that can be functionally derived** from given functional dependency **using the inference rules known as Armstrong's Rules.**

If "F" is a functional dependency then closure of functional dependency can be denoted using "{F}+".

There are three steps to calculate closure of functional dependency. These are:

Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.

Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.

Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.

Example:consider

schema R = (A, B , C , G , H , I)

and the set of all functional dependencies

{A→ B, A→C,CG → H,CG → I, B → H}.

We list several members of F + here:

1.transitivity :A—>,B—>H Then A—>H

2.union:CG—>I,CG—>H Then CG—>HI

3.pseudo transitivity:A→C,CG → H Then AG—>H

F+:{A→ B, A→C,CG → H,CG → I, B → H, A—>H,CG—>HI,AG—>H}

Closure of attribute

Closure of an attribute x is the set of all attributes that are functional dependencies on X with respect to F. It is denoted by X+ which means what X can determine.

Using closure of attribute we can find out all candidate keys present in the relation.

Once you know candidate key we can easily solve problem on 2nd normal form,3rd normal form and BCNF.

Following steps are followed to find the closure of an attribute set-

Step-01:

Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step-02:

Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

Example

A → BC

BC → DE

D → F

CF → G

Closure of attribute A-

A+ = { A }

= { A , B , C } (Using A → BC)

= { A , B , C , D , E } (Using BC → DE)

= { A , B , C , D , E , F } (Using D → F)

= { A , B , C , D , E , F , G } (Using CF → G)

Thus,A+ = { A , B , C , D , E , F , G }

Closure of attribute D-

D+ = { D }

= { D , F } (Using D → F)

We can not determine any other attribute using attributes D and F cc in the result set.

Thus,D+ = { D , F }

Closure of attribute set {B, C}-

{ B , C }+= { B , C }

= { B , C , D , E } (Using BC → DE)

= { B , C , D , E , F } (Using D → F)

= { B , C , D , E , F , G } (Using CF → G)

Thus, { B , C }+ = { B , C , D , E , F , G }

Finding the Keys Using Closure-

Super Key-

If the **closure result of an attribute set contains all the attributes of the relation,** then that attribute set is called **as a super key of that relation**.

Thus, we can say-"The closure of a super key is the entire relation schema."

Example-

In the above example,The closure of attribute A is the entire relation schema.

Thus, attribute **A is a super key for that relation.**

Candidate Key-

If there **exists no subset of an attribute set whose closure contains all the attributes of the relation,** then that **attribute set is called as a candidate key of that relation.**

Example-

In the above example,No subset of attribute A contains all the attributes of the relation.

Thus, **attribute A is also a candidate key for that relation**.

Example:Find super key and candidate key for given relation

R(A,B,C,D) with Functional dependencies-{A—>B,B—>C,C—>A}

Possible set of super keys are:

ABCD+={A,B,C,D} (Super key)

Then we try to discard B because A can determine B.

ACD+={A,B,C,D} (Super key)

Then we try to discard C because A can determine C.

AD+={A,B,C,D} (Super key)

Then we will find A+ and D+ to obtain candidate key(because candidate key is proper subset of super key)

A+={A,B,C} It's not a super key

D+={D}. It's not a super key

Hence candidate key is AD

All possible set of super keys are:ABCD,ACD,AD

Consider the given functional dependencies-

AB → CD

AF → D

DE → F

C → G

F → E

G → A

Which of the following options is false?

(A) { CF }+ = { A , C , D , E , F , G }

(B) { BG }+ = { A , B , C , D , G }

(C) { AF }+ = { A , C , D , E , F , G }

(D) { AB }+ = { A , C , D , F ,G }

Introduction to process of Normalization and denormalization

1.What is Normalization?

2.Different Normal forms.

3.De-normalization

1.What is Normalization?

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

Data redundancy can be occur at row level or column level.

Anomalies in DBMS:It occurs at column level redundancy.

There are three types of anomalies that occur when the database is not normalized. These are:

1.Insertion anomaly

2.update anomaly and

3.deletion anomaly

1.Insertion anomaly: It occurs when we cannot insert data to the table without the presence of another attribute.

Example:insert Employee information without department details(here we can not make Emp_Dept as null).

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

2.Update anomaly: It is a data inconsistency that results from data redundancy and a partial update of data.

Example:employee Rick 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.

3.Deletion Anomaly: It occurs when certain attributes are lost because of the deletion of other attributes.

Example:Let's say in future, 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.
Database Normalization is a technique of organizing the data in the database.

Normalization is a systematic approach of **decomposing tables to eliminate data redundancy**(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

Eliminating redundant(useless) data.
Ensuring data dependencies make sense i.e data is logically stored.

Normalization Rule

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF

1.First Normal Form(1NF)

A relation will be 1NF if it **contains an atomic value.**

It states that an attribute of a table cannot hold multiple values.

It must hold only single-valued attribute.

This table is not in 1NF as,colour column contains multiple values.

Table_Product

Product Id	Colour	Price
1	Black, red	Rs.210
2	Green	Rs.150
3	Red	Rs. 110
4	Green, blue	Rs.260
5	Black	Rs.100

Decompose—>

Product_id	Price
1	Rs.210
2	Rs.150
3	Rs. 110
4	Rs.260
5	Rs.100

Product_id	Colour
1	Black
1	Red
2	Green
3	Red
4	Green
4	Blue
5	Black

2.Second Normal Form (2NF)

A table is said to be in 2NF if both the following condition holds:

1.Table is in 1NF.

2.No non-prime attribute is dependent on the proper subset of any candidate key of table(i.e partial dependency should not be there.)

A attribute that is not part of ant candidate key is known as non-prime attribute.

For this table

Candidate key ={Customer_id,Store_id}

This candidate key is also use as primary key.

So non-key attribute is Location.
But location depends on Store_id which is a part (subset) of candidate/primary key.

So table is not in 2NF.

Table purchase detail

Customer_id	Store_id	Location
1	1	Patna
1	3	Noida
2	1	Patna
3	2	Delhi
4	3	Noida

After Decompostion

Table Purchase

Customer_id	Store_id
1	1
1	3
2	1
3	2
4	3

Table Store

Store_id	Location
1	Patna
2	Delhi
3	Noida

Third Normal Form

A table is said to be in 3NF if both the following condition holds:

1.Table is in 2NF.

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

I.e for each functional dependency X→Y at least one of the condition hold:

1.X is a super key of table.

2.Y is a prime attribute of table.

Table Book Details			
Bood_id	Genre_id	Genre type	Price
1	1	Fiction	100
2	2	Sports	110
3	1	Fiction	120
4	3	Travel	130
5	2	sports	140

Here ,
Bood_id→Genre_id
Genre_id→Genre_type
So Bood_id→Genre_type

Hence Transitive functional dependency occurs

After Decompostion

TABLE BOOK		
Book_id	Genre_id	Price
1	1	100
2	2	110
3	1	120
4	3	130
5	2	140

TABLE GENRE	
Genre_id	Genre type
1	Fiction
2	Sports
3	Travel

4 Boyce Codd Normal Form(BCNF)

A table is said to be in BCNF if both the following condition holds:

1.Table is in 3NF.

2.For every functional dependency X→Y,X should be the super key of the table.

Student	Course	Teacher
Aman	DBMS	AYUSH
Aditya	DBMS	RAJ
Abhinav	E-COMM	RAHUL
Aman	E-COMM	RAHUL
abhinav	DBMS	RAJ

- KEY: {Student, Course}
- Functional dependency
  {student, course} -> Teacher
  Teacher-> Course
- Problem: teacher is not superkey but determines course.

Student	Course	Teacher
Aman	DBMS	AYUSH
Aditya	DBMS	RAJ
Abhinav	E-COMM	RAHUL
Aman	E-COMM	RAHUL
abhinav	DBMS	RAJ

After Decomposition

Student	Course
Aman	DBMS
Aditya	DBMS
Abhinav	E-COMM
Aman	E-COMM
Abhinav	DBMS

Course	Teacher
DBMS	AYUSH
DBMS	RAJ
E-COMM	RAHUL

Example:Normalise the following table.

Accountant Number	Skill Number	Skill Category	Proficiency	Accountant Name	Accountant Age	Group Number	Group City	Group Supervisor
21	113	Systems	3	Ali	55	52	ISD	Babar
35	113 179 204	Systems Tax Audit	5 1 6	Daud	32	44	LHR	Ghafoor
50	179	Tax	2	Kashif	40	44	LHR	Ghafoor
77	148 179	Consulting Tax	6 6	Zahid	52	52	ISD	Babar

In this table:
For Accountant Number 35 and 77,Skill Category is having mutlivalue.
Hence table is not in 1NF.

1.First Normal Form (1NF)

Attribute of a table cannot hold multiple values.

Accountant Number	Skill Number	Skill Category	Proficiency	Accountant Name	Accountant Age	Group Number	Group City	Group Supervisor
21	113	Systems	3	Ali	55	52	ISD	Babar
35	113	Systems	5	Daud	32	44	LHR	Ghafoor
35	179	Tax	1	Daud	32	44	LHR	Ghafoor
35	204	Audit	6	Daud	32	44	LHR	Ghafoor
50	179	Tax	2	Kashif	40	44	LHR	Ghafoor
77	148	Consulting	6	Zahid	52	52	ISD	Babar
77	179	Tax	6	Zahid	52	52	ISD	Babar

2.Second Normal Form(2NF)

Accountant Number	Skill Number	Skill Category	Proficiency	Accountant Name	Accountant Age	Group Number	Group City	Group Supervisor
21	113	Systems	3	Ali	55	52	ISD	Babar
35	113	Systems	5	Daud	32	44	LHR	Ghafoor
35	179	Tax	1	Daud	32	44	LHR	Ghafoor
35	204	Audit	6	Daud	32	44	LHR	Ghafoor
50	179	Tax	2	Kashif	40	44	LHR	Ghafoor
77	148	Consulting	6	Zahid	52	52	ISD	Babar
77	179	Tax	6	Zahid	52	52	ISD	Babar

For 2NF:
1.Table is in 1NF.
2.No non-prime attribute is dependent on the proper subset of any candidate key of table(i.e partial dependency should not be there.)

In the given table following are functional dependencies:

Accountant_Number->{Accountant_Name,Accountant_Age,Group_Number,Group_City,Group_Supervisor}

Skill_Number->Skill_Category

{Accountant_Number,Skill_Number}→Proficiency

{Accountant_Number,Skill_Number} is a super key as it is used to derive all attributes of table.

Here Prime attributes :Accountant_Number,Skill_Number

Non-prime attributes:Accountant_Name,Accountant_Age,Group_Number,Group_City,Group_Supervisor,Skill_Category,Proficiency

Following are partial dependencies:

Accountant_Number->{Accountant_Name,Accountant_Age,Group_Number,Group_City,Group_Supervisor}

Skill_Number->Skill_Category

So it is not in 2NF.

So we do decomposition.

Table1

Accountant Number	Skill Number	Proficiency
21	113	3
35	113	5
35	179	1
35	204	6
50	179	2
77	148	6
77	179	6

Table2

Accountant Number	Accountant Name	Accountant Age	Group Number	Group City	Group Supervisor
21	Ali	55	52	ISD	Babar
35	Daud	32	44	LHR	Ghafoor
50	Kashif	40	44	LHR	Ghafoor
77	Zahid	52	52	ISD	Babar

Table 3

Skill Number	Skill Category
113	Systems
179	Tax
204	Audit
148	Consulting

3.Third Normal Form (3NF)

1.Table is in 2NF.

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

Table2

Accountant Number	Accountant Name	Accountant Age	Group Number	Group City	Group Supervisor
21	Ali	55	52	ISD	Babar
35	Daud	32	44	LHR	Ghafoor
50	Kashif	40	44	LHR	Ghafoor
77	Zahid	52	52	ISD	Babar

Following is Transitive functional dependency

Accountant_Number→Group_Number

Group_Number→{Group_City,Group_Supervisor}

Following is Transitive functional dependency

Accountant_Number→Group_Number

Group_Number→{Group_City,Group_Supervisor}

Hence Decompose it in two tables.

Accountant Number	Accountant Name	Accountant Age	Group Number
21	Ali	55	52
35	Daud	32	44
50	Kashif	40	44
77	Zahid	52	52

Group Number	Group City	Group Supervisor
52	ISD	Babar
44	LHR	Ghafoor

4.Boyce Codd Normal Form(BCNF)

Accountant Number	Skill Number	Proficiency
21	113	3
35	113	5
35	179	1
35	204	6
50	179	2
77	148	6
77	179	6

Skill Number	Skill Category
113	Systems
179	Tax
204	Audit
148	Consulting

For every functional dependency X→Y,

X should be the super key of the table.

For Table1:Superkey {Accountant_Number,Skill_Number}-->

{Accountant_Number,Skill_Number,Proficiency}

Same way for remaining table also.

So all tables are in BCNF.

Accountant Number	Accountant Name	Accountant Age	Group Number
21	Ali	55	52
35	Daud	32	44
50	Kashif	40	44
77	Zahid	52	52

Group Number	Group City	Group Supervisor
52	ISD	Babar
44	LHR	Ghafoor

Denormalization

Denormalization is a database optimization technique where we add redundant data in the database to get rid of the complex join operations.

It is done after normalization for improving the performance of the database.

The data from one table is included in another table to **reduce the number of joins** in the query and hence helps in speeding up the performance.

Example:Suppose after normalization we have two tables first, **Student table** and second, **Branch table**. The student has the attributes as *Roll_no*, *Student-name*, *Age*, and *Branch_id*.

Student table

Roll_no	Student_name	Age	Branch_id
1	Andrew	18	10
2	Angel	19	10
3	Priya	20	10
4	Analisa	21	11
5	Anna	21	12

Branch table

Branch_id	Branch_name	HOD
10	CSE	Mr.abc
11	EC	Dr.xyz
12	EX	Dr.pqr

If we want the name of students along with the name of the branch name then we need to perform a join operation.

The problem here is that if the table is large we need a lot of time to perform the join operations.

So, we can add the data of *Branch_name* from Branch table to the Student table and this will help in reducing the time for join operation.

Pros of Denormalization:

1. Retrieving data is faster since we do fewer joins
2. Queries to retrieve can be simpler(and therefore less likely to have bugs), since we need to look at fewer tables.

Cons of Denormalization

The following are the disadvantages of denormalization:

- It takes large storage due to data redundancy.
- It makes it expensive to updates and inserts data in a table.
- It makes update and inserts code harder to write.
- Since data can be modified in several ways, it makes data inconsistent.

Transactions

1. What is Transactions?
2. Properties of transaction
3. Transaction states
4. Issues with concurrent executions
5. Schedules
6. Serializability- Conflict and View

1. What is Transactions?

A transaction is a set of operations used to perform a logical unit of work.

A permanent change occurs when a transaction is performed.

Operations in Transaction:

1.Read:Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
2.Write:Write operation is used to write the value back to the database from the buffer.
3.Commit:It is used to save the work done permanently.
4.Rollback:It is used to undo the work done.

Transaction Example

1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);

Properties of Transaction

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

1.Atomicity

2.Consistency

3.Isolation

4.Durability

1.Atomicity:

Either all operations of the transaction are reflected properly in the database, or none are.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B

T1	T2
Read(A)	Read(B)
A:= A-100	Y:= Y+100
Write(A)	Write(B)

- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B.
- This shows the inconsistent database state.
- to ensure correctness of database state, the transaction must be executed in entirely.
- Examples of such failures include power failures, hardware failures, and software errors

2.Consistency

Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

For example:

1.The total amount must be maintained before or after the transaction.

Total before T occurs = 600+300=900

Total after T occurs= 500+400=900

2.when T1 is completed but T2 fails, then inconsistency will occur.

3.Isolation:

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of concurrent transactions Ti and Tj

1. it appears to Ti that either Tj finished execution before Ti started, or
2. Tj started execution after Ti finished.

Example:

if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.

Hence we can say isolation converts parallel transaction into serial transactions.

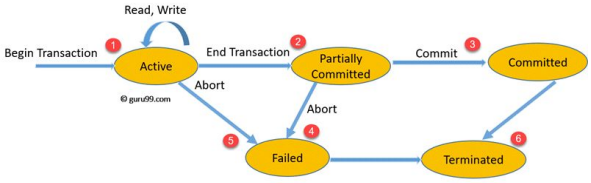
4.Durability

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

These updates now become permanent and are stored in non-volatile memory.

The effects of the transaction, thus, are never lost.

3.Transaction states



1.Active State –

When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

Partially Committed –

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to "committed state" and in case of failure it will go to the "failed state".

Failed State –

When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Data Base.

Aborted State –

After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

Committed State –

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the "terminated state".

Terminated State –

If there isn't any roll-back or the transaction comes from the "committed state", then the system is consistent and ready for new transaction and the old transaction is terminated.

Issues with concurrent executions

To increase the throughput and to reduce the waiting time, transactions are executed concurrently.

In a database transaction, the two main operations are READ and WRITE operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

1.Dirty Read Problem
2.Unrepeatable Read Problem
3.Lost Update Problem
4.Phantom Read Problem

1.Dirty Read Problem

Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

1.There is always a chance that the uncommitted transaction might roll back later.

2.Thus, uncommitted transaction might make other transactions read a value that does not even exist.

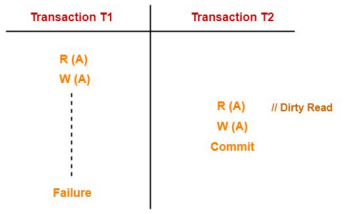
3.This leads to inconsistency of the database.

NOTE-

1.Dirty read does not lead to inconsistency always.

2.It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Example:



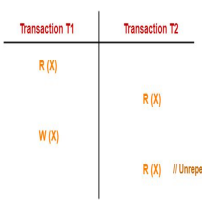
1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

T2 reads the dirty value of A written by the uncommitted transaction T1.
T1 fails in later stages and roll backs.
Thus, the value that T2 read now stands to be incorrect.
Therefore, database becomes inconsistent.

Unrepeatable Read Problem

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.



Here,

T1 reads the value of X (= 10 say).
T2 reads the value of X (= 10).
T1 updates the value of X (from 10 to 15 say) in the buffer.
T2 again reads the value of X (but = 15).

In this example,

T2 gets to read a different value of X in its second reading.
T2 wonders how the value of X got changed because according to it, it is running in isolation.

Lost Update Problem-

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Transaction T1	Transaction T2
R (A)	
W (A)	
	W (A)
	Commit
Commit	

Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.

When T1 commits, it writes A = 25 in the database.

In this example,

T2 writes the over written value of X in the database.
Thus, update from T1 gets lost.

Phantom Read Problem-

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

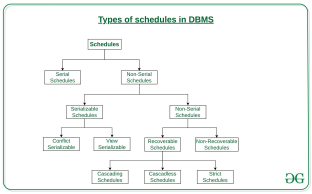
1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

T2 finds that there does not exist any variable X when it tries reading X again.
T2 wonders who deleted the variable X because according to it, it is running in isolation.

Schedules

It is a process of lining the transactions and executing them one by one(or we can say collection of transaction).
When there are multiple transactions that are running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other.
Scheduling is brought into play and the transactions are timed accordingly.



1.Serial Schedules:

Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules.

But it has waiting time problem.

2.Non-Serial Schedules:

In the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete.

As there is no waiting time ,It increases throughput(No. of transaction executed/unit of time) and improves performance of the system.

The Non-Serial Schedule can be divided further into

1. Serializable and

2.Non-Serializable.

1.Serializable

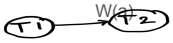
This is used to maintain the consistency of the database.

It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not

Example :

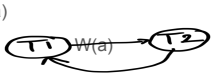
Serial schedule

T1	T2
R(a)	
W(a)	
	R(a)
	W(a)



parallel schedule

T1	T2
R(a)	
	R(a)
W(a)	



The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions.

A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

1.Conflict Serializability

2.View Serializability

1.Conflict Serializability

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Conflicting operations: Two operations are said to be conflicting if all conditions satisfy:

1. They belong to different transactions
2. They operate on the same data item
3. At Least one of them is a write operation

Conflict operations are:

1.(R1(A), W2(A)) because they belong to two different transactions on same data item A and one of them is write operation.

2.Similarly,(W1(A), W2(A)) and (W1(A), R2(A))

Non-Conflict operations are:

1.(R1(A), W2(B))

2.Similarly, ((W1(A), W2(B))

Example:

S1

T1	T2
R(a) W(a)	
	R(a) W(a)
R(b)	

S1 is conflict serializable to S2

S2

T1	T2
R(a) W(a) R(b)	
	R(a) W(a)

In S1 schedule all operations are non-conflict operations ,so we simply swap it to make transactions serializable

2.View Serializability

A schedule is said to be view serializable if it has an equivalent “view Equivalance schedule “.

View Equivalent Schedule:A schedule S’ is said to be view equivalent schedule if it satisfied following 3 conditions:

1.Initial Read

2.update Read

3.Final write operation

View equivalence conditions:

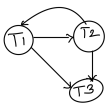
1) Initial Read

If a transaction T1 reading data item A from database in S1 then in S2 also T1 should read A from database.

2.update Read
If Ti is reading A which is updated by Tj in S1 then in S2 also Ti should read A which is updated by Tj.

3.Final Write operation
If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations.

T1	T2	T3
R(a)		
	W(a)	
W(a)		
		W(a)



This schedule creates a loop,hence it is a parallel transaction.So will check whether it is view serializable or not.

S1

T1	T2	T3
R(a) (a=100)		
	W(a) (a=a-40) 60	
W(a) (a=a-40)20		
		W(a) (a=a-20) 0

View Equivalence

S2

T1	T2	T3
R(a) (a=100) W(a) (a=a-40) 60		
	W(a) (a=a-40) 20	
		W(a) (a=a-20) 0

Introduction to Nosql

Topics to be covered

1. Overview of NoSQL
2. characteristics of NoSQL
3. Storage types of NoSQL
4. Implementing NoSQL in MongoDB -

    4.1Managing Databases and Collections from the MongoDB shell,

    4.2 Finding Documents in MongoDB collection from the MongoDB shell.

1.Overview of Nosql

1. What is NoSQL ?
2. Advantages of Nosql
3. Disadvantages of Nosql
4. A list of NoSQL databases

1.What is NoSQL ?

NoSQL originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data.

This data is modeled in means (document-based or key - value based) other than the tabular relations used in relational databases.

NoSQL databases are used in real-time web applications(online gaming,e-commerce applications) and big data (Amazon prime,healthcare applications) and their use are increasing over time.

It favours speed and flexibility over consistency and efficiency.

It uses following data model to organise the data:

1.Column-oriented

2.Document Store

3.Key Value Store

4.Graph

When to use Nosql database

- You don't know what kind of data will be stored in an application.
- Data may change frequently in an application.
- You need to develop an application as fast as possible.
- You can't relationally store data affordably.
- You need to be able to scale your database in size quickly.

Advantages of Nosql

1.Flexibility. 2.Scalability

3.High performance

4.Availability. 5.Highly Functional

1.Flexibility

With SQL databases, data is stored in a much more rigid, predefined structure.

But with NoSQL, data can be stored in a more free-form fashion without those rigid schemas.

Developers can focus on creating systems to better serve their customers without worrying about schemas.

NoSQL databases can easily handle any data format, such as structured, semi-structured, and non-structured data in a single data store.

2.Scalability

NoSQL databases are usually designed to expand by employing commodity hardware groups rather than expanding higher by introducing more servers.

3.High performance

NoSQL offers quicker, more flexible collection and execution for all users, including developers, sales teams, and customers.

NoSQL databases are built for great performance, measured in terms of both throughput (it is a measure of overall performance) and latency (it is the delay between request and actual response).

4.Availability

NoSQL databases automatically replicate data across multiple servers, data centers, or cloud resources.

In turn, this minimizes latency(delay) for users, no matter where they're located.

5.Highly Functional

NoSQL databases are designed for distributed data stores that have extremely large data storage needs. This is what makes NoSQL the ideal choice for big data, real-time web apps, customer 360, online shopping, online gaming, Internet of things, social networks, and online advertising applications.

6.Cost-effectiveness: NoSQL databases enable you to grow horizontally, effectively allocating resources and lowering expenses rapidly.

7.Replication: The replication feature of NoSQL replicates and saves data across numerous servers.

Disadvantages of Nosql

(i) Lack of Standardization:

There is no standard that defines rules and roles of NoSQL databases. The design and query languages of NoSQL databases vary widely between different NoSQL products – much more widely than they do among traditional SQL databases.

(ii) Backup of Database:

Backups are a drawback in NoSQL databases. Though some NoSQL databases like MongoDB provide some tools for backup, these tools are not mature enough to ensure proper complete data backup solution.

(iii) Consistency:

NoSQL puts a scalability and performance first but when it comes to a consistency of the data NoSQL doesn't take much consideration so it makes it little insecure as compared to the relational database e.g., in NoSQL databases if you enter same set of data again, it will take it without issuing any error whereas relational databases ensure that no duplicate rows get entry in databases.

List of Nosql databases

The following table is a list of some of the more mature, popular, and powerful NoSQL databases segregated by data model used:

Document	Key-Value	XML	Column	Graph
MongoDB	Redis	BaseX	BigTable	Neo4J
CouchDB	Membase	eXist	Hadoop / HBase	FlockDB
RavenDB	Voldemort		Cassandra	InfiniteGraph
Terrastore	MemcacheDB		SimpleDB	
			Cloudera	

Characteristics of Nosql

Following are the characteristics of Nosql:

1.Non-Relational

2.Schema-free

3.Simple API

4.Distributed

1.Non-Relational

It never follows the relational model.

It never provide table with flat fixed -column records.

It doesn't require object-relational mapping and data normalization.

No complex features like query languages,query planners and ACID properties.

2.Schema-free

NoSQL databases are independent of schemas which implies that they can be run over without any predetermined schemas.

3.Simple API

Offers easy to use interfaces for storage and querying data provided

APIs allow low-level data manipulation & selection methods

Text-based protocols mostly used with HTTP REST with JSON

Mostly used no standard based NoSQL query language

4.Distributed

NoSQL databases are designed for the distribution of data globally. This means that multiple locations, data centers, and cloud regions can be used by the NoSQL database for the read/write operations.

A key advantage of using NoSQL databases is that continuous availability can be maintained since the distribution of data is done with multiple data copies wherever needed.

Storage types of Nosql

Following are various storage types available in which the content can be modeled for NoSQL databases:

1.Key Value Store

2.Document Database

3.Column-oriented

4.Graph

1.Key Value Store

It is the simplest form of NoSQL database of all other types

The key-value database is a database that stores data in a schema-less manner.

This type of database stores data in the key-value format.

The values can be integer,string,complex data type such as sets of data.

The key in a key-value pair must be unique.

Data is retrieved via exact match on the key.

New types of data can easily be added to the database as new key - value pairs.

The three operations performed on key-value pairs are:

1.put(key,value): create new key-value pair

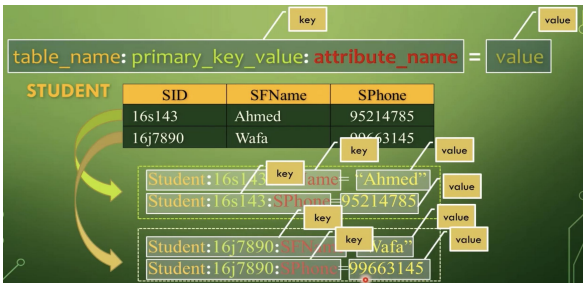
2.get(key):returns value of the given key.

3.delete (key):delete the value of key

Examples:Redis,Riak,Oracle Nosql

Key-value Database Schema

Table_name:primary_key_value:attribute_name=value



2.Document Database

Document database stores data in the form of documents.

This implies that data is grouped into files that make it easier to be recognized when it is required for building application software.

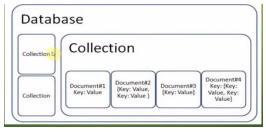
Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

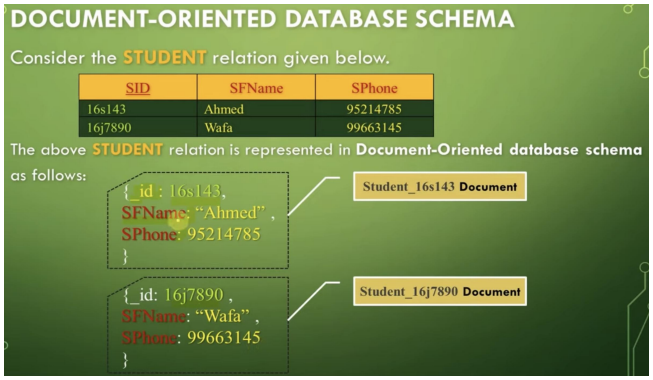
The document is stored in JSON or XML formats.

One of the major benefits of a document database is that it n

It is a semi-structured and hierarchical NoSQL database that allows efficient storage of data.

Examples:MongoDB, CouchDB, Cloudant





3.column store database

It stores data using a column-oriented model.

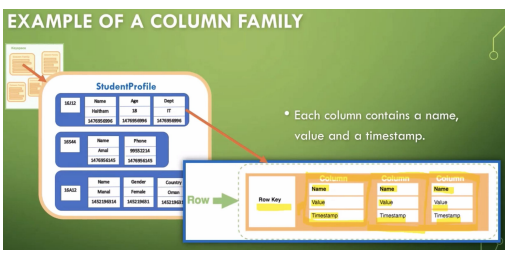
Data is stored in cells grouped in columns of data rather than as rows if data.

Examples:

BigTable

Cassandra

HBase



4.Graph Database

It organises data in the form of a graph.

A graph database contains a collection of nodes and edges.

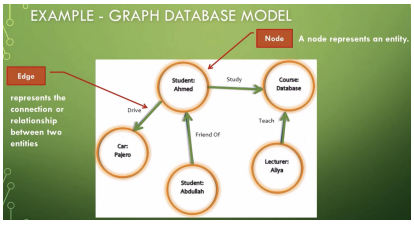
A node represents an entity ,and an edge represents the connection or relationship between two entities.

Examples:

1.Neo4J

2.OrientDB

3.ArantoDB



Relational Database	NoSQL
It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
It gives only read scalability.	It gives both read and write scalability.
It manages structured data.	It manages all type of data.
Data arrives from one or few locations.	Data arrives from many locations.
It supports complex transactions.	It supports simple transactions.
It has single point of failure.	No single point of failure.
It handles data in less volume.	It handles data in high volume.
Transactions written in one location.	Transactions written in many locations.
support ACID properties	doesn't support ACID properties
Its difficult to make changes in database once it is defined	Enables easy and frequent changes to database
schema is mandatory to store the data	schema design is not required