PANDIT DEENDAYAL ENERGY UNIVERSITY

V.I.M.A.

Variable Insurance and Mediclaim Application

By:

Tanvi K. Modi (19BCP173D)

Ravi J. Patel (19BCP172D)

Dhrumil A. Shah (19BCP164D)

Vatsal I. Prajapati (18BCP125)

Priyanshi S. Pansara (19BCP171D)

Vainavi S. Gangvekar (19BCP174D)

Subject:

DataBase Management System

Faculty:

Mr. Nishant Doshi

Assignment-2

DBMS vs File System

Advantages of DBMS over file processing system for your database on the following points. Write meaning and N example for each.

- Data redundancy and inconsistency
- Difficulty in accessing the data
- Data isolation
- Integrity problems
- Atomicity problems
- Concurrent-access anomalies
- Security problems

1) Data redundancy:

Data redundancy means duplication of the files. Suppose you have one data file stored at two different locations and if you made the change in data at one location and forget to do so at another location, it will lead to data redundancy as the data at both the places is different. Apart from this redundancy causes the wastage of storage due to the same multiple files.

--> Example:

1) Client and Claimed Table.

- ➔ Client

  Attribute: (Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)

- ➔ Claimed

  Attribute: (Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender, Date Admitted, Date Discharged.)

2) Doctor and Hospital Table.

- ➔ Doctor
  Attribute: (ID, HID, Name, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Permit), Specialist.)
- ➔ Hospital
  Attribute: HID, Name, Address, Rating, CEO, Type (Gov Private Semi-private)

3) Insurance and InsuredPaitent Table.

- ➔ Insurance
  Attribute: (**Name**, PolicyNo, Claimed or Not, DateOfCommencement, Duration, **Amount)**
- ➔ InsuredPatient (who claimed the insurance)
  Attribute: (**Name**, HID, CauseOfClaim, **Amount**, Occupation)

4) InsuredPatient and BankDetails Table.

- ➔ InsuredPatient (who claimed the insurance)
  Attribute: (**Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender,** CauseOfClaim, Amount, Occupation)
- ➔ BankDetails
  Attribute: (**Name**, AccNo, IFSCCode, BankName, Branch)

5) Insurance and InsuranceHistory Table.

- ➔ Insurance
  Attribute: (**PolicyNo,** Claimed or Not, CoID, DateOfCommencement, Duration, Amount, CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender, Name, Type)
- ➔ InsuranceHistory
  Attribute: (**PolicyNo,** Diagnosis, InsuraceStart, InsuranceEnd, Amount, Status (Claimed or not), Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender, Name, Type)

The above problem can be solved by implementing the concept of Primary Key in the table.

2) Data inconsistency:

Data inconsistency occurs when the same data is kept in different formats. It creates unreliable information because it is difficult to know which format of information is correct. So, we can say that if the data redundancy is controlled by some means the problem of inconsistent data can be solved.

--> Example:

1) Doctor and Hospital Table.

- ➔ Doctor
  Attribute: (**DoctorName**, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Permanent), Specialist)
- ➔ Hospital
  Attribute: (Name, **DoctorName,** Address, Rating, CEO, Type (Gov Private Semi-private))

If the record of the former doctor is not deleted from the Hospital table and deleted from the Doctor table then it leads to data inconsistency.

2) InsuredPatient and BankDetails Table.

- ➔ InsuredPatient (who claimed the insurance)
  Attribute: (**Name,** DOB, Address, Phone No, E-mail, Occupation, Age, Gender**,** CauseOfClaim, Amount, Occupation)
- ➔ BankDetails
  Attribute: (**Name**, AccNo, IFSCCode, BankName, Branch)

In this table, the attribute NAME causes the data inconsistency that means if the name changed in the InsuredPatient but the value of Attribute did not change in the BankDetails Table.

3) Insurance and InsuranceHistory Table.

- ➔ Insurance
  Attribute: (**PolicyNo,** Claimed or Not, CoID, DateOfCommencement, Duration, Amount, **Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)**

- ➔ InsuranceHistory

  Attribute: (**PolicyNo,** Diagnosis, InsuraceStart, InsuranceEnd, Amount, Status (Claimed or not), **Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)**

The record of Insurance and InsuranceHistory follow so many columns in common so if we change the value in the insurance table and not in InsuranceHistory then it causes the data inconsistency.

4) Insurance and InsuredPatient Table.

- ➔ Insurance

  Attribute: (**Name**, PolicyNo, Claimed or Not, DateOfCommencement, Duration, **Amount)**
- ➔ InsuredPatient (who claimed the insurance)

  Attribute: (**Name**, HID, CauseOfClaim, Amount, Occupation)

If the record of name and amount is not deleted from the insurance table and deleted from the InsuredPatient table then data inconsistency occurs.

5) Client and Claimed Table.

- ➔ Client

  Attribute: (**Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)**
- ➔ Claimed

  Attribute: (**Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender**, Date Admitted, Date Discharged)

In this table, so many attributes are common that causes the data inconsistency that means if the attribute value changed in the Client but did not change in the Claimed Table.

Thus, we can say that if we can resolve the problem of data redundancy, data inconsistency is automatically controlled.

3) Difficulty in accessing data:

In file system data is stored in files and whenever there is the need to retrieve then we need to search manually and it is time consuming and is a tedious process. So, DBMS provides some advanced concepts like query and triggers for efficient access of data.

--> Example:

1) Insurance Table.

➔ Insurance
  Attribute: (CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount)

For example, if someone wants to find the policy no in the insurance table then one needs to write a new program every time and this will lead to difficulty in accessing the data.

2) Hospital Table.

➔ Hospital
  Attribute: (HID, Name, Address, Rating, CEO, Type (Gov Private Semi-private))

If someone wants to find a doctor's name in the hospital table then they cannot find it by manually searching in the database and need to make a program.

3) Insurance Table.

➔ Insurance
  Attribute: (CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount)

If a person searches in the insurance table for an insurance amount greater than 2,00,000 then they need to write a different program.

4) BankDetails Table.

➔ BankDetails
  Attribute: (CID, Name, AccNo, IFSCCode, BankName, Branch)

Consider a person if searching for the IFSC code in the BankDetails table then to find the IFSC code that person needs to make a program.

5) InsuredPatient Table.

- ➔ InsuredPatient (who claimed the insurance)
  Attribute: (CID, Name, HID, CauseOfClaim, Amount, Occupation)

If someone wants to list the amount of insurance greater than 3,00,000 from the InsuredPatient table then a program must be made to search in large database, but this will result in data access difficulty.

DBMS provides inBuilt searching option to search in efficient way.

4)Data isolation:

In the files system data will be scattered among multiple file which also scatters among varios formates thus it makes difficult for programers to write programs. When two or more transactions are performing simultaneously and the updates made by both are different then, the update made by one of them is lost. That is Data Isolation.

--> Example:

1. Doctor:

T1	**T2**
READ TypeOfDoctor	
	READ TypeOfDoctor
UPDATE TypeOfDoctor as “Permanent”	
	UPDATE TypeOfDoctor as “Visiting”
COMMIT	
	COMMIT

Suppose we are executing two transactions T1 and T2 to update TypeOfDoctor in table Doctor. T1 updates TypeOfDoctor to “Permanent” while T2 updates TypeOfDoctor to “Visiting”. Suppose these two transactions are executed concurrently, and T1 starts first and each step of the transaction is viewed as above. Though both transactions are concurrent each step in them takes a very minute fraction of seconds to execute and the above table shows those steps. This shows how they affect the result of the transaction at those minute intervals on concurrency. Here what will be the result? TypeOfDoctor is updated to ‘Visiting’. What happens to T1’s update? It is lost!

2. InsuredPatient:

T1	T2
READ Amount	
Add Amount = 1Lac	
	READ Amount
COMMIT	
	ADD Amount = 1Lac
	COMMIT

Imagine the update is something like Adding 1Lac amount in an InsuredPatient table as above steps and T2 can read uncommitted update by T1. What happens to Amount? It will be incremented twice – once by T1 and secondly by T2 which increments the Amount updated by T1. This will lead to incorrect data.

3. Doctor

T1	T2
READ Last_Name	
UPDATE Last_Name as “S”	
	READ Last_Name
	UPDATE Last_Name as “Sharma”
COMMIT	
	COMMIT

Suppose T2 reads the data after T1 updates the Last_name but before commit. Here T2 will execute as if it is unaware of T1’s update. Hence T2 will read Last_name as ‘S’ and update it. Now Last_name is ‘Sharma’. But what happens to T1’s update? It is lost. It is nowhere saved and no record of its status!

4. Insurance

T1	T2
	READ Duration
	UPDATE Duration to 3 years
READ Duration	
	COMMIT
UPDATE Duration to 5 Years	
COMMIT	

Assume, T1 read Duration just after T2 updates it but before commit. Here T1 executes the query but because T2 commits the query so T1 gets wrong information and so T1 changes the value to 5 years. Now Duration changed to 5 Years. So, the data entered by T2 is gone forever!

5. InsuranceHistory

T1	T2
	READ InsuranceEnd
READ InsuranceEnd	
	UPDATE InsuranceEnd to 02-04-2022
	COMMIT
UPDATE InsuranceEnd to 02-08-2022	
COMMIT	

Presume, T1 wants to change InsuranceEnd date in InsuranceHistory table. Here T1 will execute as if it is unaware of T2's update. Therefore, T1 read 02-04-2022 and update it. Now, InsuranceEnd is updated to 02-08-2022. So, the data updated by T2 is replaced and not saved anywhere. It lost!

To solve this problem, there is an advanced concept called LOCKs that can be implemented after a query.

5) Integrity problems:

Integrity is mainly when the data is not consistent and not reliable. The data values that are stored in the database must satisfy Integrity constraints. Integrity Constraints helps the user whenever any data is inserted, updated or any operation is performed. Also, it is used so that the data does not do permanent damage to the whole database by giving wrong or inaccurate data.

--> Example:

1) BankDetails Table.

Attributes: (CID, Name, **AccNo**, IFSCCode, BankName, Branch)

In the file System, If the AccNo is not correct and consistent then there is a big problem to access the particular person's account.

But in the DBMS, there is a certain rule that must follow the user to add the details like if you want to write AccNo then we may apply a rule that it must be 12 numbers.

2) Client Table.

Attributes: **(CID**, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)

In the Client table if the particular members' Id is not correct and it is repeating then there is a problem for referencing a correct person.

So, for this the rule we can apply that it must be UNIQUE and NOT NULL value.

3) Doctor Table.

Attribute: (**ID, HID**, Name, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Permanent), Specialist)

For the Doctor table the ID and HID (Hospital ID) must be different because if the ID is repeated then the machine gets confused which data is correct.

4) Insurance Table.

Attribute: (CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount)

In the Insurance table, when the client claims the insurance then we can set the condition that the amount must not be a negative value as well as it's not less than 10,000 Indian Rupees.

5) Hospital Table.

Attributes: **HID**, Name, Address, Rating, CEO, Type (Gov Private Semi-private)

For the Hospital Table There are so many hospitals that contain the same name and because of that it leads to Integrity problems.

So, as a solution we can provide the HID as a unique and NOT NULL so that when we need any particular Hospital, we can access the data of that hospital.

6) Atomicity of updates:

Any operation on the database must be atomic. This means, it must happen entirely or not at all. If any operation fails due to some problem, such as system crash, then the effect of the partially executed operation can be undone.

--> Example:

1) **Hospital Table and Doctor Table.**

Doctor: ID, HID, Name, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Perment), Specialist.

Hospital: HID, Name, Address, Rating, CEO, Type (Gov Private Semi-private)

Let us say if the program for deleting a former doctor from the hospital table is executing meanwhile the system fails, it may happen that the record is deleted from the hospital table, but it is not updated in the doctor table. And hence it leads to an atomicity update problem.

2) **Client Table and Company Table.**

Client: CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender

Company: CoID, Name, Type,

If we execute a program of deleting a particular client from the company and while doing so if something goes wrong in the system and if the record is not deleted in the company table but it is deleted from the client table, it causes atomicity of updates.

3) **Client Table and Insurance Table.**

Client: CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender

Insurance: CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount.

Assume that the program of entering the records of the client who has taken the policy is executing but if the system crashes while execution and the entry of that record is not done in the company table; but it is done in client table. Here the operation is not atomic.

4) Insurance and Insurance History Table.

Insurance: CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount.

InsuranceHistory: CID, PolicyNo, Diagnosis, InsuraceStart, InsuranceEnd, Amount, Status (Claimed or not), CoID.

A program is executing for updating the ‘claimed or not’ column of the Insurance table and during execution a problem arises, and the record is updated in the Insurance table but fails to change in Insurance History Table and thus leads to the problem of atomicity updates.

5) Client Table and Bank Details Table.

Client: CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender

BankDetails: CID, Name, AccNo, IFSCCode, BankName, Branch.

Suppose a program of updating the Client’s personal information is running for both Client and Bank Details Table and due to system crash the record is updated only on Client table and the Bank details table is not updated, here this failure leaves the database in an inconsistent state with partial updates carried out.

Thus, to guarantee atomicity, the DBMs software must be able to make sure that all the transactions are committed to the database, one NONE is updated to the database.

7) Security problems:

There is no centralized control of the data in classical file organization due to which security enforcement is difficult in the File-processing system. The database management system has specialized features that help to provide shielding to its data.

--> Example:

1) **Doctor and Hospital Table.**

   ➔ Doctor

   Attributes: (ID, HID, Name, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Perment), Specialist)

   ➔ Hospital

   Attributes: (HID, Name, Address, Rating, CEO, Type (Government, Private, Semiprivate)).

Here, if the Doctor table is given access to the Hospital table, then this will result in security problems as anything that will be changed by the doctor will be changed in the Hospital table.

2) **Client and Insurance Table.**

   ➔ Client: Doctor

   Attributes: (CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)

   ➔ Insurance

   Attributes: (CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, Amount)

A client should not be able to access the insurance table as the insured company is responsible for all the details mentioned in it. So, the Insurance table should not allow or give access to the Client table.

3) **Company and BankDetails Table.**

➔ Company

Attributes: (CoID, Name, Type)

➔ BankDetails

Attributes: (CID, Name, AccNo, IFSCCode, BankName, Branch)

The insurance company should not be able to make any changes in the bank details like accessing the names or changing the bank details of any client. If it does so, then this will lead to a problem of security.

4) **Client and Hospital Table.**

➔ Client

Attributes: (CID, Name, DOB, Address, Phone No, E-mail, Occupation, Age, Gender)

➔ Hospital

Attributes: (HID, Name, Address, Rating, CEO, Type (Gov Private Semi-private))

In this example, the client should be denied editing anything in the Hospital table because if the client mistakenly adds or deletes anything from the Hospital table then that will become a major issue.

5) **InsuredPatient and Doctor Table.**

➔ InsuredPatient

Attributes: (CID, Name, HID, CauseOfClaim, Amount, Occupation)

➔ Doctor

Attributes: (ID, HID, Name, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Perment), Specialist)

A doctor can access the details related to a patient, but a patient can't be given access to change the Doctor table because the client doesn't have much knowledge regarding the doctor's table due to which it may result in problems.

To solve the security issues of file systems, DBMS has specialized features that help provide shielding to its data. Database should be accessible to users in a limited way. Each user should be allowed to access data concerning his requirements only.

8) Concurrent access by multiple users:

In classical file organization there is no central control of data. So, the concurrent access of data by many users is difficult to implement. Anomalies occur when changes made by one user get lost because of changes made by another user. File system does not provide any procedure to stop anomalies whereas DBMS provides a locking system to stop anomalies to occur.

--> Example:

1) Doctor Table.

- ➔ Doctor

  Attribute: (**DoctorName**, DOB, Address, Phone No, E-mail, Age, Gender, TypeOfDoctor (Trainee, Visiting, Permanent), Specialist)

Suppose User 1 tries to access the DoctorName Attribute to add a new row. Also, User 2 also tries to access the same attribute to do the same operation at the same time. If both the users do the same operation, Data Redundancy will increase as Data Duplication occurs. So, DBMS puts locks to the users which allows a single user to do the operation at a particular time.

2) Company and Claimed Table.

- ➔ Company

  Attributes: (**CoID**, Name, Type)

- ➔ Claimed

  Attributes: (CID, **CoID**, Date Admitted, Date Discharged. Company)

User 1 should do the operation on the CoID and after doing the necessary operation User 2 shall delete the CoID field after the operation. If User 2 deletes the CoID field before the operation, then the operation will not occur. To prevent this, DBMS handles this type of situation.

3) **BankDetails Table.**

➔ BankDetails

Attributes: (CID, Name, **AccNo**, IFSCCode, BankName, Branch)

Assume, if Bank AccNo needs to be changed before doing any transaction as it is not correct. Now, before User 1 changes the account number, User 2 tries to do operation taking the account number. So, this leads to wrong transactions and so DBMS prevents it by its solution.

4) **Insurance Table.**

➔ Insurance

Attribute: (CID, PolicyNo, Claimed or Not, CoID, DateOfCommencement, Duration, **Amount**)

If a person changes its policy (as a result changes the Insured amount) and before updating the Policy amount, if the Client claims its insurance, then the Client will be given the insurance based on the previous policy amount, which leads to Data Inconsistency and to prevent this, DBMS provides lock system so that Client could not claim any amount before the policy is updated.

5) **Doctor Table.**

➔ Doctor

Attribute: (ID, HID, Name, DOB, Address, Phone No, **E-mail**, Age, Gender, TypeOfDoctor (Trainee, Visiting, Permanent), Specialist)

When entering the Doctor’s Information, the email is not correct as it was not verified when it was entered by any means. So, when the Company sends an email about the patient to the Doctor about the patient, the Doctor does not receive it and because of this the client does not receive any amount as it is not verified by the Doctor. To prevent this, DBMS provides features to prevent these types of situations.

To prevent this type of situation, DBMS uses lock so that two users do not collide with each other's work and thus decreases Data Redundancy, Data Inconsistency and all other problems.

Assignment- 3

Development of relational model for your Database.
Development of E-R model for your Database.

- ## FEATURE:

HIPAA:
In this features the confidentialities between patients and doctor not compromised.

Policy Comparison:
In this feature we can compare two company's policies and choose best of them.

Centralized Data:
This feature helps the companies to access the data from a single source rather than looking for any other difference source every time the data changes.

Nearby Locator:
As per Current location of the user it finds our network locators. It will find nearby hospitals, pharmacy, and our Mediclaim company branches near them.

Security policy:
Any unknown person cannot access any company's data; it is because of the V.I.M.A database management system security policy rules.

- **TABLES:**

1. **Sign up**
   Attributes: User_ Name, E-Mail , Contact ,Gender, Date_Of_Birth, Password, Confirm password.

2. **Login**
   Attributes: **User_ID**, User_Name , Password

Keys	Attributes
Primary Key	User_ID
Candidate Key	User_ID
Super Key	User_ID, User_Name

3. **Client**

   Attributes: **Client_ID**, Client_Name, Client_DOB, Client_Address, Client_Contact, Client_Income, Client_Gender, Client_E-mail

Keys	Attributes
Primary Key	Client_ID
Candidate Key	Client_ID,Client_Email
Super Key	Client_ID,Client_Email,Client_Contact

4. **Company**

Attributes: **Company_ID**, Company_Name, Company_Type, Company_E-mail

Keys	Attributes
Primary Key	Company_ID
Candidate Key	Comapany_ID,Comany_Email
Super Key	Company_ID, Company_Email, Comapny_Name

5. **Policy List**

Attributes: Company_ID, Policy_List

Keys	**Attributes**
Foreign Key	Company_ID

6. **Insurance**

Attribute: **Policy_Number**, Client_ID, Company_ID, New_Policy, Renew_Policy, Insurance_Amount, DateOfCommencement, Insurance_Duration

Keys	**Attributes**
Primary Key	Policy_Number
Candidate Key	Policy_Number
Super Key	Policy_Number
Foreign Key	Client_ID, Company_ID

7. **Renew table**

Attributes: Client_ID, Company_ID, Policy_Number, New_DateOfCommencement, New_Insurance_Duration

Keys	**Attributes**
Foreign Key	Client_ID,Company_ID, Policy_Number

8. **Hospital**

Attributes: **Hospital_ID**, Hospital_Name, Hospital_Address, Hospital_Rating, Hospital_Contact, Hospital_Representative, Hospital_Type

Keys	**Attributes**
Primary Key	Hospital_ID.
Candidate Key	Hospital_ID, Hospital_Name
Super Key	Hospital_ID, Hospital_Name, Hospital_Representative

9. **Doctor**

Attribute: **Doctor_ID**, Hospital_ID, Doctor_Name, Doctor_DOB, Doctor_Address, Doctor_Contact, Doctor_E-mail, Doctor_Gender, Doctor_Type, Doctor_Specialization.

Keys	**Attributes**
Primary Key	Doctor_ID
Candidate Key	Doctor_ID,Doctor_Email
Super Key	Doctor_ID,Doctor_Email,Doctor_Contact
Foreign Key	Hospital_ID

10. **BankDetails**

Attributes: **Bank_ID**, Client_ID, Bank_Name, Client_AccNo, Client_IFSC, Bank_Branch

Keys	**Attributes**
Primary Key	Bank_ID
Candidate Key	Bank_ID, Bank_Branch
Super Key	Bank_ID, Bank_Branch ,Bank_Name
Foreign Key	Client_ID

11. Damage

Attributes: **Damage_ID**, Client_ID, Company_ID, Bank_ID, Policy_Number, Damage_Cause, Date_Admitted, Date_Discharged

Keys	Attributes
Primary Key	Damage_ID
Candidate Key	Damage_ID
Super Key	Damage_ID, Damage_Cause
Foreign Key	Client_ID,Company_ID,Bank_ID ,Policy_Number

12. Accountability

Attributes: Client_ID, Company_ID, Claimed_Amount , Paid_Amount

Keys	Attributes
Foreign Key	Client_ID,Company_ID

13. Hippa

Attributes : Client_ID, Company_ID, Hospital_ID, Doctor_ID, Law_Violation

Keys	Attributes
Foreign Key	Client_ID,Company_ID Hospital_ID, Doctor_ID

14. GroupInsurance

Attributes: **Group_ID**, Company_ID, Members, Head_DOB, Head_Address, Head_Contact, Head_Income, Head_Gender, Head_E-mail

Keys	Attributes
Primary Key	Group_ID.
Candidate Key	Group_ID,Company_ID,Head_Email
Super Key	Group_ID, Head_Email
Foreign Key	Company_ID

15. Rating

Attributes: Name, Rate, Review

16. Privileges

Attributes: Company_ID, **Employee_ID,** AccessRights

Keys	Attributes
Primary Key	Employee_ID
Foreign Key	Company_ID

17. TimeStamp

Attributes: Company_ID, Employee_ID, Data_Access, Time_Access

Keys	Attributes
Foreign Key	Company_ID, Employee_ID

18. Tariff

Attributes: Tariff_ID, Amount, Duration

Keys	Attributes
Primary Key	Tariff_ID
Candidate Key	Tariff_ID
Super Key	Tariff_ID

19. Subscription

Attributes: Company_ID, Tariff_ID, Activation_Date, Expiry_Date

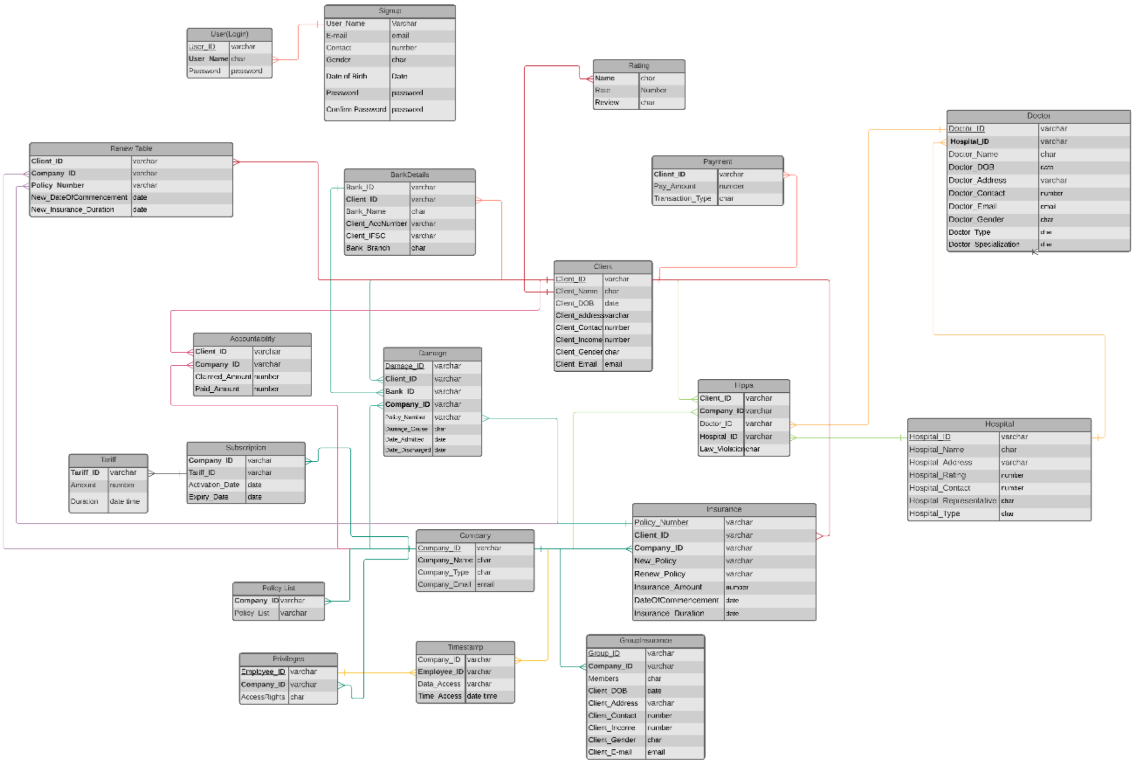
Keys	Attributes
Foreign Key	Company_ID, Tariff_ID

20. Payment

Attributes: Client_ID, Amount, Transaction_Type

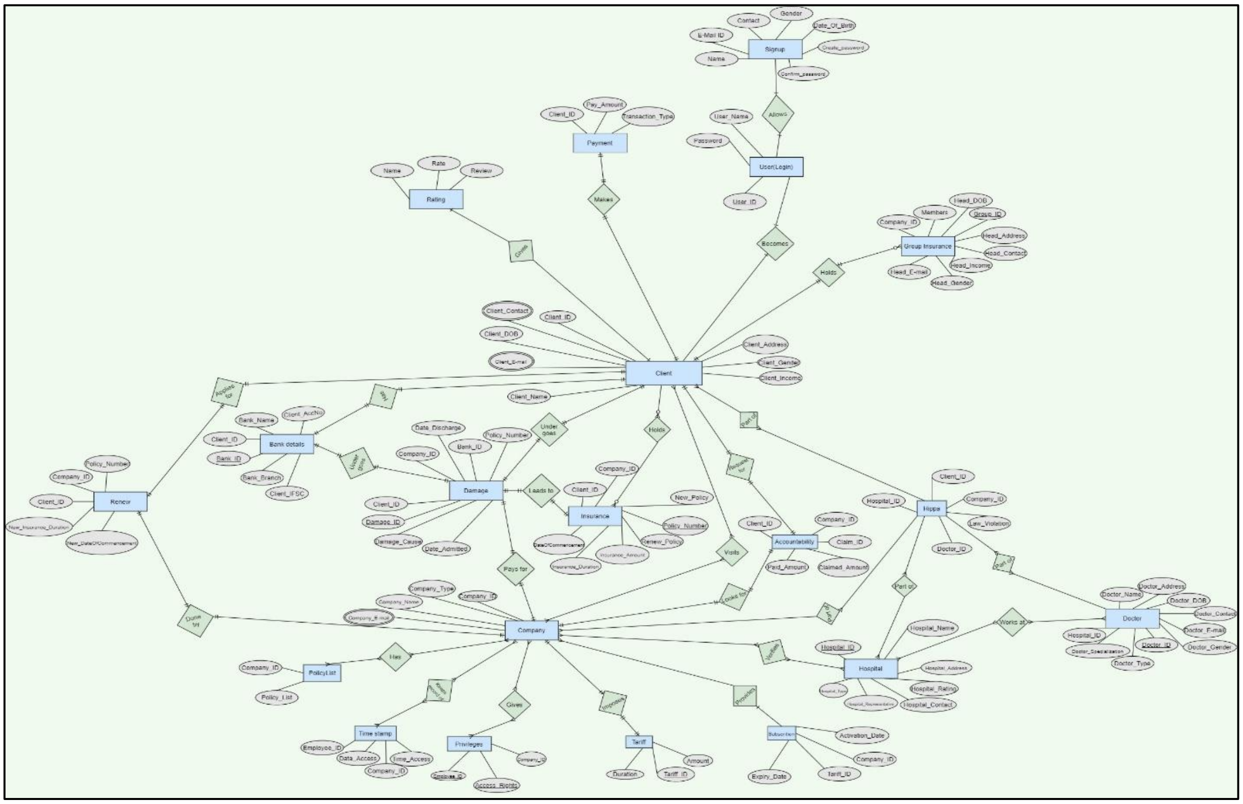
Keys	Attributes
Foreign Key	Client_ID

➢ Relational Model:



Underline represents Primary Key
BOLD represents Foreign Key

➢ Entity-Relationship Daigram:



Assignment-4

Relational Algebra queries for your system. Clearly write the definition as well as relational algebra queries for each.

1)Selection: -

- Selection Operator (σ) is a unary operator in relational algebra that used for the selection operation in database.

- It selects those rows from the relation that satisfies the condition that given in the algebra.

Example:

1. Selecting clients of ICICI bank from Bank Details Table.
   Algebra: σ_{BankName}=ICICI(Bank Details)

2. Selecting senior citizen clients from Client Table.
   Algebra: $\sigma_{\text{Age\_age>=60}}$(Client Details)

3. Selecting doctors from Ahmedabad from Doctor Table.
   Algebra: $\sigma_{\text{Address=Ahmedabad}}$(Doctor Details)

4. Selecting Insurance duration 2 years from Insurance Table.
   Algebra: $\sigma_{\text{Duaration=2years}}$(Insurance Details)

5. Selecting Private Hospitals from Hospital Table.
   Algebra: $\sigma_{\text{Type=”Private”(}}$Hospital Details)

2)Projection: -

- Projection Operator (π) is a unary operator that performs a projection operation for the algebra.
- It displays the columns of a table based on the specified attributes.

Example:

1. Retrieve the Client_ID from the Client table.
   Algebra:$\pi_{\text{Client\_ID}}$ (Client)
2. Retrieve the Company_Name and Company_Type of all the companies from the Company table.
   Algebra: $\pi_{\text{Company\_Name, Company\_Type}}$ (Company)
3. Show the Doctor_ID who gave service to the patient and also the Client_ID for the Hippa law violation details from the Hippa table.
   Algebra: $\pi_{\text{Doctor\_ID, Client\_ID, Law\_Violation}}$ (Hippa)
4. Retrieve the Policy_Number from the Insurance table.
   Algebra: $\pi_{\text{Policy\_Number}}$ (Insurance)
5. Show the Hospital_Name and Hospital_Rating from the Hospital table.
   Algebra:$\pi_{\text{Hospital\_Name, Hospital\_Rating}}$ (Hospital)

3)Cartesian Product: -

- Cartesian Product is an operation used to merge columns from two tables.
- It is never a meaningful if it performs alone but it is meaningful when other operation are performs along with them and also called Cross Product.

Example:

1. Perform Cartesian product between Company and Policy_List
   Algebra: (Company x Policy List)

2. Show the Cartesian product between Company and Privileges
   Algebra:(Company x Privileges)

3. Show the cross product between Privileges and TimeStamp Relation
   Algebra: (Privileges x TimeStamp)

4. Perform the Cartesian product of Client and Accountability relation
   Algebra:(Client x Accountability)

5. Cartesian product of Company and Insurance relations
   Algebra:(Company x Insurance)

4)Union: -

- Union operator is denoted by ∪ symbol and it is used to select all the rows from any two tables.

- If we have two tables R1 and R2 both have same columns and we want to select all the rows from these relations then we can apply the union operator on these relations.

Note: The rows that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Example:

1. Retrieve the client id of the client from the client table and their current insurance policy number from the Insurance table.
   Algebra: ∏ Client_ID(Client) ∪ ∏ Policy_number(Insurance)

2. Retrieve the client Id that have account in the bank.
   Algebra:∏ Client_ID(Client) ∪ ∏ Client_ID(Bank details)

3. Retrieve the doctor id that works at the hospital.
   Algebra:∏ Hospital_ID(Hospital) ∪ ∏ Hospital _ID(Doctor)

4. Retrieve the clients that makes the payment for the Insuranse.
   Algebra:∏ Client_ID(Client) ∪ ∏ Client_ID(Payment)

5. Retrieve the Company Id that claim the amount for an Insurance.
   Algebra: ∏ Company_ID(Company) ∪ ∏ Company_ID(Accountablity)

5)Set difference: -

- This operation is used to find data present in one relation and not present in the second relation.
- This operation is also applicable on two relations, just like Union operation.

Example:

1. Show the insurance list of LIC from Insurance table, and performing minus function on Company.
   Algebra:Insurance(Company_ID,Policy_number)Company(Company_ID,Company_name)
2. Display list of doctors that are in V.S. Hospital.
   Algebra:Doctor(Doctor_ID,Doctor_name,Hospital_ID) - Hospital(Hospital_ID)
3. Display clients that have not renewed the policy.
   Algebra:Client(Client_ID) - Renew Table(Client_ID)
4. Show employees that has write permission
   Algebra:Employee(Enployee_ID, Company_ID) - Privileges(Employee_ID, Company_ID, Acccess_Rights(Read))
5. Retreive employees of companies who does not have accounts section access
   Algebra:Employee(Employee_ID,Company_ID) -TimeStamp(Employee_ID, Company_ID, Data_Access(Accounts)

6)Natural join: -

- It is a JOIN operation that creates an implicit join for user based on the common columns.

- In the two tables being joined Common columns are columns that have the same name in both tables.

Example:

1. Attributes of Policy_List and Company table relate with each other so we can use join to use all the attributes.
   Algebra: Policy ⋈ Company

2. As there are relation between Company and Privileges we can easily give privileges if we use natural join.
   Algebra: Company ⋈ Privileges

3. We can check accountability with client table by using natural join.
   Algebra: Client ⋈ accountability

4. Any company can check their client’s insurance details if join is used.
   Algebra: Company ⋈ Insurance

5. We can join hospital and doctor table to get both hospital and doctor’s detail in a single query.
   Algebra: Doctor ⋈ Hospital

7)Composition of any two from (1-6) operators.

Example:

1. Retrieve the client id of the client from the client table and their current insurance policy number from the Insurance table.
   Algebra: $\prod$ Client_ID(Client) $\cup$ $\prod$ Policy_number(Insurance).

2. Show the Bank_branch of each Client holder, identified by their name.
   Algebra: $\Pi_{\text{bank\_Branch, Client\_Name}}$(Bank details$\bowtie$ Clent)

3. Retrieve the name of Client whose Client_id is ‘23A5’
   Algebra: $\Pi_{\text{Client\_Name}}$ ($\sigma_{\text{Client\_id='05'}}$ (Client))

4. Show the amount that pay for the Mediclaim by the client and name of client.
   Algebra: $\pi_{\text{Pay\_Amount, Client\_Name}}$ (Payment $\bowtie$ Client)

5. Retrieve the Insurance amount of the Insurance whose amount is greater than 5.00.000.
   Algebra: $\Pi_{\text{Insurance\_Amount}}$ ($\sigma_{\text{amount > 5.00.000}}$ (Insurance))

8) Composition of any three of above (1-6) operators.

Example:

1. Retrieve the Damage cause of the Client whose damage caused is car accident.
   Algebra: $\Pi_{\text{Client\_Id}}$ (Client) - $\Pi_{\text{Client\_Id}}$($\sigma_{\text{Damage_Cause = "car accident"(}}$Damage))

2. Show the names and email of Client who have made online transactions
   Algebra: $\Pi_{\text{Client\_name, Client\_email}}$ ($\sigma_{\text{type = UPI}}$ (Client $\bowtie$ Payment) U $\sigma_{\text{transaction\_ type = net-banking}}$ (Client $\bowtie$ Payment))

3. Name the company id and name that have write privilege for the database.
   Algebra:$\pi_{\text{Company\_Id,Company\_Name}}$(Hospital)U$\sigma_{\text{Access_Rights="Write"(}}$Privilages)

4. Retrieve the name of client who have new date of insurance after 05-02-2021
   Algebra: $\pi_{\text{Client\_Name}}$ (Client) U($\sigma_{\text{New\_Insurance\_Duration ="05-02-2021}}$(Renew))

5. Find names of all Doctor associated with ‘Indus’ Hospital.
   Algebra:$\Pi_{\text{Doctor\_Name}}$ ($\sigma_{\text{Hospital-Name='Indus'}}$((Doctor $\bowtie$ Hospital) $\bowtie$ (Doctor$\bowtie$Hospital))

Assignment-5

Study of SQL system and implementing various commands

- To learn basics of software that can be used for SQL Query evaluations**.**

For the database purpose, we use the **phpMyAdmin** as a server.

phpMyAdmin:

It is a free and open-source administration tool and one of the most famous tools for the database purpose.
The reason behind the using this software is it is managing centrally so that all the members from our team can do the query and perform their particular tasks.
It is very easy to do a complex query and run that on the server.

Requirement to Run the phpMyAdmin:

1. Xampp server. (we use version-3.2.4)

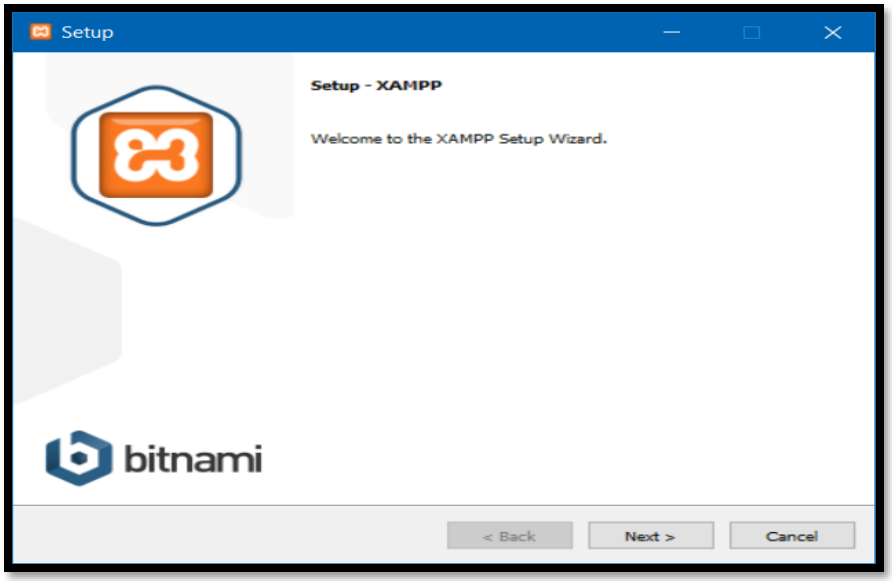
Installation steps for Xampp server:

1) Firstly, copy the following link on the browser.

https://www.apachefriends.org/download.html

select the version that you want to install and download the .exe file.

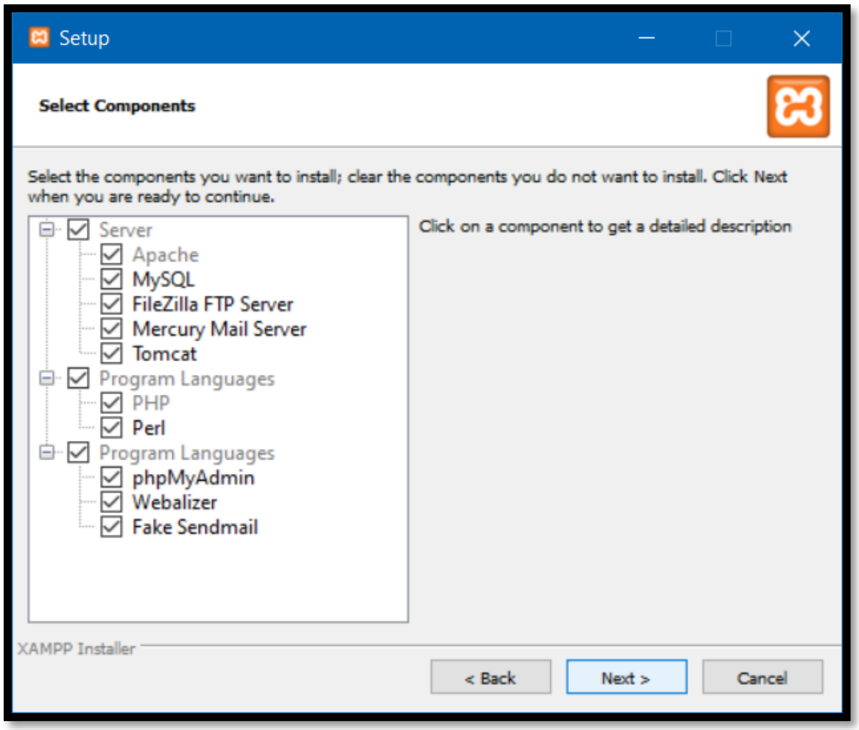
2) After that run the .exe file and you can see the following snapshot.



Setup - XAMPP

Welcome to the XAMPP Setup Wizard.

3) You have to click next until the server gets installed.



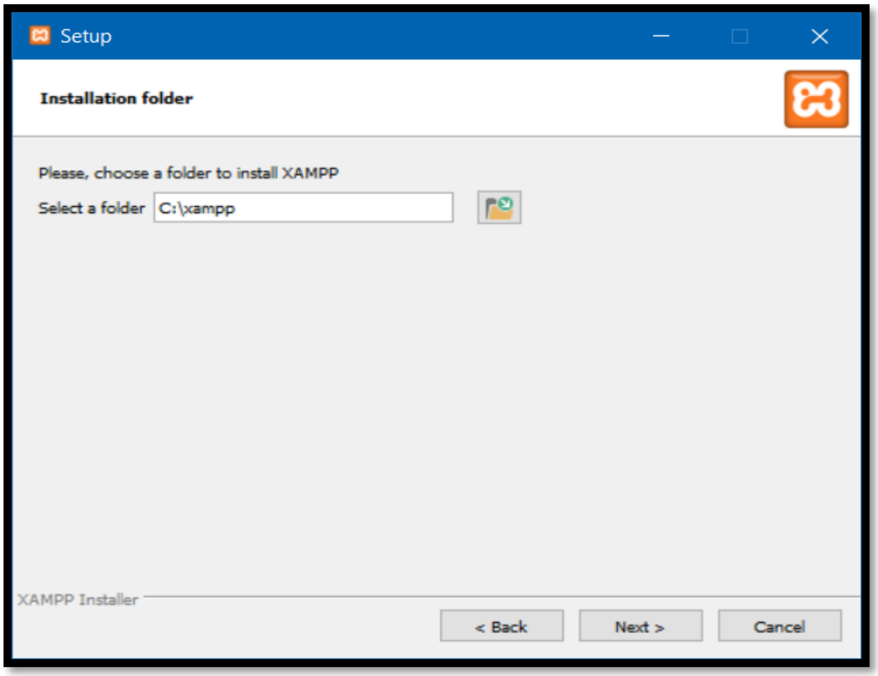
Setup

Select Components

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☑ Server
  - ☑ Apache
  - ☑ MySQL
  - ☑ FileZilla FTP Server
  - ☑ Mercury Mail Server
  - ☑ Tomcat
- ☑ Program Languages
  - ☑ PHP
  - ☑ Perl
- ☑ Program Languages
  - ☑ phpMyAdmin
  - ☑ Webalizer
  - ☑ Fake Sendmail

Click on a component to get a detailed description

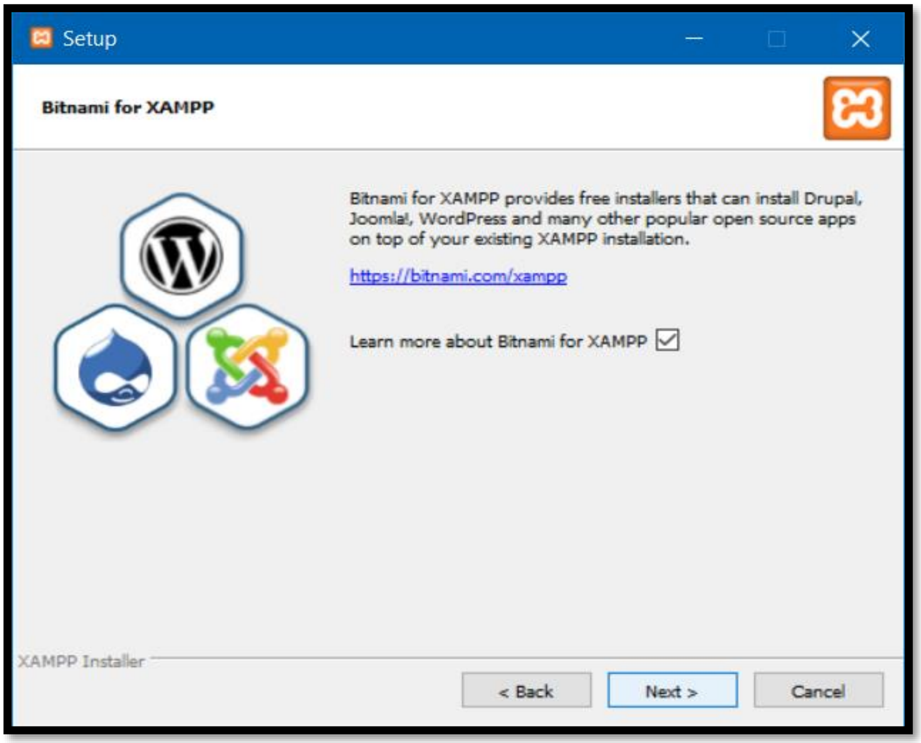


Setup

Installation folder

Please, choose a folder to install XAMPP

Select a folder C:\xampp

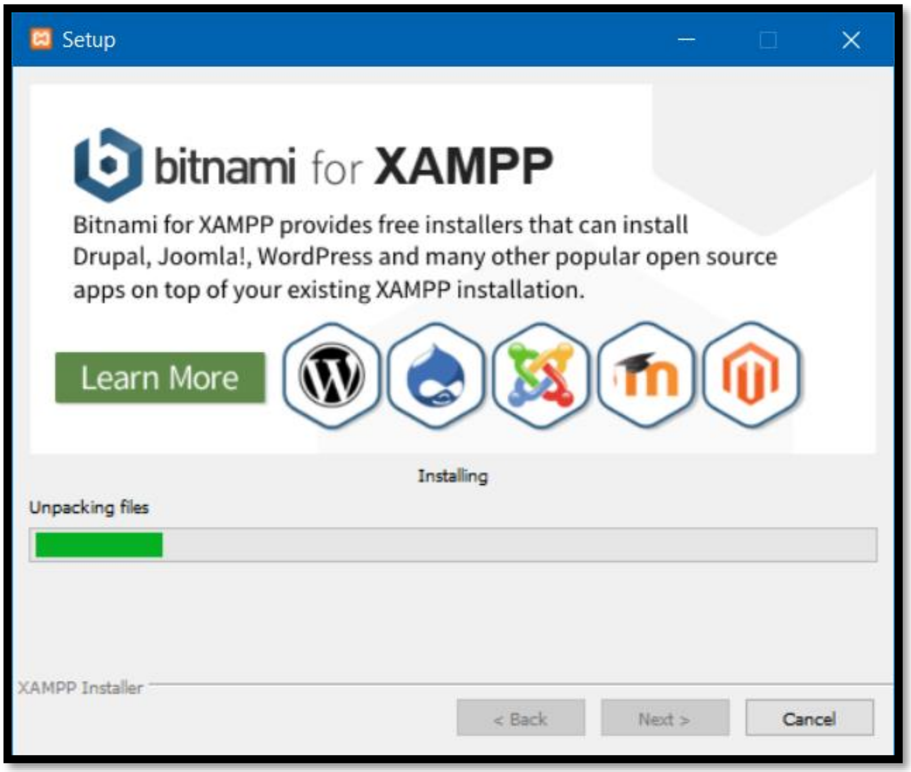


Bitnami for XAMPP

Bitnami for XAMPP provides free installers that can install Drupal, Joomla!, WordPress and many other popular open source apps on top of your existing XAMPP installation.

https://bitnami.com/xampp

Learn more about Bitnami for XAMPP ☑



Bitnami for XAMPP provides free installers that can install Drupal, Joomla!, WordPress and many other popular open source apps on top of your existing XAMPP installation.

Installing

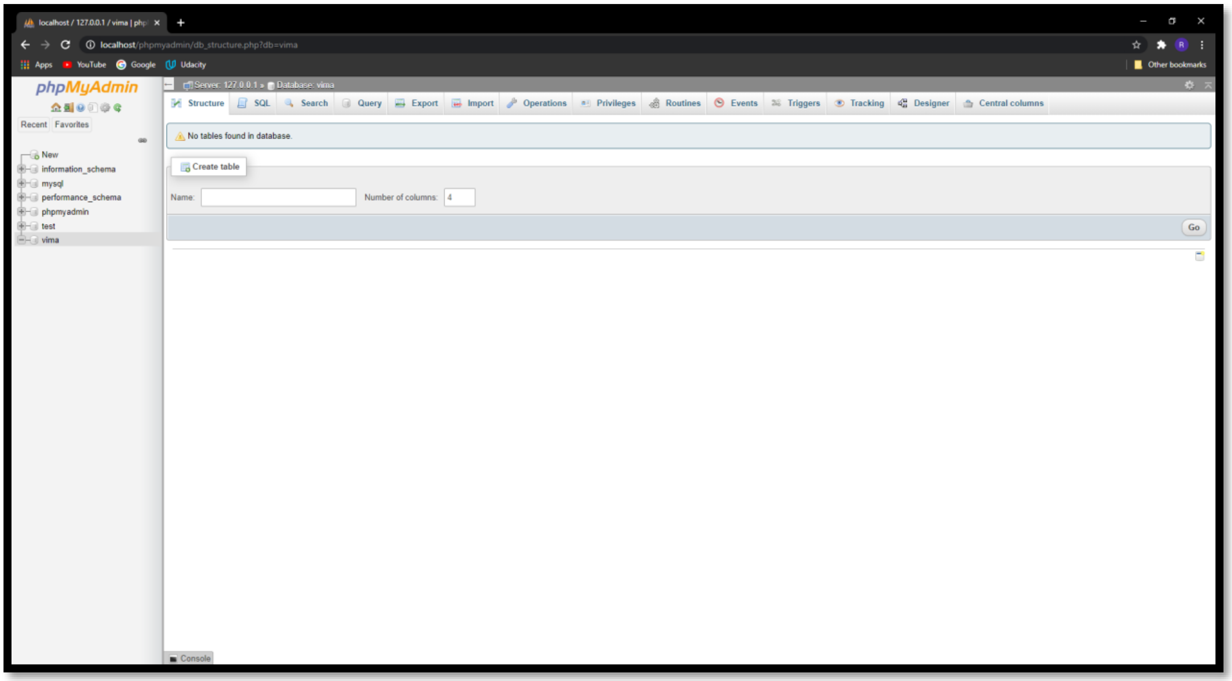
Unpacking files

After that your Xampp will be installed

4) After the installation, search for the following link on the browser.

http://localhost/phpmyadmin/

after this, you have to create the DATABASE for your system and you can see the following screen.



localhost/phpmyadmin/db_structure.php?db=vima

Server: 127.0.0.1 » Database: vima

No tables found in database.

Create table

Name:

Number of columns: 4

•Introduction to SQL, DDL, DML, DCL, database and table creation, alteration,defining Constraints, primary key, foreign key, unique, not null, check, IN Operator

DDL Commands:

It contains the following commands:-

CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

DROP – is used to delete objects from the database.

ALTER-is used to alter the structure of the database.

TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

COMMENT –is used to add comments to the data dictionary.

RENAME –is used to rename an object existing in the database.

Table Creation:

1.Client Table

```
CREATE TABLE Client(
Client_ID numeric(5) PRIMARY KEY, Client_Name CHARACTER(30) NOT
NULL,
Client_DOB DATE, Client_Address varchar(50),
Client_Contact numeric(10) UNIQUE, Client_Income numeric(10),
Client_Gender CHARACTER(6), Client_Email varchar(40) );
```

2.Sign_up Table CREATE TABLE Sign_up(

```
User_Name VARCHAR(20) NOT NULL, Email VARCHAR(40),
Contact NUMERIC(10) UNIQUE, Gender CHARACTER(6),
DOB DATE, User_Password VARCHAR(20), Confirm_Password
VARCHAR(20));
```

3.Login Table

```
CREATE TABLE Login(
User_ID VARCHAR(15) PRIMARY KEY,
User_Name VARCHAR(20), User_Password VARCHAR(20));
```

4.Policy_List Table

```
CREATE TABLE Policy_List(
Company_ID VARCHAR(5), PolicyList VARCHAR(20),
FOREIGN KEY (Company_ID) REFERENCES Company(Company_ID));
```

5.Company Table

```
CREATE TABLE Company(
Company_ID VARCHAR(5) PRIMARY KEY,
Company_Name CHARACTER(30) UNIQUE, Company_Type CHARACTER(20),
Company_Email VARCHAR(40));
```

6.Insurance Table

```
CREATE TABLE Insurance(
Policy_Number VARCHAR(10) PRIMARY KEY, Client_ID Numeric(5),
Company_ID VARCHAR(5), New_Policy VARCHAR(50), Renew_Policy
VARCHAR(50), Insurance_Amount NUMERIC(7), DateOfCommencement
DATE, Insurance_Duration DATE,
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID),
FOREIGN KEY(Company_ID) REFERENCES Company(Company_ID));
```

7.Renew Table

```
CREATE TABLE Renew(
Client_ID Numeric(5), Company_ID VARCHAR(5), Policy_Number
VARCHAR(10),
New_DateOfCommencement DATE, New_Insurance_Duration DATE,
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID), FOREIGN
KEY(Company_ID) REFERENCES Company(Company_ID),
FOREIGN KEY(Policy_Number) REFERENCES Insurance(Policy_Number));
```

8.Hospital Table

```
CREATE TABLE Hospital(
Hospital_ID VARCHAR(5) PRIMARY KEY,
Hospital_Name CHARACTER(30) UNIQUE, Hospital_Address
VARCHAR(50), Hospital_Rating NUMERIC(5), Hospital_Contact
NUMERIC(10) NOT NULL, Hospital_Representative CHARACTER(30),
Hospital_Type CHARACTER(10));
```

9.Doctor Table

```
CREATE TABLE Doctor(
Doctor_ID VARCHAR(5) PRIMARY KEY,
Hospital_ID VARCHAR(5), Doctor_Name CHARACTER(30), Doctor_DOB
DATE, Doctor_Address VARCHAR(50), Doctor_Contact NUMERIC(10),
Doctor_Email VARCHAR(20), Doctor_Gender CHARACTER(6),
Doctor_Type CHARACTER(10),
Doctor_Specialization CHARACTER(20),
FOREIGN KEY(Hospital_ID) REFERENCES Hospital(Hospital_ID));
```

10.Damage Table

```
CREATE TABLE Damage(
Damage_ID VARCHAR(5), Client_ID NUMERIC(5), Company_ID
VARCHAR(5), Bank_ID VARCHAR(5),
Policy_Number VARCHAR(10), Damage_Cause CHARACTER(50),
Date_Admitted DATE NOT NULL, Date_Discharged DATE,
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID), FOREIGN
KEY(Company_ID) REFERENCES Company(Company_ID),
FOREIGN KEY(Policy_Number) REFERENCES Insurance(Policy_Number),
FOREIGN KEY(Bank_ID) REFERENCES BankDetail(Bank_ID));
```

11.BankDetails Table

```
CREATE TABLE BankDetails(
Bank_ID VARCHAR(5) PRIMARY KEY, Client_ID NUMERIC(5),
Bank_Name CHARACTER(20), Client_AccountNumber VARCHAR(16),
Client_IFSC VARCHAR(12),
Bank_Branch CHARACTER(20),
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID));
```

12.Accountability Table

```
CREATE TABLE Accountability(
Client_ID NUMERIC(5), Company_ID VARCHAR(5), Claimed_Amount
NUMERIC(6), Paid_Amount NUMERIC(6),
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID), FOREIGN
KEY(Company_ID) REFERENCES Company(Company_ID));
```

13.Hippa Table

```
CREATE TABLE Hippa(
Client_ID NUMERIC(10),
Claimed_Amount NUMERIC(7),
Paid_Amount NUMERIC(7),
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID));
```

14.GroupInsurance Table

```
CREATE TABLE GroupInsurance( Group_ID VARCHAR(5) PRIMARY KEY,
Company_ID VARCHAR(5), Members CHARACTER(20),
Head_DOB DATE, Head_Address VARCHAR(50),
Head_Contact NUMERIC(10) UNIQUE, Head_Income NUMERIC(7),
Head_Gender CHARACTER(6), Head_Email VARCHAR(20),
FOREIGN KEY(Company_ID) REFERENCES Company(Company_ID));
```

15.Rating Table

```
CREATE TABLE Rating(
Name CHARACTER(20), Rate NUMERIC(1), Review VARCHAR(100));
```

16.Privilege Table

```
 CREATE TABLE Privilege(
Employee_ID VARCHAR(5) PRIMARY KEY,
Company_ID VARCHAR(5),
AccessRights CHARACTER(20),
FOREIGN KEY(Company_ID) REFERENCES Company(Company_ID));
```

17.TimeStamp Table

```
CREATE TABLE TimeStamp( Company_ID VARCHAR(5), Employee_ID
VARCHAR(5), Data_Acess VARCHAR(20), Time_Access DATETIME,
FOREIGN KEY(Company_ID) REFERENCES Company(Company_ID), FOREIGN
KEY(Employee_ID) REFERENCES Privilege(Employee_ID));
```

18.Tariff Table

```
CREATE TABLE Tariff(
Tariff_ID VARCHAR(5) PRIMARY KEY, Amount NUMERIC(7) NOT NULL,
Duration DATETIME);
```

19.Payment Table

```
CREATE TABLE Payment(
Client_ID NUMERIC(10), Amount NUMERIC(7),
Transaction_Type Character(10),
FOREIGN KEY(Client_ID) REFERENCES Client(Client_ID));
```

20.Subscription Table

```
CREATE TABLE Subscription( Company_ID VARCHAR(5), Tariff_ID
VARCHAR(5), Activation_Date DATE, Expiry_Date DATE,
FOREIGN KEY(Company_ID) REFERENCES Company(Company_ID), FOREIGN
KEY(Tariff_ID) REFERENCES Tariff(Tariff_ID));
```

Dropping Table:

We create the table that we don’t wont anymore so we delete the structure and also a data by this command.

- `Drop Table Car_Insurance;`
- `Drop Table Claim;`
- `Drop Table Customer;`

Alter Table:

After the table creation we want to change the structure of the table so we used the Alter command.

- ALTER TABLE Client ADD Occupation varchar(25);
- ALTER TABLE Hospiat Alter Hospital_Address varchar(25);
- ALTER TABLE Rating Drop ID numeric(10);

Truncate Table:

- Truncate Table Car_Insurance;
  Truncate Table Claim;
- Truncate Table Customer;

Rename Table:

- RENAME TABLE Group_Insurance` TO `GroupInsurance`;

Updating the value in the table: -

- UPDATE CLIENT SET Contact_Name=9997752413 WHERE CustomerID=3;

- UPDATE COMPANY SET Company_Email='hdfcEnquiry@hdfcergo' WHERE Company_ID=3;
- UPDATE BANK SET BANK_Branch='Jail Road gondal' WHERE BANK_ID=3;

Deleting the value in the table: -

- DELETE FROM Privilege WHERE Employee_ID=2;
- DELETE FROM Client WHERE Client_ID=3;
- DELETE FROM Doctor WHERE Doctor_ID=4;

DCL Commands:-

GRANT- It gives user’s access privileges to database.

REVOKE- It withdraws user’s access privileges given by using the GRANT command.

Grant Command:

-
  ```
  GRANT SELECT,INSERT,UPDATE,DELETE ON Company TO
  'ravi'@'localhost';
  ```

-
  ```
  GRANT SELECT, INSERT ON Company TO 'dhrumil'@'localhost';
  GRANT SELECT ON company TO 'vatsal'@'localhost';
  ```

Revoke Command:

-
  ```
  REVOKE ALL ON Company FROM 'ravi'@'localhost';
  ```

-
  ```
  REVOKE Insert ON Company FROM 'dhrumil'@'localhost'; REVOKE
  ALL ON Company FROM 'vatsal'@'localhost'
  ```

- **Study and use of inbuilt SQL functions - aggregate functions, Built-in functions Numeric, date, string functions.**

Aggregate Function: -

1. Find total paid amount of the company.

```
Select Company_ID , SUM(Paid_Amount) from Accountability;
```

2. Count total number of policies

```
SELECT count(PolicyList) as COUNT from Policy_List;
```

3. Find the minimum income from all clients

```
SELECT MIN(Client_Income) from Client;
```

4. Find maximum income from all clients

```
SELECT MAX(Client_Income) from Client;
```

5. Retrive average insurance amount

```
SELECT AVG(Claimed_Amount) from Accountability;
```

Numeric Function: -

1. Find round figure insurnce amount of client"

```
SELECT ROUND(Insurance_Amount) from Insurance WHERE
Client_ID = 1;
```

2. Find hospital with maximum rates"

```
SELECT
MAX(Hospital_Rating),Hospital.Hospital_Name,Hospital.Hospit
al_Rating FROM Hospital;
```

3. show Client account number whose number exceeds 15 digits"

```
SELECT Client_AccountNumber > POWER(10,7) from BankDetails;
```

String Function:-

1. Remove all spaces after client name
   ```
   SELECT TRIM(Client_Name) FROM Client;
   ```

2. Make all chacters of company name in Upper Case
   ```
   SELECT UPPER(Company_Name) FROM Company;
   ```

3. Display client name in Lower case
   ```
   SELECT LOWER(Client_Name) FROM Client;
   ```

4. Calulate the length of Client_Name
   ```
   SELECT LENGTH(Client_Name) FROM Client;
   ```

5. Replace name 'Dhrumil' with 'Dhrumik'
   ```
   SELECT REPLACE(Client_Name , 'l' ,'k') from Client WHERE
   Client_Name = 'Dhrumilshah';
   ```

Date Function:-

1. Retrieve age of client 1 from client table
   ```
   SELECT (DATEDIFF(CURDATE(),Client_DOB) DIV 365) FROM
   Client WHERE Client_ID = '1'
   ```

2. Show how much days client 2 was admitted in hospital from damage table
   ```
   SELECT (DATEDIFF(Date_Discharged,Date_Admitted)) FROM
   Damage WHERE Client_ID = '2'
   ```

3. Add precise time of client’s date admitted in hospital into damage table
   ```
   SELECT ADDTIME(Date_Admitted,'5:00') FROM Damage WHERE
   Client_ID = '6'
   ```

4. Reflect extension of expiry date of company’s subscription on successful payment from subscription table
   ```
   SELECT DATE_ADD(Expiry_Date, INTERVAL 1 year) FROM
   Subscription WHERE Company_ID = '4'
   ```

5. Check whether the policy start date was a week day or weekend from renew table
   ```
   SELECT WEEKDAY(New_DateOfCommencement) FROM Renew
   ```

- **Study, write and use the set operations, sub-queries, correlated sub-queries in SQL**

Set Operation: -

These are used to get meaningful results from data stored in the table, under different special conditions.

1)UNION:

1. This will SELECT the ID of the Client that are present in the Client Table as well as the Clients that have violated the Hippa Law.

```
SELECT Client_ID FROM Client UNION
SELECT Client_ID FROM Hippa;
```

2. This will SELECT the Bank_ID and Client_ID from the BankDetails Table and also from the Damage Table.

```
SELECT Bank_ID, Client_ID FROM BankDetails UNION
SELECT Bank_ID, Client_ID FROM Damage;
```

3. This will SELECT the ID of Hospitals from the Hospital Table and also the Hospitals that have violated the Hippa Law.

```
SELECT Hospital_ID FROM Hospital UNION
SELECT Hospital_ID
FROM Hippa;
```

4. This will SELECT the Policy_Number from the Insurance Table as well as from the Damage Table.

```
SELECT Policy_Number FROM Insurance UNION
SELECT Policy_Number
FROM Damage;
```

5. This will SELECT the ID of the Companies that are in the Company Table and also the companies that are included in the Policy list.

```
SELECT Company_ID FROM Company UNION
SELECT Company_ID FROM Policy_List;
```

2)UNION ALL:

1. This will SELECT the Policy_Number from the Insurance Table as well as from the Damage Table. (including the duplicate rows)

```
SELECT Policy_Number FROM Insurance
UNION ALL
SELECT Policy_Number
FROM Damage;
```

2. This will SELECT the ID of the Companies that are in the Company Table and also the companies that are included in the Policy list. (including the duplicate rows)

```
SELECT Company_ID FROM Company
UNION ALL
SELECT Company_ID FROM Policy_List;
```

3. This will SELECT the ID of the Client that are present in the Client Table as well as the Clients that have violated the Hippa Law. (including the duplicate rows)

```
SELECT Client_ID FROM Client
UNION ALL
SELECT Client_ID FROM Hippa;
```

4. This will SELECT the ID of Hospitals from the Hospital Table and also the Hospitals that have violated the Hippa Law. (including the duplicate rows)

```
SELECT Hospital_ID FROM Hospital
UNION ALL
SELECT Hospital_ID FROM Hippa;
```

5. This will SELECT the Bank_ID and Client_ID from the BankDetails Table and also from the Damage Table. (including the duplicate rows)

```
SELECT Bank_ID, Client_ID FROM BankDetails
UNION ALL
SELECT Bank_ID, Client_ID FROM Damage;
```

3)INTERSECT:

1. This will SELECT the ID of the Client that are present or common in both Client and Hippa Table.

   ```
   SELECT Client_ID FROM Client
   INTERSECT
   SELECT Client_ID FROM Hippa
   ```

2. This will SELECT the Bank_ID and Client_ID that are common in BankDetails Table and the Damage Table.

   ```
   SELECT Bank_ID, Client_ID FROM BankDetails
   INTERSECT
   SELECT Bank_ID, Client_ID
   FROM Damage;
   ```

3. This will SELECT the Company_ID that are present in both the Company and Accountability Table.

   ```
   SELECT Company_ID FROM Company
   INTERSECT
   SELECT Company_ID FROM Accountability;
   ```

4. This will SELECT the Company_ID that are present in both the Insurance and Client Table.

   ```
   SELECT Client_ID FROM Insurance
   INTERSECT
   SELECT Client_ID FROM Client;
   ```

5. This will SELECT the Company_ID that are common in both the Company and Policy_List Table.

   ```
   SELECT Company_ID FROM Company
   INTERSECT
   SELECT Company_ID FROM Policy_List;
   ```

4)MINUS:

1. This will combine the two select statements and will display the Client_ID from the Client Table.

   ```
   SELECT Client_ID FROM Client
    MINUS
   SELECT Client_ID FROM Hippa;
   ```

2. This will combine the two select statements and will display the Bank_ID and Client_ID from the BankDetails Table.

   ```
   SELECT Bank_ID, Client_ID FROM BankDetails
   MINUS
   SELECT Bank_ID, Client_ID FROM Damage;
   ```

3. This will combine the two select statements and will display the Hospital_ID from the Hospital Table.

   ```
   SELECT Hospital_ID FROM Hospital
   MINUS
   SELECT Hospital_ID FROM Hippa;
   ```

4. This will combine the two select statements and will display the Policy_Number from the Insurance Table.

   ```
   SELECT Policy_Number FROM Insurance
   MINUS
   SELECT Policy_Number FROM Damage;
   ```

5. This will combine the two select statements and will display the Company_ID from the Company Table.

   ```
   SELECT Company_ID FROM Company
   MINUS
   SELECT Company_ID FROM Policy_List;
   ```

Sub Queries:-

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

1. This will SELECT the client that is having the maximum income from the Client Table.

   ```
   SELECT * FROM Client
   WHERE Client_Income =(SELECT Max(Client_Income) FROM
   Client);
   ```

2. This will SELECT the client that is having the maximum income from the Client Table.

   ```
   SELECT * FROM Accountability WHERE
   Claimed_Amount IN(SELECT Max(Claimed_Amount) FROM
   Accountability GROUP BY Client_ID) ;
   ```

3. This will SELECT the maximum of all Amount from the Payment Table for the Transaction_Type equal to Cheque.

   ```
   SELECT * FROM Payment WHERE
   Transaction_Type="Cheque" AND Amount >= ALL(SELECT Amount
   FROM
   Payment WHERE Transaction_Type="Cheque") ;
   ```

4. This will SELECT the Policy_Number and Insurance_Amount along with the Client_ID from the Insurance Table.

   ```
   SELECT * FROM Insurance WHERE
   (Policy_Number,Insurance_Amount)IN(SELECT
   Policy_Number,Insurance_Amount FROM Insurance WHERE
   Client_ID=2);
   ```

5. This will SELECT the maximum tariff amount from the Tariff Table.

   ```
   SELECT * FROM Tariff WHERE
   Amount =(SELECT Max(Amount) FROM Tariff) ;
   ```

Correlated Sub Queries:

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.

1. This will SELECT the Client_ID and Client_Name from the Client Table.

```
SELECT Client_ID, Client_Name FROM Client
WHERE Client_Income >(SELECT AVG(Client_Income)
FROM Client
WHERE Client_ID = Client_ID);
```

2. This will SELECT the Hospital_ID and Hospital_Name from the Hospital Table.

```
SELECT Hospital_ID, Hospital_Name FROM Hospital
WHERE EXISTS ( SELECT Hospital_Name FROM Hospital
WHERE Hospital_ID = Hospital_ID);
```

3. This will SELECT the Client_ID and Client_Name from the Payment Table.

```
SELECT Client_ID, Client_Name FROM Client c1
WHERE NOT EXISTS (SELECT p.Client_ID
FROM Payment p
WHERE p.Client_ID = c1.Client_ID AND p.Transaction_Type =
'debit');
```

4. This will SELECT the Group_ID and Members from the GroupInsurance Table.

```
SELECT Group_ID, Members FROM GroupInsurance
WHERE EXISTS ( SELECT Members
FROM GroupInsurance
WHERE Group_ID = Group_ID);
```

5. This will SELECT the Client_ID and Insurance_Amount from the Insurance Table.

```
SELECT Client_ID, Insurance_Amount FROM Insurance
WHERE   Insurance_Amount   >(SELECT   AVG(Insurance_Amount)
FROMInsurance
WHERE Client_ID = Client_ID);
```

- **Study and use of group by, having, order by features of SQL**

Group By:-

1. Find number of clients from different banks from BankDetails Table.

```
SELECT Bank_Name, COUNT(*)
FROM BankDetails
GROUP BY Bank_Name
```

2. Find number of clients from different cities from Client Table.

```
SELECT Client_Address, count(*)
FROM Client
GROUP BY Client_Address
```

3. Find number of Doctors based on their Gender from Doctor Table

```
SELECT Doctor_Gender, COUNT(*)
FROM Doctor
GROUP BY Doctor_Gender
```

4. Find the insurance holder on bases of duration of insurance from Insurance Table.

```
SELECT Insurance_Duration, COUNT(*)
FROM Insurance
GROUP BY Insurance_Duration
```

5. Find number of Hospitals by their rating from Hospital Table.

```
SELECT Hospital_Rating, COUNT(*)
FROM Hospital
GROUP BY Hospital_Rating
```

Order By:-

1. To sort the result alphabetically based on Bank Name from Bank Details Table

```
Select * from Bank Details
order by Bank Name
```

2. To sort the results alphabetically based on Address of clients from Client Table

```
Select * from Client
order by Address
```

3. To sort the result in ascending order based on Age of Doctors from Doctor Table

```
Select * from Doctor
order by Age
```

4. To sort the results in ascending order based on Amount from Insurance Table

```
Select * from Insurance
order by Amount
```

5. To sort the results alphabetically based on Name of Hospital from Hospital Table

```
Select * from Hospital
order by Name
```

- **Study different types of join operations, Exist, Any, All and relevant features of SQL.**

Join Operation: -

1. **Natural Join**

1. Find the Client_Name and its policy_number who holds Insurance.

```
Select Client.Client_Name , Insurance.Policy_Number from
Client
NATURAL JOIN
Insurance;
```

2. Find all doctors who works in which hospital.

```
Select Doctor.Doctor_Name, Hospital.Hospital_Name from
Doctor
NATURAL JOIN
Hospital;
```

3. Find the Client Name and its bank details who holds account in Bank”

```
Select Client.Client_Name ,
BankDetails.Client_AccountNumber,BankDetails.Bank_Name from
Client
NATURAL JOIN
BankDetails ;
```

4. Find the Policy List of the Company.

```
Select Policy_List.PolicyList , Company.Company_Name from
Policy_List
NATURAL JOIN
Company;
```

5. Find the Claimed Amount of Insurance of the Client.

```
Select Client.Client_Name ,Accountability.Claimed_Amount
from Accountability
NATURAL JOIN
Client;
```

2. Left Join or Left Outer Join

1. Find the Client details who holds Insurance.

```
Select * from Client
 LEFT JOIN
Insurance On Client.Client_ID = Insurance.Client_ID;
```

2. Find all doctors details who works in any hospital.

```
Select Doctor.*,Hospital.Hospital_Name from Doctor
LEFT JOIN
Hospital On Doctor.Hospital_ID = Hospital.Hospital_ID;
```

3. Find the Client Name and its account detail.

```
Select
Client.Client_Name,BankDetails.Bank_Name,BankDetails.Client
_AccountNumber,BankDetails.Bank_Branch from BankDetails
LEFT JOIN
Client On BankDetails.Client_ID = Client.Client_ID
```

4. Find the Comapny Name and the Policy List of that Company.

```
Select  Company.Company_Name  ,  Policy_List.PolicyList  from
Policy_List
 LEFT JOIN
Company On Policy_List.Company_ID =Company.Company_ID;
```

5. Find the Claimed Amount of Insurance and the name of the Client.

```
Select Client.Client_Name, Accountability.*
from Accountability
 Left JOIN
Client On Client.Client_ID= Accountability.Client_ID;
```

3. Right Join or Right Outer Join

1. Find the payment details of client.

```
Select
Client.Client_Name,Payment.Amount,Payment.Transaction_Type
From Client
RIGHT JOIN
Payment On Client.Client_ID = Payment.Client_ID
```

2. Find the damage details of client.

```
Select Client.Client_Name   ,           Client.Client_Contact,
  Damage.Policy_Number ,
Damage.Damage_Cause, from Client
RIGHT JOIN  Damage.Date_Admitted,    Damage.Date_Discharged
Damage On Damage.Client_ID = Client.Client_ID
```

3. Find the Insurance associated with Client.

```
Select  Client.Client_Name  ,
Insurance.Policy_Number,Insurance.Insurance_Amount,
Insurance.DateOfCommencement,    Insurance.Insurance_Duration
from Client RIGHT JOIN
Insurance On Client.Client_ID= Insurance.Client_ID
```

4. Find the Employee ID and its rights in his/her Company.

```
Select      Employee_ID,      Privilege.AccessRights      ,
Company.Company_Name from Privilege
RIGHT JOIN
Company On Privilege.Company_ID = Company.Company_ID
```

5. Find the time of data access by employee in a company.

```
Select  Company.Company_Name     ,    TimeStamp.Employee_ID,
TimeStamp.Data_Acess,TimeStamp.Time_Access from Company
RIGHT JOIN
TimeStamp On Company.Company_ID = TimeStamp.Company_ID
```

4. Self-Join

1. Find all the female client from Client Table.
```
Select DISTINCT C1.Client_Name from Client as C1 , Client as C2 where C1.Client_Gender='Female' AND C1.Client_Gender ='Female'
```

2. Find all Permanent Doctors from Doctor Table.
```
Select DISTINCT D1.Doctor_Name ,D1.Doctor_Type from Doctor as D1 , Doctor as D2 where D1.Doctor_Type='Permanent' AND D2.Doctor_Type='Permanent'
```

3. Find the name of User who gave five rates”
```
Select DISTINCT R1.Name from Rating as R1 , Rating as R2 where R1.Rate=5 AND R2.Rate =5
```

4. Find the name of Client who were admitted after given date”
```
Select DISTINCT Ca1.Damage_Cause FROM Damage as Ca1, Damage as Ca2 where Ca1.Date_Admitted>'2017-12-23 ' AND Ca2.Date_Admitted>'2017-12-23'
```

5. Find the client whose Claimed amount is Specified”
```
Select DISTINCT A1.Company_ID,A1.Claimed_Amount from Accountability as A1 , Accountability as A2 where A1.Claimed_Amount=9262.
```

Exists:

```
UPDATE Client SET Client_Name = 'Priya' WHERE EXISTS (SELECT * from Client WHERE Client_Name = 'Priyancy'
```

ALL:

```
SELECT ALL Client_Name FROM Client where Client_Gender = 'Male'
```

ANY:

```
SELECT Employee_ID FROM Privilege
WHERE Employee_ID= ANY (SELECT Employee_ID FROM Privilege WHERE AccessRights = 'r-w')
```

Assignment-6

Study and implement the assignment topic from the Embedded SQL.

- Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language that are Java,C, C++, Cobol, and others, which is known as a host language.
- Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.
- Embedded SQL is a strong and favourable method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.
- The C programming language is commonly used for embedded SQL implementation.

Syntax:-

```
EXEC SQL BEGIN DECLARE SECTION;
table_name CHARACTER(30);
EXEC SQL END DECLARE SECTION;
display 'Table name? ';
read table_name;
EXEC SQL DROP TABLE :table_name; -- host variable not allowed
```

Features::

- Embedded SQL provides several advantages over a call-level interface:
- Embedded SQL is easy to use because it is simple Transact-SQL with some added features.
- It is an ANSI/ISO-standard programming language.
- It requires less coding to achieve the same results as a call-level approach.
- Embedded SQL is essentially identical across different host languages. Programming conventions and syntax change very little.
- The pre-compiler can optimize execution time by generating stored procedures for the Embedded SQL statements.

Example::

The program prompts the user for an Company name, retrieves the all information of that particular company and print on the screen.

```
int main() {
   EXEC SQL INCLUDE SQLCA;
   EXEC SQL BEGIN DECLARE SECTION;
      char Company_Name[100];     /* Company name (from user)*/
      int Company_ID;           /* Retrieved Company ID */
      char Company_Name[100]   /* Retrieved Company_Name name*/
      char Company_Type[40]   /* Retrieved Company_type name*/
      char Company_E-mail[100]      /* Retrieved Company_E-Mail*/

   EXEC SQL END DECLARE SECTION;

   /* Set up error processing */
   EXEC SQL WHENEVER SQLERROR GOTO query_error;
   EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

   /* Prompt the user for order number */
   printf ("Enter Company Name: ");
   scanf_s("%s%d", &Company_Name);

   /* Execute the SQL query */
   EXEC SQL SELECT   Company_ID, Company_Type, Company_E-mail
      FROM Company
      WHERE Company_Name = :Company_Name
      INTO :Company_ID, Company_Type, Company_E-mail;

   /* Display the results */
   printf ("Company's Id is::  %d\n", Company_ID);
   printf ("Company's Type is:: %s\n", Company_Type);
   printf ("Company's Email is:: %s\n",Company_E-mail );
   exit();
query_error:
   printf ("SQL error: %ld\n", sqlca->sqlcode);
   exit();

bad_number:
   printf ("Invalid Company Name.\n");
   exit();  }
```

Insert the data into the tables.

As per previous practical’s, we created total 20 tables for our Database and we are supposed to enter the dummy data in order to run the Queries that we made for the database.

So, for the dummy data we Maintain all the constraints and using the below website we generated the data.

https://www.generatedata.com/

Searching about research paper for our Database.

Every database has the previous records, websites, research papers and so on, so for the reference we search on the google and find the 4 most likely papers that we put in our data.

For the research paper we first go on the **GOOGLE SCHOLAR** page and we find unique and well-defined research paper that match with our database.

https://scholar.google.com/

Assignment-7

Study and apply Database Normalization techniques.

- In this assignment we have to make Functional Dependency for our tables and find the minimal covers from that Functional Dependencies.

- Also, we have to mentioned that the which normal forms are apply on that table like 1NF,2NF, 3NF or BCNF.

- Tables are as given below: -

1. **Sign up:**

Functional Dependency	minimal covers	Normal Forms
User_ Name →Contact ,Gender, Date_Of_Birth, Password, Confirm password, E-Mail	User_ Name →Contact ,Gender, Date_Of_Birth, Password, Confirm password, E-Mail	1NF

2. **Login:**

Functional Dependency	minimal covers	Normal Forms
User_ID → User_Name,Password Password → User_Name	User_ID→ User_Name, Password	1NF 2NF 3NF BCNF

3. **Client:**

Functional Dependency	minimal covers	Normal Forms
▪ Client_ID→Client_Name, Client_DOB,Client_Address, Client_Contact,Client_Income, Client_Gender, Client_Email	▪ Client_ID→Client_Name, Client_DOB, Client_Address, Client_Contact, Client_Income, Client_Gender,Client_Email	1NF 2NF

▪ Client_ID, Client_Email → Client_ID,Client_Email,Client_Address, Client_Contact ▪ Client_Name →Client_DOB,Client_Address,Client_Contact, Client_Gender ▪ Client_Contact→ Client_Name, ▪ Client_Email → Client_Income	Client_Name→Client_DOB, Client_Address,Client_Contact, Client_Gender Client_Email →Client_Address, Client_Contact, Client_Income	

4. **Company:**

Functional Dependency	minimal covers	Normal Forms
▪ Company_ID →Company_ID, Company_Name, Company_Type, Company_E-mail ▪ Comapany_ID,Comany_Email →Comapany_ID,Comany_Email, Company_Name ▪ Company_ID, Company_Email, Comapny_Name →Company_Type	▪ Company_ID→ Company_Name, Company_E-mail, Company_Type ▪ Company_Type→ Company_ Name	1NF 2NF

5. **Policy List:**

Functional Dependency	minimal covers	Normal Forms
Company_ID→ Policy_List	Company_ID→ Policy_List	1NF

6. **Insurance:**

Functional Dependency	minimal covers	Normal Forms
▪ Policy_Number→ Policy_Number, Client_ID, Company_ID, New_Policy, Renew_Policy, Insurance_Amount, DateOfCommencement, Insurance_Duration	▪ Policy_Number→ Policy_Number, Client_ID, Company_ID, New_Policy, Renew_Policy, Insurance_Amount, DateOfCommencement, Insurance_Duration	1NF 2NF 3NF BCNF

7. Renew:

Functional Dependency	minimal covers	Normal Forms
▪ Client_ID,Company_ID, Policy_Number→Client_ID, Company_ID, Policy_Number, New_DateOfCommencement, New_Insurance_Duration	▪ Client_ID,Company_ID, Policy_Number→Client_ID, Company_ID, Policy_Number, New_DateOfCommencement, New_Insurance_Duration	1NF

8. Hospital:

Functional Dependency	minimal covers	Normal Forms
▪ Hosptal_ID → Hospital_Name, Hospital_Address, Hospital_Rating, Hospital_Contact, Hospital_Representative, Hospital_Type ▪ Hospital_Address→ Hospital_Name ▪ Hospital_ID,Hospital_Representative→ Hospital_Name ▪ Hospital_Name→Hospital_Rating, Hospital_Contact	Hospital_ID→Hospital_Rating Hospital_ID→Hospital_Name Hospital_Address→Hospital_Name Hospital_Name→ Hospital_Representative Hospital_ID→ Hospital_Contact Hospital_ID→ Hospital_Representative Hospital_ID→ Hospital_Type	1NF

9. Doctor:

Functional Dependency	minimal covers	Normal Forms
▪ Doctor_ID→Doctor_Name, Doctor_DOB,Doctor_Address, Doctor_Contact,Doctor_E-mail, Doctor_Gender,Doctor_Type, Doctor_Specialization ▪ Doctor_Email→Doctor_Name, Doctor_Contact,Doctor_Address ▪ Doctor_Name → Doctor_Type, Doctor_Specialization ▪ Doctor_ID,Doctor_Type→ Doctor_Name, Doctor_Specialization	▪ Doctor_ID→ Hospital_ID ▪ Doctor_ID→ Doctor_Name ▪ Doctor_ID→ Doctor_DOB ▪ Doctor_ID→ Doctor_E-mail ▪ Doctor_ID→ Doctor_Gender ▪ Doctor_E-mail→ Doctor_Name ▪ Doctor_E-mail→ Doctor_Contact ▪ Doctor_E-mail→ Doctor_Address ▪ Doctor_Name→ Doctor_Type ▪ Doctor_Type→ Doctor_Name	1NF

▪ Doctor_Type→ Doctor_Specialization.	▪ Doctor_Type→ Doctor_Specialization	

10. BankDetails:

Functional Dependency	minimal covers	Normal Forms
▪ Bank_ID→ Client_ID, Bank_Name,Client_AccNo, Client_IFSC, Bank_Branch ▪ Bank_ID, Bank_Branch→ Bank_Name ▪ Bank_ID, Bank_Branch ,Bank_Name Client_ID→Client_AccNo, Client_IFSC	▪ Bank_ID→ Client_ID, Bank_Name, Client_AccNo, Client_IFSC, Bank_Branch	1NF

11. Damage:

Functional Dependency	minimal covers	Normal Forms
▪ Damage_ID → Client_ID, Company_ID,Policy_Number, Damage_Cause,Date_Admitted, Date_Discharged	▪ Damage_ID → Client_ID,Company_ID, Policy_Number,Damage_Cause, Date_Admitted,Date_Discharged	1NF 2NF 3NF BCNF

12. Accountability:

Functional Dependency	minimal covers	Normal Forms
▪ Client_ID, Company_ID → Claimed_Amount, Paid_Amount	▪ Client_ID, Company_ID → Claimed_Amount, Paid_Amount	1NF

13. Hippa:

Functional Dependency	minimal covers	Normal Forms
▪ Client_ID,Company_ID, Hospital_ID, Doctor_ID → Law_Violation	▪ Client_ID,Company_ID, Hospital_ID, Doctor_ID → Law_Violation	1NF

14. GroupInsurance:

Functional Dependency	minimal covers	Normal Forms
▪ Group_ID → Company_ID, Members, Client_DOB, Client_Address, Client_Contact, Client_Income, Client_Gender, Client_Email ▪ Client_Email→ Members, Client_Address, Client_Contact ▪ Client_Email → Client_Income	Group_ID→ Company_ID Group_ID→ Client_DOB Group_ID→Client_Gender Group_ID→Client_Email	1NF 2NF

15. Rating:

Functional Dependency	minimal covers	Normal Forms
▪ Name → Rate, Review	▪ Name → Rate, Review	1NF

16. Privileges:

Functional Dependency	minimal covers	Normal Forms
▪ Employee_ID → Company_ID,Access_Rights	▪ Employee_ID → Company_ID,Access_Rights	1NF 2NF 3NF BCNF

17. TimeStamp:

Functional Dependency	minimal covers	Normal Forms
▪ Company_ID →Employee_ID ▪ Employee_ID → Data_Access,Time_Access, ▪ Data_Access→Time_Access	▪ Company_ID →Employee_ID, ▪ Employee_ID→Data_Access ▪ Data_Access→Time_Access	1NF

18. Tariff:

Functional Dependency	minimal covers	Normal Forms
▪ Tariff_ID -> Amount,Duration ▪ Duration -> Amount	▪ Tariff_ID -> Amount,Duration ▪ Duration -> Amount	1NF 2NF 3NF BCNF

19. Subscription:

Functional Dependency	minimal covers	Normal Forms
▪ Tariff_ID Company_ID -> Activation_Date,Expiry_Date	▪ Tariff_ID Company_ID -> Activation_Date,Expiry_Date	1NF

20. Payment:

Functional Dependency	minimal covers	Normal Forms
▪ Client_ID -> Amount, Transaction_Type	▪ Client_ID -> Amount, Transaction_Type	1NF