# 1- Introduction of DBMS

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.

**Some key features of a DBMS include:**

1. Data modeling: A DBMS provides tools for creating and modifying data models, which define the structure and relationships of the data in a database.
2. Data storage and retrieval: A DBMS is responsible for storing and retrieving data from the database, and can provide various methods for searching and querying the data.
3. Concurrency control: A DBMS provides mechanisms for controlling concurrent access to the database, to ensure that multiple users can access the data without conflicting with each other.
4. Data integrity and security: A DBMS provides tools for enforcing data integrity and security constraints, such as constraints on the values of data and access controls that restrict who can access the data.
5. Backup and recovery: A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.
6. DBMS can be classified into two types: Relational Database Management System (RDBMS) and Non-Relational Database Management System (NoSQL or Non-SQL)
7. RDBMS: Data is organized in the form of tables and each table has a set of rows and columns. The data is related to each other through primary and foreign keys.
8. NoSQL: Data is organized in the form of key-value pairs, document, graph, or column-based. These are designed to handle large-scale, high-performance scenarios.

Database is a collection of interrelated data which helps in the efficient retrieval, insertion, and deletion of data from the database and organizes the data in the form of tables, views, schemas, reports, etc. For Example, a university database organizes the data about students, faculty, admin staff, etc. which helps in the efficient retrieval, insertion, and deletion of data from it.

2- What are the Properties of RDBMS?

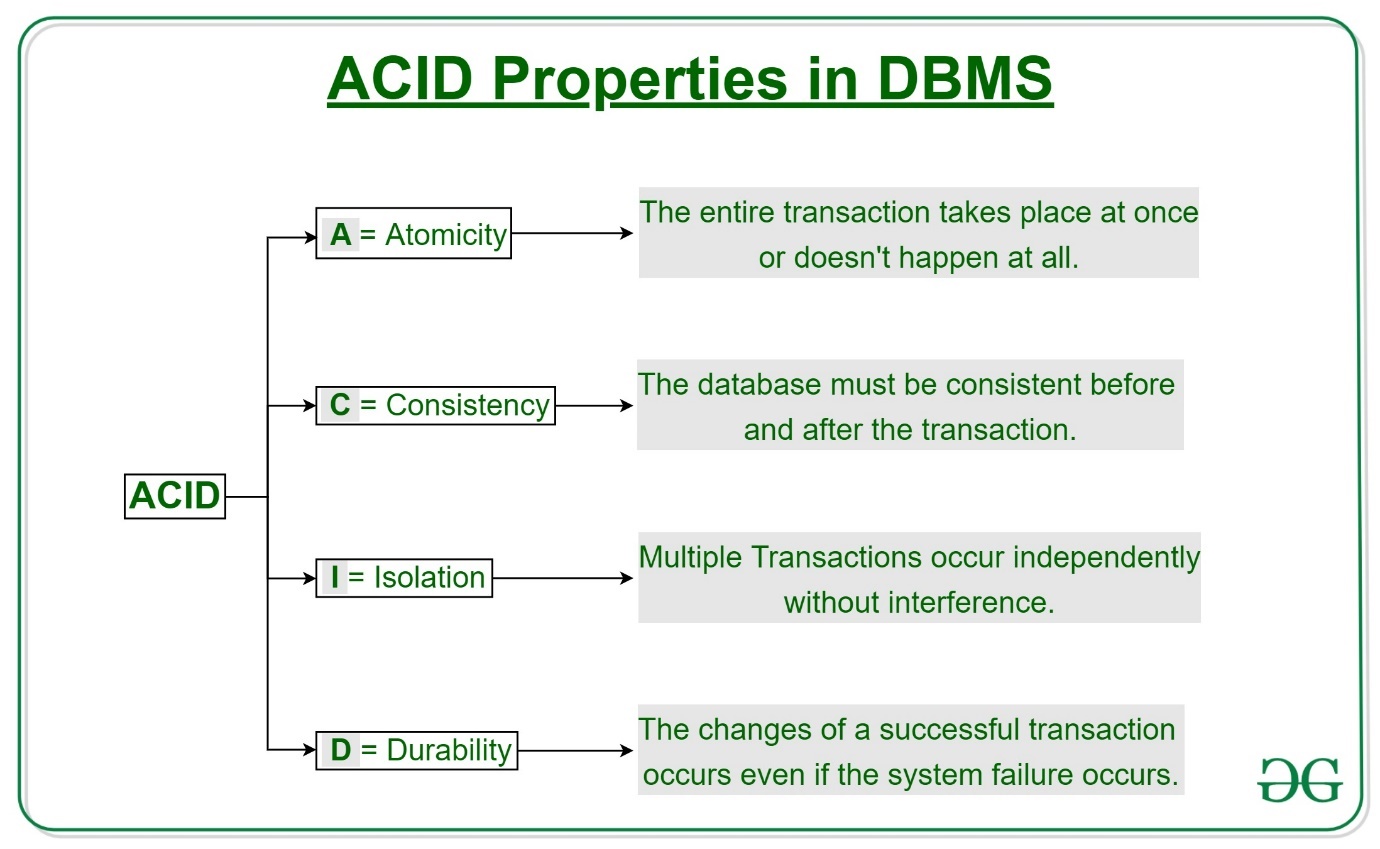
RDBMS stands for "Relational Database Management System." It is a type of database management system that organizes data into structured tables with rows and columns, where each row represents a record and each column represents a specific attribute or field of that record

**Features/ Properties of RDBMS:**

* Gives a high level of information security.
* It is quick and precise.
* Provides facility primary key, to exceptionally distinguish the rows.
* The values in RDBMS are atomic
* The values in a column of RDBMS are of the same type
* In RDMS, no two rows have the same data
* The sequence of Rows and Columns in RDBMS does not exist.
* Each column in RDBMS has a unique name.

3- Describe ACID Properties

to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.



## **1) Atomicity**

The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

 It involves the following two operations.  
--**Abort**: If a transaction aborts, changes made to the database are not visible.  
--**Commit**: If a transaction commits, changes made are visible.

## **2) Consistency**

The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial), the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

## **3) Isolation**

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

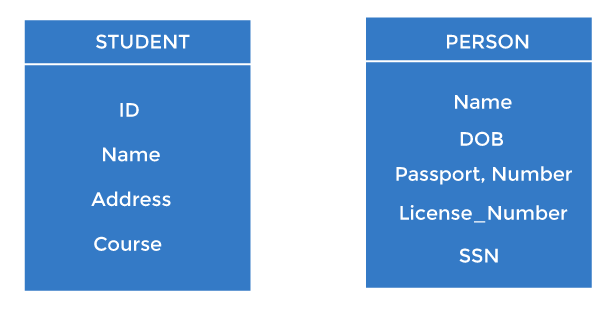
## **4) Durability**

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes

4- What are Keys in DBMS?

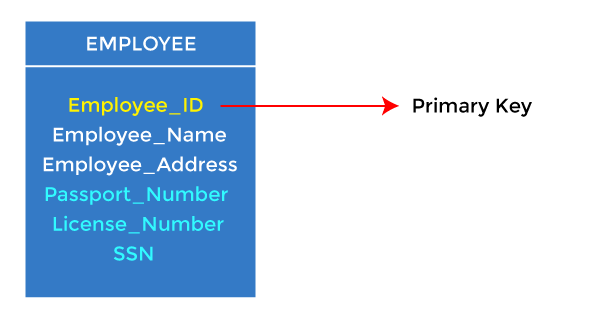
The attribute which uniquely identifies each entity in the entity set is called a **key**.

**For example,** ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport\_number, license\_number, SSN are keys since they are unique for each person.



1. Primary key

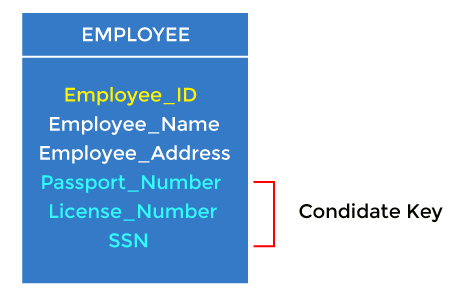
* It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
* In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License\_Number and Passport\_Number as primary keys since they are also unique.
* For each entity, the primary key selection is based on requirements and developers.



2. Candidate key

* A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
* Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

**For example:** In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport\_Number, License\_Number, etc., are considered a candidate key.



3. Super Key

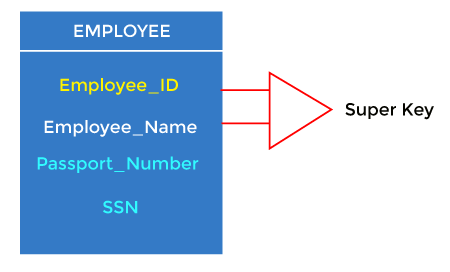
Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

We can define a super key as a set of those keys that identify a row or a tuple uniquely. The word super denotes the superiority of a key. Thus, a super key is the superset of a key known as a **Candidate key** (discussed in the next section). It means a [candidate key](https://www.javatpoint.com/candidate-key-in-dbms) is obtained from a super key only.

## **Role of Super Key**

The role of the super key is simply to identify the tuples of the specified table in the database. It is the superset where the candidate key is a part of the super key only. So, all those attributes in a table that is capable of identifying the other attributes of the table in a unique manner are all super keys.

#### **Note: Two or more attributes in a table can together identify a table uniquely, so the combination of such attributes is nothing but a super key only.**

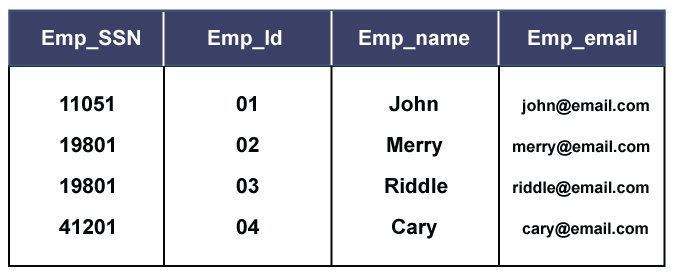


**For example:** In the above EMPLOYEE table, for(EMPLOEE\_ID, EMPLOYEE\_NAME), the name of two employees can be the same, but their EMPLYEE\_ID can't be the same. Hence, this combination can also be a key.

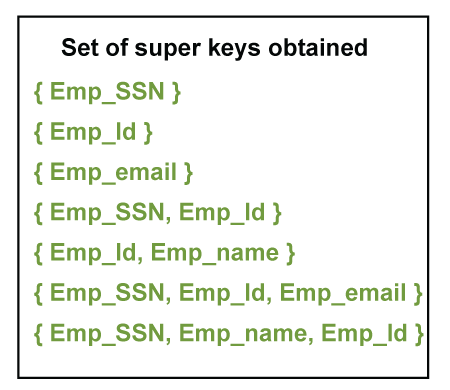
The super key would be EMPLOYEE-ID (EMPLOYEE\_ID, EMPLOYEE-NAME), etc.

Eg:

The **EMPLOYEE\_DETAIL** table is given below that will help you understand better:

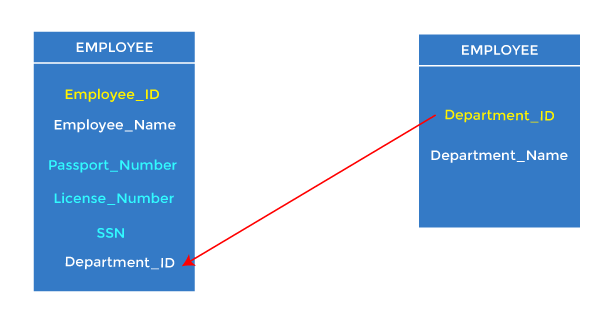


So, from the above table, we conclude the following set of the super keys:



4. Foreign key

* Foreign keys are the column of the table used to point to the primary key of another table.
* Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
* We add the primary key of the DEPARTMENT table, Department\_Id, as a new attribute in the EMPLOYEE table.
* In the EMPLOYEE table, Department\_Id is the foreign key, and both the tables are related.



5. Alternate key

The candidate key other than the primary key is called an alternate key

**For example,** employee relation has two attributes, Employee\_Id and PAN\_No, that act as candidate keys. In this relation, Employee\_Id is chosen as the primary key, so the other candidate key, PAN\_No, acts as the Alternate key.



### 6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.

* It acts as a primary key if there is no primary key in a table
* Two or more attributes are used together to make a composite key.
* Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

**Example:**

FULLNAME + DOB can be combined

together to access the details of a student.

#### \* What is a Unique Key?

* Unique Keys are the keys that define the record uniquely in the table. It is different from [Primary Keys](https://www.geeksforgeeks.org/difference-between-primary-key-and-unique-key/), as Unique Key can contain one NULL value but Primary Key does not contain any NULL values.

5- Difference between Vertical and Horizontal Scaling

| [**Horizontal Scaling**](https://www.geeksforgeeks.org/horizontal-and-vertical-scaling-in-databases/) | [**Vertical Scaling**](https://www.geeksforgeeks.org/horizontal-and-vertical-scaling-in-databases/) |
| --- | --- |
| When new server racks are added to the existing system to meet the higher expectation, it is known as horizontal scaling. | When new resources are added in the existing system to meet the expectation, it is known as vertical scaling |
| It expands the size of the existing system horizontally. | It expands the size of the existing system vertically. |
| It is difficult to implement | It is easy to implement |
| It is costlier, as new server racks comprise of a lot of resources | It is cheaper as we need to just add new resources |
| It takes more time to be done | It takes less time to be done |

6- What is Sharding?

**Sharding**is a very important concept which helps the system to keep data into different resources according to the sharding process.  
  
In DBMS, Sharding is a type of Database partitioning in which a large Database is divided or partitioned into smaller data, also known as shards. These shards are not only smaller, but also faster and hence easily manageable.

**Need for Sharding:**  
  
Consider a very large database whose sharding has not been done. For example, let's take a Database of a college in which all the student's records (present and past) in the whole college are maintained in a single database. So, it would contain a very very large number of data, say 100, 000 records.  
  
Now when we need to find a student from this Database, each time around 100, 000 transactions have to be done to find the student, which is very very costly.  
  
Now consider the same college students' records, divided into smaller data shards based on years. Each data shard will have around 1000-5000 student records only. So not only the database became much more manageable, but also the transaction cost of each time also reduces by a huge factor, which is achieved by Sharding.  
  
Hence this is why Sharding is needed.

**Features of Sharding:**

* Sharding makes the Database smaller
* Sharding makes the Database faster
* Sharding makes the Database much more easily manageable
* Sharding can be a complex operation sometimes
* Sharding reduces the transaction cost of the Database

7- What are SQL Commands? Explain types of SQL Commands

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. [SQL](https://www.geeksforgeeks.org/structured-query-language/) uses certain commands like CREATE, DROP, INSERT, etc. to carry out the required tasks.

SQL commands are like instructions to a table. It is used to interact with the database with some operations. It is also used to perform specific tasks, functions, and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create a new table in the database.

**Syntax:**

CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**Example:**

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

DROP TABLE table\_name;

**Example**

DROP TABLE EMPLOYEE;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add a new column in the table

ALTER TABLE table\_name ADD column\_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table\_name MODIFY(column\_definitions....);

**EXAMPLE**

ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

TRUNCATE TABLE table\_name;

**Example:**

TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

INSERT INTO TABLE\_NAME

(col1, col2, col3,.... col N)

VALUES (value1, value2, value3, .... valueN);

Or

INSERT INTO TABLE\_NAME

VALUES (value1, value2, value3, .... valueN);

**For example:**

INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**For example:**

UPDATE students

SET User\_Name = 'Sonoo'

WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

DELETE FROM table\_name [WHERE condition];

**For example:**

DELETE FROM javatpoint

WHERE Author="Sonoo";

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

COMMIT;

**Example:**

DELETE FROM CUSTOMERS

WHERE AGE = 25;

COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

ROLLBACK;

**Example:**

DELETE FROM CUSTOMERS

WHERE AGE = 25;

ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

SAVEPOINT SAVEPOINT\_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**a. SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

SELECT expressions

FROM TABLES

WHERE conditions;

**For example:**

SELECT emp\_name

FROM employee

WHERE age > 20;

8- What is Indexing in DBMS?

Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

--     creating a index

-- create index index\_name

-- on table\_name(column1,column2,column3,....);

--

--     deleting a index

-- drop index index\_name

-- on table\_name;

create index studage

on students (age);

show index from students;                     -- to show indexes in a table

drop index studage                             -- deleting a index

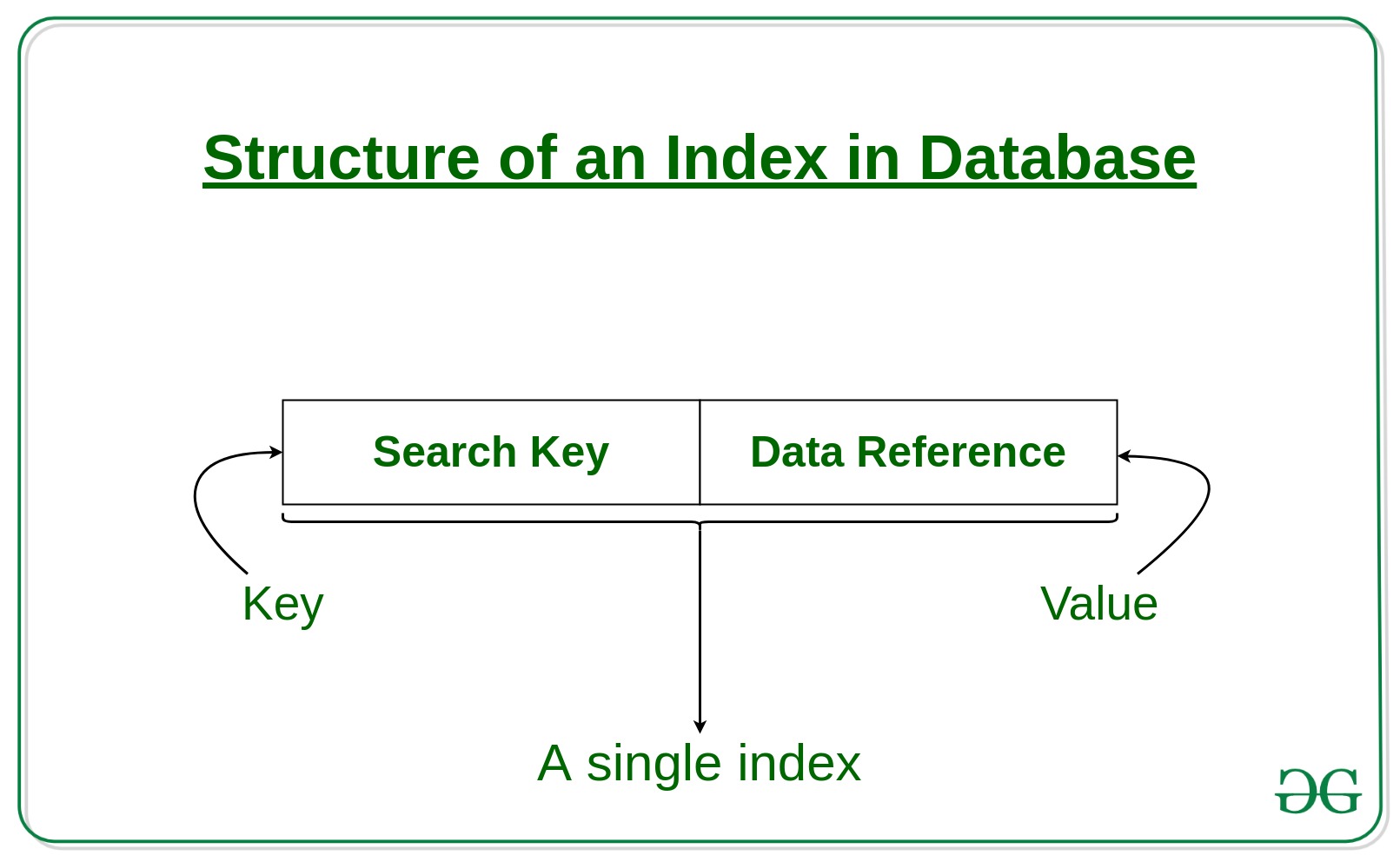
on students;

Advantage: select operation will be faster

Disadvantages: insert and update will be slower bcoz it need to update indeas as well.

Indexes are created using a few database columns.

* The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly. *Note: The data may or may not be stored in sorted order.*
* The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183525/Structure-of-an-Index-in-Database.jpg)

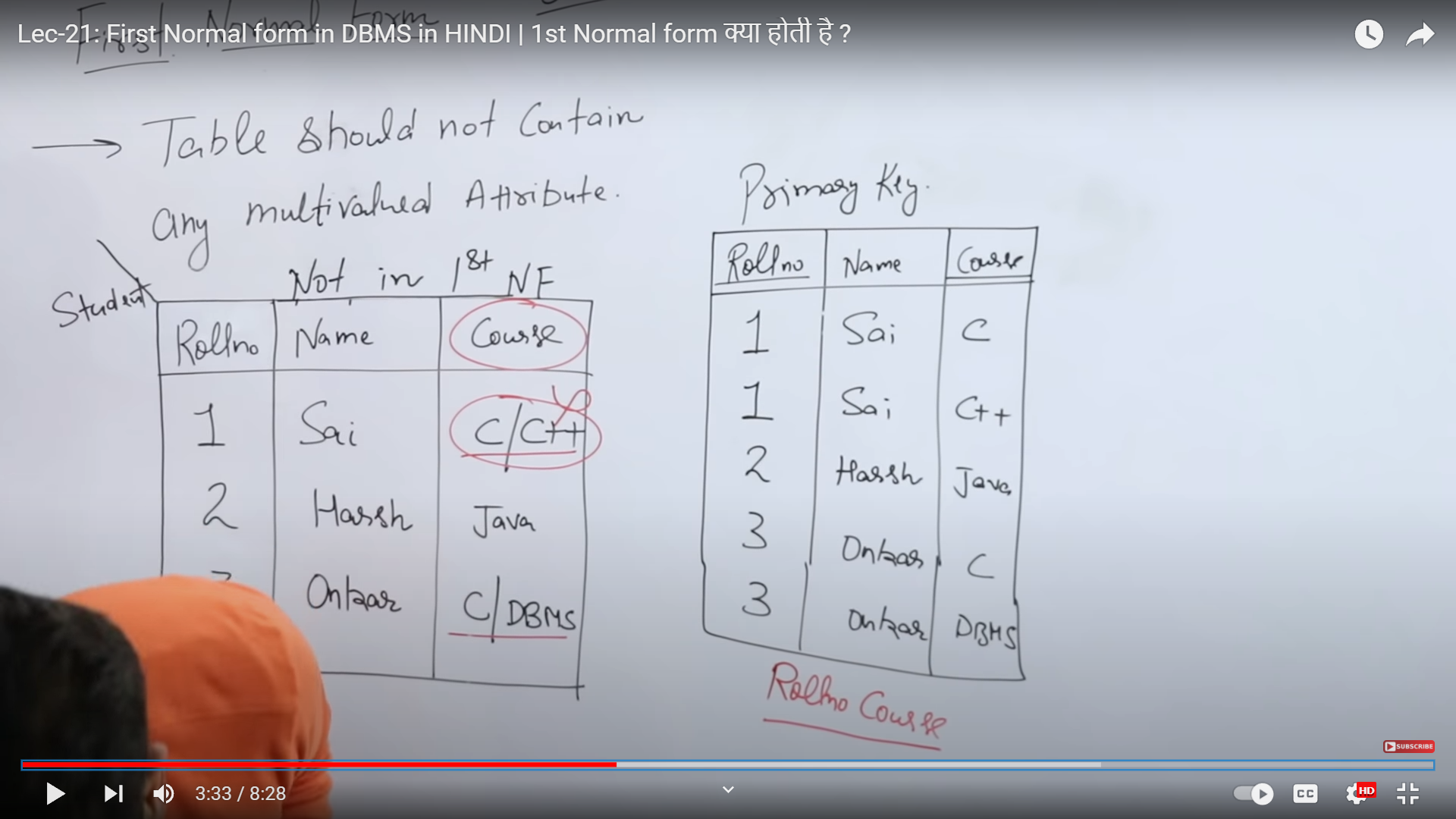
9- Explain Normal Forms in DBMS

**Normalization** is the process of minimizing **redundancy(**having several copies of the same data in the database**)** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

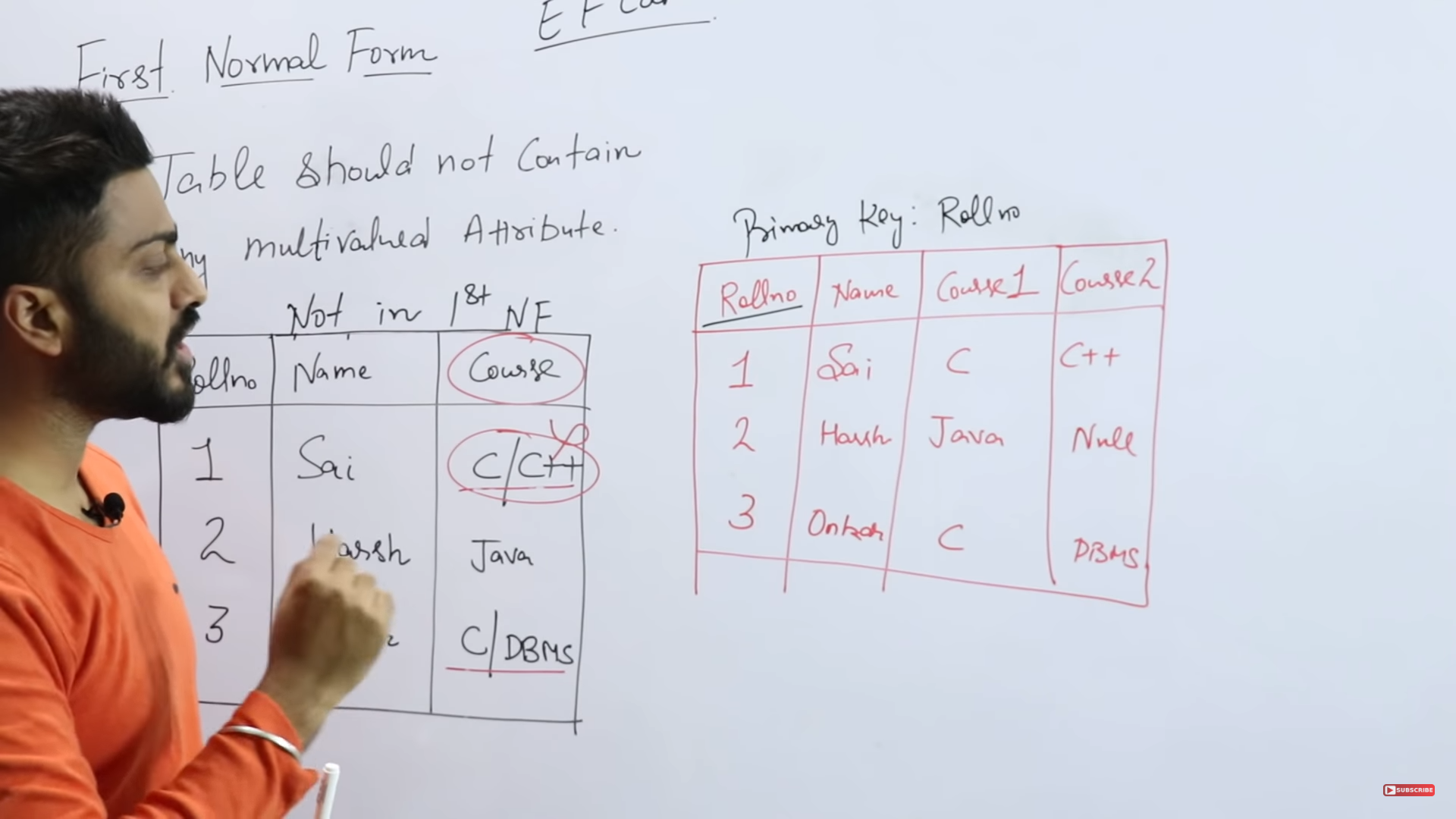
**First Normal form**

* Each column in a table must have a single, atomic data value.
* Each row in the table must be unique.

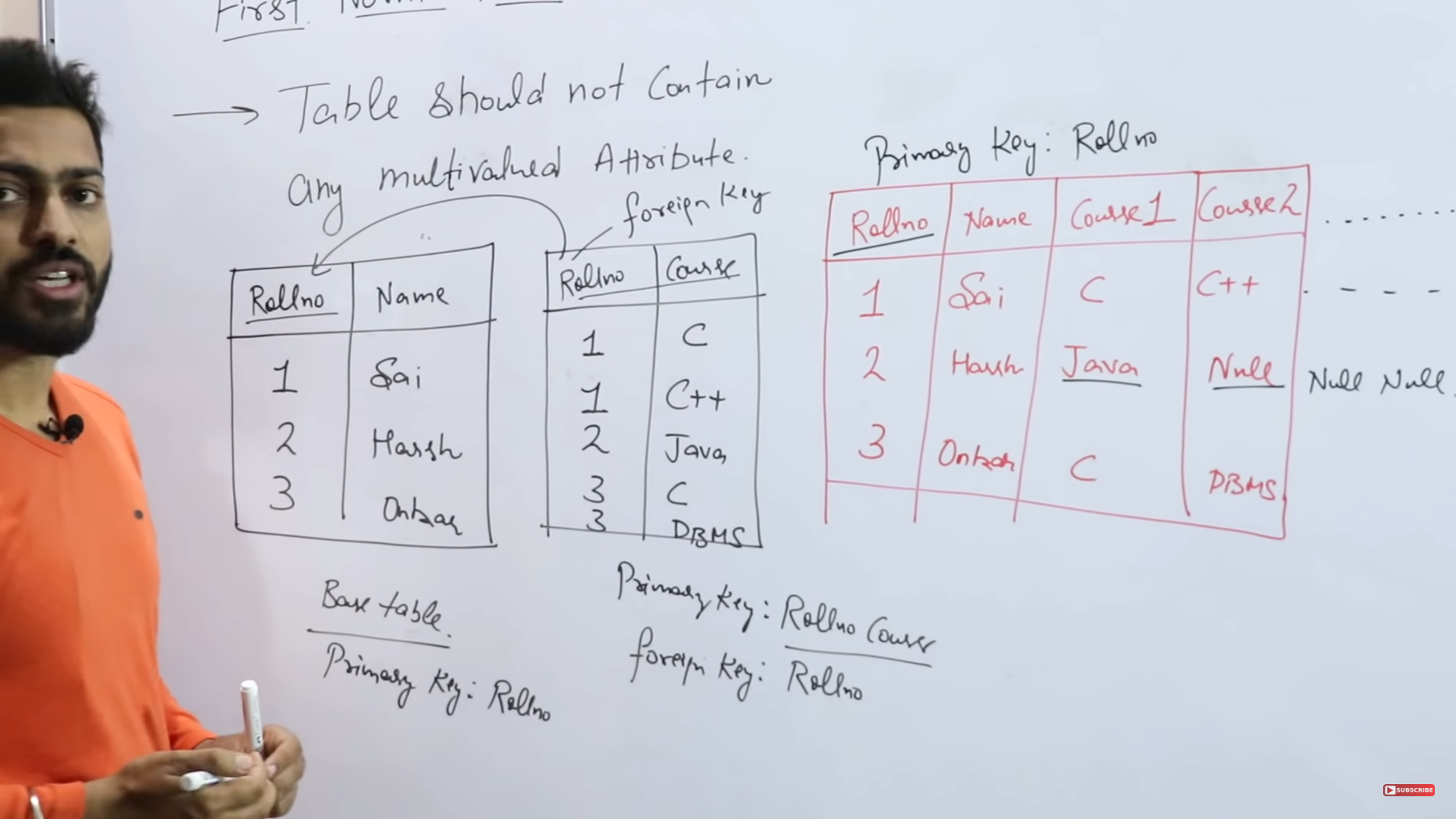
**1st representation**



2nd representation



3rd representation



**Second Normal Form (2NF):**

* It must meet the requirements of 1NF.
* It should not have partial dependencies, meaning that non-key attributes (columns) should depend on the entire primary key, not just part of it.

**Third Normal Form (3NF):**

* It must meet the requirements of 2NF.
* It should not have transitive dependencies, meaning that non-key attributes should not depend on other non-key attributes.

**Boyce-Codd Normal Form (BCNF):**

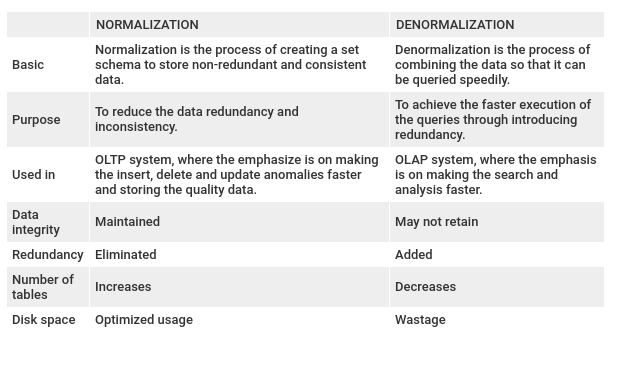
It is an extension of 3NF..

BCNF ensures that each non-key attribute is dependent only on the candidate key.  
A stronger definition of 3NF is known as Boyce Codd's normal form.

10- What are normalization and denormalization and why do we need them?

**Normalization** is the process of minimizing **redundancy(**having several copies of the same data in the database**)** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

**Denormalization** is also the method that is used in a database. It is used to add redundancy to execute the query quickly. It is a technique in which data are combined to execute the query quickly. By using denormalization the number of tables is decreased which oppose to the normalization. Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database.



11- Explain Two tier architecture

The two-tier architecture is similar to a basic **client-server** model. The application at the client end directly communicates with the database at the server-side. API's like ODBC, JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run. The application on the client-side establishes a connection with the server-side in order to communicate with the DBMS.  
An advantage of this type is that maintenance and understanding are easier, compatible with existing systems. However, this model gives poor performance when there are a large number of users.  
Faster database access because of the direct link and enhanced efficiency.



**The following are the key drawbacks of the two-tier DBMS architecture:**

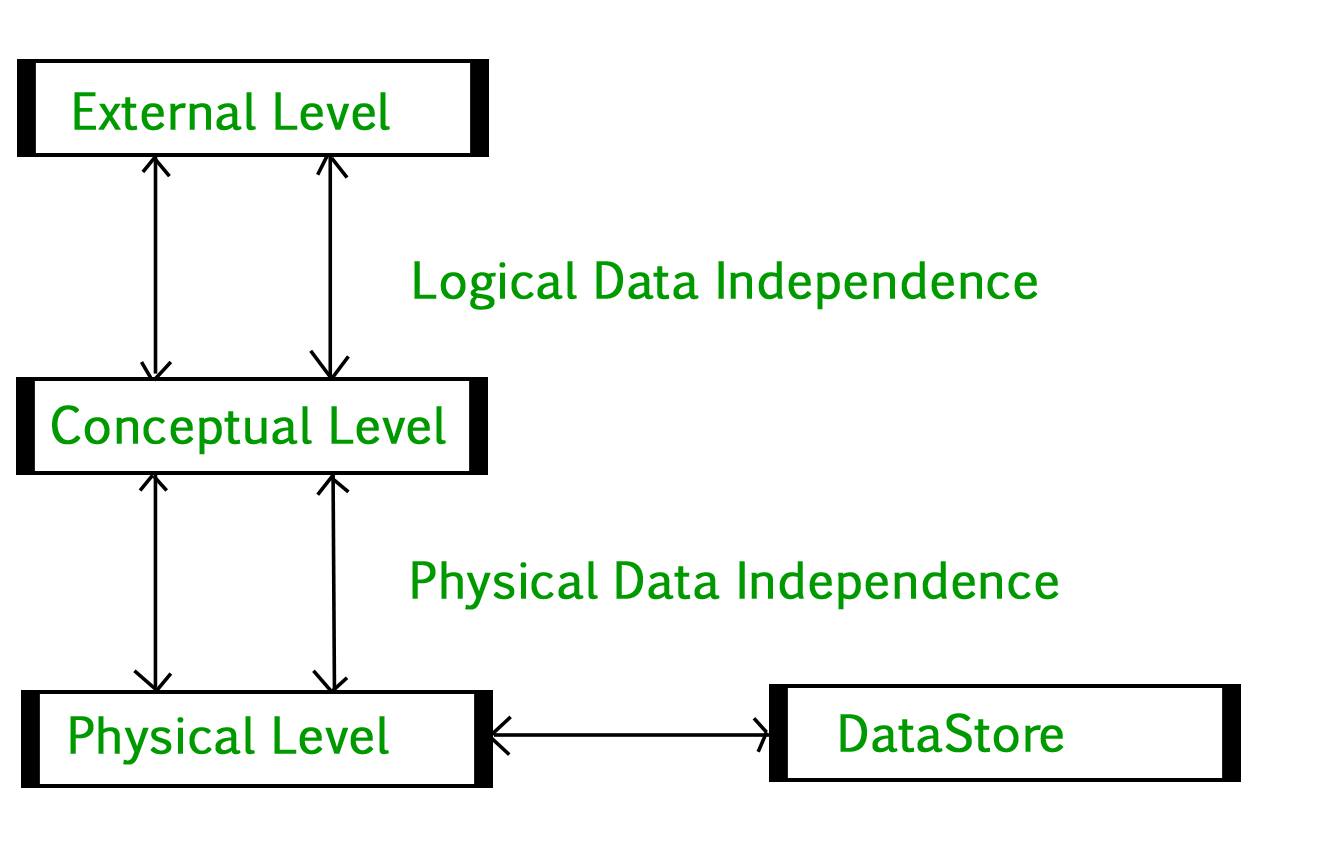
* **Scalability:**

The server is under more pressure as there are more users. Decreasing the DBMS's performance, which in turn affects the client-side application's performance.

* **Security:**

This design is prone to assaults since the client and server systems are directly connected.

12- Explain 3-Schema Architecture in DBMS

DBMS 3-tier architecture divides the complete system into three inter-related but independent modules as shown below:  
[](https://media.geeksforgeeks.org/wp-content/uploads/dbms-3tier.jpg)

1. **Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. Various users of DBMS are unaware of the locations of these objects.
2. **Conceptual Level:**At a conceptual level, data is represented in the form of various database tables. For Example, the STUDENT database may contain STUDENT and COURSE tables that will be visible to users but users are unaware of their storage.
3. **External Level:** An external level specifies a view of the data in terms of conceptual level tables.  Each external level view is used to cater to the needs of a particular category of users. For Example, FACULTY of a university is interested in looking at the course details of students, STUDENTS are interested in looking at all details related to academics, accounts, courses, and hostel details as well. So, different views can be generated for different users.

**Data Independence**  
Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

1. **Physical Data Independence:** Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.
2. **Conceptual Data Independence:** The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user’s view of the table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in conceptual schema are reflected in the user’s view.

13- What are Nested Queries in SQL?

 Nested queries or inner query or sub queries are a way to perform more complex queries by embedding one query within another. A nested query is a query that appears inside another query, and it helps retrieve data from multiple tables or apply conditions based on the results of another query. The result of inner query is used in execution of outer query.

-- syntax for nested query or subquery

-- select columns

-- from table1

-- where

-- column=(select columns from table2 where conditon);

select name from personal

where courses=(select course\_id from courses where course\_name="MBA");  -- will show name of student whose course is MBA

select name from personal

where courses IN (select course\_id from courses where course\_name IN ("MBA","Btech"));    -- will show name of student whose course is MBA or Btech

-- syntax for exists

-- select columns

-- from table1

-- where

-- exists(select columns from table2 where condition);     if any single record exists then parent command show results

select name from personal

where exists (select course\_id from courses where course\_name ="MBA");       -- this will show all names of personal table bcoz MBA record exists in 2nd table

select name from personal

where not exists (select course\_id from courses where course\_name ="Mtech");       -- this will show all names of personal table bcoz Mtech record doesnot exists in 2nd table

select name from personal

where  exists (select course\_id from courses where course\_name ="Mtech");       -- this will not show any names of personal table bcoz Mtech record doesnot exists in 2nd table

14- Explain types of JOINs in DBMS.

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

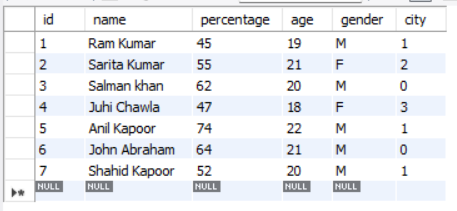
In SQL, JOIN clause is used to combine the records from two or more tables in a database.

## **Types of SQL JOIN**

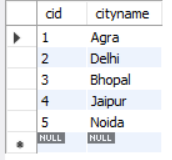
1. INNER JOIN/Simple Join
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN
5. Cross Join

### Sample Table

**Personal**

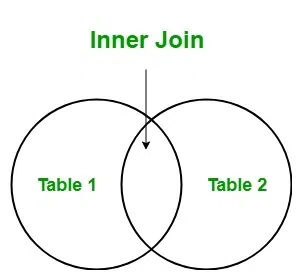


**City**



### 1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.



**Syntax**

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. INNER JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

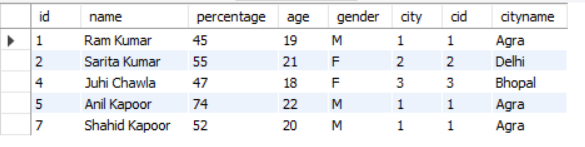
**Query**

select \* from

personal inner join city

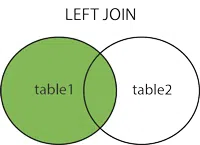
on personal.city=city.cid;

**Output**



### 2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.



**Syntax**

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. LEFT JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

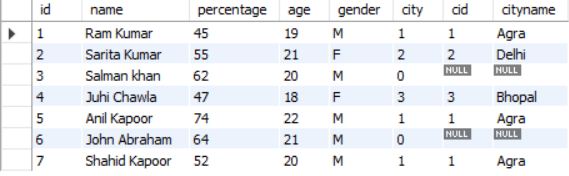
**Query**

select \* from

personal left join city

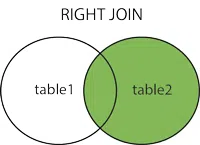
on personal.city=city.cid;

**Output**



### 3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.



**Syntax**

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. RIGHT JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

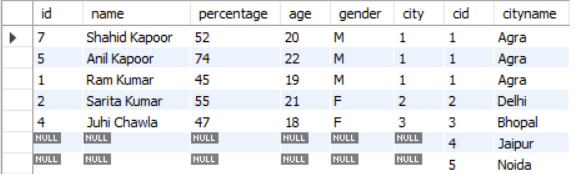
**Query**

select \* from

personal p right join city c

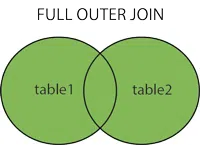
on p.city=c.cid;

**Output**



### 4. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values



**Syntax**

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. FULL JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

**Query**

select \* from

personal p full join city c

on p.city=c.cid;

### 5. Cross JOIN

The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.This kind of result is called as Cartesian Product.

-- we don't need primary of foreing key to cross join

If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

:



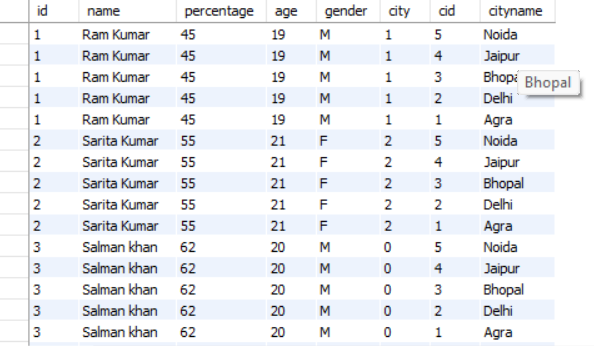
**Syntax**

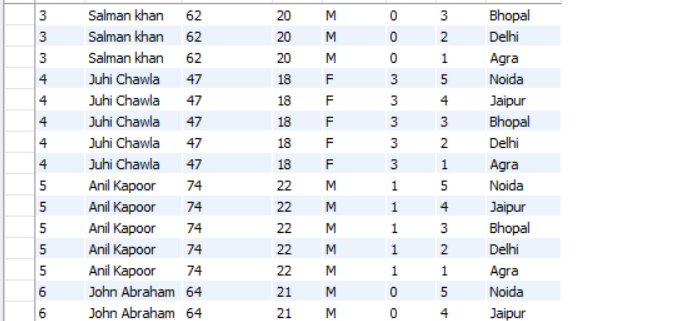
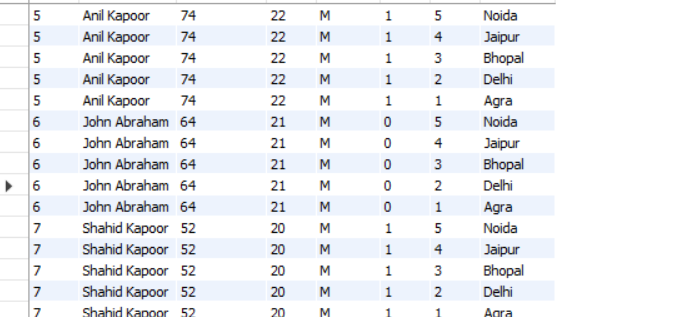
1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. RIGHT JOIN table2
4. ON table1.matching\_column = table2.matching\_column;

**Query**

select \* from

personal cross join city;



15- Difference between INNER and OUTER JOIN.

1. LEFT JOIN
2. RIGHT JOIN
3. FULL JOIN

These all are known as Outer joins.

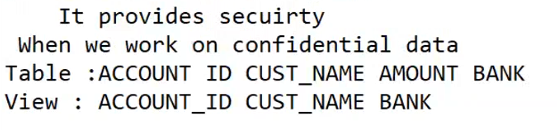
* 1. Left outer join
  2. Right outer join
  3. Full outer join

| **S.No** | **Inner Join** | **Outer Join** |
| --- | --- | --- |
| **1.** | It returns the combined tuple between two or more tables. | It returns the combined tuple from a specified table even if the join condition fails. |
| **2.** | Used clause INNER JOIN and JOIN. | Used clause LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, etc. |
| **3.** | When any attributes are not common then it will return nothing. | It does not depend upon the common attributes. If the attribute is blank then there is already placed NULL. |
| **4.** | If tuples are more. Then INNER JOIN works faster than OUTER JOIN. | Generally, The OUTER JOIN is slower than INNER JOIN. But except for some special cases. |
| **5.** | It is used when we want detailed information about any specific attribute. | It is used when we want to complete information. |
| **6.** | JOIN and INNER JOIN both clauses work the same. | FULL OUTER JOIN and FULL JOIN both clauses work the same. |
| **7.** | SQL Syntax:  select \*  from table1 INNER JOIN / JOIN table2  ON table1.column\_name = table2.column\_name; | SQL Syntax:  select \*  from table1 LEFT OUTER JOIN / RIGHT OUTER JOIN /  FULL OUTER JOIN / FULL JOIN table2 ON  table1.column\_name = table2.column\_name; |

16- What is view in database

In SQL, views contain rows and columns similar to a table, however, views don't hold the actual data.

You can think of a view as a virtual table environment that's created from one or more tables so that it's easier to work with data.



--      syntax

-- Create View view\_name

-- as

-- select columns

-- from students

-- inner join city

-- on student.city=city.cid;

create view studentdata                            -- creating view

as

select id,name,course\_name from

personal p inner join courses c

on p.courses=c.course\_id;

select \* from studentdata;                      -- accessing view data