

# Database System Architecture Work

## Database System Architecture work

AU: May-12,13,14,16,17, Dec.- 08,15,17,19, Marks 16

- The typical structure of typical DBMS is based on relational data model as shown in Fig 1.5.1.

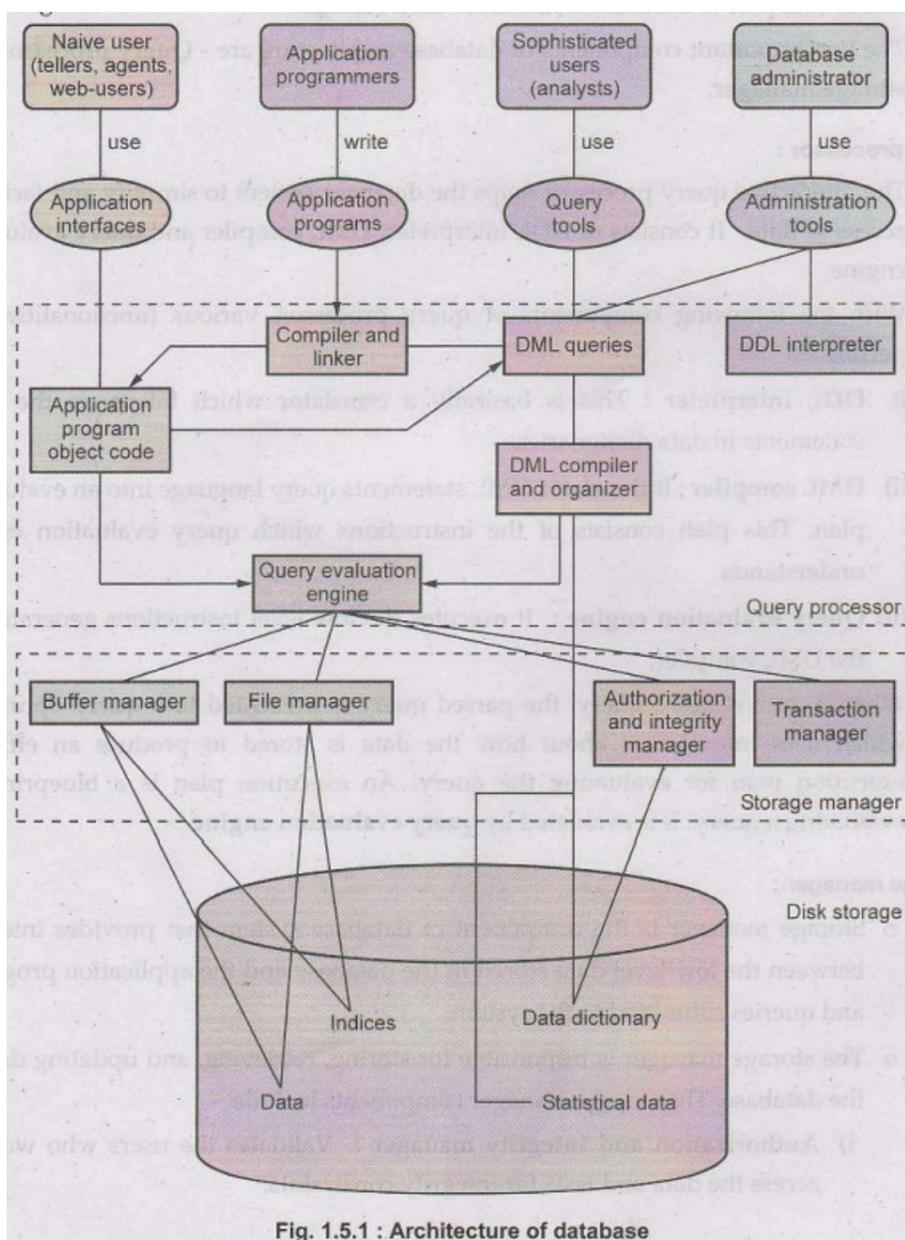


Fig. 1.5.1 : Architecture of database

- Consider the top part of Fig. 1.5.1. It shows application interfaces used by naïve users, application programs created by application programmers, query tools used by sophisticated users and administration tools used by database administrator.
- The lowest part of the architecture is for disk storage.
- The two important components of database architecture are - Query processor and storage manager.

### Query processor:

- The interactive query processor helps the database system to simplify and facilitate access to data. It consists of DDL interpreter, DML compiler and query evaluation engine.

- With the following components of query processor, various functionalities are performed -

**i) DDL interpreter:** This is basically a translator which interprets the DDL statements in data dictionaries.

**ii) DML compiler:** It translates DML statements query language into an evaluation plan. This plan consists of the instructions which query evaluation engine understands.

**iii) Query evaluation engine:** It executes the low-level instructions generated by the DML compiler.

- When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query. It is evaluated by query evaluation engine.

**Storage manager:**

- Storage manager is the component of database system that provides interface between the low level data stored in the database and the application programs and queries submitted to the system.

- The storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include -

**i) Authorization and integrity manager:** Validates the users who want to access the data and tests for integrity constraints.

**ii) Transaction manager:** Ensures that the database remains in consistent despite of system failures and concurrent transaction execution proceeds without conflicting.

**iii) File manager:** Manages allocation of space on disk storage and representation of the information on disk.

**iv) Buffer manager:** Manages the fetching of data from disk storage into main memory. The buffer manager also decides what data to cache in main memory. Buffer manager is a crucial part of database system.

- Storage manager implements several data structures such as -

**i) Data files:** Used for storing database itself.

**ii) Data dictionary:** Used for storing metadata, particularly schema of database.

**iii) Indices:** Indices are used to provide fast access to data items present in the database

### Review Questions

1. Explain the overall architecture of database system in detail. *AU: May-14,17, Dec.-17, Marks 8, May-16, Marks 16*

2. With the help of a neat block diagram explain basic architecture of a database management system. *AU May-12, May-13, Marks 16, Dec-15, Marks 8*

3. Explain component modules of a DBMS and their interactions with the architecture *AU: Dec.-08, Marks 10*

4. Sketch the typical component modules of DBMS. Indicate and explain interactions between those modules of the system. *AU: Dec.-19, Marks 7*

# Examples based on ER Diagram

## Examples based on ER Diagram

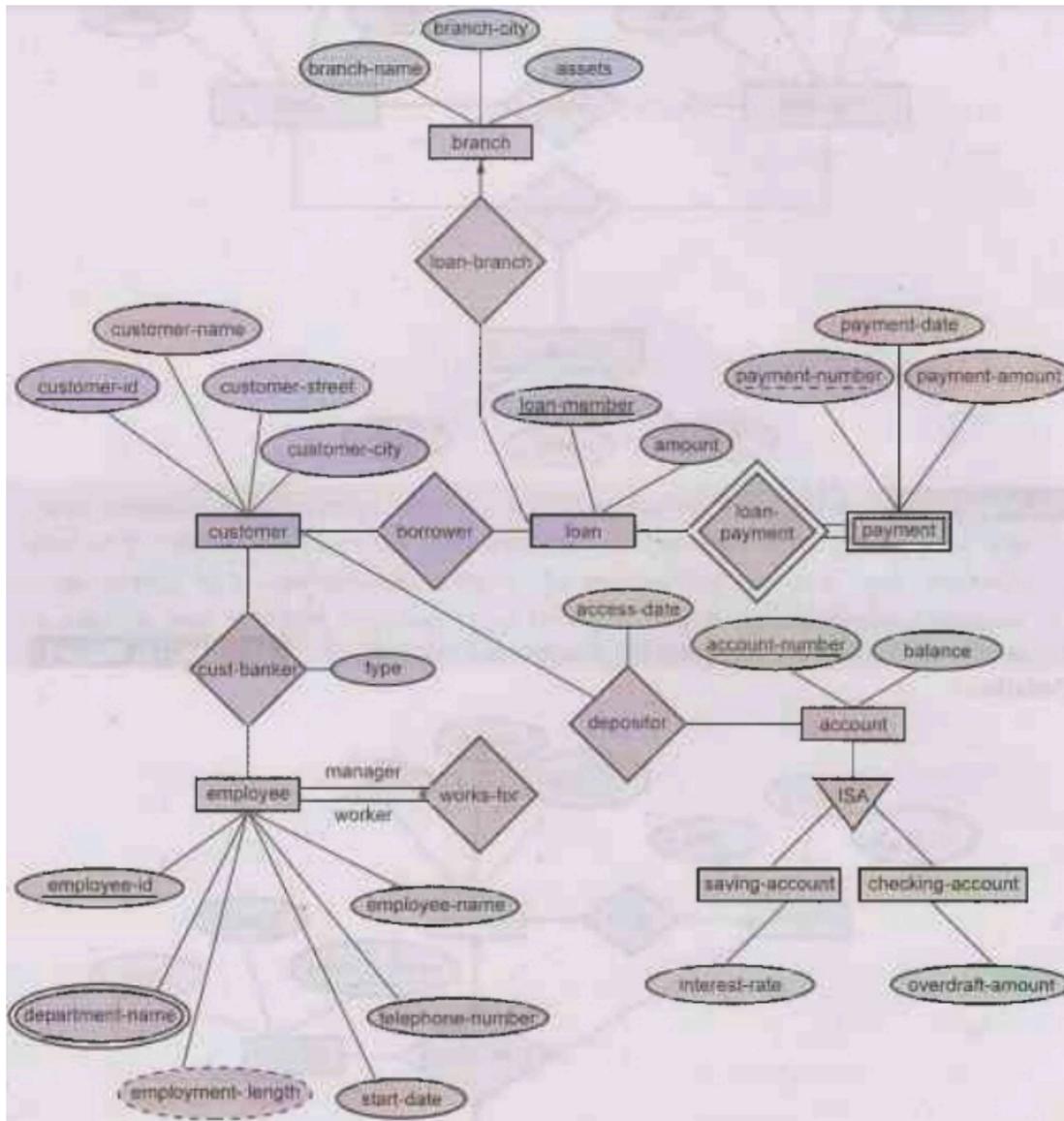
An E-R diagram can express the overall logical structure of a database graphically.

**Example 2.5.1** Draw the ER diagram for banking systems (home loan applications). AU: Dec.-17, Marks 8

OR Draw an ER diagram corresponding to customers and loans. AU: May.-14, Marks 8

OR Write short notes on: E-R diagram for banking system. AU: Dec.-14, Marks 8

**Solution:**



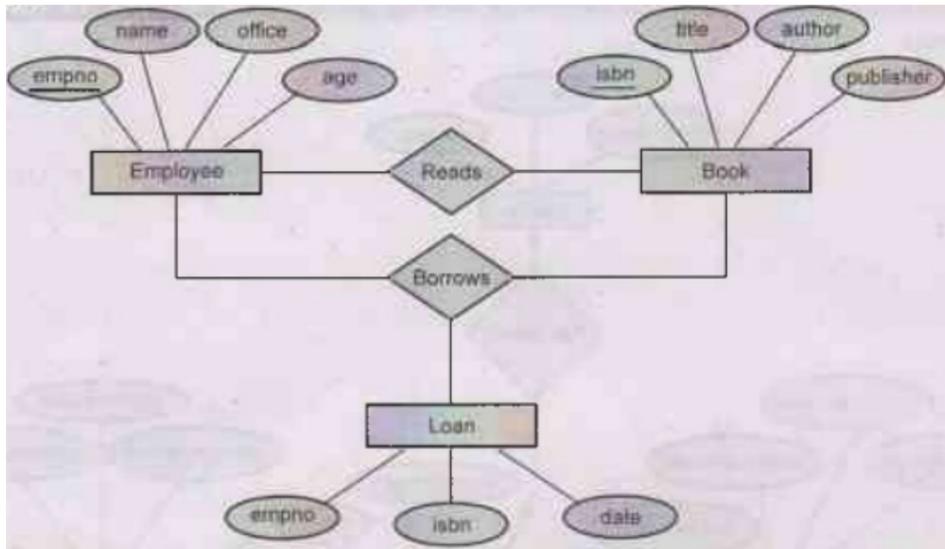
**Example 2.5.2** Consider the relation schema given in Figure. Design and draw an ER diagram *that capture the information of this schema.* AU: May-17, Marks 5

*Employee(empno,name,office,age)*

*Books(isbn,title,authors,publisher)*

*Loan(empno,isbn,date)*

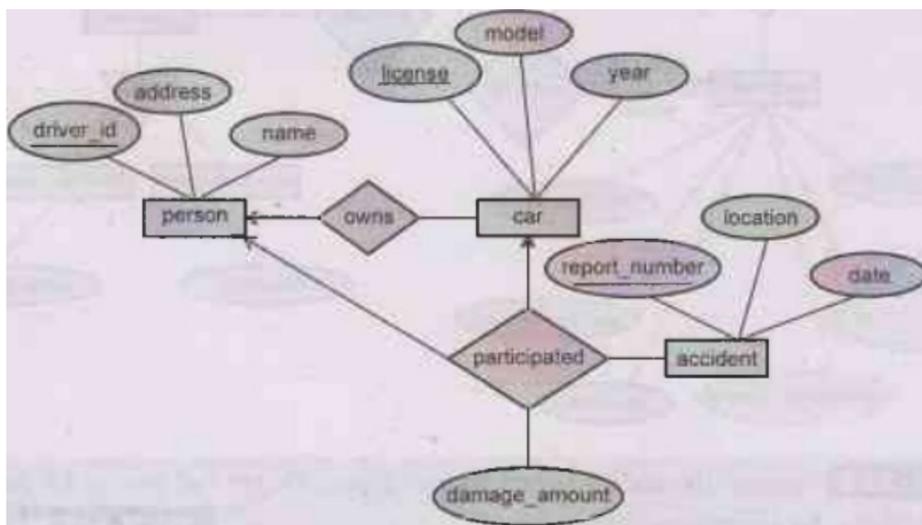
**Solution:**



**Example 2.5.3** Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Each insurance policy covers one or more cars and has one or more premium payments associated with it. Each payment is for particular period of time and has an associated due date and date when the payment was received

AU: Dec.-16, Marks 7

**Solution:**



**Example 2.5.4** A car rental company maintains a database for all vehicles in its current fleet. For all vehicles, it includes the vehicle identification number license number; manufacturer; model, date of purchase and color. Special data are included for certain types of vehicles.

*Trucks: Cargo capacity*

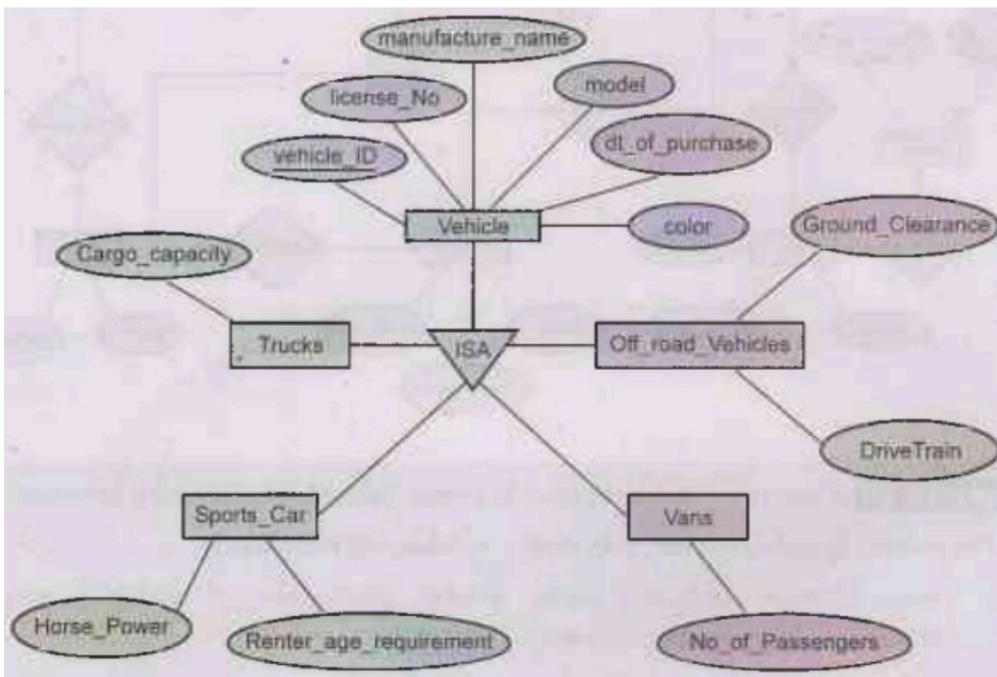
*Sports cars: horsepower, renter age requirement*

*Vans: number of passengers*

*Off-road vehicles: ground clearance, drivetrain (four-or two-wheel drive)*

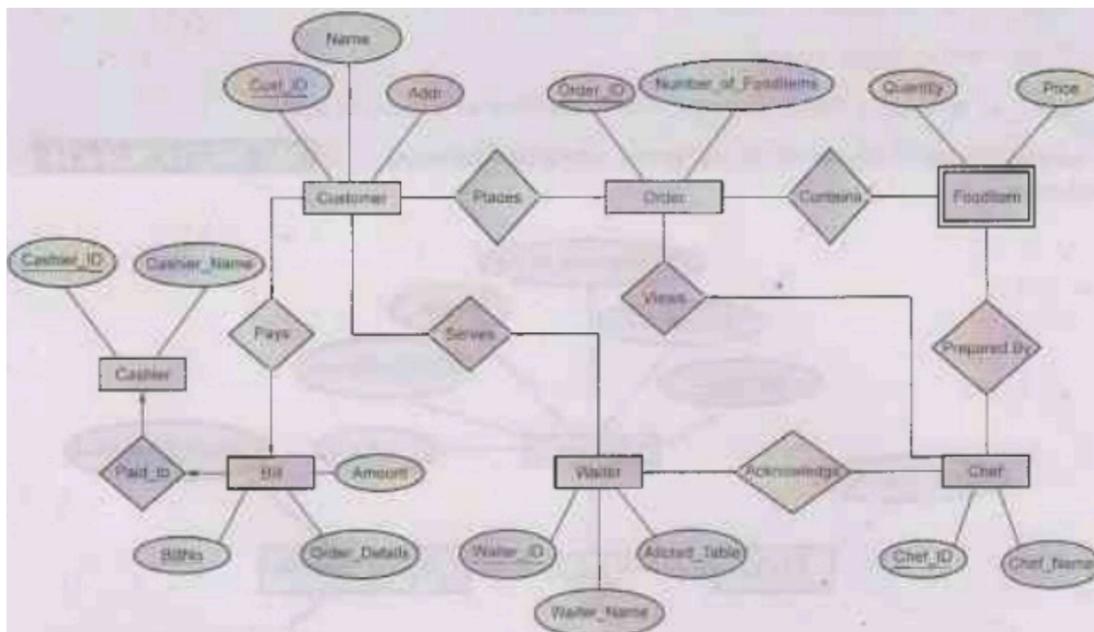
Construct an ER model for the car rental company database." AU: Dec.-15, Marks 16

**Solution:**



**Example 2.5.5** Draw E-R diagram for the "Restaurant Menu Ordering System", which will facilitate the food items ordering and services within a restaurant. The entire restaurant scenario is detailed as follows. The customer is able to view the food items menu, call the waiter, place orders and obtain the final bill through the computer kept in their table. The Waiters through their wireless tablet PC are able to initialize a table for customers, control the table functions to assist customers, orders, send orders to food preparation staff (chef) and finalize the customer's bill. The Food preparation staffs (chefs), with their touch-display interfaces to the system, are able to view orders sent to the kitchen by waiters. During preparation they are able to let the waiter know the status of each item, and can send notifications when items are completed. The system should have full accountability and logging facilities, and should support supervisor actions to account for exceptional circumstances, such as a meal being refunded or walked out on.

**Solution:**

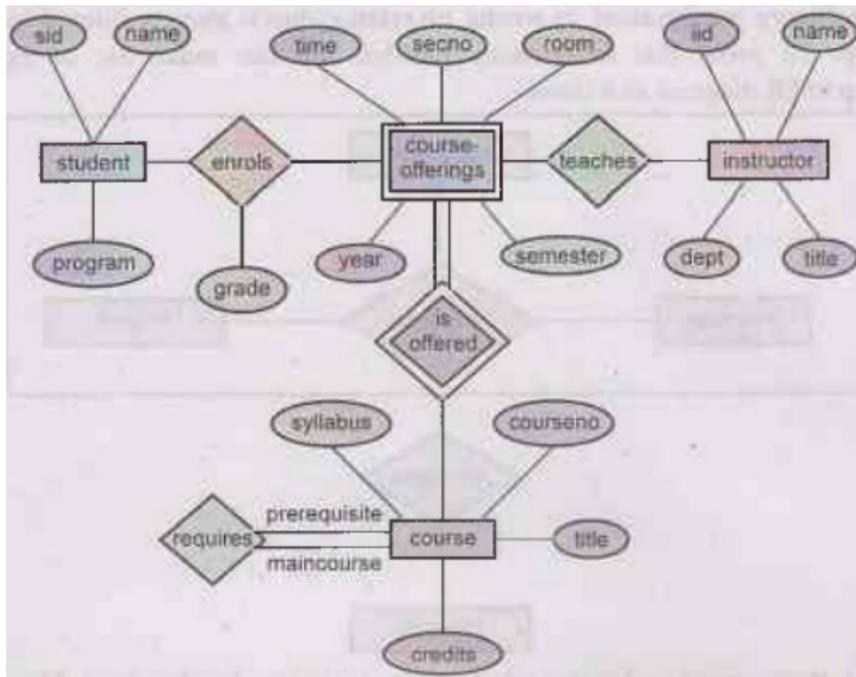


**Example 2.5.6** A university registrar's office maintains data about the following entities:

- (1) courses, including number, title, credits, syllabus, and prerequisites;
- (2) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;
- (3) students, including student-id, name, and program;
- (4) instructors, including identification number, name, department, and title.

Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints. **AU: Dec.-13, Marks 10**

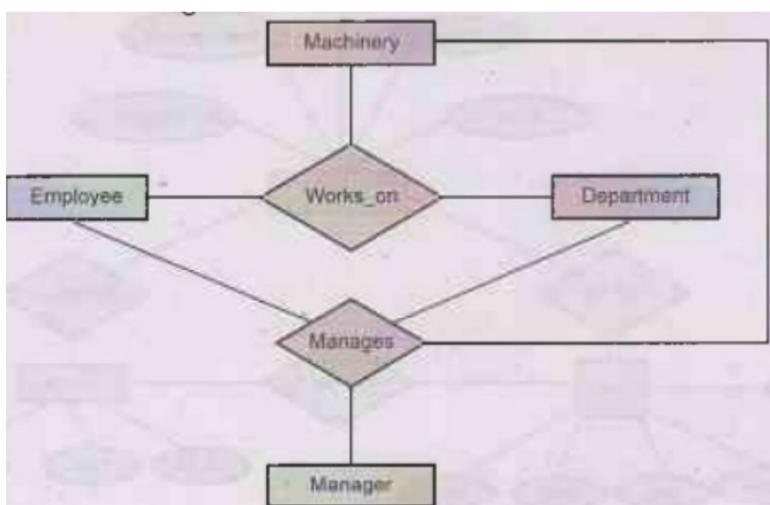
**Solution:**



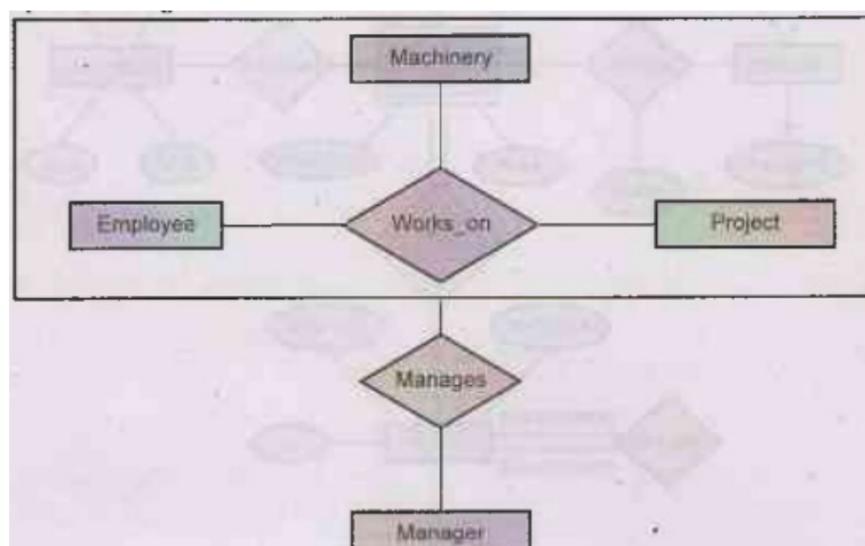
**Example 2.5.7** What is aggregation in ER model? Develop an ER diagram using aggregation that captures following information: Employees work for projects. An employee working for particular project uses various machinery. Assume necessary attributes. State any assumptions you make. Also discuss about the ER diagram you have designed. **AU: Dec.-11, Marks 8**

**Solution Aggregation: Refer section 2.4.3.**

**ER Diagram:** The ER diagram for above described scenario can be drawn as follows-



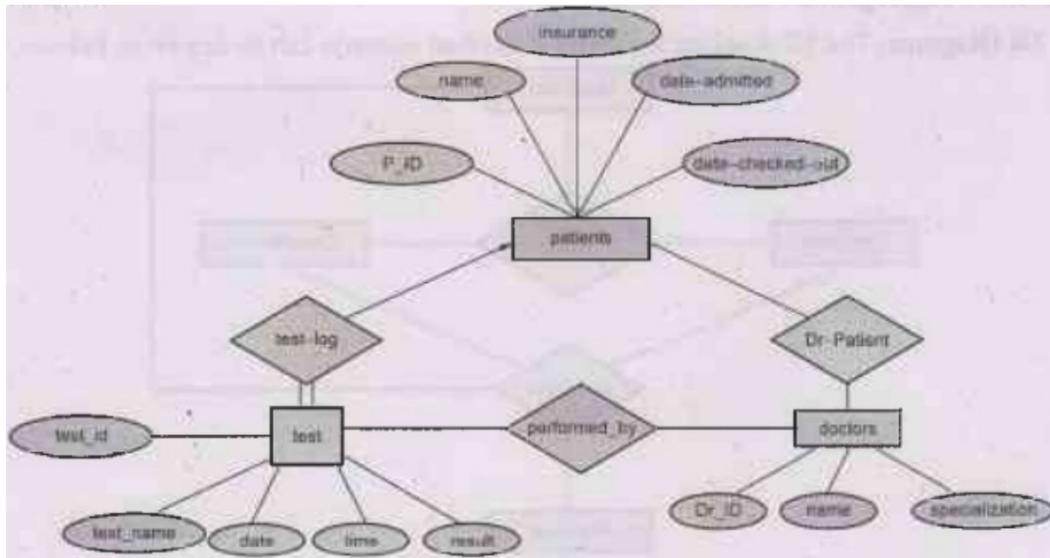
The above ER model contains the redundant information, because every Employee, Project, Machinery combination in works\_on relationship is also considered in manages relationship. To avoid this redundancy problem we can make use of aggregation relationship in ER diagram as follows -



We can then create a binary relationship manages for between Manager and (Employee, Project, Machinery).

**Example 2.5.8** Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. AU: Dec.-07, Marks 8

**Solution:**



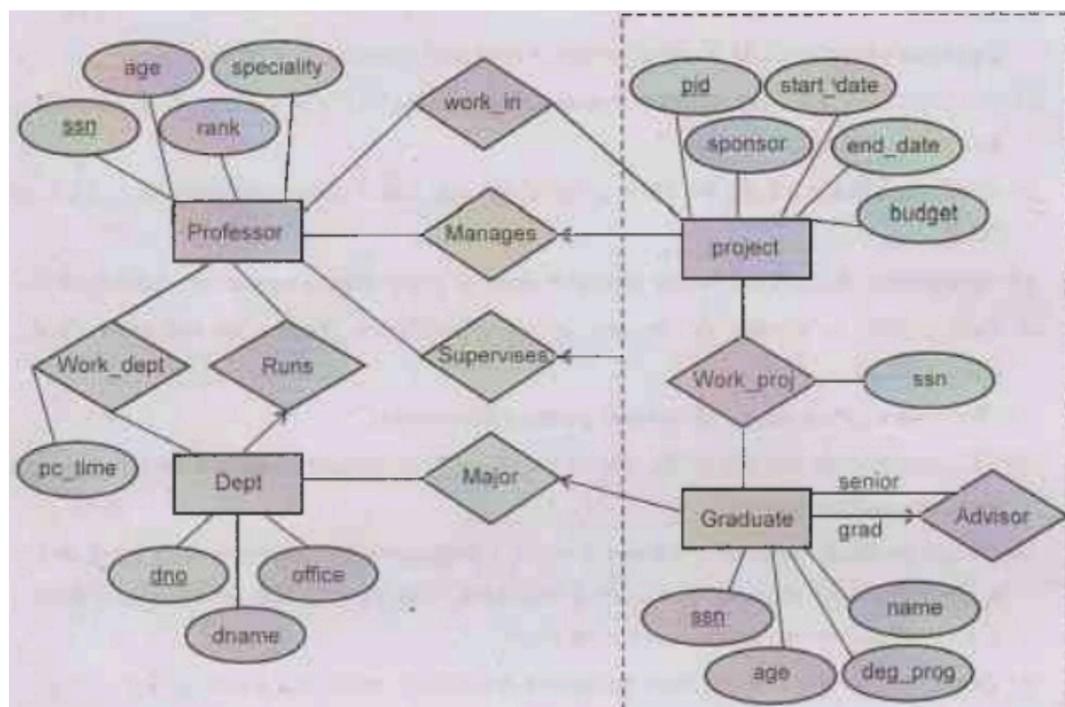
**Example 2.5.9** Consider the following information about a university database:

- i) Professors have an SSN, a name, an age, a rank and a research specialty.
- ii) Projects have a project number, a sponsor name, (e.g. NSF), a starting date, an ending date and a budget.
- iii) Graduate students have an SSN, a name, an age and a degree program (e.g. M.S. or Ph.D.).
- iv) Each project is managed by one professor (known as the project's principal investigators)
- v) Each project is worked on by one or more professors (known as the project's co- investigators).
- vi) Professors can manage and/or work on multiple projects.
- vii) Each project is worked on by one or more graduate students (known as the project's research assistants).
- viii) When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.
- ix) Departments have a department number, a department name and a main office.
- x) Departments have a professor (known as the chairman) who runs the department.
- xi) Professors work in one or more departments and for each department that they work in, a time percentage is associated with their job,
- xii) Graduate students have one major department in which they are working on their degree.
- xii) Each graduate student has another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.
- xiii) Design and draw an ER diagram that captures the information about the university.

Use only the basic ER model here; that is entities, relationship and attributes.

Be sure to indicate any key and participation constraints.

**Solution:**



## ER to Relational Mapping

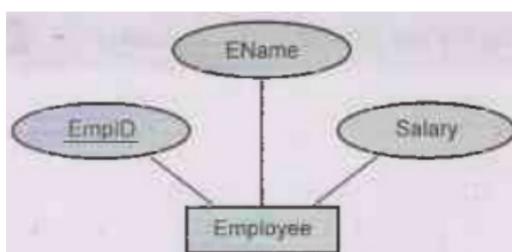
### ER to Relational Mapping

AU: May-17, Dec.-19, Marks 13

In this section we will discuss how to map various ER model constructs to Relational Model construct.

#### Mapping of Entity Set to Relationship

- An entity set is mapped to a relation in a straightforward way.
- Each attribute of entity set becomes an attribute of the table.
- The primary key attribute of entity set becomes an entity of the table.
- For example - Consider following ER diagram.



The converted employee table is as follows –

EmpID	EName	Salary
201	Poonam	30000
202	Ashwini	35000
203	Sharda	40000

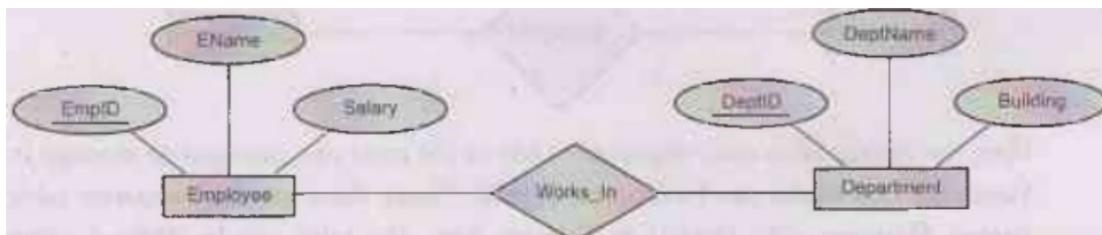
The SQL statement captures the information for above ER diagram as follows -

```
CREATE TABLE Employee( EmpID CHAR(11),
EName CHAR(30),
Salary INTEGER,
PRIMARY KEY(EmpID))
```

### Mapping Relationship Sets(Without Constraints) to Tables

- Create a table for the relationship set.
- Add all primary keys of the participating entity sets as fields of the table.
- Add a field for each attribute of the relationship.
- Declare a primary key using all key fields from the entity sets.
- Declare foreign key constraints for all these fields from the entity sets.

For example - Consider following ER model



The SQL statement captures the information for relationship present in above ER diagram as follows -

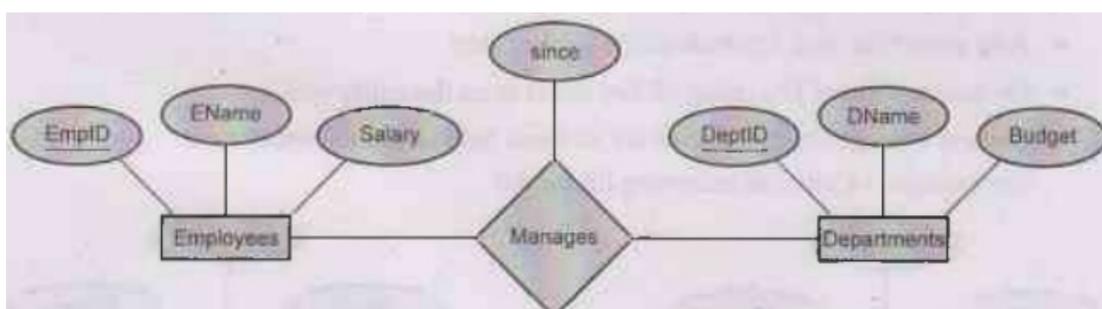
```
CREATE TABLE Works In (EmpID CHAR(11),
DeptID CHAR(11),
EName CHAR(30),
Salary INTEGER,
DeptName CHAR(20),
Building CHAR(10),
PRIMARY KEY(EmpID,DeptID),
FOREIGN KEY (EmpID) REFERENCES Employee,
FOREIGN KEY (DeptID) REFERENCES Department
)
```

### Mapping Relationship Sets( With Constraints) to Tables

- If a relationship set involves n entity sets and some m of them are linked via arrows in the ER diagram, the key for anyone of these m entity sets constitutes a key for the relation to which the relationship set is mapped.
- Hence we have m candidate keys, and one of these should be designated as the primary key.
- There are two approaches used to convert a relationship sets with key constraints into table.

#### • Approach 1:

• By this approach the relationship associated with more than one entities is separately represented using a table. For example - Consider following ER diagram. Each Dept has at most one manager, according to the key constraint on Manages.



Here the constraint is each department has at the most one manager to manage it. Hence no two tuples can have same DeptID. Hence there can be a separate table named Manages with DeptID as Primary Key. The table can be defined using following SQL statement

```
CREATE TABLE Manages (EmpID CHAR(11),
DeptID INTEGER,
Since DATE,
PRIMARY KEY (DeptID),
```

*FOREIGN KEY (EmpID) REFERENCES Employees,*  
*FOREIGN KEY (DeptID) REFERENCES Departments)*

### **Approach 2:**

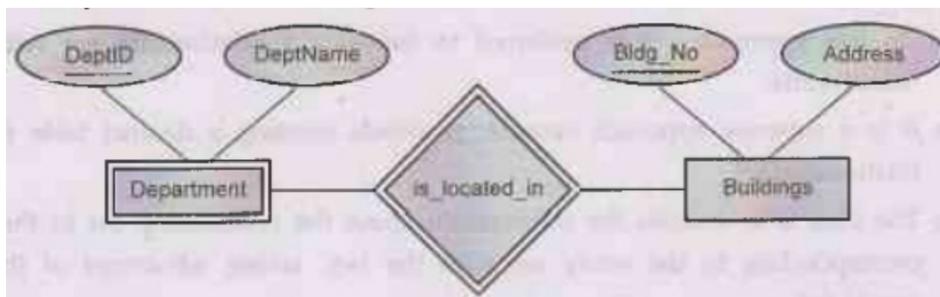
- In this approach, it is preferred to translate a relationship set with key constraints.
- It is a superior approach because, it avoids creating a distinct table for the relationship set.
- The idea is to include the information about the relationship set in the table corresponding to the entity set with the key, taking advantage of the key constraint.
- This approach eliminates the need for a separate Manages relation, and queries asking for a department's manager can be answered without combining information from two relations.
- The only drawback to this approach is that space could be wasted if several departments have no managers.
- The following SQL statement, defining a Dep\_Mgr relation that captures the information in both Departments and Manages, illustrates the second approach to translating relationship sets with key constraints:

```
CREATE TABLE Dep_Mgr (DeptID INTEGER,  
DName CHAR(20),  
Budget REAL,  
EmpID CHAR (11),  
since DATE,  
PRIMARY KEY (DeptID),  
FOREIGN KEY (EmpID) REFERENCES Employees)
```

### **Mapping Weak Entity Sets to Relational Mapping**

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity. Following steps are used for mapping Weak Entity Set to Relational Mapping
- Create a table for the weak entity set.
- Make each attribute of the weak entity set a field of the table. AI baris M
- Add fields for the primary key attributes of the identifying owner.
- Declare a foreign key constraint on these identifying owner fields.
- Instruct the system to automatically delete any tuples in the table for which there are no owners

For example - Consider following ER model,

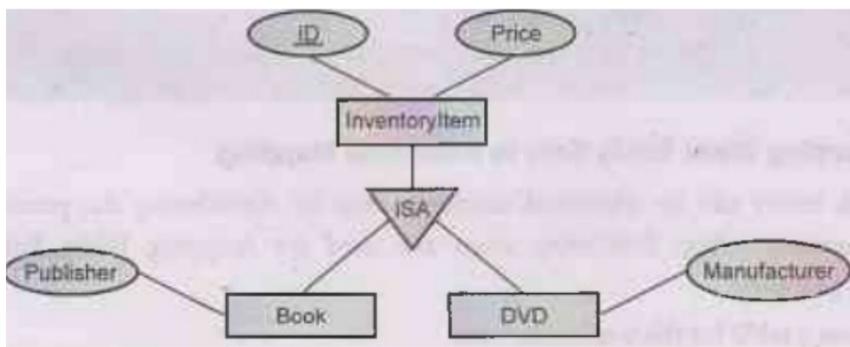


Following SQL Statement illustrates this mapping

```
CREATE TABLE Department (DeptID CHAR(11),  
DeptName CHAR(20),  
Bldg No CHAR(5),  
PRIMARY KEY (DeptID,Bldg_No),  
FOREIGN KEY(Bldg_No) References Buildings on delete cascade  
)
```

### **Mapping of Specialization / Generalization (EER Construct)to Relational Mapping**

The specialization/Generalization relationship (Enhanced ER Construct) can be mapped to database tables(relations) using three methods. To demonstrate the methods, we will take the - InventoryItem, Book, DVD



**Method 1:** All the entities in the relationship are mapped to individual tables

*InventoryItem*(ID, name)

*Book*(ID, Publisher)

*DVD*(ID, Manufacturer)

**Method 2:** Only subclasses are mapped to tables. The attributes in the superclass are duplicated in all subclasses. For example -

*Book*(ID, name, Publisher)

*DVD*(ID, name, Manufacturer)

**Method 3:** Only the superclass is mapped to a table. The attributes in the subclasses are taken to the superclass. For example -

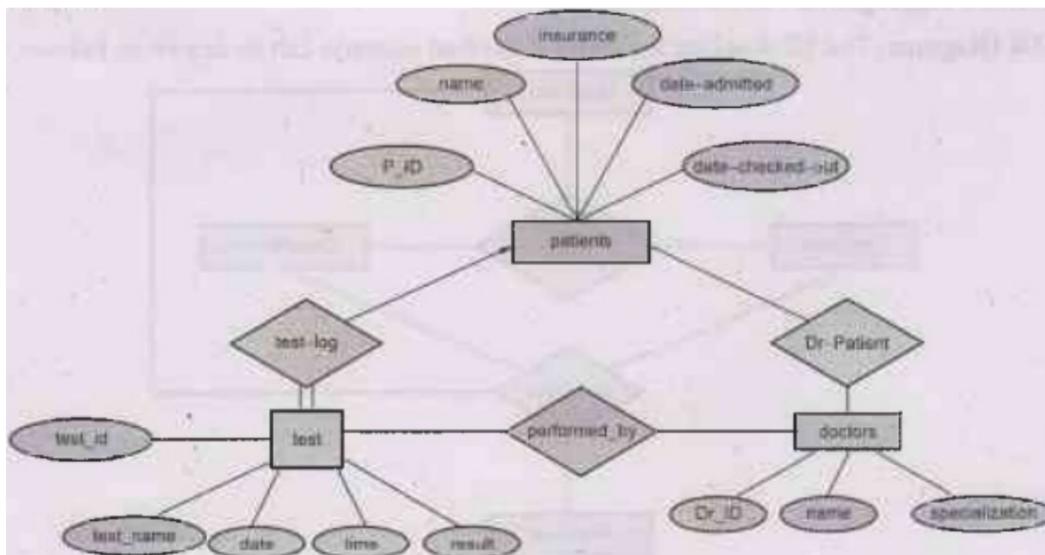
*InventoryItem*(ID, name, Publisher, Manufacturer)

This method will introduce null values. When we insert a Book record in the table, the Manufacturer column value will be null. In the same way, when we insert a DVD record in the table, the Publisher value will be null.

**Example 2.6.1** Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. Also construct appropriate tables for the ER diagram you have drawn.

**Solution:**

ER Diagram - Refer example 2.5.8.



### Relational Mapping

*Patients* (P\_id, name, insurance, date-admitted, date-checked-out)

*Doctors* (Dr\_id, name, specialization)

*Test* (testid, testname, date, time, result)

*doctor-patient* (P\_id, Dr\_id)

*test-log* (testid, P\_id) *performed-by* (testid, Dr\_id)

### Review Questions

1. Discuss the correspondence between the ER model construct and the relational model constructs. Show how each ER model construct can be mapped to the relational model. Discuss the option for mapping EER construct. **AU: May-17, Marks 13 2.**

2. Discuss in detail the steps involved in the ER to relational mapping in the process of relational database design.

# Enhanced ER Model

## Enhanced ER Model

### Specialization and Generalization AU: Dec- 19, Marks 7

- Some entities have relationships that form hierarchies. For instance, Employee can be an hourly employee or contracted employee.
- In this relationship hierarchies, some entities can act as superclass and some other entities can act as subclass.
- **Superclass:** An entity type that represents a general concept at a high level, is called superclass.
- **Subclass:** An entity type that represents a specific concept at lower levels, is called subclass.
- The subclass is said to inherit from superclass. When a subclass inherits from one or more superclasses, it inherits all their attributes. In addition to the inherited attributes, a subclass can also define its own specific attributes.
- The process of making subclasses from a general concept is called specialization. This is top-down process. In this process, the sub-groups are identified within an entity set which have attributes that are not shared by all entities.
- The process of making superclass from subclasses is called generalization. This is a bottom up process. In this process multiple sets are synthesized into high level entities.
- The symbol used for specialization/ Generalization is,



- **For example** - There can be two subclass entities namely Hourly\_Emps and Contract\_Emps which are subclasses of Employee class. We might have attributes hours\_worked and hourly wage defined for Hourly\_Emps and an attribute contractid defined for ContractEmps.

Therefore, the attributes defined for an Hourly\_Emps entity are the attributes for Employees plus Hourly\_Emps. We say that the attributes for the entity set Employees are inherited by the entity set Hourly\_Emps and that Hourly-Emps ISA (read is a) Employees. It can be represented by following **Fig. 2.4.1**.

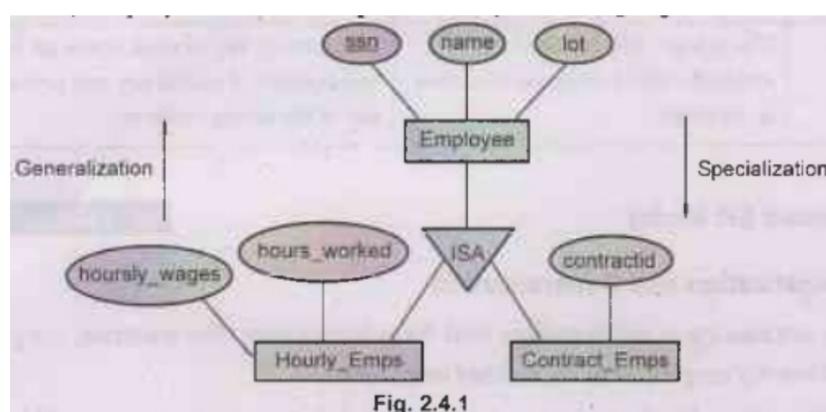


Fig. 2.4.1

### Constraints on Specialization/Generalization

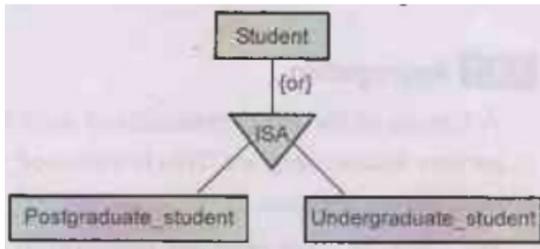
There are four types of constraints on specialization/generalization relationship. These are -

**1) Membership constraints:** This is a kind of constraints that involves determining which entities can be members of a given lower-level entity. There are two types of TO S membership constraints –

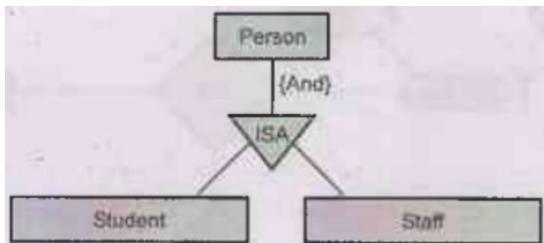
**i) Condition defined:** In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. For example - Consider the high-level entity Set Employee that has attribute Employee\_type. All Employee entities are evaluated on defining Employee\_type attribute. All entities that satisfy the condition student type = "ContractEmployee" are included in Contracted Employee. Since all the lower-level entities are evaluated on the basis of the same attribute this type of generalization is said to be attribute-defined.

ii) **User defined:** This is kind of entity set that in which the membership is manually defined.

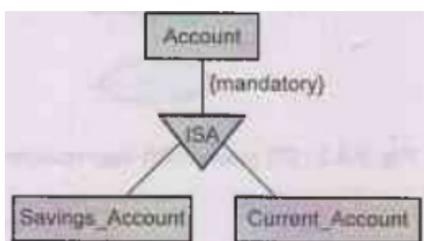
2) **Disjoint constraints:** The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are disjoint, then an entity occurrence can be a member of only one of the subclasses. For entity Student has either Postgraduate Student entity or Undergraduate Student



3) **Overlapping:** When some entity can be a member of more than one subclasses. For example - Person can be both a Student or a Staff. The And can be used to represent this constraint.

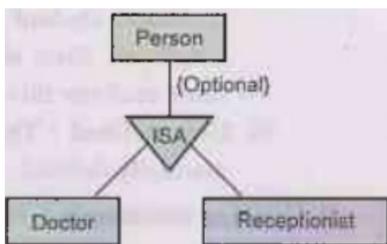


4) **Completeness:** It specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following –



i) **Total generalization or specialization:** Each higher-level entity must belong to a lower-level entity set. For example - Account in the bank must either Savings account or Current Account. The mandatory can be used to represent this constraint.

ii) **Partial generalization or specialization:** Some higher-level entities may not belong to any lower-level entity set.



## Aggregation

A feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box around the aggregation.

For example - We treat the relationship set work and the entity sets employee and project as a higher-level entity set called work.

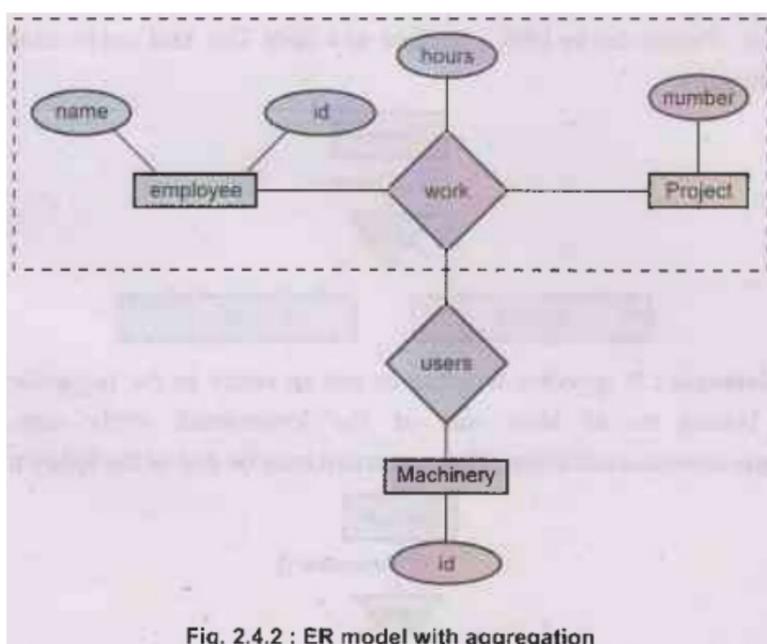


Fig. 2.4.2 : ER model with aggregation

## Review Question

1 Explain with suitable example the constraints of specialization and generalization in ER modeling AU: Dec. 19. Marks

# SQL Fundamentals

## Part III: Structured Query Language

### SQL Fundamentals

AU: Dec.-14,15,17,19, May-15,16,17,18, Marks 15

- Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS.
- Various parts of SQL are -
  - **Data Definition Language(DDL):** It consists of a set of commands for defining relation schema, deleting relations, and modifying relation schemas.
  - **Data Manipulation Language(DML):** It consists of set of SQL commands for inserting tuples into relational schema, deleting tuples from or modifying tuples in databases.
  - **Integrity:** The SQL DDL includes commands for specifying integrity constraints. These constraints must be satisfied by the databases.
  - **View definition:** The SQL DDL contains the commands for defining views for database.
  - **Transaction control:** The SQL also includes the set of commands that indicate beginning and ending of the transactions.
  - **Embedded SQL and Dynamic SQL:** There is a facility of including SQL commands in the programming languages like C,C++, COBOL or Java.
  - **Authorization:** The SQL DDL includes the commands for specifying access rights to relations and views.

### Data Abstraction

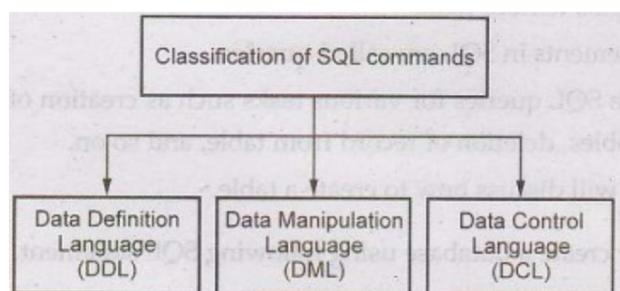
The Basic data types used in SQL are -

- (1) **char(n):** For representing the fixed length character string this data type is used. For instance to represent name, designation, coursename, we use this data type. Instead of char we can also use character. The n is specified by the user.
- (2) **varchar(n):** The varchar means character varying. That means - for denoting the variable length character strings this data type is used. The n is user specified maximum character length.
- (3) **int:** For representing the numeric values without precision, the int data type is used.
- (4) **numeric:** For representing, a fixed point number with user-specified precision this data type is used. The number consists of m digits plus sign k digits are to the right of precision. For instance the numeric(3,2) allows 333.11 but it does not allow 3333.11
- (5) **smallint:** It is used to store small integer value. It allows machine dependent subset of integer type.
- (6) **real:** It allows the floating point, double precision numbers.
- (7) **float(n):** For representing the floating point number with precision of at least n digits this data type is used.

### Basic Schema Definition

In this section, we will discuss various SQL commands for creating the schema definition.

There are three categories of SQL commands.



### Data Definition Language

Sr.No.	Command	Purpose
1.	CREATE	This command is used to create database, tables, views or any other database objects.
2.	ALTER	It modifies the existing tables.
3.	DROP	This command deletes complete table, view or any other database object.

### Data Manipulation Language

Sr. No.	Command	Purpose
1.	SELECT	This command is used to retrieve either all or desired records from one or more tables.
2.	INSERT	For inserting the records in the table, this command is used.
3.	UPDATE	For updating one or more fields of the table, this command is used.
4.	DELETE	This command is used for deleting the desired record.

### Data Control Language

Sr. No.	Command	Purpose
1.	GRANT	This command is used to give access rights or privileges to the database.
2.	INVOKE	The revoke command removes user access rights or privileges to the database objects.

## 1. Creation

- A database can be considered as a container for tables and a table is a grid with rows and columns to hold data.
- Individual statements in SQL are called queries.
- We can execute SQL queries for various tasks such as creation of tables, insertion of data into the tables, deletion of record from table, and so on.

In this section we will discuss how to create a table.

**Step 1:** We normally create a database using following SQL statement..

#### Syntax

```
CREATE DATABASE database_name;
```

#### Example

```
CREATE DATABASE Person_DB
```

**Step 2:** The table can be created inside the database as follows -

```
CREATE TABLE table name (
```

```
Coll_name datatype,
```

```
col2_name datatype,
```

```
.....
```

```
coln_name datatype
```

```
);
```

#### Example

```
CREATE TABLE person_details{
```

```
AdharNo int,
```

```
FirstName VARCHAR(20),
```

```
MiddleName VARCHAR(20),
```

```
LastName VARCHAR(20),
```

Address VARCHAR(30),

City VARCHAR(10)

}

The blank table will be created with following structure

### Person\_details

AdharNo	FirstName	MiddleName	LastName	Address	City
---------	-----------	------------	----------	---------	------

## 2. Insertion

We can insert data into the table using INSERT statement.

### Syntax

```
INSERT INTO table_name (col1, col2, ..., coln)
```

```
VALUES (value1, value2, ..., value_n)
```

### Example

```
INSERT INTO person_details (AdharNo, FirstName, MiddleName, LastName, Address, City)
```

```
VALUES (111, 'AAA', 'BBB', 'CCC', 'M.G. Road', 'Pune')
```

The above query will result into –

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune

## 3. Select

- The Select statement is used to fetch the data from the database table.
- The result returns the data in the form of table. These result tables are called resultsets.
- We can use the keyword DISTINCT. It is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

### Syntax

```
SELECT col1, col2, ..., coln FROM table_name;
```

### Example

```
SELECT AdharNo, FirstName, Address, City from person_details
```

The result of above query will be

AdharNo	FirstName	City
111	AAA	Pune

- If we want to select all the records present in the table we make use of \* character.

### Syntax

```
SELECT FROM table_name;
```

### Example

```
SELECT * FROM person_details;
```

The above query will result into

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune

## 4. Where Clause

The WHERE command is used to specify some condition. Based on this condition the data present in the table can be displayed or can be updated or deleted.

### Syntax

```
SELECT col1, col2, ..., coln
```

```
FROM table_name
```

```
WHERE condition;
```

### Example

Consider following table-

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute the following query

```
SELECT AdharNo
FROM person_details
WHERE city='Pune';
```

The result will be

AdharNo	City
111	Pune
222	Pune

If we want records of all those person who live in city Pune then we can write the query using WHERE clause as

```
SELECT *
FROM person_details
WHERE city='Pune';
```

The result of above query will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune

## 5. Update

- For modifying the existing record of a table, update query is used.

### Syntax

```
UPDATE table name
SET col1=value1, col2=value2,...
WHERE condition;
```

### Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

### Person\_details table

If we execute following query

```
UPDATE rerson_details
SET city 'Chennai'
WHERE AdharNo=333
```

The result will be

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

## 6. Deletion

We can delete one or more records based on some condition. The syntax is as follows -

### Syntax

```
DELETE FROM table_name WHERE condition;
```

### Example

```
DELETE FROM person_details
```

```
WHERE AdharNo=333
```

The result will be –

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
444	JJJ	KKK	LLL	Viman nagar	Mumbai

We can delete all the records from table. But in this deletion, all the records get deleted without deleting table. For that purpose the SQL statement will be

```
DELETE FROM person_details;
```

## 7. Logical Operators

- Using WHERE clause we can use the operators such as AND, OR and NOT.
- AND operator displays the records if all the conditions that are separated using AND operator are true.
- OR operator displays the records if any one of the condition separated using OR operator is true.
- NOT operator displays a record if the condition is NOT TRUE.

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

### Syntax of AND

```
SELECT col1, col2, ...
```

```
FROM table_name
```

```
WHERE condition1 AND condition2 AND condition3...;
```

### Example of AND

If we execute following query-

```
SELECT AdharNo, FirstName, City
```

```
FROM person_details
```

```
WHERE AdharNo=222 AND City='Pune';
```

The result will be –

AdharNo	FirstName	City
222	DDD	Pune

### Syntax of OR

```
SELECT col1, col2, ...
```

```
FROM table_name
```

```
WHERE condition1 OR condition2 OR condition3 ...;
```

### Example of OR

```
SELECT AdharNo, FirstName, City
```

```
FROM person_details
```

```
WHERE City='Pune' OR City='Mumbai';
```

The result will be –

AdharNo	FirstName	City
111	AAA	Pune
222	DDD	Pune
444	JJJ	Mumbai

### Syntax of NOT

*SELECT coll, col2, ...*

*FROM table\_name*

*WHERE NOT condition;*

### Example of NOT

*SELECT AdharNo, FirstName, City*

*FROM person\_details*

*WHERE NOT City='Pune';*

The result will be

AdharNo	FirstName	City
333	GGG	Delhi
444	JJJ	Mumbai

## 8. Order By Clause

- Many times we need the records in the table to be in sorted order.
- If the records are arranged in increasing order of some column then it is called ascending order.
- If the records are arranged in decreasing order of some column then it is called descending order.
- For getting the sorted records in the table we use ORDER BY command.
- The ORDER BY keyword sorts the records in ascending order by default.

### Syntax

*SELECT coll, col2,...,coln*

*FROM table\_name*

*ORDER BY coll,col2.... ASC | DESC*

Here ASC is for ascending order display and DESC is for descending order display.

### Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

*SELECT \**

*FROM person\_details*

*ORDER BY AdharNo DESC;*

The above query will result in

AdharNo	FirstName	MiddleName	LastName	Address	City
444	JJJ	KKK	LLL	Viman nagar	Mumbai
333	GGG	HHH	III	Chandani chowk	Delhi
222	DDD	EEE	FFF	Shivaji nagar	Pune
111	AAA	BBB	CCC	M.G. road	Pune

## 9. Alteration

There are SQL command for alteration of table. That means we can add new column or delete some column from the table using these alteration commands.

### Syntax for Adding columns

*ALTER TABLE table\_name*

*ADD column\_name datatype;*

### Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute following command

```
ALTER TABLE Customers
```

```
ADD Email varchar(30);
```

Then the result will be as follows –

AdharNo	FirstName	MiddleName	LastName	Address	City	Email
111	AAA	BBB	CCC	M.G. road	Pune	NULL
222	DDD	EEE	FFF	Shivaji nagar	Pune	NULL
333	GGG	HHH	III	Chandani chowk	Delhi	NULL
444	JJJ	KKK	LLL	Viman nagar	Mumbai	NULL

Syntax for Deleting columns

```
ALTER TABLE table_name
```

```
DROP COLUMN column name;
```

### Example

Consider following table

AdharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute following command

```
ALTER TABLE Customers
```

```
DROP COLUMN Address;
```

Then the result will be as follows –

AdharNo	FirstName	MiddleName	LastName	City
111	AAA	BBB	CCC	Pune
222	DDD	EEE	FFF	Pune
333	GGG	HHH	III	Delhi
444	JJJ	KKK	LLL	Mumbai

## 10. Defining Constraints

- We can specify rules for data in a table.
- When the table is created at that time we can define the constraints.
- The constraint can be column level i.e. we can impose constraint on the column and table level i.e we can impose constraint on the entire table.

There are various types of constraints that can be defined are as follows -

**1) Primary key:** The primary key constraint is defined to uniquely identify the records from the table.

The primary key must contain unique values. Hence database designer should choose primary key very carefully.

**For example**

Consider that we have to create a person\_details table with AdharNo, FirstName, MiddleName, LastName, Address and City.

Now making AdharNo as a primary key is helpful here as using this field it becomes easy to identify the records correctly.

The result will be

```
CREATE TABLE person_details (
```

```
AdharNo int,
```

```

FirstName VARCHAR(20),
MiddleName VARCHAR(20),
LastName VARCHAR(20),
Address VARCHAR(30),
City VARCHAR(10),
PRIMARY KEY(AdharNo)
);

```

We can create a composite key as a primary key using CONSTRAINT keyword. For example

```

CREATE TABLE person_details (
AdharNo int NOT NULL,
FirstName VARCHAR(20),
MiddleName VARCHAR(20),
LastName VARCHAR(20) NOT NULL,
Address VARCHAR(30),
City VARCHAR(10),
CONSTRAINT PK_person_details PRIMARY KEY(AdharNo, LastName)
);

```

## (2) Foreign Key

- Foreign key is used to link two tables.
- Foreign key for one table is actually a primary key of another table.
- The table containing foreign key is called child table and the table containing candidate primary key is called parent key.
- Consider

### Employee Table

EmpID	LastName	FirstName	Age
1	Khanna	Rajesh	30
2	Joshi	Sharman	23
3	Kapoor	Tushar	20

### Dept Table:

DeptID	DeptName	EmpID
1	Accounts	3
2	Production	3
3	Sales	2
4	Purchase	1

- Notice that the "EmpID" column in the "Dept" table points to the "EmpID" column in the "Employee" table.
- The "EmpID" column in the "Employee" table is the PRIMARY KEY in the "Employee" table.
- The "EmpID" column in the "Dept" table is a FOREIGN KEY in the "Dept" table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
- The purpose of the foreign key constraint is to enforce referential integrity but there are also performance benefits to be had by including them in your database design.

The table Dept can be created as follows with foreign key constraint.

```

CREATE TABLE DEPT (
DeptID int
DeptName VARCHAR(20),
EmpID int,
PRIMARY KEY(DeptID),
FOREIGN KEY (EmpID)
REFERENCES EMPLOYEE(EmpID)
);

```

### **(3) Unique**

Unique constraint is used to prevent same values in a column. In the EMPLOYEE table, for example, you might want to prevent two or more employees from having an identical designation. Then in that case we must use unique constraint.

We can set the constraint as unique at the time of creation of table, or if the table is already created and we want to add the unique constraint then we can use ALTER command.

For example -

```
CREATE TABLE EMPLOYEE(  
EmpID INT NOT NULL,  
Name VARCHAR (20) NOT NULL,  
Designation VARCHAR(20) NOT NULL UNIQUE,  
Salary DECIMAL (12, 2),  
PRIMARY KEY (EmpID)  
);
```

If table is already created then also we can add the unique constraint as follows -

```
ALTER TABLE EMPLOYEE  
MODIFY Designation VARCHAR(20) NOT NULL UNIQUE;
```

### **(4) NOT NULL**

- By default the column can have NULL values.
- NULL means unknown values.
- We can set the column values as non NULL by using the constraint NOT NULL.
- For example

```
CREATE TABLE EMPLOYEE(  
EmpID INT NOT NULL,  
Name VARCHAR (20) NOT NULL,  
Designation VARCHAR(20) NOT NULL,  
Salary DECIMAL (12, 2) NOT NULL,  
PRIMARY KEY (EmpID)  
);
```

### **(5) CHECK**

The CHECK constraint is used to limit the value range that can be placed in a column.

For example

```
CREATE TABLE parts (  
Part_no int PRIMARY KEY,  
Description VARCHAR(40),  
Price DECIMAL(10, 2) NOT NULL CHECK(cost > 0)  
);
```

### **(6) IN operator**

The IN operator is just similar to OR operator.

It allows to specify multiple values in WHERE clause.

#### **Syntax**

```
SELECT col1,col2,...  
FROM table_name  
WHERE column-name IN (value1, value2,...);
```

#### **Example**

Consider following table

#### **Employee**

empID	empName	Salary	DeptID
1	AAA	1000	D101
2	BBB	2000	D102
3	CCC	3000	D103
4	DDD	4000	D104
5	EEE	5000	D105

*SELECT FROM Employee*

*WHERE empID IN (1, 3);*

The result will be

empID	empName	Salary	DeptID
1	AAA	1000	D101
2	BBB	2000	D102
3	CCC	3000	D103

## Basic Structure of SQL Queries

The basic form of SQL queries is

*SELECT-FROM-WHERE. The syntax is as follows:*

*SELECT[**DISTINCT**] target-list*

*FROM Relation-list*

*WHERE Qualification*

- **SELECT:** This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM:** This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **WHERE:** This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.
- **Relation-list:** A list of relation names (tables)
- **Target-list:** A list of attributes of relations from relation list (tables)
- **Qualification:** Comparisons of attributes with values or with other attributes combined using AND, OR and NOT.
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

### Example

*SELECT sname*

*FROM Student*

*WHERE age > 18*

- The above query will return names of all the students from student table where age of each student is greater than 18

## Queries on Multiple Relations

Many times it is required to access multiple relations (tables) to operate on some information. For example consider two tables as Student and Reserve.

sid	sname	age	sid	isbn	day
1	Ram	21	1	005	07-07-18
2	Shyam	18	2	007	03-03-18
3	Seeta	16	3	009	
4	Geeta	23			

**Query:** Find the names of students who have reserved the books with book isbn

*Select Student.sname, Reserve.isbn*

*From Student, Reserve*

*Where Student.sid=Reserve.sid*

### Use of SQL Join

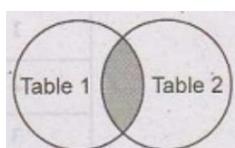
The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

**Example:** Consider two tables for using the joins in SQL. Note that cid is common column in following tables.

Student			Reserve	
sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	102	Mumbai
3	102	Seeta	103	Chennai
4	NULL	Geeta		

### 1) Inner Join:

- The most important and frequently used of the joins is the INNER JOIN. They are also known as an EQUIJOIN.
- The INNER JOIN creates a new result table by combining column values of two actual tables (Table1 and Table2) based upon the join-predicate.
- The query compares each row of table1 with each row of Table2 to find all pairs of rows which satisfy the join-predicate.
- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. It can be represented as:



- **Syntax:** The basic syntax of the INNER JOIN is as follows.

```
SELECT Table1.column1, Table2.column2...
```

```
FROM Table1
```

```
INNER JOIN Table2
```

```
ON Table1.common_field = Table2.common_field;
```

- **Example:** For above given two tables namely Student and City, we can apply inner join. It will return the record that are matching in both tables using the common column cid. The query will be

```
SELECT *
```

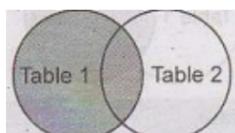
```
FROM Student Inner Join City on Student.cid=City.cid
```

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai

### 2) Left Join:

- The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- It can be represented as –



- **Syntax:** The basic syntax of a LEFT JOIN is as follows.

```
SELECT
```

```
SELECT Table1.column1, Table2.column2...
```

```
FROM Table1
```

```
LEFT JOIN Table2
```

```
ON Table1.common field Table2.common field;
```

- **Example:** For above given two tables namely Student and City, we can apply Left join. It will Return all records from the left table, and the matched records from the right table using the common column cid. The query will be

*SELECT \**

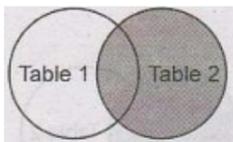
*FROM Student Left Join City on Student.cid=City.cid*

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL

### 3) Right Join:

- The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- It can be represented as follows:
- **Syntax:** The basic syntax of a RIGHT JOIN is as follow-



*SELECT Table1.column1, Table2.column2...*

*FROM Table1*

*RIGHT JOIN Table2*

*ON Table1.common\_field = Table2.common\_field;*

- **Example:** For above given two tables namely Student and City, we can apply Rightjoin. It will return all records from the right table, and the matched records from the left table using the common column cid. The query will be,

*SELECT \**

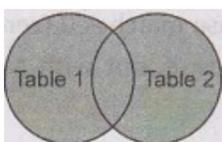
*FROM Student Right Join City on Student.cid=City.cid*

The result will be –

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
NULL	NULL	NULL	103	Chennai

### 4) Full Join:

- The SQL FULL JOIN combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLS for missing matches on either side.
- It can be represented as



- **Syntax:** The basic syntax of a FULL JOIN is as follows

*SELECT Table1.column1, Table2.column2...*

*FROM Table1 FULL JOIN Table2 ON Table1.common\_field = Table2.common\_field;*

The result will be -

- **Example:** For above given two tables namely Student and City, we can apply Full join. It will return returns rows when there is a match in one of the tables using the common column cid. The query will be -

*SELECT \**

*FROM Student Full Join City on Student.cid=City.cid*

The result will be –

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL
NULL	NULL	NULL	103	Chennai

### Additional Basic Operations

**1) The Rename Operation:** The SQL AS is used to assign temporarily a new name to a table column or table(relation) itself. One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query. For example - "Find the names of students and isbn of book who reserve the books".

#### Student

Student			Reserve		
sid	sname	age	sid	isbn	day
1	Ram	21	1	005	07-07-18
2	Shyam	18	2	007	03-03-18
3	Seeta	16	3	009	05-05-18
4	Geeta	23			

*Select S.sname,R.isbn*

*From Student as S, Reserve as R*

*Where S.sid=R.sid*

In above case we could shorten the names of tables Student and Reserve as S and R respectively.

Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself. For example-

If the query is - Find the names of students who reserve the book of isbn005. Then the SQL statement will be -

*Select S.sname, R.isbn*

*From Student as S, Reserve as R*

*Where S.sid=R.sid and S.isbn=005*

**2) Attribute Specification in Select clause:** The symbol \* is used in select clause to denote all attributes. For example - To select all the records from Student table we can write

*Select\* from Student*

**3) Ordering the display of tuples:** For displaying the records in particular order we use order by clause.

The general syntax with ORDER BY is

*SELECT column\_name(s)*

*FROM table\_name*

*WHERE condition*

*ORDER BY column\_name(s)*

• **Example:** Consider the Student table as follows-

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

**Query:** Find the names of students from highest marks to lowest

*Select sname*

*From Student*

*Order By marks*

We can also use the desc for descending order and asc for ascending order. For example:

In order to display names of the students in descending order of city - we can specify

*Select sname*

From Student

Order by city desc;

#### (4) Where clause Predicate

(i) The between operator can be used to simplify the where clause which is used to denote the value be less than or equal to some value and greater than or equal to some other value. For example - if we want the names of the students whose marks are between 80 and 90 then SQL statement will be

Select name

From Students

Where marks between 80 and 90;

Similarly we can make use of the comparison operators for various attributes. For example - If the query is - Find the names of students who reserve the book of isbn005. Then the SQL statement will be -

Select sname

From Student, Reserve

Where (Student.sid,Reserve.isbn)=(Reserve.sid,005);

(ii) We can use AND, OR and NOT operators in the Where clause. For filtering the records based on more than one condition, the AND and OR operators can be used. The NOT operator is used to demonstrate when the condition is not TRUE.

Consider following sample database - Students database, for applying AND, OR and NOT operators

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

#### Syntax of AND

SELECT column1, column2, ....

FROM table name

WHERE condition1 AND condition2 AND condition3 ...;

**Example:** Find the student having name "AAA" and lives in city "Pune"

SELECT

FROM Students

Where sname='AAA' AND city='Pune'

#### Output

sid	sname	marks	city
1	AAA	60	Pune

#### Syntax OR

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 OR condition2 OR condition3 ...;

**Example:** Find the student having name "AAA" OR lives in city "Pune"

SELECT \*

FROM Students

Where sname='AAA' OR city='Pune'

#### Output

sid	sname	marks	city
1	AAA	60	Pune
3	CCC	90	Pune

#### Syntax NOT

SELECT column1, column2, ..

FROM table name

## Example of SQL Fundamentals

### Examples of SQL Fundamentals

**AU: Dec.-14,15,17,19, May-15,16,17,18, Marks 15**

- Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS.

**Example 1.14.1** Write the DDL, DML, DCL for the students database. Which contains student details: name, id,DOB, branch, DOJ.

Course details: Course name, Course id, Stud.id, Faculty name, id, mark

**AU: Dec.-17, Marks 15**

**Solution:**

#### DDL Commands

```
CREATE TABLE Student
```

```
(  
stud_name varchar(20),  
stud_id int(3),  
DOB varchar(15),  
branch varchar(10),  
DOJ varchar(15),  
);
```

```
CREATE TABLE Course
```

```
(  
course_name varchar(20),  
course_id int(5),  
stud_id int(3),  
facult_name varchar(20),  
faculty_id varchar(5),  
marks real  
);
```

#### DML Commands

The commands which we will use here are insert and select. The insert command is used to insert the values into database tables. Using the select command, the database values can be displayed.

### (1) Inserting values into Student table

```
insert into Student(stud_name,stud_id,DOB,branch,DOJ)
values('AAA',11,'01-10-1999', 'computers','5-3-2018')
insert into Student(stud_name,stud_id,DOB,branch,DOJ)
values('BBB',12, '24-5-1988', 'Mechanical', '17-2-2016')
insert into Student(stud_name, stud_id,DOB,branch,DOJ)
values('CCC',13,'8-1-1990', 'Electrical','22-9-2017')
```

### (2) Inserting values into Course table

```
insert into Course (course_name,course_id,stud_id,faculty_name, faculty_id,marks) values('Basic',101,11,'Archana',
'F001','50')
insert into Course(course_name,course_id,stud_id,faculty_name, faculty_id,marks) values('Intermediate',102,12,'Rupali',
'F002','70')
insert into Course (course_name,course_id,stud_id,faculty_name, faculty_id,marks) values('Advanced',103,13,'Sunil','F003',
'100')
```

### (3) Displaying records of Student table

```
Select * from Student;
```

### (4) Displaying records of Course table

```
Select * from Course;
```

## DCL Commands

The DCL command is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

### (1) GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

#### Syntax

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

privilege\_name is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.

- object\_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- user\_name is the name of the user to whom an access right is being granted.
- PUBLIC is used to grant access rights to all users.
- roll\_name are a set of privileges grouped together.
- WITH GRANT OPTION - allows a user to grant access rights to other users.

#### Example

```
GRANT SELECT ON student_details TO user1
```

This query will grant the SELECT permission to student\_details table to user named user1.

Similarly we can GRANT more than one privileges to user in a table

```
GRANT SELECT, INSERT, DELETE, UPDATE ON student_details TO user1
```

For granting all privileges to user we use sysdba. The sysdba is a set of privileges which has all the permissions in it.

```
GRANT sysdba TO user1
```

### (2) REVOKE

The REVOKE command removes user access rights or privileges to the database objects.

#### Syntax

```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}
```

#### Example

To remove access right for SELECT to the table student\_details for user1 we write the query

REVOKE SELECT ON student\_details FROM user1;

**Example 1.14.2** Write the following queries in relational algebra and SQL

(i) Find the names of employee who have borrowed a book published by McGraw Hill

(ii) Find the names of employees who have borrowed all books published by McGraw-Hill **AU: May-17, Marks 10**

**Solution:**

We will assume the databases as -

member(memb\_no, name, dob)

books(isbn, title, authors, publisher)

borrowed(memb\_no, isbn, date)

**(i) Relational Algebra:**

$\Pi_{\text{name}}((\text{publisher McGraw Hill} \text{ books}) \bowtie \text{borrowed} \bowtie \text{member})$

**SQL:**

SELECT name

FROM member

WHERE member.memb\_no=borrowed.memb\_no AND books.isbn=borrowed.isbn

AND books.publisher 'McGraw Hill';

**(ii) Relational Algebra**

$\rho(\text{Tempname}, (\Pi_{\text{memb\_no, isbn}} \text{borrowed}) / \Pi_{\text{isbn}} (\sigma_{\text{publisher=McGraw Hill}} \text{books}))) \Pi_{\text{name}} (\text{Tempname member})$

**SQL:**

SELECT distinct M.name

FROM Member M,

WHERE NOT EXIST

(

(SELECT isbn

FROM books

WHERE publisher = 'McGraw Hill'

)

EXCEPT

(SELECT isbn

FROM borrowed R

WHERE R.memb\_no = M.memb\_no

)

)

**Example 1.14.3** Assume the following table.

Degree (degcode, name, subject)

Candidate (seatno, degcode, name, semester, month, year, result)

Marks (seatno, degcode, semester, month, year, papcode, marks)

[degcode - degree code, name - name of the degree (Eg. MSc.), subject - subject of the course (Eg. Physis), papcode - paper code (Eg. A1)]

Solve the following queries using SQL;

Write a SELECT statement to display,

(i) all the degree codes which are there in the candidate table but not present in degree table in the order of degcode.

(ii) the name of all the candidates who have got less than 40 marks in exactly 2 subjects.

(iii) the name, subject and number of candidates for all degrees in which there are less than 5 candidates.

(iv) the names of all the candidate who have got highest total marks in MSc. Maths. **AU: Dec.-15, Marks 4+4 +4 +4**

**Solution:**

(i) SELECT C.degcode

```

FROM Candidate C,
WHERE
NOT EXISTS
(SELECT D.degcode
FROM Degree D
WHERE D.degcode=C.degcode)
ORDER by C.degcode
(ii) SELECT C.name
FROM Candidate C, Degree D, Marks M
WHERE
C.seatno M.seatno AND C.degcode=D.degcode AND C.degcode-M.degcode AND M.marks <40 GROUP BY C.seatno
HAVING count(D.subject)=2;
(iii) SELECT D.name,D.subject,count(*)
FROM degree D, Candidate C
WHERE D.degcode=C.degcode
HAVING( SELECT count(*) FROM Candidate <5);
(iv) SELECT C.name
FROM Candidate C, Degree D, Marks M
WHERE
D.degname='MSc' AND D.subject='Maths' AND C.degcode=D.degcode AND C.seatno-M.seatno
AND
M.marks (SELECT max(M.marks) FROM Marks M)

```

**Example 1.14.4** Consider a student registration database comprising of the below given table schema.

Student File			
Student Number	Student Name	Address	Telephone
Course File			
Course Number	Description	Hours	Professor Number
Professor File			
Professor Number	Name	Office	
Registration File			
Student Number	Course Number	Date	

*Student Number Course Number Date*

Consider a suitable sample of tuples / records for the above mentioned tables and write DML statements (SQL) to answer for the queries listed below.

- i) Which courses does a specific professor teach?
- ii) What courses are taught by two specific professors?
- iii) Who teaches a specific course and where is his/her office?
- iv) For a specific student number, in which courses is the student registered and what is his/her name?
- v) Who are the professors for a specific student?
- vi) Who are the students registered in a specific course?

**Solution:**

**(i)**

```

SELECT P.name,C.description
FROM Professor P, Course C
WHERE P.ProfessorNumber=C.ProfessorNumber
HAVING count(DISTINCT P.name)=2

```

**(ii)**

```

SELECT P.name,C.description
FROM Professor P, Course C
WHERE P.Professor Number=C.ProfessorNumber

```

**(iii)**

```
SELECT P.name,P.office, C.description
FROM Professor P, Course C
WHERE P.ProfessorNumber=C.Professor Number
```

**(iv)**

```
SELECT S.StudentNumber,S.StudentNumber, C.Description
FROM Student S, Course C, Registration R
WHERE S.StudentNumber=R.StudentNumber AND C.CourseNumber=R.CourseNumber
```

**(v)**

```
SELECT S.StudentName, P.Name
FROM Student S, Course C, Professor P, Registration R
WHERE C.ProfessorNumber=P.Professor Number
AND C.CourseNumber=R.Course Number
AND S.StudentNumber=R.StudentNumber GROUP BY P.Professor Number
```

**(vi)**

```
SELECT S.StudentName, C.Description
FROM Student S, Course C, Registration R
WHERE S.StudentNumber=R.Student Number
AND R.CourseNumber=C.CourseNumber
GROUP BY C.CourseNumber
```

**Example 1.14.5** For the following database, identify primary key and foreign key wherever applicable and solve the given queries in SQL.

*Item (ino, description, unit\_price)*

*Supplier (sno, sname, address)*

*Supplied (sno, ino, sdate, qty, per\_unit\_discount)*

1. Find supplier name for the suppliers who supply every item.
2. Find distinct item names for the items supplied with total discount > 500.
3. Pair of supplier names supplying same items on same dates.

**Solution:**

- The primary key for Item table is ino.
- The primary key for supplier table is sno
- The primary key for supplied table is (sno,ino)

**(1) Find supplier name for the suppliers who supply every item.**

```
SELECT S.sname
FROM Supplier S
WHERE NOT EXISTS (
(SELECT * FROM Item I
WHERE NOT EXISTS
SELECT *
FROM Supplied SP
WHERE SP.sno S.sno
AND SP.ino I.ino)
)
```

**(2) Find distinct item names for the items supplied with total discount > 500.** SELECT DISTINCT description

```
FROM Item, Supplied
WHERE Item.ino=Supplied.ino
AND Supplied.per_unit_discount >500
```

**(3) Pair of supplier names supplying same items on same dates.**

```
SELECT sname
```

```

FROM Supplier
WHERE Supplied.sno=Supplier.sno
AND
Item.ino = Supplied.ino
)

```

**Example 1.14.6** Write SQL statements for the following (any five)

Consider the following database

*pilot (pid, pname)*

*flight (fid, ftype, capacity)*

*route (pid, fid, from\_city, to\_city)*

i) List the details of flights having capacity more than 300.

ii) List the flights between 'Surat' and 'Mumbai'.

iii) List the names of the pilots who fly from 'Pune'.

iv) List the route on which, pilot named 'Mr Kapoor' flies

v) List the pilots whose names, starts with letter 'A' %' but does not end with letter 'A'. vi) List the name of pilots who fly 'boeing 737' type of flights.

**Solution:**

**(i) List the details of flights having capacity more than 300.**

```

SELECT FROM flight

```

```

WHERE capacity>300

```

**(ii) List the flights between 'Surat' and 'Mumbai'.**

```

SELECT fid

```

```

FROM flight, route

```

```

WHERE

```

```

flight.fid=route.fid AND

```

```

route.from_city= 'Surat' AND route.to_city='Mumbai';

```

**(iii) List the names of the pilots who fly from 'Pune'.**

```

SELECT pname

```

```

FROM pilot, route

```

```

WHERE pilot.pid=route.pid

```

```

AND route.from_city= 'Pune';

```

**(iv) List the route on which, pilot named 'Mr Kapoor' flies**

```

SELECT from city, to_city

```

```

FROM route, pilot

```

```

WHERE pilot.pid=route.pid

```

```

AND pilot.pname='Mr.Kapoor';

```

**(v) List the pilots whose names, starts with letter 'A' %' but does not end with letter 'A'.**

```

SELECT pname

```

```

FROM pilot

```

```

WHERE pname LIKE 'A%' MINUS

```

```

SELECT pname FROM pilot

```

```

WHERE pname LIKE '%A';

```

**(vi) List the name of pilots who fly 'boeing 737' type of flights.**

```

SELECT pname

```

```

FROM pilot, flight, route

```

```

WHERE flight.ftype = 'boeing 737' AND

```

```

pilot.pid = route.pid AND

```

```

route.fid = flight.fid;

```

**Example 1.14.7** Consider the relation Database.

Person (SSN, Name, city)

Car (License\_no, year, Model, SSN)

Accident(drive\_no, SSN, license\_no, accidentyear, damage\_Amt) Query:

1. Find out total no of cars that had accident in 1988.
2. Find the Name of driver who did not have an accident in 'Delhi'.
3. Find the car, who don't have total damage of more than 1000.
4. Find the cars sold in 2006 and whose owner are from 'Vadodara'
5. How many different models of car are used by 'Mr.abc'?
6. Find the lucky persons who have not met any accident yet.

**Solution :**

**1. Find out total no of cars that had accident in 1988.**

```
SELECT count(License_no)
FROM Car, Accident
WHERE Accident.accidentyear=1988
```

**2. Find the Name of driver who did not have an accident in 'Delhi'.**

```
SELECT Name
FROM Person, Accident
WHERE Person.SSN=Accident.SSN AND Person.city<>'Delhi'
```

**3. Find the car, who don't have total damage of more than 1000.**

```
SELECT License_no
FROM Car, Accident
WHERE Accident.damage_amt <1000 AND
Car.License_no=Accident.licence_no;
```

**4. Find the cars sold in 2006 and whose owner are from 'Vadodara'**

```
SELECT license no
FROM Car, Person
WHERE Car.SSN = Person. SSN AND
Car.year 2006 and Person.city = 'Vadodara';
```

**5. How many different models of car are used by 'Mr.abc' ?**

```
SELECT count(model)
FROM Car, Person
WHERE Car.SSN = Person. SSN AND
Person.Name = 'Mr.abc';
```

**6. Find the lucky persons who have not met any accident yet.**

```
SELECT Name
FROM Person
WHERE person.SSN NOT IN (SELECT SSN FROM Accident);
```

**Example 1.14.8** We have following relations:

Supplier (S#, sname, status, city)

Parts (P#, pname, color, weight, city)

SP(S#, P#, quantity)

Answer the following queries in SQL,

- i) Find name of supplier for city = 'Delhi'.
- ii) Find suppliers whose name start with 'AB'.
- iii) Find all suppliers whose status is 10, 20 or 30.
- iv) Find total number of city of all suppliers.
- v) Find s# of supplier who supplies 'red' part.

vi) Count number of supplier who supplies 'red' part. vii) Sort the supplier table by sname.

**Solution:**

**i) Find name of supplier for city = 'Delhi'.**

```
SELECT sname
FROM Supplier
WHERE city= 'Delhi'
```

**ii) Find suppliers whose name start with 'AB'.**

```
SELECT sname
FROM Supplier
WHERE sname='AB%'
```

**iii) Find all suppliers whose status is 10, 20 or 30.**

```
SELECT sname
FROM Supplier
WHERE status BETWEEN 10 AND 30
```

**iv) Find total number of city of of all suppliers.**

```
SELECT count(*) FROM (SELECT DISTINCT CITY FROM Supplier);
```

**v) Find s# of supplier who supplies 'red' part.**

```
SELECT DISTINCT supplier.S#
FROM Supplier, Parts, SP
WHERE Supplier.S# = SP.S#
AND SP.P# = Parts.P# AND Parts.color = 'red';
```

**vi) Count number of supplier who supplies 'red' part. SELECT count(\*)**

```
FROM (SELECT DISTINCT Supplier.S#
FROM Supplier, Parts, SP
WHERE Supplier.S# = SP.S#
AND SP.P# = Parts.P# AND Parts.color = 'red');
```

**vii) Sort the supplier table by sname.**

```
SELECT *
FROM Supplier
ORDER BY sname;
```

**Example 1.14.9** We have following relations:

*Supplier (S#, sname, status, city)*

*Parts (P#, pname, color, weight, city)*

*SP(S#, P#, quantity)*

Answer the following queries in SQL,

i) Delete records in supplier table whose status is 40.

ii) Add one field in supplier table.

**Solution:**

**i) Delete records in supplier table whose status is 40.**

```
DELETE FROM Supplier
WHERE status=40
```

**ii) Add one field in supplier table.**

```
ALTER TABLE Supplier
ADD(PhoneNo number);
```

**Example 1.14.10** We have following relations:

*Supplier (S#, sname, status, city)*

*Parts (P#, pname, color, weight, city)*

*SP(S#, P#, quantity)*

Answer the following queries in SQL,

i) Find name of parts whose color is 'red'.

ii) Find parts name whose weight less than 10 kg.

iii) Find all parts whose weight from 10 to 20 kg.

iv) Find average weight of all parts.

v) Find S# of supplier who supply part 'p2'.

vi) Find name of supplier who supply maximum parts. vii) Sort the parts table by pname.

**Solution:**

**i) Find name of parts whose color is 'red'.**

```
SELECT pname
FROM Parts
WHERE Parts.color = 'red';
```

**ii) Find parts name whose weight less than 10 kg.**

```
SELECT pname
FROM Parts
WHERE Parts.weight < 10;
```

**iii) Find all parts whose weight from 10 to 20 kg.**

```
SELECT pname, weight
FROM Parts
WHERE Parts.weight BETWEEN 10 AND 20;
```

**iv) Find average weight of all parts.**

```
SELECT AVG (weight)
FROM Parts;
```

**v) Find S# of supplier who supply part 'p2'.**

```
SELECT S#
FROM SP
WHERE p# = 'p2';
```

**vi) Find name of supplier who supply maximum parts.**

```
SELECT sname, MAX(quantity)
FROM Supplier, Parts, SP
WHERE Supplier.S# = SP.S#
AND SP.P# = P.P#
GROUP BY sname;
```

**vii) Sort the parts table by pname.**

```
SELECT *
FROM Parts
ORDER BY pname;
```

**Example 1.14.11** Consider following schema and write SQL for given statements.

*Student (rollno, name, branch)*

*exam(rollno, subject\_code, obtained\_marks, paper\_code)*

*papers(paper\_code, paper\_satter\_name, university)*

i) Display name of student who got first class in subject '130703'.

ii) Display name of all student with their total mark.

iii) Display list number of student in each university.

iv) Display list of student who has not given any exam.

**Solution:**

**i) Display name of student who got first class in subject '130703'.**

```
SELECT name
```

```
FROM student,exam
WHERE exam.obtained_marks>60 AND exam.subject_code='130703' AND
student.rollno = exam.rollno
```

**ii) Display name of all student with their total mark.**

```
SELECT name, SUM(obtained_marks)
FROM student,exam
WHERE student.rollno=exam.rollno
```

**iii) Display list number of student in each university.**

```
SELECT COUNT(Rollno)
FROM student, exam, papers
WHERE student.rollno=exam.rollno AND exam.paper_code=papers.paper_code;
```

**iv) Display list of student who has not given any exam.**

```
SELECT name
FROM student NOT IN
(SELECT rollno
FROM student, exam
WHERE student.rollno=exam.rollno)
```

**Example 1.14.12** Write down the query for the following table where primary keys are underlined.

*Person(ss#, name, address)*

*Car(license, year, model)*

*Accident(date, driver, damage-amount)*

*Owns(ss#, license)*

*Log(license, date, driver)*

1) Find the total number of people whose cars were involved in accidents in 2009.

2) Find the number of accidents in which the cars belonging to "S.Sudarshan"

3) Add a new customer to the database.

4) Add a new accident recorded for the santro belonging to "KORTH"

**Solution:**

**1) Find the total number of people whose cars were involved in accidents in 2009.**

```
SELECT count(ss#)
FROM person,owns, accident,log
WHERE person.ss#=owns.ss# AND
owns.licence=log.licence AND
log.driver accident.driver AND
accident.date >= '01-jan-2009' and accident.date<='31-dec-09;
```

**2) Find the number of accidents in which the cars belonging to "S.Sudarshan"**

```
SELECT COUNT(accident.date)
FROM person,owns, accident,log
WHERE log.date=accident.date AND log.driver accident.driver AND
own.licence=log.licence AND
person.ss#=own.ss# AND
person.name = 's.sudarshan';
```

**3) Add a new customer to the database.**

```
INSERT INTO person (ss#, name, address) values (101, 'Madhav', 'Pune');
```

```
INSERT INTO owns (ss#, license) values (101, 'L101');
```

```
INSERT INTO car (license, year, model) values ('L101', 2017, 'Honda'); INSERT INTO log (license, date, driver) values
('L101', NULL, 'Shankar');
```

**4) Add a new accident recorded for the santro belonging to "KORTH"**

```
INSERT INTO accident (date, driver, damage-amount) VALUES ('31-Dec-2016', 'Shankar', 10000);
```

```
INSERT INTO log (license, date, driver) VALUES ('L111','31-Dec-2016','Shankar');
```

```
INSERT INTO car (license, year, model) VALUES ('L111', '2005', 'SANTRO'); INSERT INTO person (ss#, name, address) VALUES (111, 'Korth', 'Mumbai');
```

```
INSERT INTO owns (ss#, license) VALUES (111, 'L111');
```

**Example 1.14.13** Consider the employee data. Give an expression in SQL for the following query:

*Employee(employee\_name, street,city)*

*Works (employee\_name,company\_name,salary)*

*Company(company\_name,city) Manages(employee\_name,manager\_name)*

1) Find the name of all employees who work for State Bank.

2) Find the names and cities of residence of all employees who work for State Bank.

3) Find all employee in the database who do not work for State Bank.

4) Find all employee in the database who earn more than every employee of UCO bank.

**Solution:**

**1) Find the name of all employees who work for State Bank.**

```
SELECT employee_name
```

```
FROM Works
```

```
WHERE company_name='State Bank';
```

**2)Find the names and cities of residence of all employees who work for State Bank.**

```
SELECT employee_name,city
```

```
FROM Employee, Works
```

```
WHERE company_name='State Bank' AND
```

```
Works.employee_name=Employee.employee_name;
```

**3) Find all employee in the database who do not work for State Bank.**

```
SELECT employee_name
```

```
FROM Works
```

```
WHERE company_name<>'State Bank';
```

**4) Find all employee in the database who earn more than every employee of UCO bank.**

```
SELECT employee_name from Works
```

```
WHERE salary>(SELECT MAX(salary) FROM Works
```

```
WHERE company_name='UCO bank');
```

**Example 1.14.14** Consider following schema and write SQL for given statements. Student (Rollno,Name,Age, Sex, City).

*Student\_marks (Rollno,Sub1,Sub2,Sub3, Total,Average)*

*Write query to*

i) Calculate and store total and average marks from sub1, sub2 and sub3.

ii)Display name of students who got more than 60 marks in subject Sub1.

iii)Display name of students with their total and average marks.

iv) Display name of students who got equal marks in subject sub2.

**Solution:**

**i) Calculate and store total and average marks from sub1, sub2 and sub3.**

```
UPDATE Student_marks
```

```
SET Total=sub1+sub2+sub3,
```

```
Avarage (sub1+sub2+sub3)/3;
```

**ii) Display name of students who got more than 60 marks in subject Sub1. SELECT Name**

```
FROM Student, Student marks
```

```
WHERE Student.Rollno=Student_marks. Rollno AND Student_marks.sub1>60
```

**iii) Display name of students with their total and average marks.**

```
SELECT Name, Total, Average
```

FROM Student, Student\_marks

WHERE Student.Rollno=Student\_marks.Rollno

**iv) Display name of students who got equal marks in subject sub2.**

**SELECT Name**

FROM Student S, Student\_marks SM1, Student\_marks SM2

WHERE S.Rollno=SM1.Rollno AND SM1.sub2=SM2.sub2;

**Example 1.14.15** We have following relations.

*EMP (empno, ename, jobtitle, manager no, hiredate, sal, comm, dept no)*

*DEPT (dept no, dname, loc)*

*i) The employees who are getting salary greater than 3000 for those persons belongings to the department 20*

*ii) Employees who are not getting any commission.*

*iii) Find how many job titles are available in employee table.*

*iv) Display total salary spent for each job category.*

*v) Display number of employees working in each department and their department name.*

*vi) List ename whose manager is NULL.*

*vii) List all employee names and their salaries, whose salary lies between 1500/- and 3500/- both inclusive.*

**Solution:**

**i) The employees who are getting salary greater than 3000 for those persons belongings to the department 20**

SELECT ename

FROM EMP

WHERE EMP.sal>3000 AND EMP.dept\_no=20

**ii) Employees who are not getting any commission.**

SELECT ename

FROM EMP

WHERE EMP.comm IS NULL;

**iii) Find how many job titles are available in employee table.**

SELECT count(jobtitle)

FROM EMP

**iv) Display total salary spent for each job category.**

SELECT ename, SUM(sal)

FROM EMP

GROUP BY jobtitle;

**v) Display number of employees working in each department and their department name.**

SELECT COUNT(EMP.eno), DEPT.dname

FROM EMP, DEPT

WHERE EMP.dept\_no=DEPT.dept\_no

GROUP BY DEPT.dept\_no;

**vi) List ename whose manager is NULL.**

SELECT ename

FROM EMP

WHERE manager\_no IS NULL;

**vii) List all employee names and their salaries, whose salary lies between 1500/- and 3500/- both inclusive.**

SELECT ename, sal

FROM EMP

WHERE sal>=1500 AND sal<=3500;

**Example 1.14.16** We have following relations.

*EMP (empno, ename, jobtitle, manager no, hiredate, sal, comm, dept no) DEPT (dept no, dname, loc)*

*Answer the following queries in SQL*

- i) Find the employees working in the department 10, 20, 30 only.
- ii) Find employees whose names start with letter A or letter a. iii) Find employees along with their department name.
- iv) Find employees whose manager is KING.
- v) Find the employees who are working in smith's department.
- vi) Find the employees who get salary more than Allen's salary.
- vii) Display employees who are getting maximum salary in each department. **Solution:**

**i) Find the employees working in the department 10, 20, 30 only.**

```
SELECT empno
FROM EMP
WHERE dept_no BETWEEN 10 AND 30
```

**ii) Find employees whose names start with letter A or letter a.**

```
SELECT ename
FROM EMP
WHERE ename='A%' OR ename='a%';
```

**iii) Find employees along with their department name.**

```
SELECT EMP.ename,DEPT.dname
FROM EMP, DEPT
WHERE EMP.dept_no=DEPT.dept_no ;
```

**iv) Find employees whose manager is KING.**

```
SELECT ename
FROM EMP
WHERE managerno = (SELECT empno FROM EMP WHERE ename='KING');
```

**v) Find the employees who are working in Smith's department.**

```
SELECT ename
FROM EMP, DEPT
WHERE EMP.dept_no=DEPT.dept_no AND ename='Smith';
```

**vi) Find the employees who get salary more than Allen's salary.**

```
SELECT ename
FROM EMP
WHERE sal > (SELECT sal FROM EMP WHERE ename='Allen');
```

**vii) Display employees who are getting maximum salary in each department.**

```
SELECT ename, MAX(sal)
FROM EMP
GROUP BY dept_no;
```

**Example 1.14.17** Write queries for the following table.

T1 (Empno, Ename, Salary, Designation),

T2 (Empno, Deptno.)

- i) Display all rows for salary greater than 5000
- ii) Display the deptno for the ename='syham'.
- iii) Add a new column deptname in table T2.
- iv) Change the designation of ename='ram' from 'clerk' to 'senior clerk'.
- v) Find the total salary of all the rows.
- vi) Display Empno, Ename, Deptno and Deptname.
- vii) Drop the table T1.

**Solution:**

**i) Display all rows for salary greater than 5000 SELECT \***

```
FROM T1
WHERE Salary > 5000
```

**ii) Display the deptno for the ename='shyam'. SELECT deptno**

FROM T1,T2

WHERE T1.Empno=T2.Empno AND T1.Ename='shyam';

**iii) Add a new column deptname in table T2.**

ALTER TABLE T2

ADD (deptname VARCHAR(20));

**iv) Change the designation of ename='ram' from 'clerk' to 'senior clerk'. UPDATE T1**

SET T1.designation='Senior Clerk'

WHERE T1.ename='ram';

**v) Find the total salary of all the rows.**

SELECT SUM(Salary)

FROM T1;

**vi) Display Empno, Ename, Deptno and Deptname.**

SELECT E.Empno, E.Ename, D.Deptno, D.Deptname

FROM T1 E, T2 D

WHERE E.Empno=D.Empno;

**vii) Drop the table T1.**

DROP Table T1;

**Example 1.14.18** Solve following queries with following table, where underlined attribute is primary key.  
*primery key.*

*Person(SS#, name, address)*

*Car(license, year, model)*

*Accident(date, driver,damage-amount).*

*Owns(SS##,license)*

*Log(licence, date, driver)*

*i) Find the name of person whose license number is '12345'.*

*ii) Display name of driver with number of accidents done by that driver.*

*iii) Add a new accident by 'Ravi' for 'BMW' car on 01/01/2013 for damage amount of 1.5 lakh rupees.*

**Solution:**

**i) Find the name of person whose license number is '12345'.**

SELECT name

FROM Person, Owns, Car

WHERE Person.SS#=Owns.SS# AND

Car.license=Owns.license AND

Car.license='12345';

**ii) Display name of driver with number of accidents done by that driver.**

SELECT driver,COUNT(\*)

FROM Accident

GROUP BY driver

**iii) Add a new accident by 'Ravi' for 'BMW' car on 01/01/2013 for damage amount of 1.5 lakh rupees.**

INSERT INTO Person (SS#, name, address) VALUES (111, 'Ravi', 'Mumbai');

INSERT INTO Car (license, year, model) VALUES ('L111', '2008', 'BMW');

INSERT INTO Owns (SS#, license) VALUES(111, 'L111');

INSERT INTO Accident('01/01/2013', 'Ravi', 150000);

INSERT INTO Log('L111', '01/01/2013','Ravi');

**Example 1.14.19** For Supplier - Parts database

*Supplier(S#, sname, status, city)*

*Parts(P#, pname,color,weight,city)*

*SP(S#, P#, quantity)*

*Answer the following queries in SQL.*

*i) Display the name of supplier who lives in 'Ahmedabad'.*

*ii) Display the parts name which is not supplied yet.*

*iii) Find all suppliers whose status is either 20 or 30. Solution:*

**i) Display the name of supplier who lives in 'Ahmedabad'.**

```
SELECT sname
FROM Supplier
WHERE city='Ahmedabad';
```

**ii) Display the parts name which is not supplied yet.**

```
SELECT pname
FROM Parts, SP
WHERE SP.P# <> Parts.P#
```

**iii) Find all suppliers whose status is either 20 or 30.**

```
SELECT sname
FROM Supplier
WHERE status = 20 or status = 30;
```

**Example 1.14.20** *For Supplier - Parts database*

*Supplier(S#, sname, status, city)*

*Parts(P#, pname,color,weight,city)*

*SP(S#, P#, quantity)*

*Answer the following queries in SQL.*

*i) Find the name of parts having 'Red' colour.*

*ii) Delete parts whose weight is more than 100 gram.*

*iii) Count how many times each supplier has supplied part 'P2'.*

*iv) How much times shipment is for more than 100 quantities?*

**Solution:**

**i) Find the name of parts having 'Red' colour.**

```
SELECT pname
FROM Parts
WHERE color='Red';
```

**ii) Delete parts whose weight is more than 100 gram.**

```
DELETE FROM Parts
WHERE weight>100
```

**iii) Count how many times each supplier has supplied part 'P2'.**

```
SELECT S#, COUNT(*)
FROM SP
WHERE P#='P2'
```

**iv) How much times shipment is for more than 100 quantities?**

```
SELECT S#, COUNT(*)
FROM SP
WHERE quantity>100;
```

**Example 1.14.21** *Consider following schema and write SQL for given statements. Student(RollNo, Name, Age, Sex, City)*

*Student\_marks(RollNo, Sub1, Sub2, Sub3, Total, Average)*

*Write query to*

*i) Display name and city of students whose total marks are greater than 225.*

*ii) Display name of students who got more than 60 marks in each subject.*

*iii) Display name of city from where more than 10 students come from.*

iii) Display a unique pair of male and female students.

**Solution:**

**i) Display name and city of students whose total marks are greater than 225. SELECT name, city**

FROM student, student\_marks

WHERE student.RollNo = student\_marks.RollNo AND student\_marks.Total>225

**ii) Display name of students who got more than 60 marks in each subject. SELECT name**

FROM student, student\_marks

WHERE student.RollNo = student\_marks.RollNo AND

student\_marks.Sub1>60 OR student\_marks.Sub2>60 OR student\_marks.Sub3>60;

**iii) Display name of city from where more than 10 students come from. SELECT city**

FROM student

WHERE count(RollNo)>10

**iv) Display a unique pair of male and female students.**

SELECT S1.name

FROM Student S1, Student S2

WHERE S1.Name=S2.Name AND S1.sex='M' S2.sex='F'

**Example 1.14.22 C Write queries for the following tables:**

*T1 (Empno, Ename, Salary, Designation)*

*T2 (Empno, Deptno.)*

1) Display all the details of the employee whose salary is lesser than 10 K.

2) Display the Deptno in which employee Seeta is working.

3) Add a new column Deptname in table T2.

4) Change the designation of Geeta from 'Manager' to 'Senior Manager'.

5) Find the total salary of all the employees.

6) Display Empno, Ename, Deptno and Deptname

7) Drop the table T1.

**Solution:**

**1) Display all the details of the employee whose salary is lesser than 10 K. SELECT \***

FROM T1

WHERE Salary<10000

**2) Display the Deptno in which employee Seeta is working.**

SELECT Deptno

FROM T2, T1

WHERE T1.Empno=T2.Empno AND T1.Ename='Seeta';

**3) Add a new column Deptname in table T2.**

ALTER TABLE T2

ADD (Deptname VARCHAR(20));

**4) Change the designation of Geeta from 'Manager' to 'Senior Manager'.**

UPDATE T1

SET Designation 'Senior Manager'

WHERE Ename='Geeta';

**5) Find the total salary of all the employees.**

SELECT SUM(Salary)

FROM T1

6) Display Empno, Ename, Deptno and Deptname

SELECT T1.Empno, T1.Ename, T2.Deptno, T2.Deptname

FROM T1, T2

**7) Drop the table T1.**

DROP TABLE T1

**Example 1.14.23** We have following relations:

*EMP(empno, ename, jobtitle, managerno, hiredate, sal, comm, deptno) DEPT(deptno, dname, loc)*

Answer the following queries in SQL.

i) Find the Employees working in the department 10, 20, 30 only.

ii) Find Employees whose names start with letter A or letter a.

iii) Find Employees along with their department name.

iv) Insert data in EMP table.

v) Find the Employees who are working in Smith's department

vi) Update Department name of Department No = 10

vii) Display employees who are getting maximum salary in each department **Solution:**

**i) Find the Employees working in the department 10, 20, 30 only.**

```
SELECT ename
FROM EMP
WHERE deptno BETWEEN 10 AND 30
```

**ii) Find Employees whose names start with letter A or letter a.**

```
SELECT ename
FROM EMP
WHERE ename='A%' OR ename='a%';
```

**iii) Find Employees along with their department name.**

```
SELECT EMP.ename,DEPT.dname
FROM EMP, DEPT
WHERE EMP.deptno=DEPT.deptno;
```

**iv) Insert data in EMP table.**

```
INSERT INTO EMP( empno, ename, jobtitle, managerno, hiredate, sal, comm, deptno)
VALUES ('E111', 'AAA', 'Manager', M123,01-01-2010,20000,2000, 'D111');
```

**v) Find the Employees who are working in Smith's department**

```
SELECT ename
FROM EMP,DEPT
WHERE EMP.deptno=DEPT.deptno AND ename='Smith';
```

**vi) Update Department name of Department No = 10**

```
UPDATE DEPT
SET dname='Accounts'
WHERE deptno=10;
```

**vii) Display employees who are getting maximum salary in each department**

```
SELECT ename, MAX(sal)
FROM EMP
GROUP BY deptno;
```

**Example 1.14.24** Consider following Hotel database, primary keys are underlined:

*hotel(hotel\_no,name,type,price)*

*room(room-no,hotel-no,type,price)*

*booking(hotel-no,guest-no,date-from,date-to,room-no)*

*guest(guest-no,name,address)*

Give an expression in SQL for each of the following queries

(1) List the names and addresses of all guests in London, alphabetically ordered by name.

(2) List out hotel name and total number of rooms available

(3) List the details of all the rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.

(4) List all guests currently staying at the Grosvenor Hotel.

(5) List the rooms that are currently unoccupied at the Grosvenor Hotel.

(6) List the number of rooms in each hotel in London.

(7) List out all guests who have booked room for three or more days.

**Solution:**

**(1) List the names and addresses of all guests in London, alphabetically ordered by name.**

```
SELECT name, address
FROM guest
WHERE address LIKE '%London%'
ORDER BY name;
```

**(2) List out hotel name and total number of rooms available**

```
SELECT name, COUNT(room-no)
FROM hotel, room
WHERE hotel.hotel_no= room.hotel_no
GROUP by hotel no;
```

**(3) List the details of all the rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.**

```
SELECT r.* FROM Room r LEFT JOIN
(SELECT g.guestName, h.hotelNo, b.roomNo
FROM guest g, booking b, hotel h
WHERE g.guest_no = b.guest_no AND
b.hotel_no = h.hotel_no AND
h.name= 'Grosvenor Hotel' AND
date_from <= CURRENT_DATE AND
date_to >= CURRENT DATE)
AS XXX ON r.hotel_no = XXX.hotel_no AND r.room_no = XXX.room_no;
```

**(4) List all guests currently staying at the Grosvenor Hotel.**

```
SELECT FROM guest
WHERE guest_no =
(SELECT guest_no FROM booking
WHERE
date-from <= CURRENT DATE AND date-to >= CURRENT DATE AND
hotel_no =
(SELECT hotel_no FROM hotel
WHERE name = 'Grosvenor Hotel'));
```

**(5) List the rooms that are currently unoccupied at the Grosvenor Hotel.**

```
SELECT (r.hotel_no, r.room_no, r.type, r.price)
FROM room r, hotel h
WHERE r.hotel_no = h.hotel no AND
h.name 'Grosvenor Hotel' AND
NOT EXIST
(SELECT *
FROM booking b, hotel h
WHERE (date from <= 'CURRENT DATE'
AND date_to >= 'CURRENT DATE')
AND r.hotel_no=b.hotel_no
AND r.room_no=b.room_no
AND r.hotel_no=h.hotel_no
```

AND name = 'Grosvenor Hotel');

**(6) List the number of rooms in each hotel in London.**

```
SELECT hotel_no, COUNT(room_no) AS count
FROM room r, hotel h
WHERE r.hotel_no = h.hotel_no AND city = 'London'
GROUP BY hotel_no;
```

**(7) List out all guests who have booked room for three or more days.**

```
SELECT guest-no, name
FROM guest g, booking b
WHERE b.date-to Minus b.date-from >=3
```

**Example 1.14.25** Consider following schema and write SQL for given statements

*employee(employee-name, street, city)*

*works(employee-name, company-name, salary)*

*company(company-name, city)*

*manages(employee-name, manager-name)*

(1) Find the names of all employees who work for First Bank Corporation.

(2) Give all employees of First Bank Corporation a 10-percent raise.

(3) Find the names and cities of residence of all employees who work for First Bank Corporation.

(4) Find the names and Street addresses, cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000

(5) Find all employees in the database who live in the same cities as the companies for which they work.

(6) Find all employees in the database who do not work for First Bank Corporation.

(7) Find the company and number of employees in company that has more than 30 employees.

**Solution:**

**(1) Find the names of all employees who work for First Bank Corporation.**

```
SELECT employee_name
FROM works
WHERE company_name='First Bank Corporation';
```

**(2) Give all employees of First Bank Corporation a 10-percent raise.**

```
UPDATE works
SET salary salary*1.1
WHERE company_name='First Bank Corporation';
```

**(3) Find the names and cities of residence of all employees who work for First Bank Corporation.**

```
SELECT employee_name, city
FROM employee
WHERE employee_name IN
(SELECT employee_name
FROM works
WHERE company_name='First Bank Corporation');
```

**(4) Find the names and Street addresses, cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000**

```
SELECT employee_name, city
FROM employee
WHERE employee_name IN
(SELECT employee_name
FROM works
WHERE company_name='First Bank Corporation' AND salary>10000);
```

**OR**

```

SELECT E.employee_name, E.street, E.city
FROM employee as E, works as W
WHERE
E.employee_name=W.employee_name AND
W.company_name='First Bank Corporation' AND W.salary>10000

```

**(5) Find all employees in the database who live in the same cities as the companies for which they work.**

```

SELECT E.employee_name
FROM employee as E, works as W, company as C
where E.employee_name=W.employee_name AND E.city=C.city AND
W.company_name=C.company_name;

```

**(6) Find all employees in the database who do not work for First Bank Corporation.**

```

SELECT employee_name
FROM works
WHERE company_name <>'First Bank Corporation';

```

**(7) Find the company and number of employees in company that has more than 30 employees**

```

SELECT company-name,COUNT(employee-name)
FROM employee E, company C,works W
WHERE E.employee-name=W.employee-name AND
W.company-name=C.company-name
HAVING COUNT(employee-name)>30;

```

**Example 1.14.26** Consider the following Schema: Suppliers(sid:integer, sname:string, address: string) Parts(pid:integer, pname:string, color: string) Catalog(sid:integer, pid: integer, cost: real)

Write the following queries in SQL

- 1) Find the names of suppliers who supply some red part.
- 2) Find the sids of suppliers who supply some red part and some green part.
- 3) Find the sids of suppliers of supply every red part.
- 4) Find the pids of parts supplied by at least two different suppliers.

**Solution:**

```

1) SELECT S.sname
FROM Suppliers S, Parts P, Catalog C
WHERE P.color = 'red' AND C.pid = P.pid AND C.sid = S.sid

```

```

2) SELECT C.sid
FROM Catalog C, Parts P
WHERE (P.color = 'red' OR P.color = 'green') AND P.pid = C.pid

```

```

3) SELECT C.sid.
FROM catalog C
WHERE NOT EXISTS(SELECT P.pid
FROM Parts P
WHERE P.color='red'
AND (NOT EXISTS (SELECT C1.sid
FROM Catalog C1
WHERE C1.sid = C.sid AND C1.pid = P.pid)))

```

```

4) SELECT C.pids
FROM Catalog C
WHERE EXISTS(SELECT C1.sid
FROM Catalog C1
WHERE C1.pid = C.pid AND C1.sid + C.sid)

```

# Functional Dependencies

## Functional Dependencies

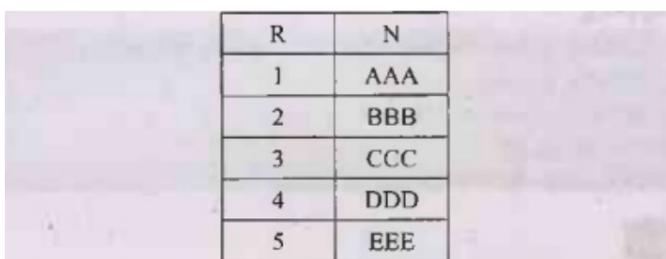
**Definition:** Let P and Q be sets of columns, then: P functionally determines Q, written  $P \rightarrow Q$  if and only if any two rows that are equal on (all the attributes in) P must be equal on (all the attributes in) Q.

In other words, the functional dependency holds if

$T1.P = T2.P$ , then  $T1.Q = T2.Q$

Where notation  $T1.P$  projects the tuple  $T1$  onto the attribute in P.

**For example:** Consider a relation in which the roll of the student and his/her name is stored as follows:



R	N
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 2.8.1 : Table which holds functional dependency i.e.  $R \rightarrow N$

Here,  $R \rightarrow N$  is true. That means the functional dependency holds true here. Because for every assigned RollNumber of student there will be unique name. For instance: The name of the Student whose RollNo is 1 is AAA. But if we get two different names for the same roll number then that means the table does not hold the functional dependency. Following is such table –



R	N
1	AAA
2	BBB
3	CCC
1	XXX
2	YYY

Fig. 2.8.2 : Table which does not hold functional dependency

In above table for RollNumber 1 we are getting two different names - "AAA" and "XXX". Hence here it does not hold the functional dependency.

## Computing Closure Set of Functional Dependency

The closure set is a set of all functional dependencies implied by a given set F. It is denoted by  $F^+$

The closure set of functional dependency can be computed using basic three rules which are also called as Armstrong's Axioms.

These are as follows -

**i) Reflexivity:** If  $XY$ , then  $X \rightarrow Y$

**ii) Augmentation:** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any Z

iii) **Transitivity:** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$

In addition to above axioms some additional rules for computing closure set of functional dependency are as follows -

• **Union:** If  $XY$  and  $X \rightarrow Z$  then  $XYZ$

• **Decomposition:** If  $X \rightarrow YZ$ , then  $XY$  and  $X \rightarrow Z$

**Example 2.8.1** Compute the closure of the following set of functional dependencies for a relation scheme  $R(A,B,C,D,E)$ ,  $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$

**Solution:** Consider  $F$  as follows

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

The closure can be written for each attribute of relation as follows:

•  $(A)^+ =$  **Step 1:**  $\{A\}$  -> the attribute itself

**Step 2:**  $\{ABC\}$  as  $A \rightarrow BC$

**Step 3:**  $\{ABCD\}$  as  $B \rightarrow D$

**Step 4:**  $\{ABCDE\}$  as  $CD \rightarrow E$

**Step 5:**  $\{ABCDE\}$  as  $E \rightarrow A$  and  $A$  is already present

Hence  $(A)^+ = \{ABCDE\}$

•  $(B)^+ =$  **Step 1:**  $\{B\}$

**Step 2:**  $\{BD\}$  as  $B \rightarrow D$

**Step 3:**  $\{BD\}$  as there is no  $BD$  pair on LHS of  $F$

Hence  $(B)^+ = \{BD\}$

•  $(C)^+ =$  **Step 1:**  $\{C\}$

**Step 2:**  $\{C\}$  as there is no single  $C$  on LHS of  $F$

Hence  $(C)^+ = \{C\}$

•  $(D)^+ =$  **Step 1:**  $\{D\}$

**Step 3:**  $\{D\}$  as there is no  $BD$  pair on LHS of  $F$

Hence  $(D)^+ = \{D\}$

•  $(E)^+ =$  **Step 1:**  $\{E\}$

**Step 2:**  $\{EA\}$  as  $E \rightarrow A$

**Step 3:**  $\{EABC\}$  as  $A \rightarrow BC$

**Step 4:**  $\{EABCD\}$  as  $B \rightarrow D$

**Step 5:**  $\{EABCD\}$  as  $CD \rightarrow E$  and  $E$  is already present

By rearranging we get  $\{ABCDE\}$

Hence  $(E)^+ = \{ABCDE\}$

•  $(CD)^+ =$  **Step 1:**  $\{CD\}$

**Step 2:**  $\{CDE\}$

**Step 3:**  $\{CDEA\}$

**Step 4:**  $\{CDEAB\}$

By rearranging we get  $\{ABCDE\}$

Hence  $(CD)^+ = \{ABCDE\}$

**Example 2.8.2** Compute the closure of the following set of functional dependencies for a relation scheme  $R(A,B,C,D,E)$ ,  $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$  and Find the candidate key.

**Solution:** For finding the closure of functional dependencies - Refer example 2.8.1.

We can identify candidate from the given relation schema with the help of functional dependency. For that purpose we need to compute the closure set of attribute. Now we will find out the closure set which can completely identify the relation  $R(A,B,C,D)$ .

Let,  $(A)^+ = \{ABCDE\}$

$(B)^+ = \{BD\}$

$(C)^+ = \{C\}$

$(D)^+ = \{ABCDE\}$

$(E)^+ = \{ABCDE\}$

$(CD)^+ = \{ABCD\}$

Clearly, only (A), (E) and (CD)\* gives us (ABCDE) i.e. complete relation R. Hence these are the candidate keys.

### Canonical Cover or Minimal Cover

**Formal Definition:** A minimal cover for a set F of FDs is a set G of FDs such that:

- 1) Every dependency in G is of the form  $X \rightarrow A$ , where A is a single attribute.
- 2) The *closure*  $F^*$  is equal to the *closure*  $G^*$ .
- 3) If we obtain a set H of dependencies from G by deleting one or more dependencies or by deleting attributes from a dependency in G, then  $F^* \neq H^*$ .

### Concept of Extraneous Attributes

**Definition:** An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies. The formal definition of extraneous attributes is as follows:

Consider a set F of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in F

- Attribute A is extraneous in  $\alpha$  if  $A \in \alpha$  and F logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$
- Attribute A is extraneous in  $\beta$  if  $A \in \beta$  and the set of functional dependencies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$  logically implies F.

### Algorithm for computing Canonical Cover for set of functional Dependencies F

**Fc = F**

**repeat**

Use the union rule to replace any dependencies in Fc of the form

$\alpha_1 \rightarrow \beta_1$  and  $\alpha_1 \rightarrow \beta_2$  and  $\alpha_1 \rightarrow \beta_1\beta_2$

Find a functional dependency  $\alpha \rightarrow \beta$  in Fc with an extraneous attribute either in  $\alpha$  or in  $\beta$

/\* The test for extraneous attributes is done using Fc, not F \*/

If an extraneous attribute is found, delete it from  $\alpha \rightarrow \beta$  in Fc.

until (Fc does not change)

**Example 2.8.3** Consider the following functional dependencies over the attribute set  $R(ABCDE)$  for finding minimal cover  
 $FD = \{A \rightarrow C, AC \rightarrow D, B \rightarrow ADE\}$

**Solution: Step 1:** Split the FD such that R.H.S contain single attribute. Hence we get

$A \rightarrow C$

$AC \rightarrow D$

$B \rightarrow A$

$B \rightarrow D$

$B \rightarrow E$

**Step 2:** Find the redundant entries and delete them. This can be done as follows –

**For  $A \rightarrow C$ :** We find  $(A)^*$  by assuming that we delete  $A \rightarrow C$  temporarily. We get  $esimab (A)^* = \{A\}$ . Thus from A it is not possible to obtain C by deleting  $A \rightarrow C$ . This means we can not delete  $A \rightarrow C$

• **For  $AC \rightarrow D$ :** We find  $(AC)^*$  by assuming that we delete  $AC \rightarrow D$  temporarily. We get  $(AC)^* = (AC)$ . Thus by such deletion it is not possible to obtain D. This means we can not delete  $AC \rightarrow D$

• **For  $B \rightarrow A$ :** We find  $(B)^*$  by assuming that we delete  $B \rightarrow A$  temporarily. We get  $(B)^* = \{BDE\}$ . Thus by such deletion it is not possible to obtain A. This means we can not delete  $B \rightarrow A$

• **For  $B \rightarrow D$ :** We find  $(B)^*$  by assuming that we delete  $B \rightarrow D$  temporarily. We get  $(B)^* = (BEACD)$ . This shows clearly that even if we delete  $B \rightarrow D$  we can obtain D. This means we can delete  $B \rightarrow A$ . Thus it is redundant.

• **For  $B \rightarrow E$ :** We find  $(B)^*$  by assuming that we delete  $B \rightarrow E$  temporarily. We get  $(B)^* = \{BDAC\}$ . Thus by such deletion it is not possible to obtain E. This means we can not delete  $B \rightarrow E$

To summarize we get now

$A \rightarrow C$

$AC \rightarrow D$

$B \rightarrow A$

$B \rightarrow E$

Thus R.H.S gets simplified.

**Step 3:** Now we will simplify L.H.S.

Consider  $AC \rightarrow D$ . Here we can split A and C. For that we find closure set of A and C.

$$(A)^+ = \{AC\}$$

$$(C)^+ = \{C\}$$

Thus C can be obtained from both A as well as C. That also means we need not have to have AC on L.H.S. Instead, only A can be allowed and C can be eliminated. Thus after simplification we get

$$A \rightarrow D$$

To summarize we get now

$$A \rightarrow C$$

$$A \rightarrow D$$

$$B \rightarrow A$$

$$B \rightarrow E$$

Thus L.H.S gets simplified.

**Step 3:** The simplified L.H.S. and R.H.S can be combined together to form

$$A \rightarrow CD$$

$$B \rightarrow AE$$

This is a minimal cover or Canonical cover of functional dependencies.

# Concept of Redundancy and Anomalies

## Concept of Redundancy and Anomalies

**Definition:** Redundancy is a condition created in database in which same piece of data is held at two different places.

Redundancy is at the root of several problems associated with relational schemas.

**Problems caused by redundancy:** Following problems can be caused by redundancy-

i) **Redundant storage:** Some information is stored repeatedly.

ii) **Update anomalies:** If one copy of such repeated data is updated then inconsistency is created unless all other copies are similarly updated.

iii) **Insertion anomalies:** Due to insertion of new record repeated information get added to the relation schema.

iv) **Deletion anomalies:** Due to deletion of particular record some other important information associated with the deleted record get deleted and thus we may lose some other important information from the schema.

**Example:** Following example illustrates the above discussed anomalies or redundancy problems

Consider following Schema in which all possible information about Employee is stored.

EmpID	EName	Salary	DeptID	DeptName	DeptLoc
1	AAA	10000	101	XYZ	Pune
2	BBB	20000	101	XYZ	Pune
3	CCC	30000	101	XYZ	Pune
4	DDD	40000	102	PQR	Mumbai

Redundancy!!!

1) **Redundant storage:** Note that the information about DeptID, DeptName and DeptLoc is repeated.

2) **Update anomalies:** In above table if we change DeptLoc of Pune to Chennai, then it will result inconsistency as for DeptID 101 the DeptLoc is Pune. Or otherwise, we need to update multiple copies of DeptLoc from Pune to Chennai. Hence this is an update anomaly.

3) **Insertion anomalies:** For above table if we want to add new tuple say (5, EEE,50000) for DeptID 101 then it will cause repeated information of (101, XYZ,Pune) will occur.

4) **Deletion anomalies:** For above table, if we delete a record for EmpID 4, then automatically information about the DeptID 102, DeptName PQR and DeptLoc Mumbai will get deleted and one may not be aware about DeptID 102. This causes deletion anomaly.

# Decomposition

## Decomposition

- Decomposition is the process of breaking down one table into multiple tables.
- Formal definition of decomposition is -
- A decomposition of relation Schema R consists of replacing the relation Schema by two relation schema that each contain a subset of attributes of R and together include all attributes of R by storing projections of the instance.
- For example - Consider the following table

Employee\_Department table as follows -

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Hyderabad	25000	D005	Human Resource

We can decompose the above relation Schema into two relation schemas as Employee (Eid, Ename, Age, City, Salary) and Department (Deptid, Eid, DeptName) as follows –

### Employee Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Hyderabad	25000

### Department Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- The decomposition is used for eliminating redundancy.

- **For example:** Consider following relation Schema R in which we assume that the grade determines the salary, the redundancy is caused

### Schema R

Name	eid	deptname	Grade	Salary
AAA	121	Accounts	2	8000
AAA	132	Sales	3	7000
BBB	101	Marketing	4	7000
CCC	106	Purchase	2	8000

**Redundancy!!!**

- Hence, the above table can be decomposed into two Schema S and T as follows:

Schema S			
Name	eid	deptname	Grade
AAA	121	Accounts	2
AAA	132	Sales	3
BBB	101	Marketing	4
CCC	106	Purchase	2

Schema T	
Grade	Salary
2	8000
3	7000
4	7000
2	8000

### Problems Related to Decomposition:

Following are the potential problems to consider:

- 1) Some queries become more expensive.
- 2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
- 3) Checking some dependencies may require joining the instances of the decomposed relations.
- 4) There may be loss of information during decomposition.

### Properties Associated With Decomposition

There are two properties associated with decomposition and those are -

- 1) **Loss-less Join or non Loss Decomposition:** When all information found in the original database is preserved after decomposition, we call it as loss less or non loss decomposition.
- 2) **Dependency Preservation:** This is a property in which the constraints on the original table can be maintained by simply enforcing some constraints on each of the smaller relations.

### Non-loss Decomposition or Loss-less Join

The lossless join can be defined using following three conditions:

- i) Union of attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

$$\text{Att}(R1) \cup \text{Att}(R2) = \text{Att}(R)$$

- ii) Intersection of attributes of R1 and R2 must not be NULL.

$$\text{Att}(R1) \cap \text{Att}(R2) \neq \Phi$$

- iii) Common attribute must be a key for at least one relation (R1 or R2)

$$\text{Att}(R1) \cap \text{Att}(R2) \rightarrow \text{Att}(R1)$$

$$\text{Att}(R1) \cap \text{Att}(R2) \rightarrow \text{Att}(R2)$$

**Example 2.10.1** Consider the following relation  $R(A,B,C,D)$  and FDs  $A \rightarrow BC$ , is the decomposition of R into  $R1(A,B,C)$ ,  $R2(A,D)$ . Check if the decomposition is lossless join or not.

**Solution:**

**Step 1:** Here  $\text{Att}(R1) \cup \text{Att}(R2) = \text{Att}(R)$  i.e  $R1(A,B,C) \cup R2(A,D) = (A,B,C,D)$  i.e R.

**Step 2:** Here  $R1 \cap R2 = \{A\}$ . Thus  $\text{Att}(R1) \cap \text{Att}(R2) \neq \Phi$ . Here the second condition gets satisfied.

**Step 3:**  $\text{Att}(R1) \cap \text{Att}(R2) \rightarrow \{A\}$ . Now  $(A)^* = \{A,B,C\} \in$  attributes of R1. Thus the third condition gets satisfied.

This shows that the given decomposition is a lossless join.

**Example 2.10.2** Consider the following relation  $R(A,B,C,D,E,F)$  and FDs  $A \rightarrow BC$ ,  $C \rightarrow A$ ,  $D \rightarrow E$ ,  $F \rightarrow A$ ,  $E \rightarrow D$  is the decomposition of R into  $R1(A,C,D)$ ,  $R2(B,C,D)$ , and  $R3(E,F,D)$ . Check for lossless.

**Solution:**

**Step 1:**  $R1 \cup R2 \cup R3 = R$ . Here the first condition for checking lossless join is satisfied as  $(A,C,D) \cup (B,C,D) \cup (E,F,D) = \{A,B,C,D,E,F\}$  which is nothing but R.

**Step 2:** Consider  $R1 \cap R2 = \{CD\}$  and  $R2 \cap R3 = \{D\}$ . Hence second condition of intersection not being gets satisfied.

**Step 3:** Now, consider  $R1(A,C,D)$  and  $R2(B,C,D)$ . We find  $R1/R2 = \{CD\}$

$(CD)^* = \{ABCDE\}$  = attributes of R1 i.e.  $\{A,C,D\}$ . Hence condition 3 for checking lossless join for R1 and R2 gets satisfied.

**Step 4:** Now, consider  $R2(B,C,D)$  and  $R3(E,F,D)$ . We find  $R2 \cap R3 = \{D\}$ .  $(D)^* = \{D,E\}$  which is neither complete set of attributes of R2 or R3. [Note that F is missing for being attribute of R3].

Hence it is not lossless join decomposition. Or in other words we can say it is a lossy decomposition.

**Example 2.10.3** Suppose that we decompose schema  $R=(A,B,C,D,E)$  into  $(A,B,C)$   $(C,D,E)$  Show that it is not a lossless decomposition.

**Solution:**

**Step 1:** Here we need to assume some data for the attributes A, B, C, D, and E. Using this data we can represent the relation as follows -

**Relation R**

A	B	C	D	E
a	1	x	p	q
b	2	x	r	s

**Relation R1 = (A,B,C)**

A	B	C
a	1	x
b	2	x

**Relation R2 = (C,D,E)**

C	D	E
x	p	q
x	r	s

**Step 2:** Now we will join these tables using natural join, i.e. the join based on common attribute C. We get  $R1 \bowtie R2$  as

A	B	C	D	E
a	1	x	p	q
a	1	x	r	s
b	2	x	p	q
b	2	x	r	s

Here we get more rows or tuples than original relation R

Clearly  $R1 \bowtie R2 \neq R$ . Hence it is not lossless decomposition.

## Dependency Preservation

• **Definition:** A Decomposition  $D = \{R1, R2, R3, \dots, Rn\}$  of R is dependency preserving for a set F of Functional dependency if  $(F1 \cup F2 \cup \dots \cup Fm) = F$ .

• If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

**Example 2.10.4** Consider the relation R (A, B, C) for functional dependency set  $(A \rightarrow B \text{ and } B \rightarrow C)$  which is decomposed into two relations  $R1 = (A, C)$  and  $R2 = (B, C)$ . Then check if this decomposition dependency preserving or not.

**Solution:** This can be solved in following steps:

**Step 1:** For checking whether the decomposition is dependency preserving or not we need to check following condition

$$F^+ = (F1 \cup F2)^+$$

**Step 2:** We have with us the  $F^+ = \{A \rightarrow B \text{ and } B \rightarrow C\}$

**Step 3:** Let us find  $(F1)^+$  for relation R1 and  $(F2)^+$  for relation R2

R1(A,C)	R2(B,C)
A → A Trivial	B → B Trivial
C → C Trivial	C → C Trivial
A → C ∵ In $(F1)^+$ A → B → C and it is Nontrivial	B → C ∵ In $(F2)^+$ B → C and it is Non-Trivial
AC → AC Trivial	BC → BC Trivial
A → B but is not useful as B is not part of R1 set	We can not obtain C → B
We can not obtain C → A	

**Step 4:** We will eliminate all the trivial relations and useless relations. Hence we can obtain R1 and R2 as,



$(F1 \cup F2)^+ = \{A \rightarrow C, B \rightarrow C\} \neq \{A \rightarrow B, B \rightarrow C\}$  i.e.  $(F)^+$

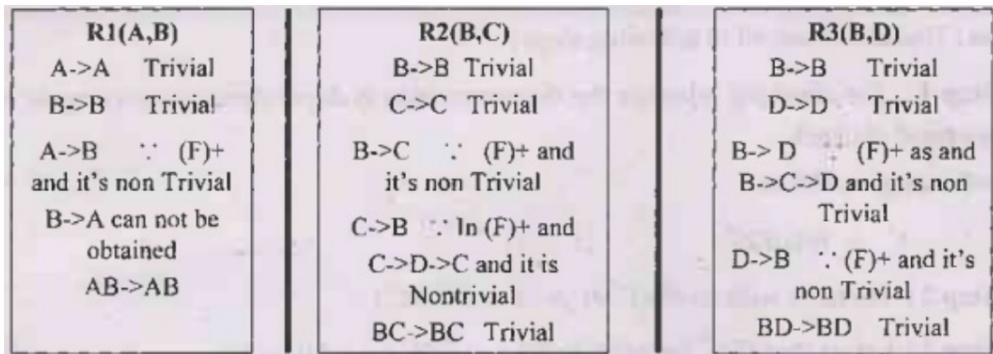
Thus the condition specified in step 1 i.e.  $F^+ = (F1 \cup F2)^+$  is not true. Hence it is not dependency preserving decomposition.

**Example 2.10.5** Let relation  $R(A,B,C,D)$  be a relational schema with following functional dependencies  $(A \rightarrow B, B \rightarrow C, C \rightarrow D, \text{ and } D \rightarrow B)$ . The decomposition of  $R$  into  $(A,B), (B,C)$  and  $(B,D)$ . Check whether this decomposition is dependency preserving or not.

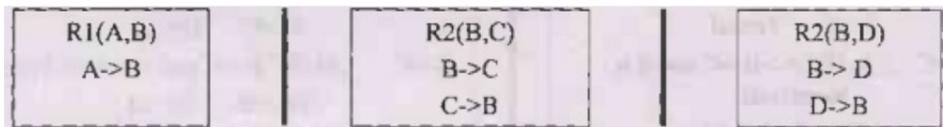
**Solution:**

**Step 1:** Let  $(F) = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$ .

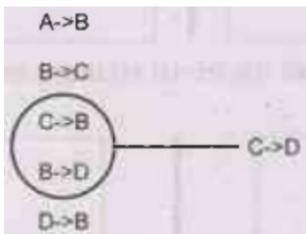
**Step 2:** We will find  $(F1)^+, (F2)^+, (F3)^+$  for relations  $R1(A,B), R2(B,C)$  and  $R3(B,D)$  as follows -



**Step 3:** We will eliminate all the trivial relations and useless relations. Hence we can obtain  $R1 \cup R2 \cup R3$  as,



**Step 4:** As from above FD's we get



**Step 5:** This proves that  $F^+ = (F1 \cup F2 \cup F3)^+$ . Hence given decomposition is dependency preserving.

**Example 2.10.6** Differentiate lossy decomposition and lossless decomposition.

**Solution:**

Sr. No.	Lossy decomposition	Lossless decomposition
1.	In this decomposition, there is loss of data.	In this decomposition, there is no loss of any data.
2.	This technique cannot restore the original amount and quality of the data contained in the original file.	This technique can easily restore the original quality and amount of data in an available file after decompression.
3.	The original size of data reduces after lossy data compression.	The original size of data stays intact after lossless data compression.
4.	This technique can hold more data because it does not need to restore them back to their original form.	This technique can hold less data because it restores the original quality and amount of data.

# Normal Forms

## Normal Forms

• Normalization is the process of reorganizing data in a database so that it meets two basic requirements:

- 1) There is no redundancy of data (all data is stored in only one place), and
- 2) data dependencies are logical (all related data items are stored together)

### Need for normalization

- 1) It eliminates redundant data.
- 2) It reduces chances of data error.
- 3) The normalization is important because it allows database to take up less disk space.
- 4) It also help in increasing the performance.
- 5) It improves the data integrity and consistency.

## First Normal Form

The table is said to be in 1NF if it follows following rules –

- i) It should only have single (atomic) valued attributes/columns.
- ii) Values stored in a column should be of the same domain
- iii) All the columns in a table should have unique names.
- iv) And the order in which data is stored, does not matter.

Consider following Student table

### Student

sid	sname	Phone
1	AAA	11111 22222
2	BBB	33333
3	CCC	44444 55555

As there are multiple values of phone number for sid 1 and 3, the above table is not in 1NF. We can make it in 1NF. The conversion is as follows -

sid	sname	Phone
1	AAA	11111
1	AAA	22222
2	BBB	33333
3	CCC	44444
3	CCC	55555

## Second Normal Form

Before understanding the second normal form let us first discuss the concept of functional dependency and prime and non prime attributes.

### Concept of Partial Functional Dependency

Partial dependency means that a nonprime attribute is functionally dependent on part of a candidate key.

**For example:** Consider a relation  $R(A,B,C,D)$  with functional dependency  $\{AB \rightarrow CD, A \rightarrow C\}$

Here  $(AB)$  is a candidate key because

$$(AB)^+ = \{ABCD\} = \{R\}$$

Hence  $\{A,B\}$  are prime attributes and  $\{C,D\}$  are non prime attribute. In  $A \rightarrow C$ , the non prime attribute  $C$  is dependent upon  $A$  which is actually a part of candidate key  $AB$ . Hence due to  $A \rightarrow C$  we get partial functional dependency.

### Prime and Non Prime Attributes

- **Prime attribute:** An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute:** An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
- **Example :** Consider a Relation  $R = \{A,B,C,D\}$  and candidate key as  $AB$ , the Prime attributes:  $A, B$

Non Prime attributes:  $C, D$

### The Second Normal Form

For a table to be in the Second Normal Form, following conditions must be followed

- i) It should be in the First Normal form.
- ii) It should not have partial functional dependency.

**For example:** Consider following table in which every information about a the Student is maintained in a table such as student id(sid), student name(sname), course id(cid) and course name(cname).

### Student\_Course

sid	sname	cid	cname
1	AAA	101	C
2	BBB	102	C++
3	CCC	101	C
4	DDD	103	Java

This table is not in 2NF. For converting above table to 2NF we must follow the following steps -

**Step 1:** The above table is in 1NF.

**Step 2:** Here sname and sid are associated similarly cid and cname are associated with each other. Now if we delete a record with sid=2, then automatically the course C++ will also get deleted. Thus,

sid  $\rightarrow$  sname or cid  $\rightarrow$  cname is a partial functional dependency, because  $\{sid, cid\}$  should be essentially a candidate key for above table. Hence to bring the above table to 2NF we must decompose it as follows:

### Student:

sid	sname	cid
1	AAA	101
2	BBB	102
3	CCC	101
4	DDD	103

Here candidate key is (sid,cid) and (sid,cid)  $\rightarrow$  sname

### Course:

cid	cname
101	C
102	C++
101	C
103	Java

Here candidate key is cid  
Here cid  $\rightarrow$  cname

Thus now table is in 2NF as there is no partial functional dependency

## Third Normal Form

Before understanding the third normal form let us first discuss the concept of transitive dependency, super key and candidate key.

### Concept of Transitive Dependency

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

X → Z is a transitive dependency if the following functional dependencies hold true:

X → Y

Y → Z

### Concept of Super key and Candidate Key

**Superkey:** A super key is a set or one of more columns (attributes) to uniquely identify rows in a table.

**Candidate key:** The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For example consider following table:

RegID	RollNo	Sname
101	1	AAA
102	2	BBB
103	3	CCC
104	4	DDD

### Superkeys

- {RegID}
- {RegID, RollNo}
- {RegID, Sname}
- {RollNo, Sname}
- {RegID, RollNo, Sname}

### Candidate Keys

- {RegID}
- {RollNo}

### Third Normal Form

A table is said to be in the Third Normal Form when,

- It is in the Second Normal form. (i.e. it does not have partial functional dependency)
- It doesn't have transitive dependency.

Or in other words

In other words 3NF can be defined as: A table is in 3NF if it is in 2NF and for each functional dependency

X → Y

at least one of the following conditions hold :

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

**For example: Consider following table Student\_details as follows -**

sid	sname	zipcode	cityname	state
1	AAA	11111	Pune	Maharashtra
2	BBB	22222	Surat	Gujarat
3	CCC	33333	Chennai	Tamilnadu
4	DDD	44444	Jaipur	Rajasthan
5	EEE	55555	Mumbai	Maharashtra

Here

**Super keys:** {sid}, {sid, sname}, {sid, sname, zipcode}, {sid, zipcode, cityname}... and so on.

**Candidate keys:** {sid}

**Non-Prime attributes:** {sname, zipcode, cityname, state}

The dependencies can be denoted as,

sid → sname

sid → zipcode

zipcode → cityname

cityname → state

The above denotes the transitive dependency. Hence above table is not in 3NF. We can convert it into 3NF as follows:

**Student**

sid	sname	zipcode
1	AAA	11111
2	BBB	22222
3	CCC	33333
4	DDD	44444
5	EEE	55555

### Zip

zipcode	cityname	state
11111	Pune	Maharashtra
22222	Surat	Gujarat
33333	Chennai	Tamilnadu
44444	Jaipur	Rajasthan
55555	Mumbai	Maharashtra

**Example 2.11.1** Consider the relation  $R = \{A, B, C, D, E, F, G, H, I, J\}$  and the set of functional dependencies  $F = \{A, B\} \rightarrow C, A \rightarrow \{D, E\}, B \rightarrow F, F \rightarrow \{G, H\}, D \rightarrow \{I, J\}$

1. What is the key for R? Demonstrate it using the inference rules.
2. Decompose R into 2NF, then 3NF relations.

**Solution: Let,**

$A \rightarrow DE$  (given)

$A \rightarrow D, A \rightarrow E$

$AsD \rightarrow IJ, A \rightarrow IJ$

Using union rule we get

$A \rightarrow DEIJ$

As  $A \rightarrow A$

we get  $A \rightarrow ADEIJ$

Using augmentation rule we compute AB

$AB \rightarrow ABDEIJ$

But

$AB \rightarrow C$  (given)

$AB \rightarrow ABCDEIJ$

$B \rightarrow F$  (given)  $F \rightarrow GH$  ..  $B \rightarrow GH$  (transitivity)

$AB \rightarrow AGH$  is also true

Similarly  $AB \rightarrow AF$   $B \rightarrow F$  (given)

Thus now using union rule

$AB \rightarrow ABCDEFGHIJ$

AB is a key

The table can be converted to 2NF as,

$R_1 = (A, B, C)$

$R_2 = (A, D, E, I, J)$  smoot

$R_3 = (B, F, G, H)$

The above 2NF relations can be converted to 3NF as follows:

$R_1 = (A, B, C)$

$R_2 = (A, D, E)$

$R_1 = (D, I, J)$

$R_1 = (B, E)$

$R_3 = (E, G, H).$

### Review Questions

1. What is database normalization? Explain the first normal form, second normal form and third normal form. **AU: May-18, Marks 13; Dec.-15, Marks 16**
2. What are normal forms. Explain the types of normal form with an example. **AU: Dec.-14, Marks 16**
3. What is normalization? Explain in detail about all Normal forms. **AU: May-19, Marks 13**

## Example on Normalization

### Examples on Normalization

AU: Dec.-19, Marks 9

**Example 2.15.1** Study the relation given below and state what level of normalization can be achieved and normalize it upto that level.

Order no.	Order Date	Item Lines		
		Item Code	Quantity	Price/Unit
1456	26-12-1999	3687	52	50.4
		4627	38	60
		3214	20	20.00
1886	04-03-1999	4629	45	20.25
		4627	30	60.20
1788	04-04-1999	4627	40	60.20

**Solution:**

**Reason for the given relation being unnormalized**

1. Observe order for many items.
2. Item lines has many attributes-called composite attributes.
3. Each tuple has variable length.
4. Difficult to store due to non-uniformity.
5. Given item code difficult to find qty-ordered and hence called Unnormalized relation.

**For conversion to First Normal Form -**

- Identify the composite attributes, convert the composite attributes to individual attributes.
- Duplicate the common attributes as many times as lines in composite attribute.
- Every attribute now describes single property and not multiple properties, some data will be duplicated.
- Now this is called First normal form (1NF) also called flat file.

Order no.	Order Date	Item Lines		
		Item Code	Quantity	Price/Unit
1456	26-12-1999	3687	52	50.4
1456	26-12-1999	4627	38	60
1456	26-12-1999	3214	20	20.00
1886	04-03-1999	4629	45	20.25
1886	04-03-1999	4627	30	60.20
1788	04-04-1999	4627	40	60.20

Fig. 2.15.1 : Table in first normal form

- The above table has insertion, deletion and update anomalies. For instance - if we delete order no. 1886, then the item code 4629 gets lost. Similarly if we update 4627, then all instances of 4627 need to be changed.
- We need to convert 2NF if it is in 1NF. The non-key attributes are functionally dependent on key attribute and if there is a composite key then no non-key attribute is functionally depend on one part of the key.
- The table can be converted to 2NF as follows -

#### Orders

OrderNo	OrderDate
1456	26-12-1999
1886	04-03-1999
1788	04-04-1999

#### Order Details

OrderNo	ItemCode	Qty
1456	3687	52
1886	4629	45
1788	4627	40

#### Prices

ItemCode	Price/Unit
3687	50.4
4627	60
3214	20
4629	20.25

**Example 2.15.2** A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise:

- Employee Number
- Employee Name
- Date of Birth
- Department Code
- Department Name Project Code
- Project Description
- Project Supervisor

Assume the following:

- Each employee number is unique.
- Each department has a single department code.
- Each project has a single code and supervisor.
- Each employee may work on one or more projects.
- Employee names need not necessarily be unique.
- Project Code, Project Description and Project Supervisor are repeating fields.

Normalise this data to Third Normal Form.

#### Solution:

##### Un-Normalized Form

Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description, Project Supervisor

**1NF**

Employee Number, Employee Name, Date of Birth

Department Code, Department Name

Employee Number, Project Code, Project Description, Project Supervisor

**2NF**

Employee Number, Employee Name, Date of Birth, Department Code, Department Name Employee Number, Project Code,

Project Code, Project Description, Project Supervisor

**3NF**

Employee Number, Employee Name, Date of Birth, Department Code

Department Code, Department Name

Employee Number, Project Code

Project Code, Project Description, Project Supervisor

**Example 2.15.3** What is normalization? Normalize below given relation upto 3NF STUDENT.

StudID	StudName	City	Pincode	ProjectID	ProjectName	Course	Content
S101	Ajay	Surat	326201	P101	Health	Programming	C++, Java, C
S102	Vijay	Pune	325456	P102	Social	WEB	HTML, PHP, ASP

**Solution:** For converting the given schema to first normal form, we will arrange it in such a way that have each tuple contains single record. For that purpose we need to split the schema into two tables namely Student and Projects

**1NF**

**Student**

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

**Projects**

StudID	ProjectID	ProjName	Course	Content
S101	P101	Health	Programming	C++
S101	P101	Health	Programming	Java
S101	P101	Health	Programming	C
S102	P102	Social	WEB	HTML
S102	P102	Social	WEB	PHP
S102	P102	Social	WEB	ASP

**2NF**

For a table to be in 2NF, there should not be any partial dependency.

**Student**

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

**Project**

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

### CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
C103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

**3NF:** There was a transitive dependency in 2NF tables because city is associated with student ID and city depends upon zip code. Hence the transitive dependency is removed to convert table into 3NF. The required 3NF schema is as below -

### Student

StudID	StudName	Pincode
S101	Ajay	326201
S102	Vijay	325456

### Student\_Address

Pincode	City
326201	Surat
325456	Pune

### Project

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

### CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
wC103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

**Example 2.15.4** Consider table  $R(A,B,C,D,E)$  with FDs as  $A \rightarrow B$ ,  $BC \rightarrow E$ , and  $ED \rightarrow A$ . The table is in which normal form? Justify your answer.

### Solution:

**Step 1:** We will first find out the candidate keys for given relation R

$$(ACD)^+ = \{A,B,C,D,E\}$$

$$(BCD)^+ = \{A, B, C, D, E\}$$

$$(CDE)^+ = \{A, B, C, D, E\}$$

**Step 2:** Let  $A \rightarrow B$ , the ACD is candidate key and A is a partial key, B is a prime attribute (i.e. it is also part of candidate key). Hence  $A \rightarrow B$  is not a partial functional dependency.

Similarly in  $BC \rightarrow E$  and  $ED \rightarrow A$ ,

E and A are prime-attributes and hence both are not partial functional dependencies. Hence R is in 2NF.

**Step 3:** According to 3NF, every non-prime attribute must be dependent on the candidate key.

In the given functional dependencies, all dependent attributes are prime-attributes. Hence the relation R is in 3NF.

**Step 4:** For R being in BCNF for  $X \rightarrow Y$  the X should be candidate key or super key.

The table is not in BCNF, none of A, BC and ED contain a key.

**Example 2.15.5** Prove the statement "Every relation which is in BCNF is in 3NF but the converse is not true".

**Solution:** For a relations to be in 3NF

A table is said to be in the Third Normal Form when,

- i) It is in the Second Normal form. (i.e. it does not have partial functional dependency)
- ii) It doesn't have transitive dependency.

Or in other words

In other words 3NF can be defined as: A table is in 3NF if it is in 2NF and for each functional dependency

$$X \rightarrow Y$$

At least one of the following conditions hold:

- iii) X is a super key of table
- iv) Y is a prime attribute of table

For a relation to be in BCNF

- (1) It should be in 3NF
- (2) A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

For proving that the table can be in 3NF but not in BCNF consider following relation R(Student, Subject, Teacher) . Consider following are FDs

**(Subject, Student)  $\rightarrow$  Teacher**

Because subject and student combination gives unique teacher. A

**Teacher  $\rightarrow$  Subject**

Because each teacher teaches only Subject.D.

**(Teacher, Student)  $\rightarrow$  Subject**

- So, this relation is in 3NF as every non-key attribute is non-transitively fully dependent on the primary key.
- But it is not in BCNF. Because this is a case of overlapping of candidate keys because there are two composite candidate keys:

- (Subject, Student)
- (Teacher, Student)

And Student is a common attribute in both the candidate keys.

So we need to normalize the above table to BCNF. For that purpose we must set Teacher to be a candidate key as TME of

The decomposition of above takes place as follows

R1(Student, Teacher)

R2(Teacher, Subject)

Now table is in 3NF, as well as in BCNF.

This show that the relation Every relation which is in BCNF is in 3NF but the converse is not true.

**Example 2.15.6** Consider the relation  $R = \{A, B, C, D, E, F, G, H, I, J\}$  and the set of functional dependencies  $F = \{A, B\} - C, A(D, E), B \rightarrow F, F \{G, H), D \rightarrow \{I, J\}$

1. What is the key for R? Demonstrate it using the inference rules.
2. Decompose R into 2NF, then 3NF relations.

**Solution: Let,**

$A \rightarrow DE$  (given)

$A \rightarrow D, A \rightarrow E$  (decomposition rule)

As  $D \rightarrow IJ, A \rightarrow IJ$

Using union rule we get

$A \rightarrow DEIJ$

As  $A \rightarrow A$

we get  $A \rightarrow ADEIJ$

Using augmentation rule we compute AB

$AB \rightarrow ABDEIJ$

But

$AB \rightarrow C$  (given)

$AB \rightarrow ABCDEIJ$

$B \rightarrow F$  (given)  $F \rightarrow GH$   $B \rightarrow GH$  (transitivity)

$AB \rightarrow AGH$  is also true

Similarly  $AB \rightarrow AF$   $B \rightarrow F$  (given)

Thus now using union rule

$AB \rightarrow ABCDEFGHIJ$

AB is a key

The table can be converted to 2NF as

$R_1 = (A, B, C)$

$R_2 = (A, D, E, I, J)$

$R_3 = (B, F, G, H)$

The above 2NF relations can be converted to 3NF as follows:

$R_1 = (A, B, C)$

$R_2 = (A, D, E)$

$R_3 = (D, I, J)$

$R_4 = (B, F)$

$R_5 = (E, G, H)$ .

**Example 2.15.7** Consider a relation  $R(ABC)$  with following FD  $A \rightarrow B, B \rightarrow C$  and  $C \rightarrow A$ . What is the normal form of R?

**Solution:**

**Step 1:** We will find the candidate key

$(A)^+ = \{ABC\} = R$

$(B)^+ = \{ABC\} = R$

$(C)^+ = \{ABC\} = R$

Hence A, B and C all are candidate keys

**Prime attributes** =  $\{A, B, C\}$

**Non prime attribute**  $\{\}$

**Step 2:** For R being in BCNF for  $X \rightarrow Y$  the X should be candidate key or super key. From above FDs

- Consider  $A \rightarrow B$  in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider  $B \rightarrow C$  in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider  $C \rightarrow A$  in which C is a candidate key or super key. Condition for BCNF is satisfied.

This shows that the given relation R is in BCNF.

**Example 2.15.8** Consider the Table 2.15.1 and answer to queries given below.

Userid	U_Email	Fname	Lname	City	State	Zip
MA12	mani@ymail.com	Manish	Jain	Bilaspur	Chhattisgarh	458991
PO45	pujag@gmail.com	Pooja	Magg	Kacch	Gujrat	832212
LA33	lavle98@jj.com	Lavleen	Dhalla	Raipur	Chhattisgarh	853578
CH99	checki9j@ih.com	Chimal	Bedi	Trichy	Tamil Nadu	632011
DA74	danu58@g.com	Dany	James	Trichy	Tamil Nadu	645018

Table 2.15.1 User\_personal

- 1) In this table in first normal form - 1NF? Justify and normalize to 1NF if needed.
- 2) Is this table in second normal form - 2NF? Justify and normalize to 2NF if needed.
- 3) Is User\_personal in third normal form - 3NF? Justify and normalize to 3NF if needed.

**Solution:**

1) All the rows contain only one atomic value.

Hence table is in 1NF.

2) For identifying if table is in 2NF, we must check two rules -

**Rule 1:** The table must be in 1NF.

**Rule 2:** There should not be any partial key dependency.

As we know, that table is in 1NF, Rule 1 is said to be satisfied.

For checking Rule 2, first find out the primary keys.

Assume that, User\_id and zip are to primary keys.

F = {User\_id → U\_Email Fname Lname City State Zip

Zip → City State

}

Note that Userid can uniquely identify all the attributes of given relation. There is no partial dependency for identifying all the attributes. Hence rule 2 is said to be fulfilled. Therefore table is in 2NF.

3) To verify 3NF, the conditions are -

**Rule 1:** Table should be in 2NF

**Rule 2:** There should not be transitive dependency in the table. The table is already in 2NF, hence rule 1 is already satisfied.

Given table shows transitive dependency. It is as follows:

Userid → Zip and Zip → City State

To bring the relation in 3NF, we have to decompose table into two tables

User\_personal (Userid, U\_Email, Fname, Lname, Zip)

Address (Zip, City, State)

The underlined fields are primary keys of respective tables. The tables are as follows:

Userid	U_Email	Fname	Lname	Zip
MA12	mani@ymail.com	Manish	Jain	458991
PO45	pujag@gmail.com	Pooja	Magg	832212
LA33	lavle98@jj.com	Lavleen	Dhalla	853578
CH99	checki9j@ih.com	Chimal	Bedi	632011
DA74	danu58@g.com	Dany	James	645018

**User\_personal table**

Zip	City	State
458991	Bilaspur	Chhattisgarh
832212	Kacch	Gujrat
853578	Raipur	Chhattisgarh
632011	Trichy	Tamil Nadu
645018	Trichy	Tamil Nadu

**Address Table**

**Example 2.15.9** What is the difference between 3NF and BCNF?

**Solution:**

Sr. No.	3NF	BCNF
1.	3NF stands for third normal form.	BCNF stands for Boyce Codd normal form.
2.	The table is in 3NF if it is in 2NF and for each functional dependency $X \rightarrow Y$ at least following condition hold : (i) X is a superkey, (ii) Y is prime attribute of table.	The table is in BCNF if it is in 3rd normal form and for each relation $X \rightarrow Y$ X should be super key.
3.	3NF can be obtained without sacrificing all dependencies.	Dependencies may not be preserved in BCNF.
4.	Lossless decomposition can be achieved in 3NF.	Lossless decomposition is hard to obtain in BCNF.
5.	3NF can be achieved without losing any information from the old table.	For obtaining BCNF we may lose some information from old table.

## Boyce / Codd Normal Form (BCNF)

### Boyce / Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF.

A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

Or in other words,

For a table to be in BCNF, following conditions must be satisfied:

- i) R must be in 3rd Normal Form
- ii) For each functional dependency ( $X \rightarrow Y$ ), X should be a super Key. In simple words if Y is a prime attribute then X can not be non prime attribute.

For example - Consider following table that represents that a Student enrollment for the course -

#### Enrollment Table

sid	Course	Skill
1	C	English
	C++	German
2	Java	English
		French

From above table following observations can be made:

- One student can enroll for multiple courses. For example student with sid=1 can enroll for C as well as Java.
- For each course, a teacher is assigned to the student.
- There can be multiple teachers teaching one course for example course C can be taught by both the teachers namely - Ankita and Archana.
- The candidate key for above table can be (sid,course), because using these two columns we can find
- The above table holds following dependencies,
  - (sid,course)->Teacher
  - Teacher->courseann
- The above table is not in BCNF because of the dependency teacher->course. Note NS that the teacher is not a superkey or in other words, teacher is a non prime attribute and course is a prime attribute and non-prime attribute derives the prime attribute.
- To convert the above table to BCNF we must decompose above table into Student and Course tables

#### Student

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

## Course

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Now the table is in BCNF

**Example 2.12.1** Consider a relation  $(A,B,C,D)$  having following FDs.  $(AB \rightarrow C, AB \rightarrow D, C \rightarrow A, B \rightarrow D)$ . Find out the normal form of  $R$ .

**Solution:**

**Step 1:** We will first find out the candidate key from the given FD.

$$(AB)^+ = \{ABCD\} = R$$

$$(BC)^+ = \{ABCD\} = R$$

$$(AC)^+ = \{AC\} R$$

There is no involvement of  $D$  on LHS of the FD rules. Hence  $D$  can not be part of any candidate key. Thus we obtain two candidate keys  $(AB)^*$  and  $(BC)^*$ . Hence

**prime attributes** =  $\{A,B,C\}$

**Non prime attributes** =  $\{D\}$

**Step 2:** Now, we will start checking from reverse manner, that means from BCNF, so then 3NF, then 2NF.

**Step 3:** For  $R$  being in BCNF for  $X \rightarrow Y$  the  $X$  should be candidate key or super key. From above FDs consider  $C \rightarrow D$  in which  $C$  is not a candidate key or super key. Inabu Hence given relation is not in BCNF.

**Step 4:** For  $R$  being in 3NF for  $X \rightarrow Y$  either i) the  $X$  should be candidate key or super key or ii)  $Y$  should be prime attribute. (For prime and non prime attributes refer step 1)

- For  $AB \rightarrow C$  or  $AB \rightarrow D$  the  $AB$  is a candidate key. Condition for 3NF is satisfied.
- Consider  $C \rightarrow A$ . In this FD the  $C$  is not candidate key but  $A$  is a prime attribute. Condition for 3NF is satisfied.
- Now consider  $B \rightarrow D$ . In this FD, the  $B$  is not candidate key, similarly  $D$  is not a prime attribute. Hence condition for 3NF fails over here.

Hence given relation is not in 3NF.

**Step 5:** For  $R$  being in 2NF following condition should not occur.

Let  $X \rightarrow Y$ , if  $X$  is a proper subset of candidate key and  $Y$  is a non prime attribute. This is a case of partial functional dependency.

For relation to be in 2NF there should not be any partial functional dependency.

- For  $AB \rightarrow C$  or  $AB \rightarrow D$  the  $AB$  is a complete candidate key. Condition for 2NF is satisfied.
- Consider  $C \rightarrow A$ . In this FD the  $C$  is not candidate key. Condition for 2NF is satisfied.
- Now consider  $B \rightarrow D$ . In this FD, the  $B$  is a part of candidate key ( $AB$  or  $BC$ ), similarly  $D$  is not a prime attribute. That means partial functional dependency occurs here. Hence condition for 2NF fails over here.

Hence given relation is not in 2NF.

Therefore we can conclude that the given relation  $R$  is in 1NF.

**Example 2.12.2** Consider a relation  $R(ABC)$  with following FD  $A \rightarrow B, B \rightarrow C$  and  $C \rightarrow A$ . What is the normal form of  $R$ ?

**Solution:**

**Step 1:** We will find the candidate key

$$(A)^+ = \{ABC\} = R$$

$$(B)^+ = \{ABC\} = R$$

$$(C)^+ = \{ABC\} = R$$

Hence A, B and C all are candidate keys

**Prime attributes** = {A,B,C}

**Non prime attribute** {}

**Step 2:** For R being in BCNF for  $X \rightarrow Y$  the X should be candidate key or super key. From above FDs

- Consider  $A \rightarrow B$  in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider  $B \rightarrow C$  in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider  $C \rightarrow A$  in which C is a candidate key or super key. Condition for BCNF is satisfied.

This shows that the given relation R is in BCNF.

**Example 2.12.3** Prove that any relational schema with two attributes is in BCNF.

**Solution:** Here, we will consider  $R = \{A, B\}$  i.e. a relational schema with two attributes. Now various possible FDs are  $A \rightarrow B$ ,  $B \rightarrow A$ .

From the above FDs

- Consider  $A \rightarrow B$  in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider  $B \rightarrow A$  in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider both  $A \rightarrow B$  and  $B \rightarrow A$  with both A and B is candidate key or super key. Condition for BCNF is satisfied.
- No FD holds in relation R. In this {A,B} is candidate key or super key. Still condition for BCNF is satisfied.

This shows that any relation R is in BCNF with two attributes.

# Transaction Concepts

## Transactions

### Syllabus

*Transaction Concepts - ACID Properties - Schedules - Serializability - Transaction support in SQL - Need for Concurrency - Concurrency control - Two Phase Locking- Timestamp - Multiversion Validation and Snapshot isolation - Multiple Granularity locking - Deadlock Handling - Recovery Concepts - Recovery based on deferred and immediate update - Shadow paging - ARIES Algorithm.*

## Part I: Introduction to Transactions

### Transaction Concepts

**AU: Dec.-14, Marks 4**

**Definition:** A transaction can be defined as a group of tasks that form a single logical unit.

**For example** - Suppose we want to withdraw 100 from an account then we will follow following operations:

- 1) Check account balance
- 2) If sufficient balance is present request for withdrawal.
- 3) Get the money
- 4) Calculate Balance = Balance -100
- 5) Update account with new balance.

The above mentioned four steps denote one transaction.

In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

### Review Question

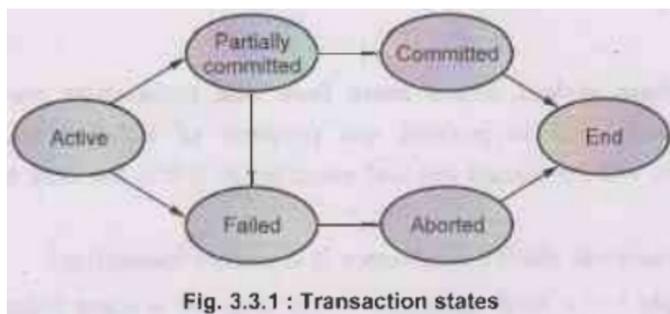
1. Write a short note on - Transaction Concept. **AU: Dec- 14,Mark 4**

## Transaction States

### Transaction States

AU: May-14,18, Dec.-11,19, Marks 8

Each transaction has following five states:



**1) Active:** This is the first state of transaction. For example: insertion, deletion or updation of record is done here. But data is not saved to database.

**2) Partially Committed:** When a transaction executes its final operation, it is said to be in a partially committed state.

**3) Failed:** A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

**4) Aborted:** If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction.

**5) Committed:** If a transaction executes all its operations successfully, it is said to be committed. This is the last step of a transaction, if it executes without fail.

**Example 3.3.1** Define a transaction. Then discuss the following with relevant examples:

1. A read only transaction 2. A read write transaction 3. An aborted transaction

**Solution:**

**(1) Read only transaction**

T1
Read(A)
Read(B)
Display(A-B)

**(2) A read write transaction**

T1
Read(A)
A=A+100
Write(A)

**(3)**

T1	T2	
Read(A)		Assume A=100
A=A+50		A=150
Write(A)		
	Read(A)	A=150
	A=A+100	A=250
RollBack		A=100 (restore back to original value which is before Transaction T1)
	Write(A)	

**Review Questions**

1. During execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which transaction may pass. Explain why each state transaction may occur? **AU: May-18, Marks 6**
2. With a neat sketch explain the states of transaction. **AU: May-14, Marks 8**

## ACID Properties

AU: May 14,18,19, Dec- 19,Mark 15

### 1) Atomicity:

This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.

- No transaction in the database is left half completed.
- Database should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.
- For example In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

### 2) Consistency:

- The database must remain in consistent state after performing any transaction.
- For example: In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

### 3) Isolation:

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- For example: If a bank manager is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

### 4) Durability:

- The database should be strong enough to handle any system failure.
- If there is any set of insert /update, then it should be able to handle and commit to the database.
- If there is any failure, the database should be able to recover it to the consistent state.
- For example: In ATM withdrawal example, if the system failure happens after Customer getting t should be strong enough to update Database with his new balance, after system recovers. For that keep the log of each transaction and its failure. So when the system recovers, it should be able to failed and if there is any pending transaction, then it should be updated to Database.

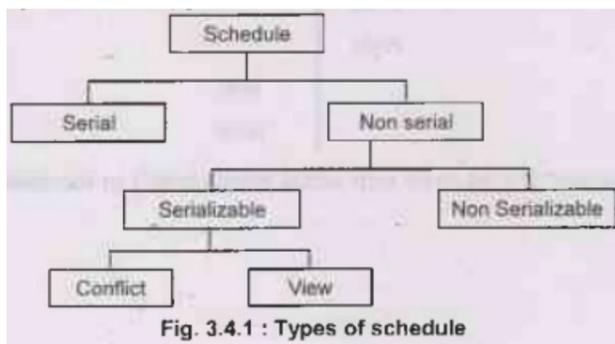
## Review Questions

1. Explain with an example the properties that must be satisfied by transaction. AU: May-18, Marks 7
2. Explain the ACID properties of transaction AU: May-14, Marks 8
3. Discuss in detail about the ACID properties of transaction. AU: May-19, Marks 15
4. Discuss the properties of a transaction that ensure integrity of data in the database system. AU: Dec 19, Marks 4

# Schedules

## Schedules

Schedule is an order of multiple transactions executing in concurrent environment. Following figure represents the types of schedules.



**Serial schedule:** The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions  $T_1$  and  $T_2$

$T_1$	$T_2$
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction  $T_1$  on data items A and then B executes and then in transaction  $T_2$  all the operations on data items A and B execute. The R stands for Read operation and W stands for write operation.

**Non serial schedule:** The schedule in which operations present within the transaction are intermixed. This may lead to conflicts in the result or inconsistency in the resultant data. For example-

Consider following two transactions,

$T_1$	$T_2$
R(A)	
W(A)	
	R(A)
	W(B)
R(A)	
W(B)	
	R(B)
	W(B)

The above transaction is said to be non serial which result in inconsistency or conflicts in the data.

# Serializability



**Serializability**

- When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transaction).
- Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.
- For example:

T <sub>1</sub>	A	B	T <sub>2</sub>
Initial Value	100	100	
A=A-10			
W(A)			
B=B+10			
W(B)			
	90	110	
			A=A-10
			W(A)
	80	110	

• In above transaction initially T<sub>1</sub> will read the values from database as A= 100, B= 100 and modify the values of A and B, transaction T<sub>2</sub> will read the modified value i.e. 90 and will modify it to 80 and perform write operation. Thus at the end of transaction T<sub>1</sub> value of A will be 90 but at end of transaction T<sub>2</sub> value of A will be 80. Thus conflicts or inconsistency occurs here. This sequence can be converted to a sequence which may give us consistent result. This process is called serializability.

**Difference between Serial schedule and Serializable schedule**

Serial schedule	Serializable schedule																																						
No concurrency is allowed in serial schedule.	Concurrency is allowed in serializable schedule.																																						
In serial schedule, if there are two transactions executing at the same time and no interleaving of operations is permitted, then following can be the possibilities of execution – (i) Execute all the operations of transactions T1 in a sequence and then execute all the operations of transactions T2 in a sequence. (ii) Execute all the operations of transactions T2 in a sequence and then execute all the operations of transactions T1 in a sequence.	In serializable schedule, if there are two transactions executing at the same time and interleaving of operations is allowed there can be different possible orders of executing an individual operation of the transactions.																																						
<b>Example of serial schedule</b>	<b>Example of serializable schedule</b>																																						
<table border="1"> <thead> <tr> <th>T1</th> <th>T2</th> </tr> </thead> <tbody> <tr><td>Read(A)</td><td></td></tr> <tr><td>A=A-50</td><td></td></tr> <tr><td>Write(A)</td><td></td></tr> <tr><td>Read(B)</td><td></td></tr> <tr><td>B=B+100</td><td></td></tr> <tr><td>Write(B)</td><td></td></tr> <tr><td></td><td>Read(A)</td></tr> <tr><td></td><td>A=A+10</td></tr> <tr><td></td><td>Write(A)</td></tr> </tbody> </table>	T1	T2	Read(A)		A=A-50		Write(A)		Read(B)		B=B+100		Write(B)			Read(A)		A=A+10		Write(A)	<table border="1"> <thead> <tr> <th>T1</th> <th>T2</th> </tr> </thead> <tbody> <tr><td>Read(A)</td><td></td></tr> <tr><td>A=A-50</td><td></td></tr> <tr><td>Write(A)</td><td></td></tr> <tr><td></td><td>Read(B)</td></tr> <tr><td></td><td>B=B+100</td></tr> <tr><td></td><td>Write(B)</td></tr> <tr><td>Read(B)</td><td></td></tr> <tr><td>Write(B)</td><td></td></tr> </tbody> </table>	T1	T2	Read(A)		A=A-50		Write(A)			Read(B)		B=B+100		Write(B)	Read(B)		Write(B)	
T1	T2																																						
Read(A)																																							
A=A-50																																							
Write(A)																																							
Read(B)																																							
B=B+100																																							
Write(B)																																							
	Read(A)																																						
	A=A+10																																						
	Write(A)																																						
T1	T2																																						
Read(A)																																							
A=A-50																																							
Write(A)																																							
	Read(B)																																						
	B=B+100																																						
	Write(B)																																						
Read(B)																																							
Write(B)																																							

- There are two types of serializabilities: conflict serializability and view serializability

### Conflict Serializability

• **Definition:** Suppose  $T_1$  and  $T_2$  are two transactions and  $I_1$  and  $I_2$  are the instructions in  $T_1$  and  $T_2$  respectively. Then these two transactions are said to be conflict Serializable, if both the instruction access the data item  $d$ , and at least one of the instruction is write operation.

• **What is conflict?:** In the definition three conditions are specified for a conflict in conflict serializability -

- 1) There should be different transactions
- 2) The operations must be performed on same data items
- 3) One of the operation must be the Write(W) operation

• We can test a given schedule for conflict serializability by constructing a precedence graph for the schedule, and by searching for absence of cycles in the graph.

• Precedence graph is a directed graph, consisting of  $G=(V,E)$  where  $V$  is set of vertices and  $E$  is set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges  $T_i \rightarrow T_j$  for which one of three conditions holds :

1.  $T_i$  executes write(Q) before  $T_j$  executes read(Q).
2.  $T_i$  executes read(Q) before  $T_j$  executes write(Q).
3.  $T_i$  executes write(Q) before  $T_j$  executes write(Q).

• A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

### Testing for serializability

Following method is used for testing the serializability: To test the conflictserializability we can draw a graph  $G = (V,E)$  where  $V$  = vertices which represent the number of transactions.  $E$  = edges for conflicting pairs.

**Step 1:** Create a node for each transaction.

**Step 2:** Find the conflicting pairs(RW, WR, WW) on the same variable(or data item) by different transactions.

**Step 3:** Draw edge for the given schedule. Consider following cases

1.  $T_i$  executes write(Q) before  $T_j$  executes read(Q), then draw edge from  $T_i$  to  $T_j$ .

2.  $T_i$  executes read(Q) before  $T_j$  executes write(Q), then draw edge from  $T_i$  to  $T_j$

3.  $T_i$  executes write(Q) before  $T_j$  executes write(Q), then draw edge from  $T_i$  to  $T_j$

**Step 4:** Now, if precedence graph is cyclic then it is a non conflict serializable schedule and if the precedence graph is acyclic then it is conflict serializable schedule.

**Example 3.5.1** Consider the following two transactions and schedule (time goes from top to bottom). Is this schedule conflict-serializable? Explain why or why not.

$T_1$	$T_2$
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	

**Solution :**

**Step 1:** To check whether the schedule is conflict serializable or not we will check from top to bottom. Thus we will start reading from top to bottom as

$T_1: R(A) \rightarrow T_1: W(A) \rightarrow T_2: R(A) \rightarrow T_2: R(B) \rightarrow T_1: R(B) \rightarrow T_1: W(B)$

**Step 2:** We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them-

- i) Both the operations belong to different transactions.
- ii) Both the operations are on same data item.
- iii) At least one of the two operations is a write operation

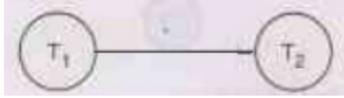
From above given example in the top to bottom scanning we find the conflict as  $T_1: W(A) \rightarrow T_2: R(A)$ .

- i) Here note that there are two different transactions  $T_1$  and  $T_2$ ,
- ii) Both work on same data item i.e. A and
- iii) One of the operation is write operation.

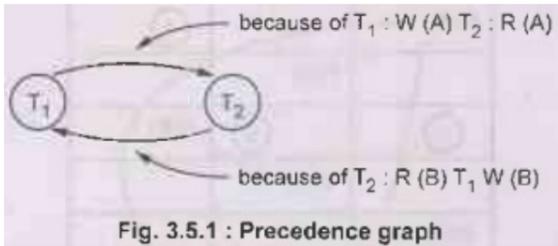
**Step 3:** We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions, there will be two nodes namely  $T_1$  and  $T_2$



**Step 4:** Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from  $T_1:W(A)$  to  $T_2:R(A)$ . Hence edge must be from  $T_1$  to  $T_2$ .



**Step 5:** Repeat the step 4 while reading from top to bottom. Finally the precedence graph will be as follows



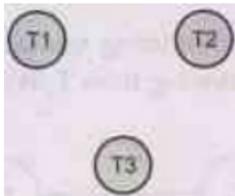
**Step 6:** Check if any cycle exists in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the is cycle found then schedule is not conflict serializable. In the step 5 we get a graph with cycle, that means given schedule is not conflict serializable.

**Example 3.5.2** Check whether following schedule is conflict serializable or not. If it is not conflict serializable then find the serializability order.

$T_1$	$T_2$	$T_3$
R(A)		
	R(B)	
		R(B)
	W(B)	
W(A)		
		W(A)
	R(A)	
	W(A)	

**Solution:**

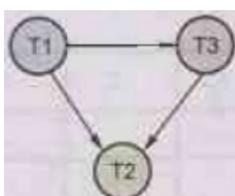
**Step 1:** We will read from top to bottom, and build a precedence graph for conflicting entries. We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are three transactions, there will be two nodes namely T1 T2, and T3



**Step 2:** The conflicts are found as follows –

T1	T2	T3
R(A)		I
	R(B)	
III	II	R(B)
	W(B)	
W(A)		W(A)
IV	R(A)	
	W(A)	

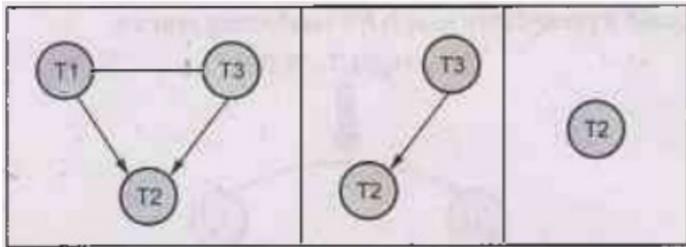
**Step 3:** The precedence graph will be as follows –



**Step 4:** As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

**Step 5:** A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

**Step 6:** Find the vertex which has no incoming edge which is T1. If we delete T1 node then T3 is a node that has no incoming edge. If we delete T3, then T2 is a node that has no incoming edge.



Thus the nodes can be deleted in a order T1, T3 and T2. Hence the order will be T1-T3-T2

**Example 3.5.3** Check whether the below schedule is conflict serializable or not.  
 $\{B2, r_2(X), b_1, r_1(X), W_1(X), r_1(Y), W_1(Y), W_2(X), e_1, C_1, e_2, C_2\}$

**Solution:** b2 and b1 represents begin transaction 2 and begin transaction 1. Similarly, e1 and e2 represents end transaction 1 and end transaction 2.

We will rewrite the schedule as follows-

T <sub>1</sub>	T <sub>2</sub>
	r <sub>2</sub> (X)
r <sub>1</sub> (X)	
W <sub>1</sub> (X)	
r <sub>1</sub> (Y)	
W <sub>1</sub> (Y)	
	W <sub>2</sub> (X)

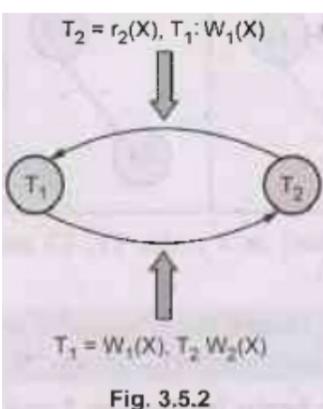
**Step 1:** We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

- i) Both the operations belong to different transactions.
- ii) Both the operations are on same data item.
- iii) At least one of the two operations is a write operation.

The conflicting entries are as follows –

T <sub>1</sub>	T <sub>2</sub>
	R <sub>2</sub> (X)
R <sub>1</sub> (X)	
W <sub>1</sub> (X)	
R <sub>1</sub> (Y)	
W <sub>1</sub> (Y)	
	W <sub>2</sub> (X)

**Step 2:** Now we build a precedence graph for conflicting entries.



As there are two transactions only two nodes are present in the graph.

**Step 3:** We get a graph with cycle, that means given schedule is not conflict serializable.

**Example 3.5.4** Consider the three transactions  $T1$ ,  $T2$ , and  $T3$  and schedules  $S1$  and  $S2$  given below. Determine whether each schedule is serializable or not? If a schedule is serializable write down the equivalent serial schedule( $S$ ).

$T1$ :  $R1(x) R1(z); W1(x);$

$T2$ :  $R2(x); R2(y); W2(z); W2(y)$

$T3$ :  $R3(x); R3(y); W3(y);$

$S1$ :  $R1(x); R2(z); R1(z); R3(x); R3(y); W1(x); W3(y); R2(y); W2(z); W2(y);$

$S2$ :  $R1(x); R2(z); R3(x); R1(z); R2(y); R3(y); W1(x); W2(z); W3(y); W2(y);$

**Solution:**

**Step 1:** We will represent the schedule  $S1$  as follows

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

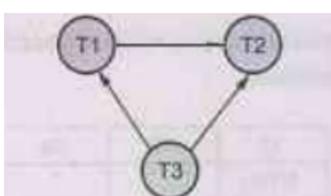
**Step (a):** We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

- i) Both the operations belong to different transactions.
- ii) Both the operations are on same data item.
- iii) At least one of the two operations is a write operation

The conflicting entries are as follows -

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

**Step (b):** Now we will draw precedence graph as follows-



**Fig. 3.5.3 : Precedence graph**

As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

**Step (c):** A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

**Step (d):** Find the vertex which has no incoming edge which is T3. If we delete T3, then T1 is the edge that has no incoming edge. Finally find the vertex having no outgoing edge which is T2. Hence the order will be T3-T1-T2.

**Step 2:** We will represent the schedule S2 as follows -

T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

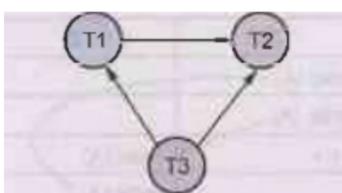
**Step (a):** We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

- i) Both the operations belong to different transactions.
- ii) Both the operations are on same data item.
- iii) At least one of the two operations is a write operation

The conflicting entries are as follows -

T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

**Step (b):** Now we will draw precedence graph as follows-



**Fig. 3.5.4 : Precedence graph**

As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

**Step (c):** A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

**Step (d):** Find the vertex which has no incoming edge which is T3. Finally find the vertex having no outgoing edge which is T2. So in between them is T1. Hence the order will be T3-T1-T2

**Example 3.5.5** Explain the concept of conflict serializability. Decide whether following schedule is conflict serializable or not. Justify your answer.

T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

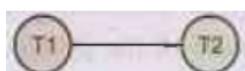
**Solution:**

**Step 1:** We will read from top to bottom, and build a precedence graph for conflicting entries.

The conflicting entries are as follows-

T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

**Step 2:** Now we will build precedence graph as follows



**Step 3:** There is no cycle in the precedence graph. That means this schedule is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow the following steps to find the serializable order.

- 1) Find the vertex which has no incoming edge which is T1.
- 2) Then find the vertex having no outgoing edge which is T2. In between them there is no other transaction.
- 3) Hence the order will be T1-T2.

### View Serializability

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.
- **View Equivalent Schedule:** Consider two schedules  $S_1$  and  $S_2$  consisting of transactions  $T_1$  and  $T_2$  respectively, then schedules  $S_1$  and  $S_2$  are said to be view equivalent schedule if it satisfies following three conditions:
  - If transaction  $T_1$  reads a data item A from the database initially in schedule  $S_1$ , then in schedule  $S_2$  also,  $T_1$  must perform the initial read of the data item X from the database. This is same for all the data items. In other words –the initial reads must be same for all data items.
  - If data item A has been updated at last by transaction  $T_1$  in schedule  $S_1$ , then in schedule  $S_2$  also, the data item A must be updated at last by transaction  $T_1$ .

- If transaction  $T_1$  reads a data item that has been updated by the transaction  $T_1$  in schedule  $S_1$  then in schedule  $S_2$  also, transaction  $T_1$  must read the same data item that has been updated by transaction  $T_1$ . In other words the Write-Read sequence must be same.

Sr.No.	Conflict serializability	View serializability
1	Every conflict serializable is view serializable.	Every view serializable schedule is not necessarily conflict serializable.
2.	It is easy to test conflict serializability.	It is complex to test view serializability.

### Steps to check whether the given schedule is view serializable or not

**Step 1:** If the schedule is conflict serializable then it is surely view serializable because conflict serializability is a restricted form of view serializability.

**Step 2:** If it is not conflict serializable schedule then check whether there exist any blind write operation. The blind write operation is a write operation without reading a value. If there does not exist any blind write then that means the given schedule is not view serializable. In other words if a blind write exists then that means schedule may or may not be view conflict.

**Step 3:** Find the view equivalence schedule

**Example 3.5.6** Consider the following schedules for checking if these are view serializable or not.

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
		W(C)
	R(A)	
	W(B)	
R(C)		
		W(B)
W(B)		

**Solution:**

i) The initial read operation is performed by T<sub>2</sub> on data item A or by T<sub>1</sub> on data item C. Hence we will begin with T<sub>2</sub> or T<sub>1</sub>. We will choose T<sub>2</sub> at the beginning.

ii) The final write is performed by T<sub>1</sub> on the same data item B. Hence T<sub>1</sub> will be at the last position.

iii) The data item C is written by T<sub>3</sub> and then it is read by T<sub>1</sub>. Hence T<sub>3</sub> should be before T<sub>1</sub>.

Thus we get the order of schedule of view serializability as T<sub>2</sub> - T<sub>3</sub> - T<sub>1</sub>

**Example 3.5.7** Consider following two transactions:

T<sub>1</sub>: read(A)

read(B)

if A=0 then B:=B+1;

write(B)

T<sub>2</sub>: read(B);

read(A);

if B=0 then A:=A+1;

write(A)

Let consistency requirement be A=0 V B=0 with A=B=0 the initial values.

1) Show that every serial execution involving these two transactions preserves the consistency of the Database?

2) Show a concurrent execution of T<sub>1</sub> and T<sub>2</sub> that produces a non serializable schedule?

3) Is there a concurrent execution of T<sub>1</sub> and T<sub>2</sub> that produces a serializable schedule?

**Solution:** 1) There are two possible executions: T<sub>1</sub> ->T<sub>2</sub> or T<sub>2</sub>->T<sub>1</sub>

Consider case T<sub>1</sub>->T<sub>2</sub> then

A	B
0	0
0	1
0	1

A∨B =A OR B=FVT-T. This means consistency is met.

Consider case T<sub>2</sub>->T<sub>1</sub> then

A	B
0	0
1	0
1	0

$A \vee B = A \text{ OR } B = F \vee T = T$ . This means consistency is met.

2) The concurrent execution means interleaving of transactions  $T_1$  and  $T_2$ . It can be

$T_1$	$T_2$
R(A)	
	R(B)
	R(A)
R(B)	If B=0 then
If A=0 then	A=A+1
B=B+1	W(A)
W(B)	

This is a non-serializable schedule.

(3) There is no concurrent execution resulting in a serializable schedule.

**Example 3.5.8** Test serializability of the following schedule:

i)  $r_1(x); r_2(x); w_1(x); r_2(x); w_2(x)$  ii)  $l_1(x); l_2(x); W_1(x); l_1(x); w_1(x)$

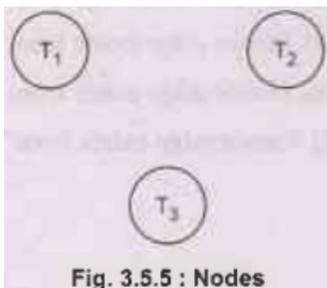
**Solution:**

i)  $r_1(x); r_3(x); w_1(x); r_2(x); w_3(x)$

The  $r_1$  represents the read operation of transaction  $T_1$ ,  $w_3$  represents the write operation on transaction  $T_3$  and so on. Hence from given sequence the schedule for three transactions can be represented as follows:

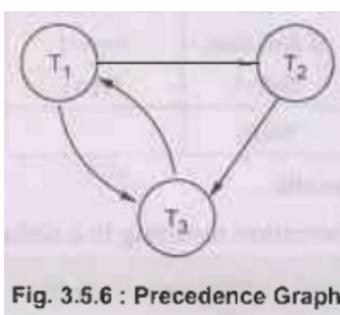
$T_1$	$T_2$	$T_3$
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

**Step 1:** We will use the precedence graph method to check the serializability. As there are three transactions, three nodes are created for each transaction.



**Step 2:** We will read from top to bottom. Initially we read  $r_1(x)$  and keep on moving bottom in search of write operation. Here all the transactions work on same data item i.e.  $x$ . Now we get a write operation in  $T_1$  as  $w_3(x)$ . Hence the dependency is from  $T_1$  to  $T_3$ . Therefore we draw edge from  $T_1$  to  $T_3$ .

Similarly, for  $r_3(x)$  we get  $w_1(x)$  pair. Hence there will be edge from  $T_3$  to  $T_1$ . Continuing in this fashion we get the precedence graph as



**Step 3:** As cycle exists in the above precedence graph, we conclude that it is not serializable.

ii)  $r_3(x); r_2(x); w_3(x); r_1(x); w_1(x)$

From the given sequence the schedule can be represented as follows:

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>
		r <sub>3</sub> (x)
	r <sub>2</sub> (x)	
		w <sub>3</sub> (x)
r <sub>1</sub> (x)		
w <sub>1</sub> (x)		

**Step 1:** Read the schedule from top to bottom for pair of operations. For r<sub>3</sub>(x) we get w<sub>1</sub>(x) pair. Hence edge exists from T<sub>1</sub> to T<sub>3</sub> in precedence graph.

**There is a pair from r<sub>2</sub>(x):** w<sub>3</sub>(x). Hence edge exists from T<sub>2</sub> to T<sub>3</sub>.

**There is a pair from r<sub>2</sub>(x):** w<sub>1</sub>(x). Hence edge exists from T<sub>2</sub> to T<sub>1</sub>.

**There is a pair from w<sub>2</sub>(x):** r<sub>1</sub>(x). Hence edge exists from T<sub>3</sub> to T<sub>1</sub>.

**Step 2:** The precedence graph will then be as follows-

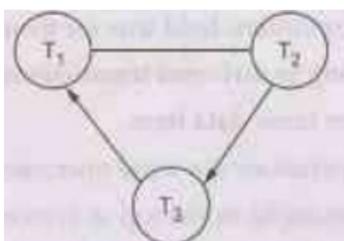
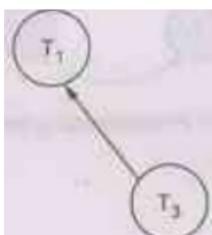


Fig. 3.5.7 : Precedence graph

**Step 3:** As there is no cycle in the above graph, the given schedule is serializable.

**Step 4:** The serializability order for consistent schedule will be obtained by applying topological sorting on above drawn precedence graph. This can be achieved as follows,

**Sub-Step 1:** Find the node having no incoming edge. We obtain T<sub>2</sub> is such a node. Hence T<sub>2</sub> is at the beginning of the serializability sequence. Now delete T<sub>2</sub>. The Graph will be



**Sub-Step 2:** Repeat sub-Step 1, We obtain T<sub>3</sub> and T<sub>1</sub> nodes as a sequence.

Thus we obtain the sequence of transactions as T<sub>2</sub>, T<sub>3</sub> and T<sub>1</sub>. Hence the serializability order is

r<sub>2</sub>(x);r<sub>3</sub>(x);w<sub>3</sub>(x);r<sub>1</sub>(x);w<sub>1</sub>(x)

**Example 3.5.9** Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.

S1: T1: R(X), T2: R(X), T1: W(Y), T2: W(Y) T1: R(Y), T2: R(Y)

S2: T3: W(X), T1: R(X), T1: W(Y), T2: R(Z), T2: W(Z) T3: R(Z)

For each of the schedules, answer the following questions:

i) What is the precedence graph for the schedule?

ii) Is the schedule conflict-serializable? If so, what are all the conflict equivalent serial schedules?

iii) Is the schedule view-serializable? If so, what are all the view equivalent serial schedules? **AU: May-15, Marks 2 +7 +7**

**Solution:** i) We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

- Both the operations belong to different transactions.
- Both the operations are on same data item.
- At least one of the two operations is a write operation

**For S1:** From above given example in the top to bottom scanning we find the conflict as

- T1: W(Y), T2: W(Y) and
- T2: W(Y), T1: R(Y)

Hence we will build the precedence graph. Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T<sub>1</sub>:W(Y) to T<sub>2</sub>:W(Y). Hence edge must be from T<sub>1</sub> to T<sub>2</sub>. Similarly for second conflict, there will be the edge from T<sub>2</sub> to T<sub>1</sub>



Fig. 3.5.8 : Precedence graph for S1

For S2: The conflicts are

- T3: W(X), T1: R(X)
- T2: W(Z) T3: R(Z)

Hence the precedence graph is as follows –



Fig. 3.5.9 : Precedence graph for S2

i)

- S1 is not conflict-serializable since the dependency graph has a cycle.
- S2 is conflict-serializable as the dependency graph is acyclic. The order T2-T3-T1 is the only equivalent serial order

ii)

- S1 is not view serializable.
- S2 is trivially view-serializable as it is conflict serializable. The only serial order allowed is

**T2-T3-T1.**

**Example 3.5.10** Check whether following schedule is view serializable or not. Justify your answer. (Note: T1 and T2 are transactions). Also explain the concept of view equivalent schedules and conflict equivalent schedule considering the example schedule given below :

T1	T2
read (A)	
A := A - 50	
write (A)	
	read (A)
	temp := A * 0.1
	A := A - temp
	write (A)
read (B)	
B := B + 50	
write (B)	
	read (B)
	B := B + temp
	write (B)

**Solution:**

**Step 1:** We will first find if the given schedule is conflict serializable or not. For that purpose, we will find the conflicting operations. These are as shown below –

T1	T2
read (A)	
A := A - 50	
write (A)	
	read (A)
	temp := A * 0.1
	A := A - temp
	write (A)
read (B)	
B := B + 50	
write (B)	
	read (B)
	B := B + temp
	write (B)

The precedence graph is as follows -



Fig. 3.5.10 : Precedence graph

As there exists no cycle, the schedule is conflict serializable. The possible serializability order can be T1 - T2

Now we check it for view serializability. As we get the serializability order as T1 - T2, we will find the view equivalence with the given schedule as serializable schedule.

Let S be the given schedule as given in the problem statement. Let the serializable schedule is S' = {T1, T2}. These two schedules are represented as follows:



# Transaction Support in SQL

## Transaction Support in SQL

The COMMIT, ROLLBACK, and SAVEPOINT are collectively considered as Transaction Commands

(1) **COMMIT:** The COMMIT command is used to save permanently any transaction to database.

When we perform, Read or Write operations to the database then those changes can be undone by rollback operations. To make these changes permanent, we should make use of commit

(2) **ROLLBACK:** The ROLLBACK command is used to undo transactions that have not already saved to database. For example

Consider the database table as

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 3.6.1 : Student Table

Following command will delete the record from the database, but if we immediately performs ROLLBACK, then this deletion is undone.

For instance -

```
DELETE FROM Student
```

```
WHERE RollNo = 2;
```

```
ROLLBACK;
```

Then the resultant table will be

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

**(3) SAVEPOINT:** A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction. The SAVEPOINT can be created as

SAVEPOINT savepoint\_name;

Then we can ROLLBACK to SAVEPOINT as

ROLLBACK TO savepoint\_name;

For example - Consider Student table as follows –

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 3.6.2 : Student Table

Consider Following commands

SQL> SAVEPOINT S1 SQL>DELETE FROM Student

Where RollNo=2;

SQL> SAVEPOINT S2

SQL>DELETE FROM Student

Where RollNo=3;

SQL> SAVEPOINT S3

SQL>DELETE FROM Student

Where RollNo=4

SQL> SAVEPOINT S4

SQL>DELETE FROM Student

Where RollNo=5

SQL> ROLLBACK TO S3;

Then the resultant table will be

RollNo	Name
1	AAA
2	BBB
3	CCC

Thus the effect of deleting the record having RollNo 2, and RollNo3 is undone.

# Concurrency Control

## Part II Concurrency Control

### Concurrency Control

AU: May-19, Marks 15

- One of the fundamental properties of a transaction is isolation.
- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.
- A database can have multiple transactions running at the same time. This is called concurrency.
- To preserve the isolation property, the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called concurrency control schemes.
- **Definition of concurrency control:** A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called concurrency control mechanism.
- The concurrency control can be achieved with the help of various protocols such as - lock based protocol, Deadlock handling, Multiple Granularity, Timestamp based protocol, and validation based protocols.

### Review Questions

1. Discuss the violations caused by each of the following: dirty read, non repeatable read and phantoms with suitable example. AU: May-17, Marks 13
2. What is concurrency control? How it is implemented in DBMS ? Briefly elaborate diagrams and examples. AU: May-19, Marks 15

## Need for Concurrency

### Need for Concurrency

AU: MAY-17,19, Marks 15

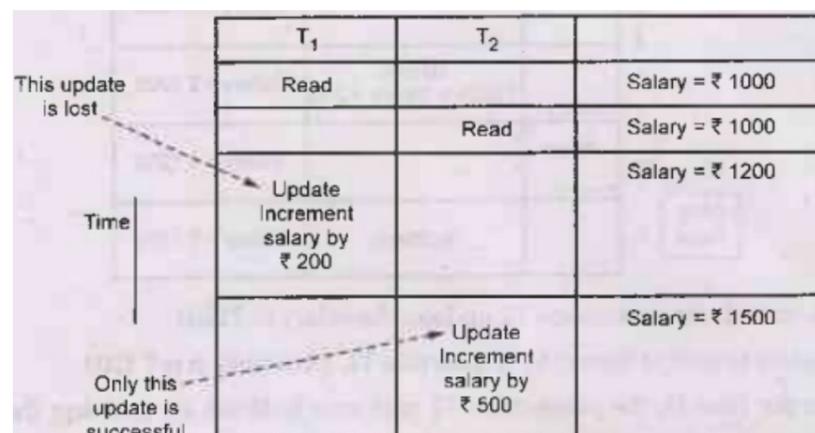
Following are the purposes of concurrency control -

- To ensure isolation
- To resolve read-write or write-write conflicts
- To preserve consistency of database
- Concurrent execution of transactions over shared database creates several data integrity and consistency problems - these are

**(1) Lost update problem:** This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

For example - Consider following transactions

- (1) Salary of Employee is read during transaction T1.
- (2) Salary of Employee is read by another transaction T2.
- (3) During transaction T1, the salary is incremented by 200
- (4) During transaction T2, the salary is incremented by 500



The result of the above sequence is that the update made by transaction T1 is completely lost. Therefore this problem is called as lost update problem.

**(2) Dirty read or Uncommitted read problem:** The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction

T <sub>1</sub>	T <sub>2</sub>
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A-20
	W(A)
	Commit
Commit	

Dirty read

For example - Consider following transactions -

Assume initially salary is = 1000

Time	T <sub>1</sub>	T <sub>2</sub>	Salary
	...	...	Salary = ₹ 1000
t <sub>1</sub>		Update Salary = Salary + 200	Salary = ₹ 1200
t <sub>2</sub>	Read		Salary = ₹ 1200
t <sub>3</sub>		Rollback	Salary = ₹ 1000

(1) At the time t<sub>1</sub>, the transaction T<sub>2</sub> updates the salary to 1200

(2) This salary is read at time t<sub>2</sub> by transaction T<sub>1</sub>. Obviously it is 1200

(3) But at the time t<sub>3</sub>, the transaction T<sub>2</sub> performs Rollback by undoing the changes made by T<sub>1</sub> and T<sub>2</sub> at time t<sub>1</sub> and t<sub>2</sub>.

(4) Thus the salary again becomes = 1000. This situation leads to Dirty Read or Uncommitted Read because here the read made at time t<sub>2</sub>(immediately after read update of another transaction) becomes a dirty read.

### (3) Non-repeatable read problem

This problem is also known as inconsistent analysis problem. This problem occurs when a particular transaction sees two different values for the same row within its lifetime. For example-

Time	T <sub>1</sub>	T <sub>2</sub>	Salary
t <sub>1</sub>	Read		Salary = ₹ 1000
t <sub>2</sub>		Update salary from ₹ 1000 to ₹ 1200	Salary = ₹ 1200
t <sub>3</sub>		Commit	
t <sub>4</sub>	Read		Salary = ₹ 1200

(1) At time t<sub>1</sub>, the transaction T<sub>1</sub> reads the salary as 1000

(2) At time t<sub>2</sub> the transaction T<sub>2</sub> reads the same salary as 1000 and updates it to 1200

(3) Then at time t<sub>3</sub>, the transaction T<sub>2</sub> gets committed.

(4) Now when the transaction T<sub>1</sub> reads the same salary at time t<sub>4</sub>, it gets different value than what it had read at time t<sub>1</sub>. Now, transaction T<sub>1</sub>, cannot repeat its reading operation. Thus inconsistent values are obtained.

Hence the name of this problem is non-repeatable read or inconsistent analysis problem.

### (4) Phantom read problem

The phantom read problem is a special case of non repeatable read problem.

This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently. For example -

Time	T <sub>1</sub>	T <sub>2</sub>	Salary
t <sub>1</sub>	Read		Salary = ₹ 1000
t <sub>2</sub>		Read	Salary = ₹ 1000
t <sub>3</sub>	Delete salary		No salary
t <sub>4</sub>		Read	"Can not find salary"

(1) At time t<sub>1</sub>, the transaction T<sub>1</sub> reads the value of salary as 1000

(2) At time t<sub>2</sub>, the transaction T<sub>2</sub> reads the value of the same salary as 1000

(3) At time t<sub>3</sub>, the transaction T<sub>1</sub> deletes the variable salary.

(4) Now at time t<sub>4</sub>, when T<sub>2</sub> again reads the salary it gets error. Now transaction T<sub>2</sub> can not identify the reason why it is not getting the salary value which is read just few time back.

This problem occurs due to changes in the database and is called phantom read problem.

# Locking Protocols

## Locking Protocols

AU: Dec-15,17, May-16, Marks 16

### Why Do We Need Locks?

- One of the method to ensure the isolation property in transactions is to require that data items be accessed in a mutually exclusive manner. That means, while one transaction is accessing a data item, no other transaction can modify that data item.
- The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.
- Thus the lock on the operation is required to ensure the isolation of transaction.

### Simple Lock Based Protocol

- Concept of Protocol: The lock based protocol is a mechanism in which there is exclusive use of locks on the data item for current transaction.
- **Types of Locks:** There are two types of locks used -

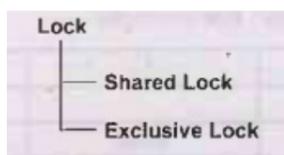


Fig. 3.9.1 : Types of locks

i) **Shared Lock:** The shared lock is used for reading data items only. It is denoted by Lock-S. This is also called as read lock.

ii) **Exclusive Lock:** The exclusive lock is used for both read and write operations. It is denoted as Lock-X. This is also called as write lock.

- The compatibility matrix is used while working on set of locks. The concurrency control manager checks the compatibility matrix before granting the lock. If the two modes of transactions are compatible to each other then only the lock will be granted.

- In a set of locks may consists of shared or exclusive locks. Following matrix represents the compatibility between modes of locks.

	S	X
S	T	F
X	F	F

Fig. 3.9.2 : Compatibility matrix for locks

Here T stands for True and F stands for False. If the control manager get the compatibility mode as True then it grant the lock otherwise the lock will be denied.

- **For example:** If the transaction  $T_1$  is holding a shared lock in data item A, then the no control manager can grant the shared lock to transaction  $T_2$  as compatibility is True.

But it cannot grant the exclusive lock as the compatibility is false. In simple words if transaction  $T_1$  is reading a data item A then same data item A can be read by another transaction  $T_2$  but cannot be written by another transaction.

- Similarly if an exclusive lock (i.e. lock for read and write operations) is hold on the data item in some transaction then no other transaction can acquire Share or exclusive lock as the compatibility function denotes F. That means of some transaction is writing a data item A then another transaction can not read or write that data item A.

Hence the rule of thumb is

i) Any number of transactions can hold shared lock on an item.

ii) But exclusive lock can be hold by only one transaction.

- **Example of a schedule denoting shared and exclusive locks:** Consider following schedule in which initially  $A=100$ . We deduct 50 from A in T, transaction and Read the data item A in transaction T2. The scenario can be represented with the help of locks and concurrency control manager as follows:

	$T_1$	$T_2$	Concurrency control manager
Exclusive Lock	Lock-X(A)		
			Grant X(A,T1) because in T1 there is write operation.
	R(A)		
	A=A-50		
	W(A)		
	Unlock(A)		
Shared Lock		Lock-S(A)	
			Grant S(A,T2) because in T2 there is Read operation
		R(A)	
		Unlock(A)	

### Review Questions

1. State and explain the lock based concurrency control with suitable example. AU: Dec-17, Marks 13, May-16, Marks 16

2. What is Concurrency control? How is implemented in DBMS? Illustrate with suitable example. AU: Dec-15, Marks 8

## Two Phase Locking

### Two Phase Locking

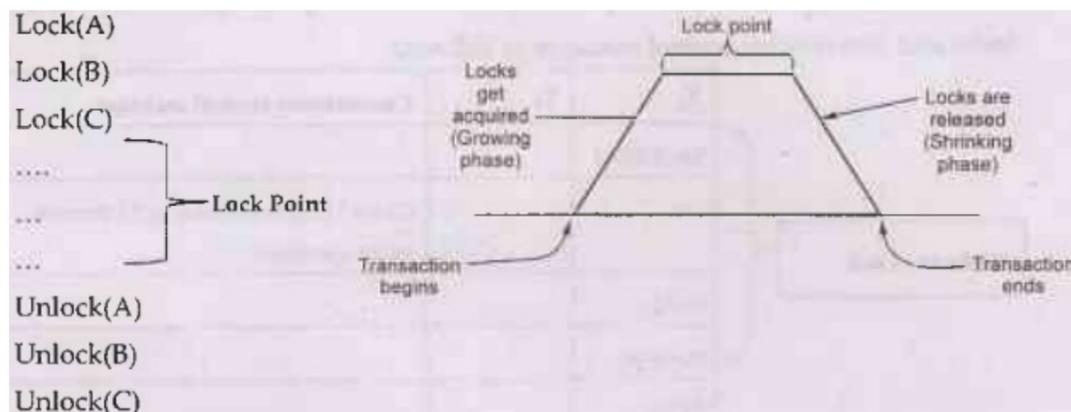
AU : May-14,18, Dec.-11,16,19, Marks 9

• The two phase locking is a protocol in which there are two phases:

i) **Growing phase (Locking phase):** It is a phase in which the transaction may obtain locks but does not release any lock.

ii) **Shrinking phase (Unlocking phase):** It is a phase in which the transaction may release the locks but does not obtain any new lock.

• **Lock Point:** The last lock position or first unlock position is called lock point. For example



Consider following transactions

T1	T2
Lock-X(A)	Lock-S(B)
Read(A)	Read(B)
A=A-50	Unlock-S(B)
Write(A)	
Lock-X(B)	
Unlock-X(A)	
B=B+100	Lock-S(A)
Write(B)	Read(A)
Unlock-X(B)	Unlock-S(A)

The important rule for being a two phase locking is - All Lock operations precede all the unlock operations.

In above transactions T<sub>1</sub> is in two phase locking mode but transaction T<sub>2</sub> is not in two phase locking. Because in T<sub>2</sub>, the Shared lock is acquired by data item B, then data item B is read and then the lock is released. Again the lock is acquired by data item A, then the data item A is read and the lock is then released. Thus we get lock-unlock-lock-unlock sequence. Clearly this is not possible in two phase locking.

**Example 3.10.1** Prove that two phase locking guarantees serializability. AU: Dec.-11, Marks 8

**Solution:**

- Serializability is mainly an issue of handling write operation. Because any inconsistency may only be created by write operation.
- Multiple reads on a database item can happen parallelly.
- 2-Phase locking protocol restricts this unwanted read/write by applying exclusive lock.
- Moreover, when there is an exclusive lock on an item it will only be released in shrinking phase. Due to this restriction there is no chance of getting any inconsistent state.

The serializability using two phase locking can be understood with the help of following example

Consider two transactions

T <sub>1</sub>	T <sub>2</sub>
R(A)	
	R(A)
R(B)	
W(B)	

**Step 1:** Now we will apply two phase locking. That means we will apply locks in growing and shrinking phase

T <sub>1</sub>	T <sub>2</sub>
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-X(B)	
R(B)	
W(B)	
Unlock-X(B)	
	Unlock-S(A)

Note that above schedule is serializable as it prevents interference between two transactions.

The serializability order can be obtained based on the lock point. The lock point is either last lock operation position or first unlock position in the transaction.

The last lock position is in T<sub>1</sub>, then it is in T<sub>2</sub>. Hence the serializability will be T<sub>1</sub>->T<sub>2</sub> based on lock points. Hence the sequence can be **R1(A);R2(A);R1(B);W1(B)**

**Limitations of Two Phase Locking Protocol**

The two phase locking protocol leads to two problems - deadlock and cascading roll back.

**(1) Deadlock:** The deadlock problem can not be solved by two phase locking. Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

For example

T1	T2
Lock-X(A)	Lock-X(B)
Read(A)	Read(B)
A=A-50	B=B+100
Write(A)	Write(B)
<div style="border: 1px dashed black; border-radius: 50%; padding: 10px; width: 100px; margin: 0 auto;">                     Delayed, wait for T2 to release Lock on B                 </div>	<div style="border: 1px dashed black; border-radius: 50%; padding: 10px; width: 100px; margin: 0 auto;">                     Delayed, wait for T1 to release Lock on A                 </div>

**(2) Cascading Rollback:** Cascading rollback is a situation in which transaction failure leads to a series of transaction rollback. For example -

T1	T2	T3
Read(A)		
Read(B)		
C=A+B		
Write(C)		
	Read(C)	
	Write(C)	
		Read(C)

When T<sub>1</sub> writes value of C then only T<sub>2</sub> can read it. And when T<sub>2</sub> writes the value of C then only transaction T<sub>3</sub> can read it. But if the transaction T<sub>1</sub> gets failed then automatically transactions T<sub>2</sub> and T<sub>3</sub> gets failed.

The simple two phase locking does not solve the cascading rollback problem. To solve the problem of cascading Rollback two types of two phase locking mechanisms can be used.

### Types of Two Phase Locking

**(1) Strict two phase locking:** The strict 2PL protocol is a basic two phase protocol but all the exclusive mode locks be held until the transaction commits. That means in other words all the exclusive locks are unlocked only after the transaction is committed. That also means that if T<sub>1</sub> has exclusive lock, then T<sub>2</sub> will release the exclusive lock only after commit operation, then only other transaction is allowed to read or write. For example Consider two transactions

T <sub>1</sub>	T <sub>2</sub>
W(A)	
	R(A)

If we apply the locks then

T <sub>1</sub>	T <sub>2</sub>
Lock-X(A)	
W(A)	
<b>Commit</b>	
Unlock(A)	
	Lock-S(A)
	R(A)
	Unlock-S(A)

Thus only after commit operation in T<sub>1</sub>, we can unlock the exclusive lock. This ensures the strict serializability.

Thus compared to basic two phase locking protocol, the advantage of strict 2PL protocol is it ensures strict serializability.

**(2) Rigorous two phase locking:** This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits. The transactions can be serialized in the order in which they commit.

example - Consider transactions

T <sub>1</sub>
R(A)
R(B)
W(B)

If we apply the locks then

T <sub>1</sub>
Lock-S(A)
R(A)
Lock-X(B)
R(B)
W(B)
Commit
Unlock(A)
Unlock(B)

Thus the above transaction uses rigorous two phase locking mechanism

**Example 3.10.2** Consider the following two transactions:

T1:read(A)

Read(B);

If A=0 then B=B+1;

Write(B)

T2:read(B); read(A)

If B=0 then A=A+1

Write(A)

Add lock and unlock instructions to transactions T1 and T2, so that they observe two phase locking protocol. Can the execution of these transactions result in deadlock? **AU: Dec.-16, Marks 6**

**Solution:**

T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Read(B)	Read(A)
if A=0 then B=B+1	if B=0 then A=A+1
Write(B)	Write(A)
Unlock(A)	Unlock(B)
Commit	Commit
Unlock(B)	Unlock(A)

This is lock-unlock instruction sequence help to satisfy the requirements for strict two phase locking for the given transactions.

The execution of these transactions result in deadlock. Consider following partial execution scenario which leads to deadlock.

T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Now it will wait for T2 to release exclusive lock on A	Now it will wait for T1 to release exclusive lock on B

## Lock Conversion

Lock conversion is a mechanism in two phase locking mechanism - which allows conversion of shared lock to exclusive lock or exclusive lock to shared lock.

**Method of Conversion :**

**First Phase:**

- can acquire a lock-S on item
- can acquire a lock-X on item
- can convert a lock-S to a lock-X (upgrade)

**Second Phase:**

- can release a lock-S

- can release a lock-X
- can convert a lock-X to a lock-S (downgrade)

This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

For example - Consider following two transactions -

T <sub>1</sub>	T <sub>2</sub>
R(A)	R(A)
R(B)	R(B)
...	
R(C)	
...	
W(A)	

Here if we start applying locks, then we must apply the exclusive lock on data item A, because we have to read as well as write on data item A. Another transaction T<sub>2</sub> does not get shared lock on A until transaction T<sub>1</sub> performs write operation on A. Since transaction T<sub>1</sub> needs exclusive lock only at the end when it performs write operation on A, it is better if T<sub>1</sub> could initially lock A in shared mode and then later change it to exclusive mode lock when it performs write operation. In such situation, the lock conversion mechanism becomes useful.

When we convert the shared mode lock to exclusive mode then it is called upgrading and when we convert exclusive mode lock to shared mode then it is called downgrading.

Also note that upgrading takes place only in growing phase and downgrading takes place only in shrinking phase. Thus we can refine above transactions using lock conversion mechanism as follows -

T <sub>1</sub>	T <sub>2</sub>
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-S(B)	
R(B)	
	Lock-S(B)
	R(B)
	Unlock(A)
	Unlock(B)
...	
Lock-S(C)	
R(C)	
...	
Upgrade(A)	
W(A)	
Unlock(A)	
Unlock(B)	
Unlock(C)	

### Review Questions

1. What is concurrency control? Explain two phase locking protocol with an example. AU: May-18, Marks 7
2. Illustrate two phase locking protocol with an example. AU: May-14, Dec.-16, Marks 6
3. Discuss elaborately the two phase locking protocol that ensures serializability. AU: Dec.-19, Marks 9

## **UNIT V -Object Relational and No-SQL Databases**

Mapping EER to ODB schema - Object identifier-reference types-row types- UDTs-Subtypes and supertypes user-defined routines- Collection types - Object Query Language, No-SQL: CAPtheorem -Document-based: MongoDB data model and CRUD operations: Column-based: Hbase data model and CRUD operations

### **Part -A**

#### **Two Mars Questions with Answers**

##### **1. What is object identifier(OID)?**

Any real world entity is modeled as object. Associated with each object there is an unique ID maintained which is called object identifier(OID)

##### **2. Explain the structure of object**

Object is a fundamental unit of object oriented programming in which data var and code is encapsulated in a single unit. Conceptually all interactions among the objects and rest of the system are carried out by using messages.

##### **3 .What is reference type?**

Reference types are all types other than value types. Variables of reference types store a reference to the memory address of the value.

A reference type value can be stored in one table and used as a reference to specific row in some other table.

##### **4.What is row type?**

A row type is a sequence of name-datatype pair. It is used to provide a datatype to the rows in the table. Using row type complete row can be stored in a variable.

##### **5 .What is supertype and subtype?**

Supertype: It an entity type that has got the relationship as parent to child with one or more subtypes. It contains attributes that are common to its subtypes.

Subtype: The subtypes are group of subtype entity and have unique attributes but they are different from each subtype.

##### **6. What is NoSQL ?**

NoSQL stands for not only SQL. It is nontabular database system that store data differently than relational tables. There

are various types of NoSQL databases such document, key-value, wide column and graph.

Using NoSQL we can maintain flexible schemas and these schemas can be scaled easily with large amount of data.

### **7.What is CAP theorem?**

The CAP theorem states that it is not possible to guarantee all three of the desirable properties Consistency, availability and partition tolerance at the same time in a distributed system with data replication.

### **8.Enlist features of MongoDB.**

- 1) It is a schema-less, document based database system.
- 2) It provides high performance data persistence.
- 3) It supports multiple storage engines.
- 4) It has a rich query language support.

### **9.Enlist the features of NoSQL.**

1. The NoSQL does not follow any relational mode
2. It is either schema free or have relaxed schema. That means it does not require specific
3. definition of schema.
4. Multiple NoSQL databases can be executed in distributed fashion.
5. It can process both unstructured and semi-structured data
6. The NoSQL have higher scalability.

### **10.Enlist the features of MongoDB.**

7. It is a schema-less, document based database system.
8. It provides high performance data persistence.
9. It supports multiple storage engines.
10. It has a rich query language support.
11. MongoDB provides high availability and redundancy with the help of replication.

That means it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.

### **11.What is mongoDB?**

- MongoDB is an open source, document based database.
- It is developed and supported by a company named 10gen which is now known MongoDB Inc.
- The first ready version of MongoDB was released in March 2010.

### **12.Why MongoDB is needed?**

There are so many efficient RDBMS products available in the market, then why do we need MongoDB? Well, all the modern applications require Big data, faster development and flexible

deployment. This need is satisfied by the document based database like MongoDB.

### **13.How the terms in MongoDB are different from SQL?**

The terms in SQL are treated differently in MongoDB. In MongoDB the data is not stored in tables, instead of that, there is a concept called collection which is analogous to the tables. In the same manner the rows in RDBMS are called documents in MongoDB, likewise the columns of the record in RDBMS are called fields.

### **14.Define collections in MongoDB.**

MongoDB groups data together through collections. A *collection* is simply a grouping of documents that have the same or a similar purpose.

A collection acts similarly to a table in a traditional SQL database, with one major difference.

In MongoDB, a collection is not enforced by a strict schema; instead, documents in a collection can have a slightly different structure from one another as needed.

This reduces the need to break items in a document into several different tables, which is often done in SQL implementations.

### **15.Define the term document in MongoDB.**

A *document* is a representation of a single entity of data in the MongoDB database. A collection is made up of one or more related objects.

A major difference between MongoDB and SQL is that documents are different from rows.

Row data is flat, meaning there is one column for each value in the row. However, in MongoDB, documents can contain embedded subdocuments, thus providing a much closer inherent data model to your applications.

### **16.Enlist any four data types in MongoDB.**

Following are various types of data types supported by MongoDB.

1. **Integer:** This data type is used for storing the numerical value.
2. **Boolean:** This data type is used for implementing the Boolean values  
i.e. true or false.
3. **Double:** Double is used for storing floating point data.
4. **String:** This is the most commonly used data type used for storing the string values.

## 17.Explain how to create collection in MongoDB?

After creating some sample database, we must create some collection inside that database.

### **Syntax**

```
db.create Collection(name, options)
```

where

We can create collection explicitly using createCollection command name is the name of collection options is an optional field. This field is maximum number of documents and so on.

## 18.Define CRUD Operation

After creating some sample database, we must create some collection inside that database. We can perform various operations such as insert, delete and update on this collection. These operations are called CRUD operations. CRUD stands for Create, Read, Update and Delete operation.

## 19.How to create Create Collection in MongoDB

### **Syntax**

We can create collection explicitly using createCollection command.  
`db.create Collection(name, options)`

where

**name** is the name of collection

**options** is an optional field. This field is used to specify some parameters such as Size, maximum number of documents and so on.

## 20.Give the syntax for insert the document. The document is analogous to rows in database.

The document is inserted within the collection. The document is analogous to rows in database

### **Syntax**

```
db.collection name.insert( {key, value }
```

## 21.How to insert multiple documents in a MongoDB database?

It is possible to insert multiple documents at a time using a single command. Following command shows how to insert multiple documents in the existing collection

```
> var =  
allStudents =[  
{  
"name": AAA,  
'age": 20  
},
```

```
{
  "name": BBB,
  "age": 21
},
{
  "name": CCC
  "age": 22
}
];
>db.Student
>db.Student details.insert(alliStudents),
```

## **22.How to delete Documents in MongoDB**

For deleting the document, the remove command is used. This is the simplest command

Syntax

```
db.collection_name.remove(delete_criteria)
```

## **23.What are the methods used in Update the document in MongoDB**

The methods used in Update the document in MongoDB

- 1.updateOne,
2. updateMany or bulkwrite.

## **24.. What is column-oriented database model? Give example of it.**

The column-oriented database is a database system that stores the data table by column rather than by rows. The advantage of column-oriented Database is that we can access the data more efficiently when only subset of columns is used for querying. For example - HBase

## **25.. Enlist the features of HBase data model.**

1. HBase is horizontally scalable. That means we can add any number of columns anytime.
2. The data is stored in key value format.
3. It is a platform for storing and retrieving data with random access.
4. It does not enforce relationship within the data.
5. It is designed to run on cluster of computers.

## Part B & C Questions

### 1.Explain about Object Database Concept and Mapping EER and ODB Schema

#### Object Database Concept

Traditional applications are designed for business applications such as inventory, employee, university, bank, library, air-line reservation systems, and so on.

These applications require relatively simple data types that are well suited to the relational data model. But database systems were applied to a wider range of applications, such as computer-aided design and geographical information systems. Hence such systems require complex data type and processing.

The object based database provide the solution to model the real world object and their behavior. It is an alternative to relational database model.

#### Mapping EER to ODB Schema

Object oriented paradigm encapsulate data and corresponding methods in a single entity called object.

The object oriented data model is a logical data model just like ER model. Loosely object can be thought as an entity of E-R model.

Object oriented data model helps in adapting object oriented paradigm to database systems,

In object oriented database, information is represented in the form of objects.

Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.

The core object oriented Data model consists of following object oriented concepts-

- **Object and Object Identifier**

Any real world entity is modeled as object. Associated with each object there is an unique ID maintained which is called object identifier(OID). Object identifiers used to uniquely identify objects. Object identifiers are unique. No two objects have the same identifier, each object has only one object identifier.

- **Attribute and method**

Every object has state and behavior. The state is a set of values for attributes of the object. The behavior is set of methods. For example - Object named circle has state radius and computing its area is a behavior.

## Class

Class is a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance-of relationship). A class is similar to an abstract data type.

- **Class Hierarchy and Inheritance**

The class hierarchy is used to represent the Inheritance property of object oriented paradigm. The inheritance is a mechanism in which a new class is derived from existing class.

## 1.Object Structure

- Object is a fundamental unit of object oriented programming in which data value and code is encapsulated in a single unit.
- Conceptually all interactions among the objects and rest of the system are carried out by using messages. Thus the interface among various objects and rest of the system is messages. Messages can be implemented as procedure invocations.
  - In general, an object has associated with it:
    - A set of variables that contain the data for the object. The value of each variable is itself an object.
    - A set of messages to which the object responds.
  - A set of methods, each of which is a body of code to implement each message; a method returns a value as the response to the message.
  - Methods are programs written in general-purpose language with the following features-
    - Only variables in the object itself may be referenced directly.
    - Data in other objects are referenced only by sending messages.
    - Methods can be read-only or update methods. Read-only methods do not change the value of the object.
  - **For example**-the object Circle can have methods such as compute\_area)

## 2.Object Classes

Similar objects are grouped into **classes**. In other words, object is an instance of a class. For example

```
class Student {  
  
    /*variables*/  
  
    Int RollNo:
```

```
String Name;  
String address;
```

```
/*Messages*/
```

```
String get name();  
String get_address();  
Int get RollNo();
```

```
}
```

Methods can be defined separately. For example

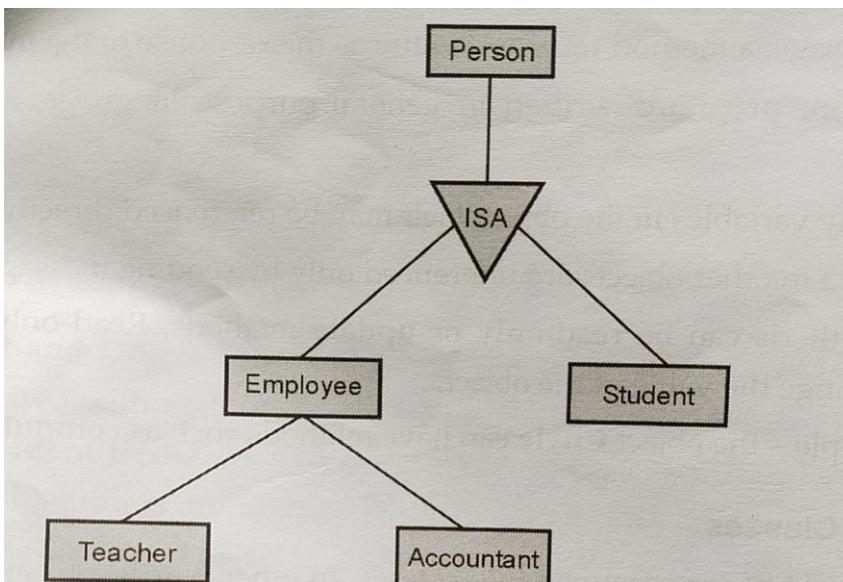
```
int get roliNo()
```

```
{  
Return RollNO;  
}
```

### 3.Inheritance

In object oriented database schema, there can be large number of classes. Some classes are similar and can be derived from old existing classes. The classes are placed into specialization or Is-a hierarchy.

For example - Consider following ER model –



### Class Hierarchy

The class hierarchy can be defined in pseudo-code in which the variables associated

with each class are as follows

```
class Person  
{
```

```

        String name;
        String address;
    }

class Student isa Person //Inheritance: Deriving child class
{
    int RollNo;
    String Course taken
}

class Employee isa Person
{
    date join-date;
    int salary;
}
class Teacher isa Employee
{
    String Subject:
}
class Accountant isa Employee
{
    int office no
}

```

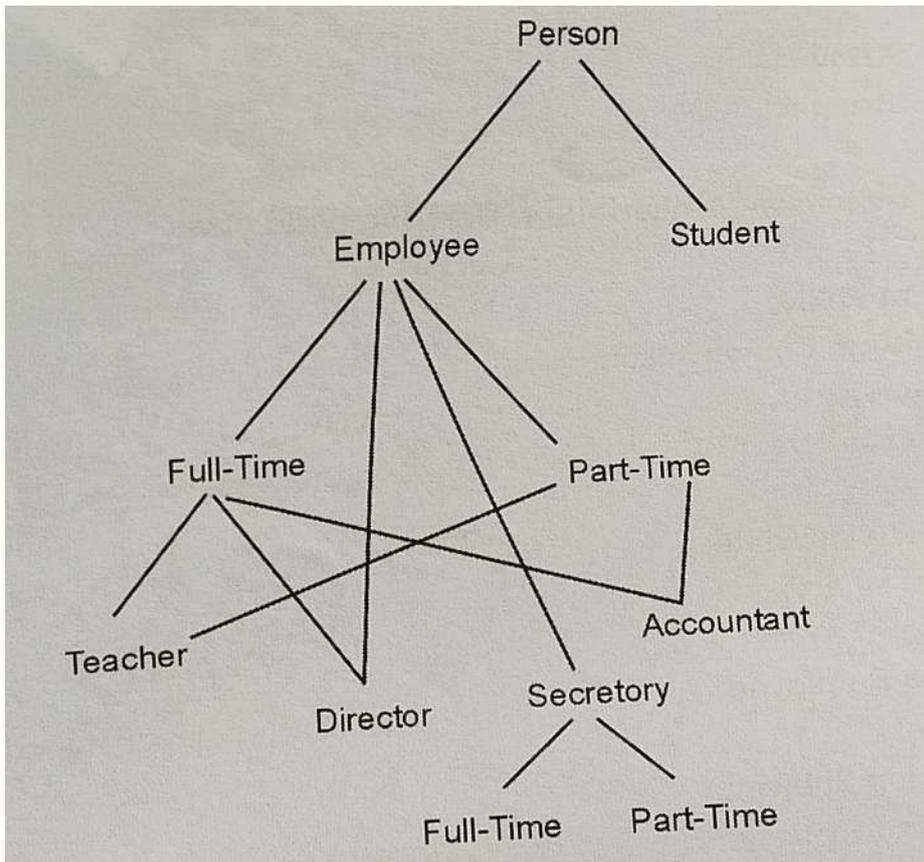
- The keyword **isa** is used to indicate that a class is a specialization of another class. The specialization of a class are called subclasses. For example - Employee is a subclass of Person. Conversely, Person is a superclass of class Employee.
- Class hierarchy and inheritance of properties from more general classes.
- The important benefit of using inheritance is code-reusability. This can be achieved by means of substitutability property.
- **Substitutability** means any method of a class, say person, can be invoked equally well with any object belonging to any subclass, such as subclass Teacher of Person.

#### 4. Multiple Inheritance

In most cases, tree-structured organization of classes is adequate to describe applications. In such cases, all super classes of a class are ancestors of descendants of another in the hierarchy. However, there are situations that cannot be represented well in a tree-structured class hierarchy.

To avoid the conflicts between two occurrences, we use multiple inheritance.

Multiple inheritance is an ability of class to inherit variables from multiple super classes. The class subclass relationship is represented by directed acyclic graph(DAG). For example-



**Fig. 5.2 Multiple inheritance**

Handling name conflicts: When multiple inheritance is used, there is potential ambiguity if the same variable or method can be inherited from more than one superclass.

## 5 .Object Identity

- For identifying the RDBMS record uniquely in the table, we use primary key associated with it. In object oriented database management system we use Object Id to identify the record.

Associated with each object there is a unique ID maintained which is called object identifier(OID).

An object retains its identity even if some or all the values of the variables or definitions of methods change over time.

The concept of object identity does not apply to tuples of relational database. It is a stronger notion of identity.

The difference between primary of RDBMS and object ID of OODBMS is that primary key is explicit; that means it is visible to the user whereas the object ID is implicit; that means it is hidden from external world.

It has several forms of identity:

- Value: The data value is used for identity. For example the primary key of the tuple in a relational database.
- Name: The user supplied name is used for identity. For example file name can be used for identity of object.
- Built-in: A notion of identity is built-into the data model or programming languages, and no user-supplied identifier is required.

Object identity is typically implemented via a unique, system-generated OID. The value of the OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references.

## **6. Object Containment**

- The objects that contain another objects are called composite objects or complex objects.
- There can be multiple levels of object containment. These levels can be represented using containment hierarchy.
- For example The following Fig. 5.2.3 represents the design of simple document The document contains the text in the form of paragraphs. Each paragraph contains word. Each word is made up of come characters.
- Also note that there can be some size and style for each paragraph, word and each character.
- The links between classes must be interpreted as is-part-of, rather than the is-a interpretation of links in an inheritance hierarchy.
- Containment allows data to be viewed at different granularities by different users. For example - document can be viewed by a reader for simply reading purpose, the size and style of paragraphs can be viewed by DTP operator or proof editor for editing purpose.
- The containment hierarchy can be used to find all object contained in document object.
- In certain applications, an object may be contained in several objects. In such cases, the containment relationship is represented by a DAG rather than by a hierarchy.

## **2. Write about Reference types in detail**

- Reference types are all types other than value types. Variables of reference types store a reference to the memory address of the value.
- A reference type value can be stored in one table and used as a reference to specific row in some other table.

- To use a reference type in an SQL statement, use REF(type-name), where type-name represents the referenced type.
- REF IS SYSTEM GENERATED in a CREATE TYPE statement indicates that the actual values of associated REF type are provided by the system.

### 3. Write about Row Types in detail.

- A row type is a sequence of name-datatype pair. It is used to provide a datatype to the rows in the table.
- Using row type complete row can be stored in a variable. This variable can be passed as an argument to the routine or we can get the returned value from a function call within this variable.
- **For example** Consider an Employee table containing Emp\_Id and address and insert into the new table

```
CREATE TABLE Employee( empID CHAR(5),
address ROW( Street VARCHAR(30),
City VARCHAR(25),
Pincode ROW(city id VARCHAR(6), Region VARCHAR(10))));
INSERT INTO Employee
VALUES('Account 1234', ROW('11 Shankar Rd', "Chennai", ROW('ch11','Anand
Nagar')));
```

To create a subtype StudentType of the supertype PersonType we write:

```
CREATE TYPE StudentType UNDER PersonType AS (
rollNo VARCHAR(5),
name VARCHAR(10),
branchID CHAR(4))
```

### 4. Explain about User Defined Types in detail

- The user defined type(UDT) is a data type derived from an existing data type
- UDT allows the database developer to extend the built in types and to create a customized data type.
- There are two types of user defined types-

#### **Distinct Type**

A distinct type is a user-defined data type that shares its internal representation with an existing built-in data type. For example-

```
CREATE TYPE empNumberType AS VARCHAR(5) FINAL
```

#### **Structured Type**

A structured type is a user-defined data type containing one or more named attributes, each of which has a data type. Attributes are properties that describe an instance of a type. A structured type also includes a set of method

specifications. Methods enable you to define behaviors for structured types. For example -

```
CREATE TYPE person AS OBJECT (  
  name VARCHAR(30),  
  phone VARCHAR(20));  
CREATE TYPE purchase_order AS OBJECT (  
  id NUMBER, contact person,  
  MEMBER FUNCTION  
  get value RETURN NUMBER);
```

In above example we have defined an object named person and created a structure type purchase\_order. The members of this structure are attributes such as id, contact and one member function named get\_value

- The value of the attribute can be accessed using the dot operator. For example if p is an instance of person then,
  - `p.name = 'Anand'`
  - allows to assign name to the attribute of the person data type.

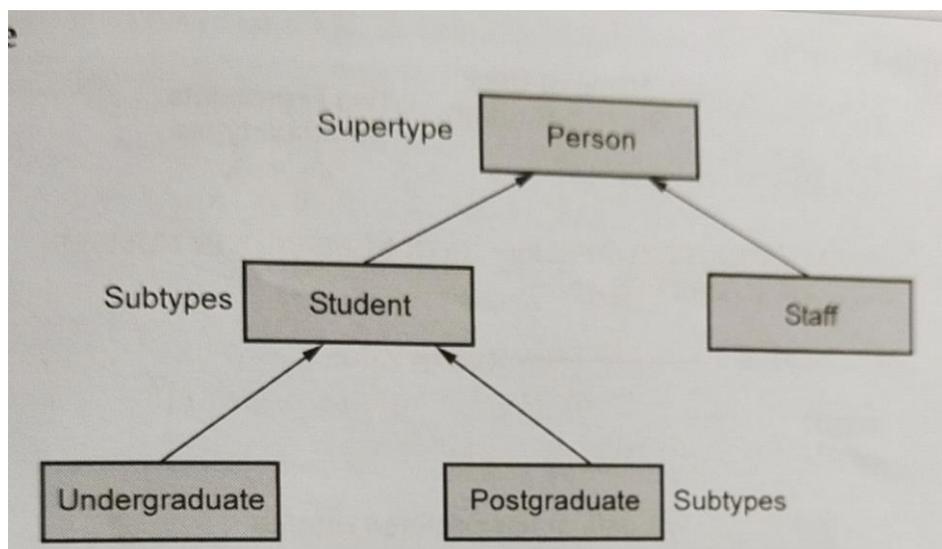
## 5. Write a short notes on Subtypes and Supertypes

### Supertype:

It is an entity type that has got the relationship as parent to child with one or more subtypes. It contains attributes that are common to its subtypes.

### Subtype:

The subtypes are a group of subtype entity and have unique attributes but they are different from each subtype.



**Figure.6 Example of Subtype and Super type**

To create a subtype **StudentType** of the **supertype** PersonType we write:

```
CREATE TYPE StudentType UNDER PersonType AS (  
rollNo VARCHAR(5),  
name VARCHAR(10),  
branchID CHAR(4))
```

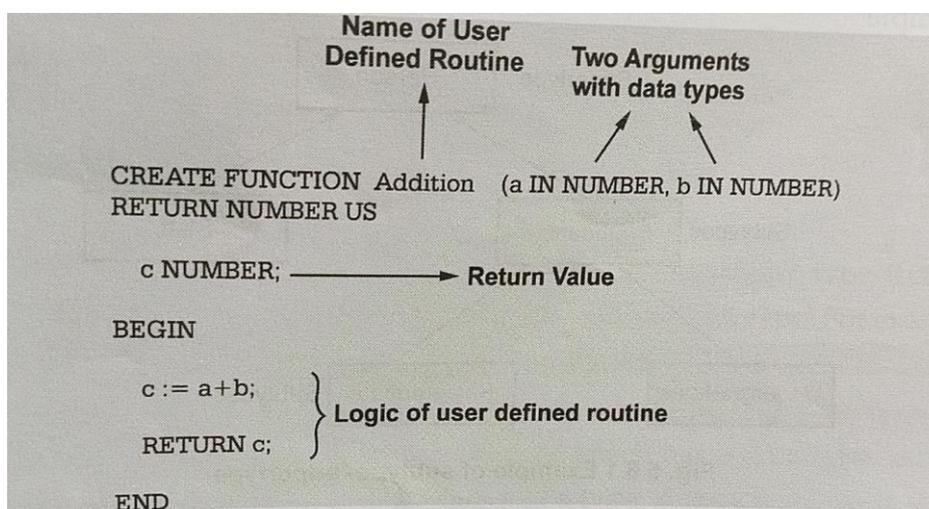
### Advantages of Subtype and super type

1. Avoids data redundancy, as each subtype contains only unique information. This in turn leads to a reduction in the size of the database itself.
2. Reduction of errors in programming operations on a table that corresponds to a specific subtype of an entity.
3. Flexibility in modifying the database structure. There is no need to modify the super type table if you add / change the structure of the subtype table.

### 6. Write about User-Defined Routines in detail

- User-defined routines are routines that users create themselves. User-defined routines provide a means for users to extend the SQL language.
- The User Defined routines capture the functionality of most commonly used arithmetic, string, and casting functions. However, these also allows to create routines to encapsulate logic of your own.
- User-defined procedures, functions and methods are created in the database by executing the appropriate CREATE statement for the routine type. These routine creation statements include:

```
CREATE PROCEDURE  
CREATE FUNCTION  
CREATE METHOD
```



The clauses specific to each of the CREATE statements define characteristics of the routine, such as the routine name, the number and type of routine arguments, and details about the routine logic.

## 7. Write about the collection types in detail

- A collection is an ordered group of elements having the same data type.
- The collections are used to store multiple values in a single column of table.
- The collection add flexibility to the database design.
- Following are the ways by which the collection types can be used-
  - ARRAY: It is an entity in that represents one dimensional array with
    - maximum number of elements
  - LIST: Ordered collection that allows duplicates.
  - SET: Unordered collection that does not allow duplicates

For example-Array can be created as follows-

```
VARCHAR(20) ARRAY(4):  
ARRAY['Archana','Ashwini', 'Aishwarya', "Sharda"]:
```

## 8. Give the detail about Object Query Language(OQL) in Detail

- Object Query Language (OQL) is a query language standard for object-oriented databases modeled after SQL...
- The following rules apply to OQL statements
  - (1) All complete statements must be terminated by a semi-colon.
  - (2) A list of entries in OQL is usually separated by commas but not terminated by a comma(,).
  - (3) Strings of text are enclosed by matching quotation marks.
- Basic From of OQL: SELECT, FROM, WHERE

### Syntax

```
SELECT <list of values>  
FROM <list of collections and variable assignments>  
WHERE <condition>
```

Where the SELECT clause extracts those elements of a collection meeting a specific condition. By using the keyword DISTINCT duplicated elements in the resulting collection get eliminated. Collections in FROM can be either extents (persistent names sets) or expressions that evaluate to a collection (a set).

Example: Give the names of people who are older than 30 years old:

```
SELECT SName: p.name
FROM p in People
WHERE page > 30
```

## **DOT notations and Path expressions**

We use the dot notation and path expressions to access components of complex values. For example

ta.salary -> real

t.students-> set of tuples of type tuple(name: string, fee: real) representing students

t.salary-> real

## **9.What is NoSQL? What is the need for it. Enlist various feature of NoSQL**

### **1.Introduction**

- NoSQL stands for not only SQL.
- It is nontabular database system that store data differently than relational tables. There are various types of NoSQL databases such as document, key-value, wide column and graph.
- Using NoSQL we can maintain flexible schemas and these schemas can be scaled easily with large amount of data

### **2. Need**

The NoSQL database technology is usually adopted for following reasons-

- 1) The NoSQL databases are often used for handling big data as a part of fundamental architecture.
- 2) The NoSQL databases are used for storing and modelling structured, semi-structured and unstructured data.
- 3) For the efficient execution of database with high availability, NoSQL. is used.
- 4) The NoSQL database is non-relational, so it scales out better than relational databases and these can be designed with web applications.
- 5) For easy scalability, the NoSQL is used.

### **3. Features**

- 1) The NoSQL does not follow any relational model.
- 2) It is either schema free or have relaxed schema. That means it does not require specific definition of schema.
- 3) Multiple NoSQL databases can be executed in distributed fashion.
- 4) It can process both unstructured and semi-structured data.

- 5) The NoSQL have higher scalability.
- 6) It is cost effective.
- 7) It supports the data in the form of key-value pair, wide columns and graphs.

## 10.Explain different types of NoSQL databases.

There are four types of NoSQL databases and those are-

1. Key-value store
2. Document store
3. Graph based
4. Wide column store

Let us discuss them in detail.

### 1. Key-Value Store

- Key-value pair is the simplest type of NoSQL database.
- It is designed in such a way to handle lots of data and heavy load.
- In the key-value storage the key is unique and the value can be JSON, string or Binary objects.
- For example-

```
{
  Customer:
  [
    {"id":1,"name":"Ankita"},
    {"id":2,"name":"Kavita"}
  ]
}
```

Here id, name are the keys and 1,2, "Ankita", "Prajakta" are the values corresponding to those keys.

Key-value stores help the developer to store schema-less data. They work best for Shopping Cart Contents.

The DynamoDB, Riak, Redis are some famous examples of key-value store

### 2 .Document Store

- The document store make use of key-value pair to store and retrieve data.
- The document is stored in the form of XML and JSON.
- The document stores appear the most natural among NoSQL. database types.
- It is most commonly used due to flexibility and ability to query on any field.
- For example -

```
{
  "id": 101.
```

```
"Name": "AAA",  
"City": "Pune"  
}
```

MongoDB and CouchDB are two popular document oriented NoSQL database

### 3. Graph

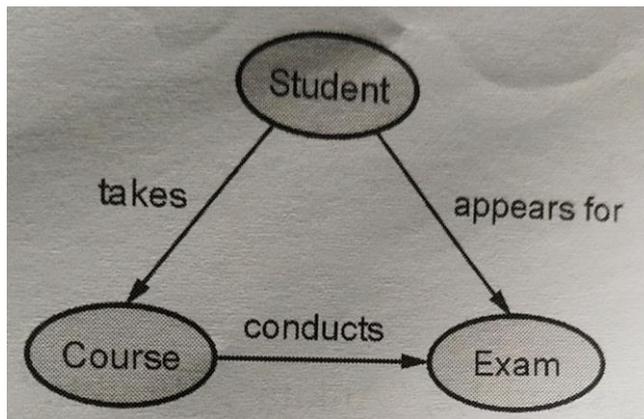
The graph database is typically used in the applications where the relationships among the data elements is an important aspect.

The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.

The graph database has two components-

- 1) **Node:** The entities itself. For example - People, student,
- 2) **Edge:** The relationships among the entities.

For example-



Graph base database is mostly used for social networks, logistics, spatial data. The graph databases are Neo4J, Infinite Graph, OrientDB.

### 4. Wide Column Store

- Wide column store model is similar to traditional relational database. In this model, the columns are created for each row rather than having predefined by the table structure.
- In this model number of columns are not fixed for each record.
- Columns databases can quickly aggregate the value of a given column.
- For example-

Row ID	Columns...		
1	<b>Name</b>	<b>City</b>	
	Ankita	Pune	
2	<b>Name</b>	<b>City</b>	<b>email</b>
	Kavita	Mumbai	<a href="mailto:kavita123@gmail.com">kavita123@gmail.com</a>

The column store databases are widely used to manage data warehouses, business intelligence, HBase, Cassandra are examples of column based databases

### 11. Give the difference between RDBMS and NoSQL

S.No	RDBMS	NoSQL
1	The relational database system is based on relationships among the tables.	It is non-relational database system. It can be used in distributed environment
2	It is vertically scalable.	It is horizontally scalable.
3	It has predefined schema.	It does not have schema or it may have related schema.
4	It uses SQL to query the database.	It uses unstructured query language.
5	It is a table based database.	It is document based, graph based on key-value pair.
6	It emphasizes on ACID properties (Atomicity, consistency, isolation and durability)	It follows Brewer's CAP theorem (Consistency, availability and partition tolerance)
7	Schema is fixed or rigid.	Schema is dynamic
8	Pessimistic.	Optimistic
9	Examples: MySQL, Oracle, PostgreSQL	Examples: MongoDB, Big Table, Redis

## 12. Write a short note on CAP Theorem

- Cap theorem is also called as brewer's theorem.
- The CAP Theorem is comprised of three components (hence its name) as they relate to distributed data stores:
- - Consistency: All reads receive the most recent write or an error.
  - Availability: All reads contain data, but it might not be the most recent
  -
- Partition tolerance: The system continues to operate despite network failures (i.e.; dropped partitions, slow network connections, or unavailable network connections between nodes.)
- The CAP theorem states that it is not possible to guarantee all three of the desirable properties - consistency, availability, and partition tolerance at the same time in a distributed system with data replication.

## 13. Understanding NoSQL and MongoDB in Detail

### NoSQL

- NoSQL stands for not only SQL.
- It is **nontabular database system** that store data differently than relational tables. are various types of NoSQL databases such as document, key-value, wide column and graph.
- Using NoSQL we can maintain flexible schemas and these schemas can be scaled easily with large amount of data.

### Need

1. The NoSQL database technology is usually adopted for following reasons -
2. The NoSQL databases are often used for handling big data as a part of fundamental architecture.
3. The NoSQL databases are used for storing and modelling structured, semi-structured and unstructured data.
4. For the efficient execution of database with high availability, NoSQL is used.
5. The NoSQL database is non-relational, so it scales out better than relational
6. databases and these can be designed with web applications.
7. For easy scalability, the NoSQL is used.

### Features

1. The NoSQL does not follow any relational model.
2. It is either schema free or have relaxed schema. That means it does not require specific definition of schema.
3. Multiple NoSQL databases can be executed in distributed fashion.
4. It can process both unstructured and semi-structured data.
5. The NoSQL have higher scalability.
6. It is cost effective.
7. It supports the data in the form of key-value pair, wide columns and graphs.

## **MongoDB**

- MongoDB is an open source, document based database.
- It is developed and supported by a company named 10gen which is now known MongoDB Inc.
- The first ready version of MongoDB was released in March 2010.

### **Why MongoDB is needed?**

There are so many efficient RDBMS products available in the market, then why do we need MongoDB? Well, all the modern applications require Big data, faster development and flexible deployment. This need is satisfied by the document based database like MongoDB.

### **Features of MongoDB**

1. It is a schema-less, document based database system.
2. It provides high performance data persistence.
3. It supports multiple storage engines.
4. It has a rich query language support.
5. MongoDB provides high availability and redundancy with the help of replication. That means it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
6. MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers.
7. In MongoDB, every field in the document is indexed as primary or secondary. Due to which data can be searched very efficiently from the database.

### **Understanding Collections**

MongoDB groups data together through collections. A *collection* is simply a grouping of documents that have the same or a similar purpose.

A collection acts similarly to a table in a traditional SQL database, with one major difference.

In MongoDB, a collection is not enforced by a strict schema; instead, documents in a collection can have a slightly different structure from one another as needed.

This reduces the need to break items in a document into several different tables, which is often done in SQL implementations.

### **Understanding Documents**

A *document* is a representation of a single entity of data in the MongoDB database. A collection is made up of one or more related objects.

A major difference between MongoDB and SQL is that documents are different from rows.

Row data is flat, meaning there is one column for each value in the row. However, in MongoDB, documents can contain embedded subdocuments, thus providing a much closer inherent data model to your applications.

The records in MongoDB that represent documents are stored as BSON, which is a lightweight binary form of JSON, with field:value pairs corresponding to JavaScript property:value pairs.

These field:value pairs define the values stored in the document.

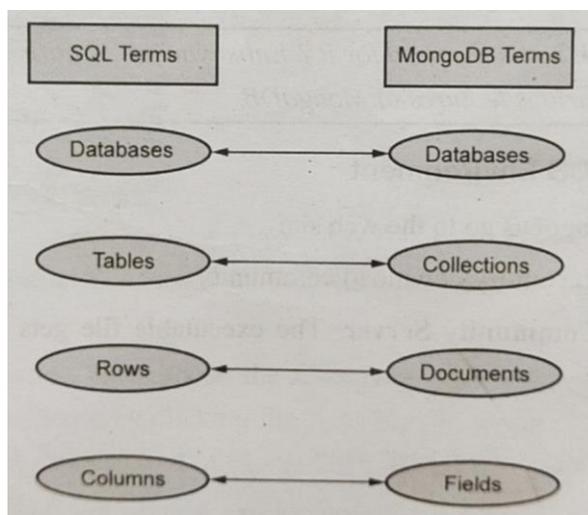
That means little translation is necessary to convert MongoDB records back into the JavaScript object that you use in your Node.js applications.

For example, a document in MongoDB may be structured similarly to the following with name, version, languages, admin, and paths fields:

```
{  
  name: "New Project",  
  version: 1,  
  languages: ["JavaScript", "HTML", "CSS"],  
  admin: {name: "Brad", password: "*****"},  
  paths: {temp: "/tmp", project: "/opt/project", html: "/opt/project/html"}  
}
```

### SQL Structure Vs. MongoDB

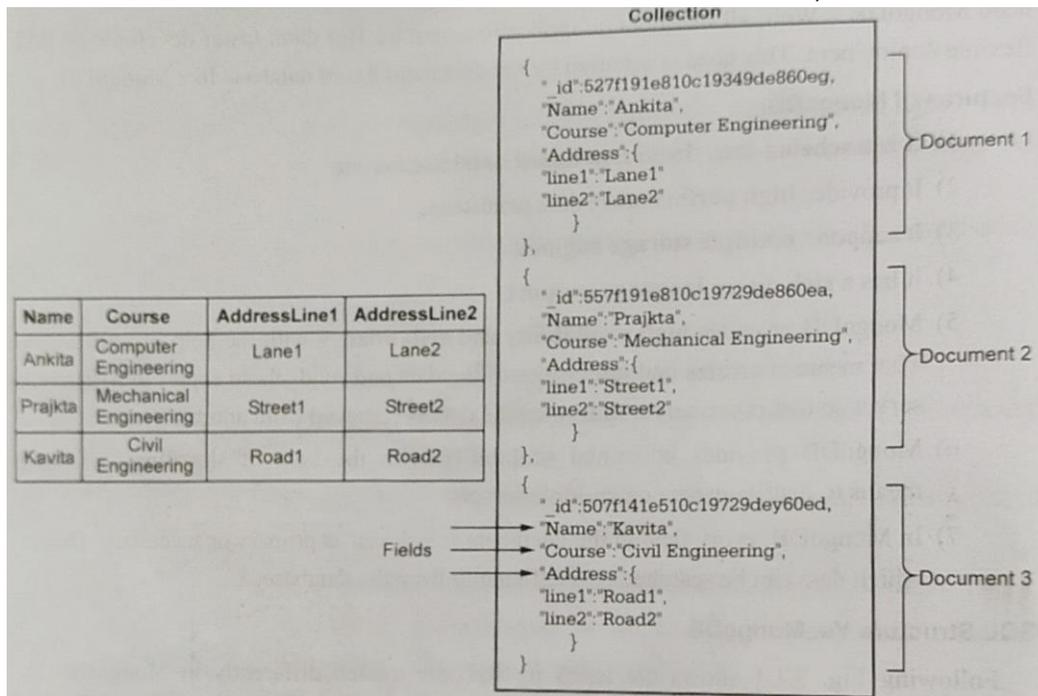
Following Fig. 3.1 shows the terms in SQL are treated differently in MongoDB. In MongoDB the data is not stored in tables, instead of that, there is a concept called collection which is analogous to the tables. In the same manner the rows in RDBMS are called documents in MongoDB, likewise the columns of the record in RDBMS are called fields.



**Figure 3.1 SQL Structure Vs. MongoDB**

## Consider a student database as follows

To the left hand side we show the database in the form of table and to the right hand side the database is shown in the form of collection,



## 14. What are the data types used in MongoDB? Explain in detail.

Following are various types of data types supported by MongoDB.

- 1. Integer:** This data type is used for storing the numerical value.
- 2. Boolean:** This data type is used for implementing the Boolean values ie.true or false.
- 3. String:** This is the most commonly used data type used for storing the string values
- 4. Double:** Double is used for storing floating point data.
- 5. Min/Max keys:** This data type is used to compare a value against the highest BSON element.
- 6. Arrays:** For storing an array or list of multiple values in one key, this data used.
- 7. Object:** The object is implemented for embedded documents.
- 8. Symbol:** This data type is similar to string data type. This data type is used specific symbol type.
- 9. Null:** For storing the null values this data type is used.
- 10. Date:** This data type is used to store current date or time. We can also own date or time object.
- 11. Binary data:** In order to store binary data, we need to use this data type.
- 12. Regular expression:** This data type is used to store regular expression.

**15. Write the MongoDB command to create and drop the database.**

**Administering Databases.**

Write the MongoDB command to create and drop the database operations.

In this section we will discuss how to create and handle database in MongoDB using various commands.

**(1) Create Database**

For example

Open the command prompt and type the command mongosh for starting the MongoDB shell.

The test> prompt will appear. For creating a database we need to “use” the command.

**Syntax**

## (2) Displaying all the databases present in the system

We can see the list of databases present in the MongoDB using the command `show dbs`

For example

```
mystudents> show
dbsAdmin 180.06
KiB
config 72.00 KiB
Local 72.00
kiBmystudents>
```

Note that in above listing we can not see the `mystudents` database. This is because we have not inserted any document into it. To get the name of the database in the listing by means of `show` command, there should be some record present in the database.

## (3) Drop Database

The `dropDatabase()` command is used to delete the database. For example

```
-Mystudents> db.dropDatabase()
{ ok: 1,dropped:
'mystudents' }mystudents>
```

## 16.Explain with suitable examples, CRUD operations in

### MongoDB. Managing Collections

After creating some sample database, we must create some collection inside that database. We can perform various operations such as insert, delete and update on this collection. These operations are called CRUD operations. CRUD stands for Create, Read, Update and Delete operation.

### 1.Create Collection

#### Syntax

We can create collection explicitly using `createCollection` command. `db.createCollection(name, options)`

where

**name** is the name of collection

**options** is an optional field. This field is used to specify some parameters such as Size, maximum number of documents and so on.

Following is a list of such options.

Field	Type	Description
-------	------	-------------

Capped	Boolean	Capped collection is a fixed size collection. It automatically overwrites the oldest entries when it reaches to maximum size. If it is set to true, enabled a capped collection. When you specify this value is true, you need to specify the size parameter.
autoIndexID	Boolean	This field is required to create index id automatically. Its default value is false.
size	Number	This value indicates the maximum size in bytes for a capped collection.
Max	Number	It specifies the maximum number of documents allowed in capped collection.

**Table 5.3 Options for create collection**

**For example** - Following command shows how to create collection in a database using explicit command

```
test> use myStudents
Switched to db
myStudents
myStudents> db.createCollection("Student_details")
{ ok: 1 }
MyStudents>
```

## 2) Display Collection

To check the created collection use the command "show collections" at the command prompt

```
myStudents> show
collections Student_details
myStudents>
```

## 3) Drop Collection

The drop collection command is actually used to remove the collection completely from the database. The drop command removes a collection completely from database.

### Syntax

```
db.collection name.drop()
```

### For Example

```
my Students>
db.Student_details.drop() true
my Students>
```

We can verify the deletion of the collection by using "show collections" in the database.

```
myStudents> show
collections myStudents>
```

#### 4) Insert documents within the collection

The document is inserted within the collection. The document is analogous to rows in database

##### **Syntax**

```
db.collection name.insert( {key, value }
```

##### **For example**

```
myStudents> show collections
nyStudents> db.create Collection("stedent details)
{ ok: 1}

myStudents> db.Student_details.insert(nameAAA" age":2)
acknowledged : true,

myStudents>
insertedIds:{ '0': ObjectId("643e4b63436497399alala66")}
}
myStudents>
```

We can verify this insertion by issuing following command myStudents> db. Student.details.find()

```
{_id: Object Id ("643e4b63436497399al a2 a66"), name: AAA, age: 22
}myStudents>
```

#### 5) Inserting Multiple Documents

It is possible to insert multiple documents at a time using a single command. Following program shows how to insert multiple documents in the existing collection.

```
my Students> var
all_students. =[
{
"name":
"BBB"
"age":20
},
{
"name" :
"DDD"
"age":21
},
];
acknowledged
:true,ent
```

```

    details.find()
    myStudents> db.Student details . Insert(allstudents);
    insertedIds: { '0' : ObjectId
    ("643e518d436197399ala2a67") '1':
    "ObiectId"643e518d436497399aia2a68")
    }
    }
myStudents>

```

To verify the existence of these documents in the collection you can use find command as follows -

```

myStudents> db.Student_details. find()
{-id: ObjectId ("643e4b63436497399ala2a66"), name: 'AAA' ,age: 22 }
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'BBB', age:
21,}
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'DDD', age:
20,}
}
myStudents>

```

## 6) Delete Documents

For deleting the document, the remove command is used. This is the simplest command

Syntax

```
db.collection_name.remove(delete_criteria)
```

### For example

First of all we can find out the documents presents in the collection using find() command

```

myStudents> db.Student_details. find()[
{-id: ObjectId ("643e4b63436497399ala2a66"), name: 'AAA' ,age: 22 }
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'BBB', age: 21,}
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'DDD', age: 20,}
]
myStudents>

```

Now to delete a record with name "DDD" we can issue the command as follows-

```

myStudents> db.Student details.deleteOne( {"name": "DDD" });
{ acknowledged: true, deletecount: 0}
myStudents>

```

Now using find() command we can verify if the desired data is deleted or not.

```

myStudents> db.Student_details. find()
{-id: ObjectId ("643e4b63436497399ala2a66"), name: 'AAA' ,age: 22 }
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'BBB', age: 21,}
}

```

## 7.Delete all documents

For deleting all the documents from the collection,we can use.deleteMany()  
For example

```
myStudents> db.Student_details.deleteMany( { });
{ acknowledged: true,
deletecount:2} myStudents>
```

We can verify it using db.Student\_details.find() command,we can verify that records are deleted or not.

```
myStudents> db.Student_details.find()
myStudents>
```

## 8.Update Documents

For updating the document, we have to provide some criteria based on which the document can be updated.

```
db.collection_name.update(criteria, update_data)
```

For example - Suppose the collection "Student\_details" contain following documents

```
myStudents> db.Student_details.find() [
{_id: ObjectId ("643e4b63436497399ala2a66"), name: 'AAA', age: 22 }
{_id: ObjectId ("643e518d436497399ala2a67 "), name: 'BBB', age: 21,}
{_id: ObjectId ("643e518d436497399ala2a67 "), name: 'DDD', age: 20,}
{_id: ObjectId ("643e518d436497399ala2a67 "), name: 'CCC', age: 23,}
```

**And we want to change the name CCC" to WWW", then the command can be issued as**

```
myStudents>db.Student_details.update({"name:"CCC"},{set:{"name":"WWW"}}) DeprecationWarning: Collection.update () is deprecated. Use updateOne, updateMany, or bulkwrite.
{
  acknowledged :
  true,InsertedId:
  null,
  matchedCount: 1,
  modifiedCount:1,
  upsertedCount:0
}
```

It can be verified using db.Student\_details.find() command  
my Students> db.Student\_details.find().

```
[
{-id: ObjectId ("643e4b63436497399ala2a66"), name: 'AAA', age: 22 }
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'BBB', age: 21,}
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'DDD', age: 20,}
{ -id: ObjectId ("643e518d436497399ala2a67 "), name: 'WWW', age: 23,}
]
```

myStudents>

Thus the document gets updated.

By default the update command updates a single document. But we can update multiple document as well. For that purpose we have to add

```
db. Student details. update({ "age":21}, {$set:{"age":23},{multi:true})
```

## 17. Write about Column Oriented Database: HBase

### Concept of Column Oriented Database

The column-oriented database is a database system that stores the data table by column rather than by rows. The advantage of column-oriented Database is that we can access the data more efficiently when only subset of columns is used for querying, For example-Consider the customer table

	<b>email address</b>	<b>gender</b>	<b>age</b>
Archana	<a href="mailto:archana123@gmail.com">archana123@gmail.com</a>	Female	30

	<b>email address</b>	<b>gender</b>
Archana	<a href="mailto:ankit123@gmail.com">ankit123@gmail.com</a>	Male

	<b>email address</b>	<b>Country</b>
Aswini	<a href="mailto:aswini123@gmail.com">aswini123@gmail.com</a>	India

we can see that a column family has several rows. Within each row, there can be several different columns, with different names, links, and even sizes

## Advantages of Column oriented database

- a. The most important benefit of column oriented database is faster performance compared to row oriented database
- b. The column-oriented database can store more data in smaller amount of memory.
- c. The column-oriented database is highly used for Big data applications.
- d. Another benefit of a column-based DBMS is self-indexing, which uses less disk space than Relational database schema.
- e. We can use aggregation for vast amount of data.

## 18. Write about Introduction to Hbase Data Model in detail.

- HBase is a column oriented non-relational database management system. It runs on top of Hadoop Distributed File System(HDFS)
- HBase is an Open-source technology developed by Apache Software Foundation.
- The HBase can store massive amount of data from terabyte to petabytes

### 1.Features of HBase Data Model

- 1) HBase is horizontally scalable. That means we can add any number of columns anytime.
- 2) The data is stored in key value format.
- 3) It is a platform for storing and retrieving data with random access.
- 4) It does not enforce relationship within the data.
- 5) It is designed to run on cluster of computers.

### 2.Difference between RDBMS and HBase

S.No	RDBMS	HBase
1	It requires SQL.	There is no SQL. in HBase.
2	It has fixed schema.	It does not have a fixed schema
3	It is row-oriented.	It is column oriented.
4	It is static in nature	It is dynamic.
5	The data retrieval is slower.	Data can be retrieved fast in HBase
6	In RDBMS, the data is normalized.	The HBase is not normalized
7	It accommodates small tables.	It accommodates large tables

### 3 .HBASE CRUD Operations

The CRUD stands for Create, Read, Update and Delete Operations. In order to communicate with HBase, the HBase Shell is used. The CRUD operations are verified by issuing the Create, put, get alter, delete, list and many more such commands. Let us discuss some commonly used commands

#### 1) Create Table:

The create command is used to create a table.

```
CREATE <table-table>',<column-family>
```

For example

```
> CREATE customer ef
```

#### 2) Read Operation:

The 'get' command is used to read the contents of row or cell.

The syntax is-

```
get<table-name>', '<row>'"
```

#### For example-

```
hbase(main):015:0> get 'customer', '1'
```

```
COLUMN      CELL
cf: ge timestamp = 1417621885277, value - 30
ef:city timestamp =1417521848375, value-Pune
cf: name timestamp = 1417521785385, value- Supriya
```

Reading a specific column

```
hbase(main):018:0> get customer', 'T', (COLUMN 'cfname')
```

```
COLUMN      CELL
```

```
cf: name timestamp = 1418035791555, value Supriya
```

```
1 row(s) in 0.0080 seconds
```

#### 3) Insert Data:

We can insert data to the table using the put command.

Now, we can create a table by inserting some data into it in column-based manner using **put** command

```
>put customer 1,'ct name", "Supriya
>put 'customer',1,'cf:city', 'Pune'
>put 'customer',1,'cfage', '30"
```

We get the table as follows

Row Key	cf:name	cf:city	cf:age
1	Supriya	Pune	30

We can check if table is created using **list** command. For example

```
> list 'customer'
```

#### 4) Update Operation

We can modify some properties of existing table using alter command. We add more column families or we can modify the table attributes.

The syntax is alter <tablename>, NAME> <column family name> VERSIONS> <new version no>

To add the column family 'status' in table 'customer' and to keep a maximum of 5 cells VERSIONS

```
>alter 'customer, NAME'status', VERSIONS =>5
```

To delete some column family 'status' we use alter command as

```
>alter customer, NAME->'status', METHOD=>delete
```

#### 5) Deletion Operation

We can delete a particular cell by using the delete command. The syntax is

For example-We can delete a cell for the column city from the customer table as follows  
hbase(main) 0060 delete customer, T. of city,  
1417621948376  
0 row 0.0060 seconds

Delete all: It deletes all the cells in a row

```
deleteall table name>,<row>
```

For example-

```
hbase(main) 0070 deleteall customer. T
```

#### (E) Display the Contents of Table

For displaying the contents of the table '**scan**' command is used. For example

```
hBase(main) 040:0> scan 'customer' NOW COLUMN+CELL
1 column =cf:age, timestamp 1505200004953, value-30
1 column =cf:city, timestamp-1505200291903, value-Pune
1 column=ct:name, timestamp=1505200278058, value-Supriya
```