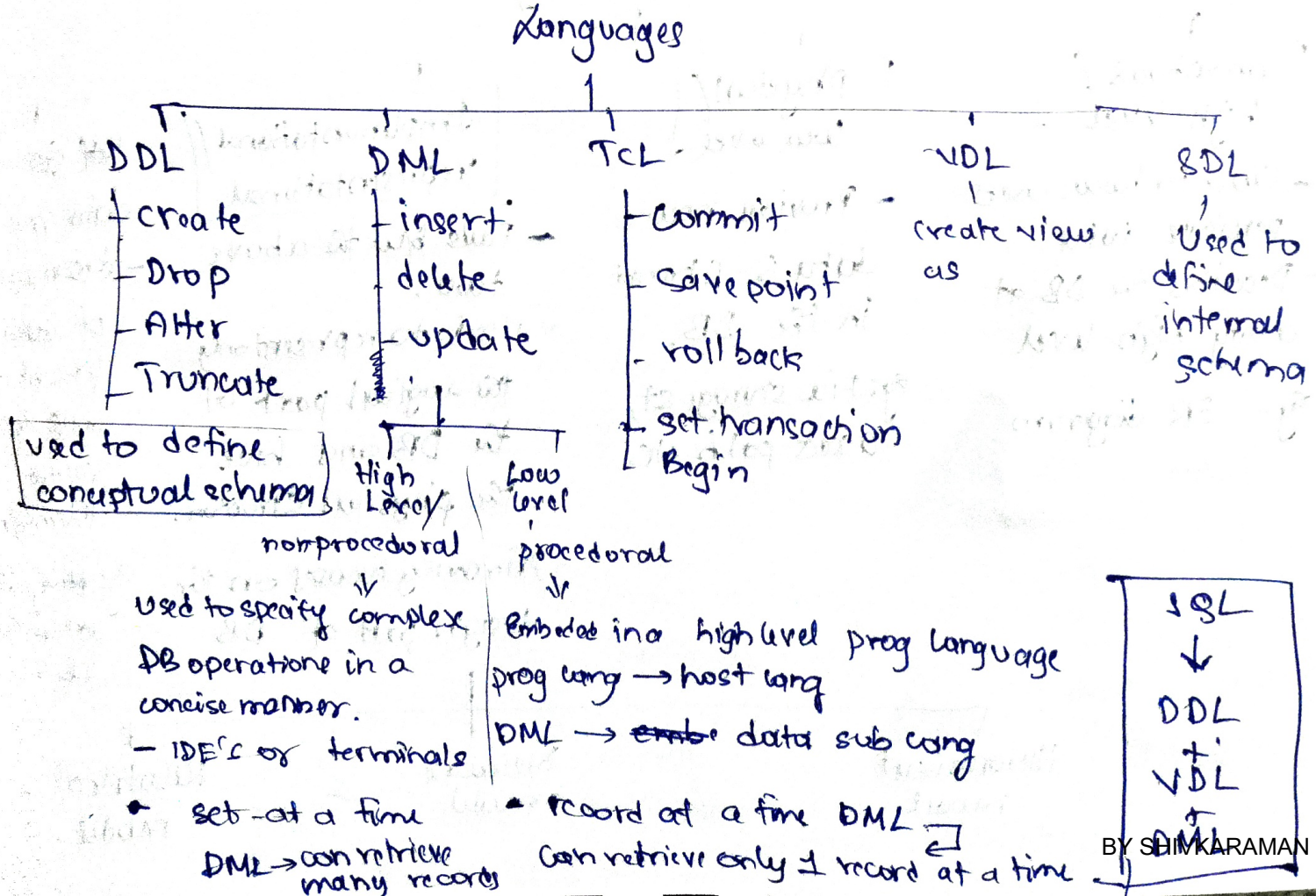
DBMS Important

* DBMS → Software to manage database.
* Functionality of DBMS → ~~Creati~~ Defining DB, creating DB, Manipulating DB and Sharing the DB.
* Limitation of file based approach
  - Data redundancy and inconsistance
  - Concurrent users not allowed
  - Multiple file formats
  - File path ie Data dependence.
* Characters of DBMS
  - Metadata and file structure stored in db catalog
  - Data is separate from program
  - Supports multiple views
  - Access controls
* DBA → Schema design, access control, selection of appropriate software to handle DB, etc



* instance / current state → Snapshot of DB at current time
* Schema → Overall design / description of DB. (fields, attributes)
* Sub-schema / view → Subset of schema
* Schema diagram → Diagramatic representation of schema.
* Schema construct → A component of schema or object within schema. Eg: STUDENT
* DB state → Empty, initial, current, valid state.
* schema evolution → adding a new field to schema.

- Relational → means arranges data into tables containing rows / records / tuples and columns / attributes. Relation b/w tables is through foreign key.
- Data model → Collection of concepts that can be used to describe the structure of DB (data types, relations, constraints etc)

Data model

Conceptual / high-level
- Close to how users percieve data
- Describes the DB at a very high level

Eg: ER diagram

Physical / low level
- Provides how data is stored in the DB.

Eg: File storage, access path etc.

Implementational / representational
- Falls b/w the above two.
- Used to represent only the logical part of the DB and hide the physical structure.
- Primarily focuses on the design part of DB

Self describing
- data model
- description of data is present along with data values themselves.

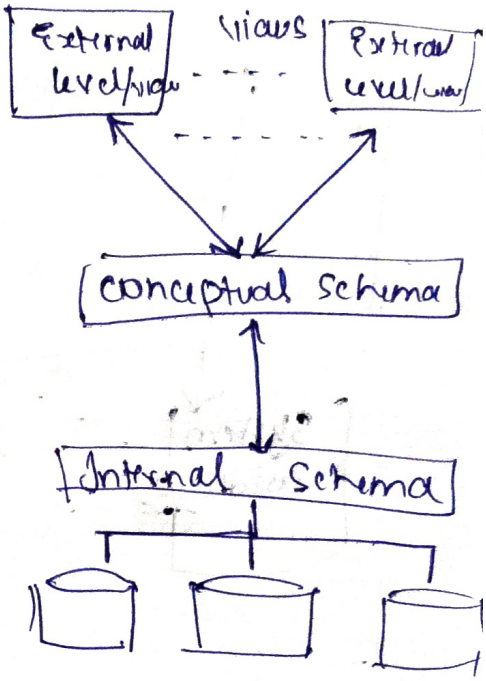
Eg: MongoDB, other NoSQL DB's

Hierarchical model | Network model | Relational model.

Conceptual data model

- **3-schema architecture**

Goal → To separate user applications from physical database. The user application is at the top-most level and the physical database is at bottom-most level.



1) Internal Level
   - Has internal schema
   - Describes the physical storage structure of DB
   - Uses physical data model.

2) Conceptual level
   - uses conceptual schema
   - Represents structure of whole DB
   - Uses representational data model.
   - Hides phy storage structure and focuses on entities, relation etc.

3) External level
   - has a no of views / external schema
   - representational data model.

Transformation of request from external → conceptual → internal level is called as **mapping**

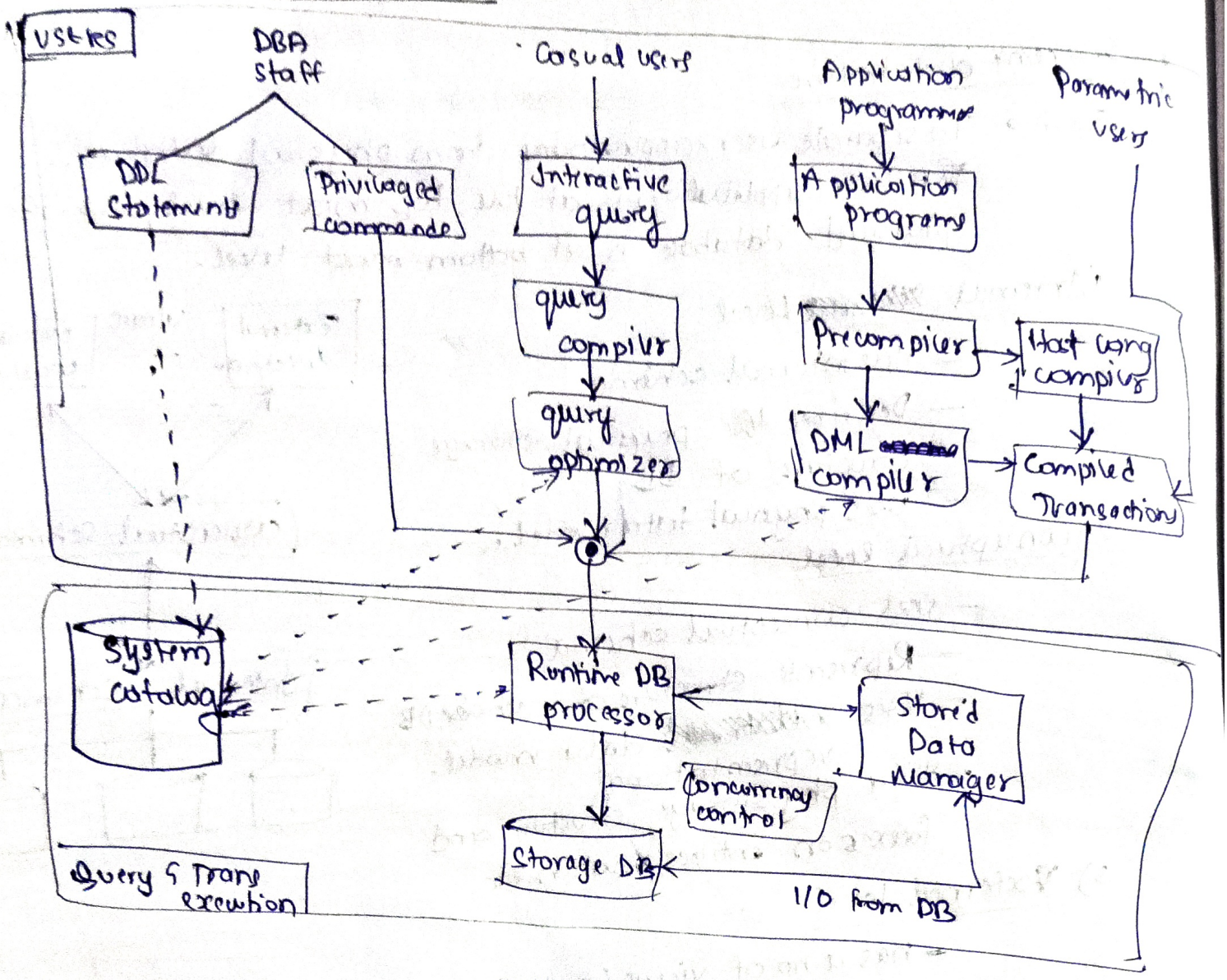
- **Data independance** → Capacity to change schema at one level of the DBMS system without having to change schema at the next higher level.

Logical data independance → Change in conceptual schema without change in external schema.

physical data independance → Change in physical / internal schema without change in conceptual schema.

→ Harder to achieve

- DBMS component modules

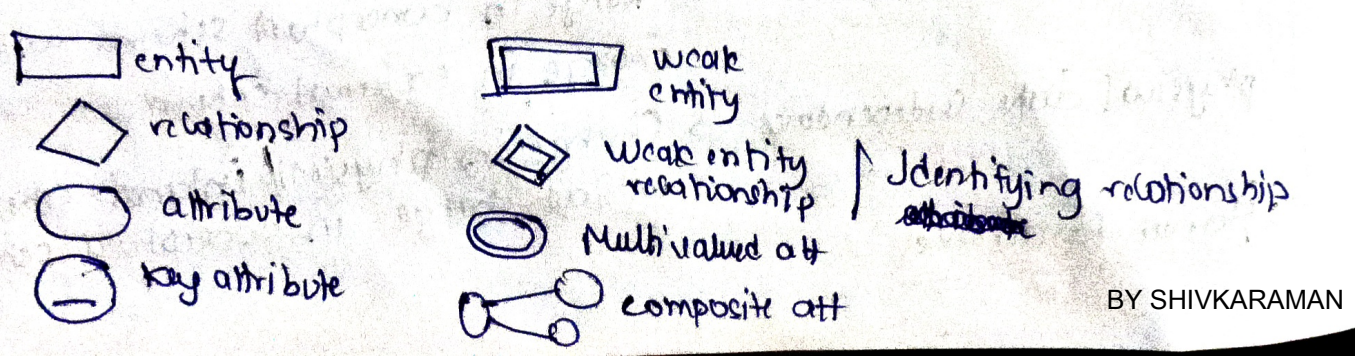


ER diagrams

- Entity → A thing in real world
- Attributes → Properties of an entity.

Types of attributes

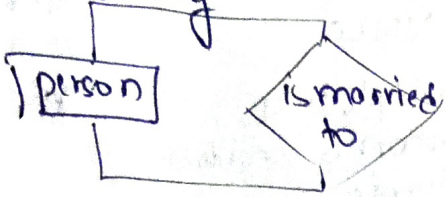
- Simple attributes - Weight
- Composite attributes - DOB (Day + Month + Year)
- Single valued
- Multi valued
- Stored attr
- Derived attribute
- NULL value
- Key attribute



• relationship: association among 2 or more entities.

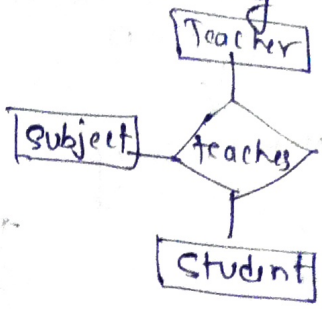
Degree of relationship

Unary



Binary

Ternary



Relationship constraints

Cardinality Ratio
- 1:1
- 1:N
- N:1
- N:M

Participation constraint
- Total
- Partial

Keys

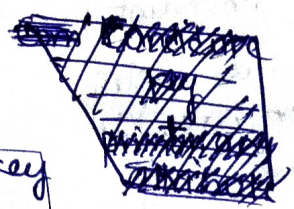
Candidate key
- Attribute whose value is unique and is capable of identifying a relationship → key
- Collection of all keys

Primary key
Most appropriate candidate key

Alternate keys
keys other than the PK

Foreign key.
Attribute or set of attributes that refering to PK of same table or different table.

Super key - candidate key + any attribute



Relational Model

- rows → tuples
  cols → attributes
  table → relation

domain → datatype + range + format
Degree → No of attributes
Cardinality → No of tuples

Characteristics of a relation

- **Ordering of tuples inside a relation** → Not necessary
- **Order of attribute inside a tuple** → Important and necessary
  - T = ⟨(Roll No, 2), (Name, Ben)⟩
  - Not necessary when value along with attribute is given.
- **Value and NULLS**
  - Each tuple in a relation is considered atomic
    - ⇒ Multivalued att and composite att not allowed
    - Multival → Separate rel
    - comp → Break down into simple components
  - NULL ⇒ Value is unknown or it does not apply
- **Interpretation of a relation**

Constraints

- **Inherent model based constraints**
  - already present in data model.
  - Eg: Duplicate tuples not allowed
- **Schema based / explicit constraint**
  - Defined by the schema of data model.
  - **Domain constraints**
    - data type + range + format
  - **Key constraint**
    - No two tuples can have the same key.
  - **Entity integrity constraint**
    - Key cannot be NULL
  - **Referential Integrity constraint**
    - Tuple in one relation that refers to another relation must refer to an existing tuple in that relation. ie
      - FK = NULL or
      - FK = value of PK of another relation
- **Application based constraints**

Update operation and constraint violation

- anomaly → Unexpected behavior during update/delete/insert operations.

1) Insert → Domain, Key, Entity integrity, Referential integrity
2) Delete → Only referential integrity
3) Update → Referential integrity, Key constraint

Relational Algaebra

Unary

- SELECT (σ)
  - * Used to select a columns
  - * $\sigma_{col\text{-}list}(Table)$
- Project (π)
  - * used to select rows based on condition
  - * $\sigma_{condition}(table)$
- Rename (ρ)

Set

- union ($\cup$)
- Intersection ($\cap$)
- set difference ($-$)
- Cartisian Product ($\times$)

- Two sets are union compatable if
  - Both have same no of attributes
  - Domain's are same
- Two sets must be union compatable to carry out $\cup$, $\cap$ and $-$ operations

Binary

- JOIN
  - equi Join
  - Natural Join
  - Cross Join
  - outer Join

$R \bowtie_{condition} S$

- Division operation

Other operation

- Aggregate functions etc.

- Restrictions on updating views →
  1) Not allowed when view is created by joining more than table
  2) Not allowed on views defined with group by and aggregate function.
  3) Not allowed on views which donot include * PK of base table.

Sql Injection

Based on 1=1

user id: `105 OR 1=1`

password: `105 OR 1=1`

```
SELECT * FROM Users
WHERE USERID=105 OR 1=1

SELECT * FROM password
WHERE password=105 OR 1=1
```

1=1 is always true and will fetch all user id & password

Based on "" = ""

user id: `"" OR "" = ""`

password: `"" OR "" = ""`

```
SELECT * FROM USERS
WHERE USERID= "" OR "" = ""
AND pass = "" OR "" = ""
```

Will return all records of userid and password

Delete V/S Truncate

DELETE

- DML command
- Deletes a row based on given condition.

```
DELETE FROM table-name
WHERE condition;
```

- Deleted rows are logged in MySQL transaction log, making it possible to rollback if needed.

TRUNCATE

- DDL command
- Used to delete all rows from table.

```
TRUNCATE TABLE
  table-name;
```

- TRUNCATE is not logged in transaction log which makes it unable to be rolled back.
- Cannot use WHERE clause.
- Faster than DELETE

Relational DB design

- Design goals
  - Information preservation
  - Minimizing redundancy ...

Design Guidelines

Imparting clear semantics to attributes.

- Don't combine attributes from multiple entity types ie tables

Redundant info in tuples & update anomalies.

- anamoly - Unusual behavior during DML operations.

Sol → Reduce redundancy
→ Design schema that doesn't suffer from any anamoly

- Insertion anamoly
  - Certain information cannot be stored until some other info is also stored
- Deletion anamoly
  - Deletion of data representing certain information results in loosing some other info as well.
- Update anomaly
  - Presence of redundant data will lead to inconsistency

NULL values

- Problems
  - Wasted storage
  - Problem in understanding
  - Problem when joining operation

Sol → Avoid fat tables

Generation of spurious tuples

- Bad Join

• Functional dependency :- It is a constraint or relationship between two sets of attribute from DB.

$X \rightarrow Y$: X is functionally dependent on Y

Trivial FD: Y is a subset of X $\Rightarrow$ $X \cap Y \neq$ NULL

Non-trivial FD: $X \cap Y =$ NULL

• Normalization

Process of analyzing the given schema based on their FD's and PK's to achieve desirable properties of minimizing redundancies and information preservation.

1NF: No multivalued attributes. i.e all attributes are atomic

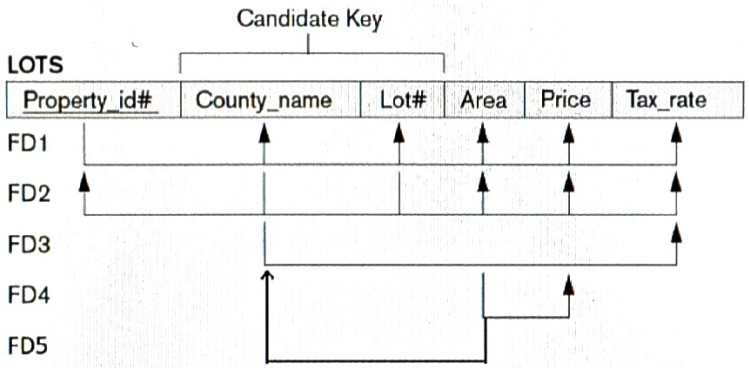
2NF: No partial dependency ie non-prime attribute not partially dependent on any of the key.

3NF: No transitive dependency i.e no nonprime attribute is transitively dependent any key.

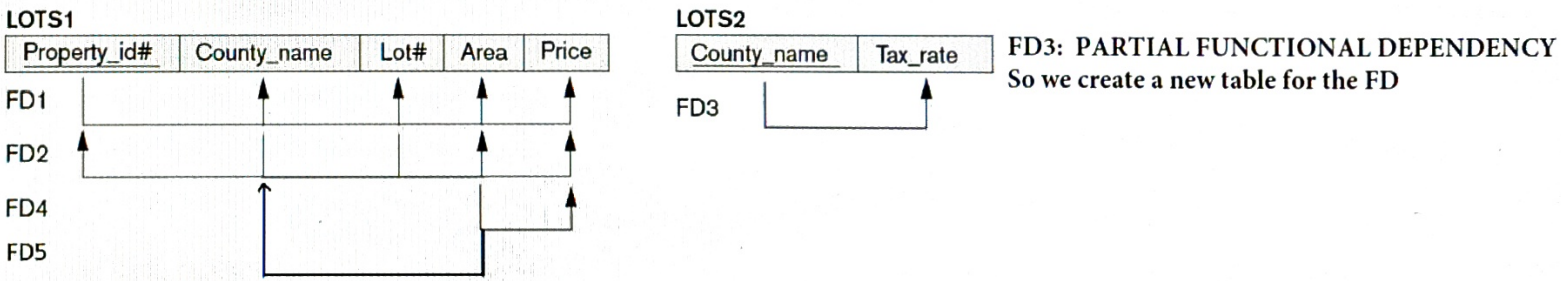
BCNF: candidate key → Non-prime attribute dependency should not be present

We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2because its attributes no longer coexist in the same relation after decomposition

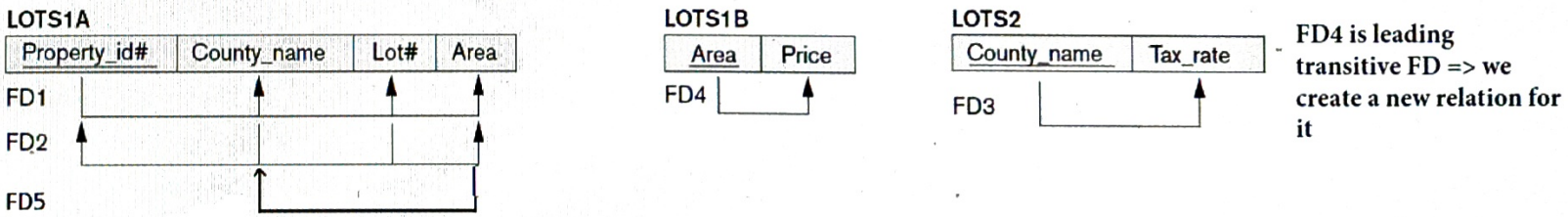
1NF



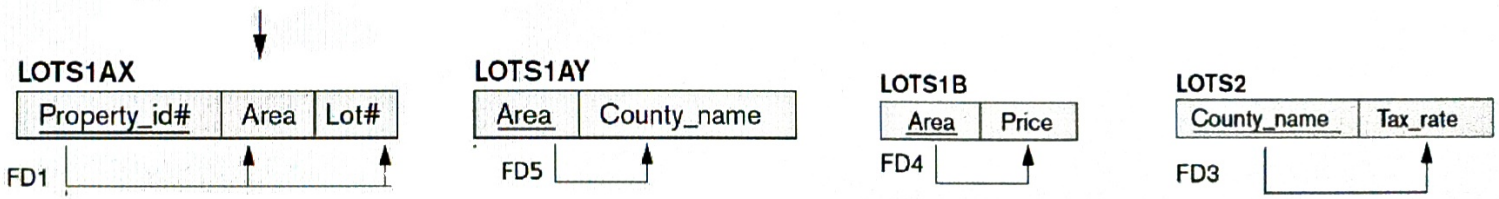
2NF



3NF



BCNF



FD5: A candidate key is FD on a non prime attributs

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

Schedule — A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

Serial Schedule — It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.- **view equivalence and conflict equivalence**

1> Conflict Equivalence

Conflicts between transactions typically arise when they access shared data concurrently.

Read-Write, Write-Read, Write-Write → Conflicts

Read-Read → Non conflict Pair

Compare adj-operations → If non-conflict → swap
conflicting → NO SWAP

Check if S and S' are conflict equivalent

Eg:

S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	

Non conflict ⇒ Swap

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)

Non conflict ⇒ Swap

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

= S'

S'

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

∴ S ≡ S'

Concurrency Control

Lock Based Protocols

Time stamp based protocols

- Binary locks : locked or unlocked
- Shared / exclusive locks: This type of locking mechanism differentiates the lock based on their uses.
  write operation → exclusive lock
  read operation → shared lock.

1) Simplistic lock protocol: lock before write operation and unlock after write operation is completed.

2) Preclaiming lock protocol: Pre claiming locks protocols evaluate their operations and create a list of data items on which they need locks. Before initializing an execution, the transaction requests the system for all the locks it needs before hand. If all locks are granted, the transaction executes and releases all locks when all operations are over. If all locks are not granted, the transaction rolls backs and waits until all locks are available.

3) 2-Phase locking: Divides the execution phase of a transaction into three parts
   1) Growing phase: As transaction starts executing, it seeks permission for the lock it requires
   2) Second part: Transaction acquires all the locks.
   3) Shrinking phase: As soon as the transaction releases its first lock, third phase starts. In this phase, the transaction cannot demand any new lock, it only releases acquired locks.

4) Strict 2-P-Locking: Same as 2-PL but here it doesnot release any lock until the commit point and releases at that time.

How data is stored in DBMS

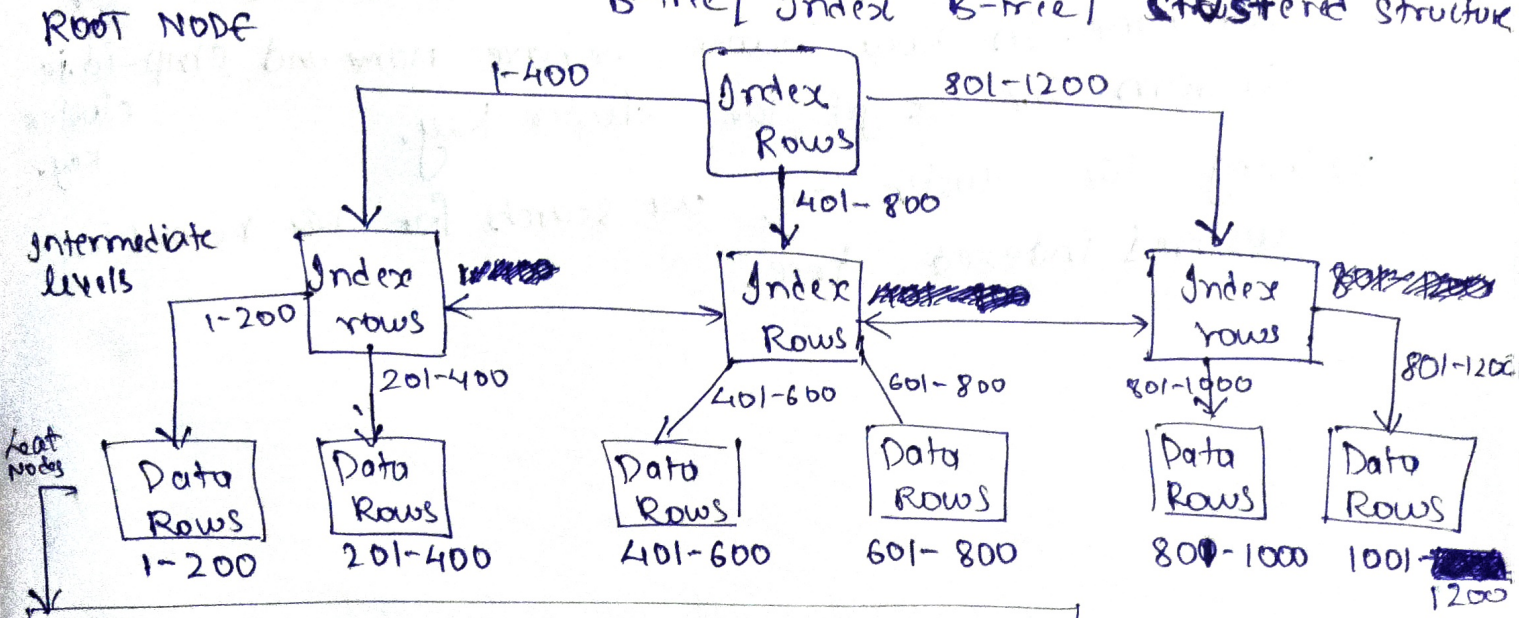
1) Clustered Index Structure

For given table Employee

Emp-id	Name	Email	Dept

→ 1200 rows

* First, the tuples/rows are sorted according to the PK
* The data is stored in a series of data pages in a tree like structure ⟶ Called as B-tree / Index B-tree / clustered structure



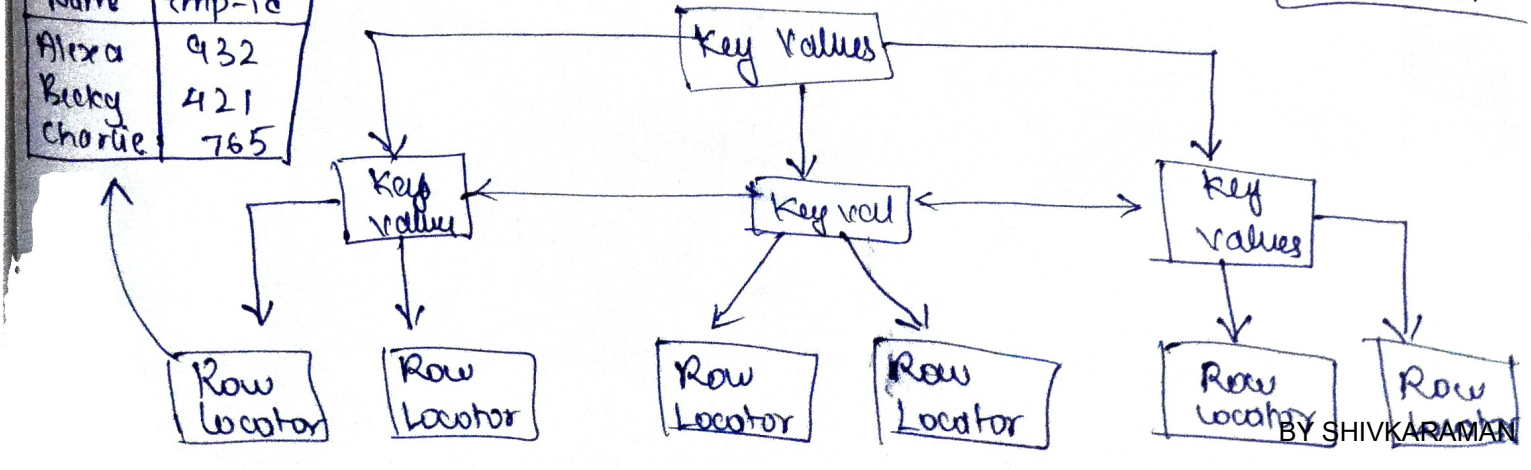
Data pages: 8 KB ⟶ Table data actually resides here

* So in clustered index, indexing is done on the primary key. So this is helpful when we search by primary key.

2) Non-Clustered index structure

* What if we search by Employees name? There is no index for name column ⇒ SQL server has to scan every record in the table. ↓ Inefficient
* Solution ⟶ form a non clustered index on name column

Name	Emp-id
Alexa	432
Becky	421
Charlie	765



* Here since we are searching on name column, the key values ie name is sorted in alphabetical order. The row locator present at the leaf nodes contain employee names and its cluster key i-e Employee ID.

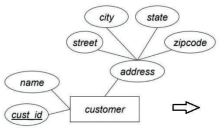
* So now when we search employee by name, both non-clustered index and clustered index are used to fetch data.

1) Uses non-clustered index on name column to search for given name. In Row locator, we have name and emp-id. ie So from here we get the cluster key. cluster key.

2) Using the cluster key, we search for the record in clustered indexed tree.

ER Model -> Relational Model

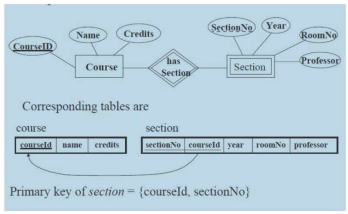
Step 1: Mapping of Regular Entity Types.



1. Attributes must be simple
2. Should have primary key

customer(*cust_id*, name, street, city, state, zipcode)

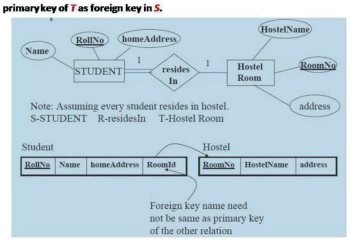
Step 2: Mapping of Weak Entity Types



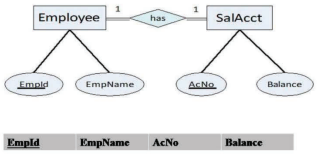
Create a foreign in weak entity refering to primary key of strong entity and make the partial key and foreign key as primary key in weak entity

Step 3: Mapping of 1:1 Relation Types

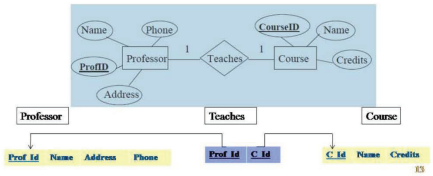
1. **Foreign Key approach :** Choose one of the relations-say *S*, and include the primary key of *T* as foreign key in *S*.



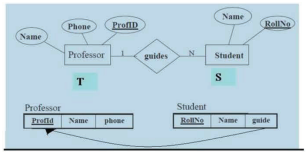
2. **Merged relation option : merging the two entity types** and the relationship into a single relation. This may be appropriate when both participations are total.



3. **Cross-reference or relationship relation option:** Set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. **(Partial on both)**

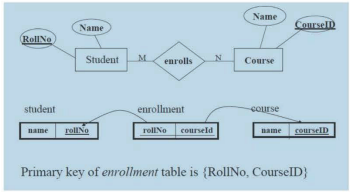


Step 4: Mapping of 1:N Relation Types



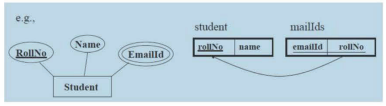
Include primary key of **1-side** as foreign key of **N-side**

Step 5: Mapping of M:N Relationship Types.



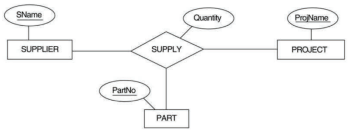
Make a new table for the relation and add 2 foreign key referencing the primary keys of the two ettities

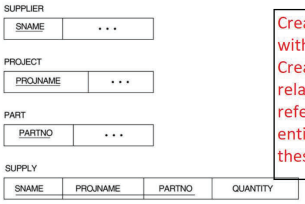
Step 6: Mapping of Multivalued attributes.



Create a new relation for the multivalued attribute and add a foregin key referencing to primary key of entity

Step 7: Mapping of N-ary Relationship Types.

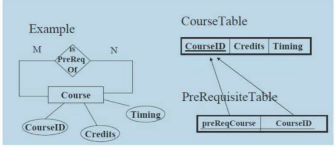




Create a relation for each entity with their attributes and primary keys
Create a new relation for the relationship and create foreign keys referencing to each of the related entity and the composite key of all these foreign keys is the primary key

Handling Recursive relationships

- Make a table T for the participating entity set E and one table for recursive relationship R.



JOIN = CROSS PRODUCT + SELECTION BASED ON CONDITION

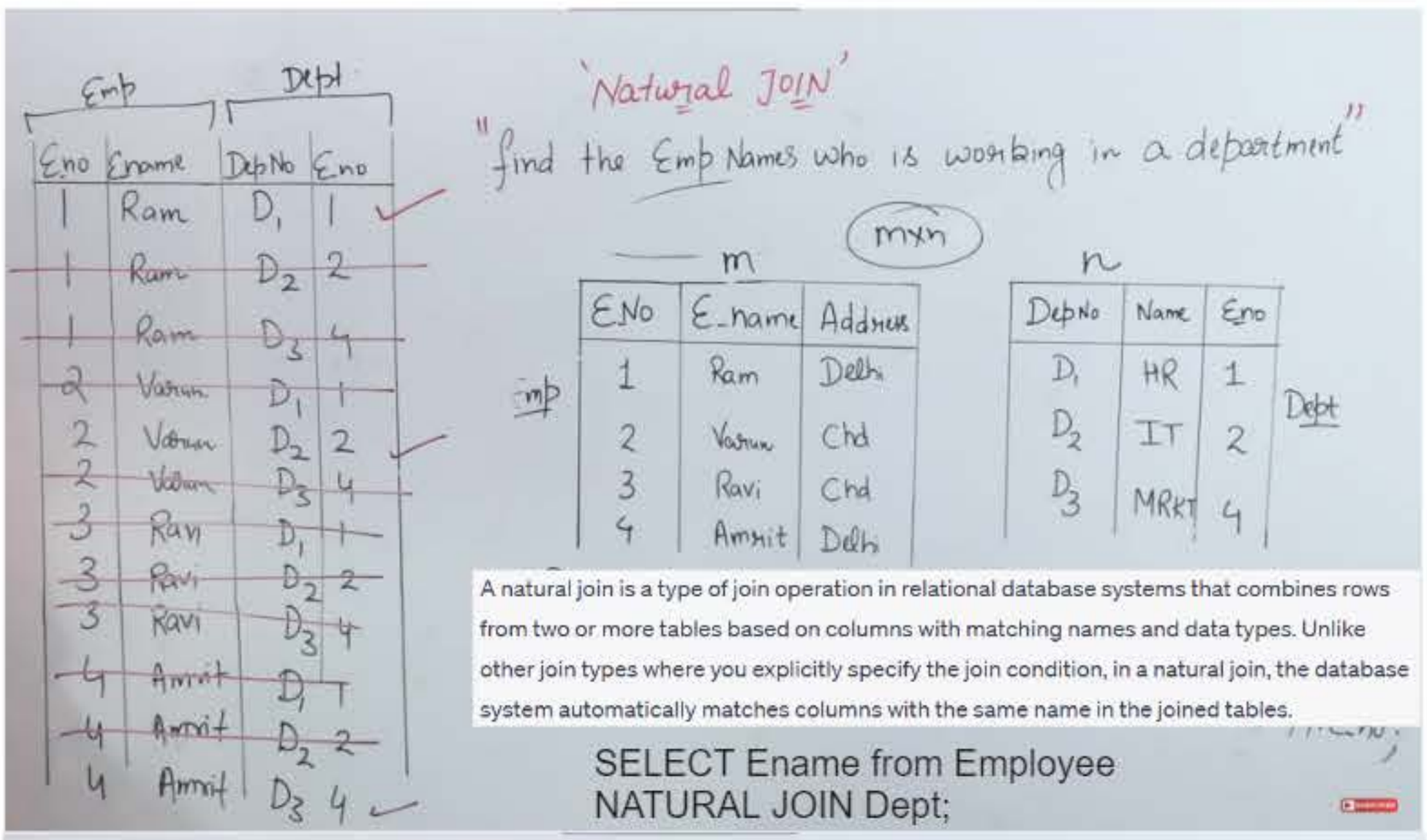
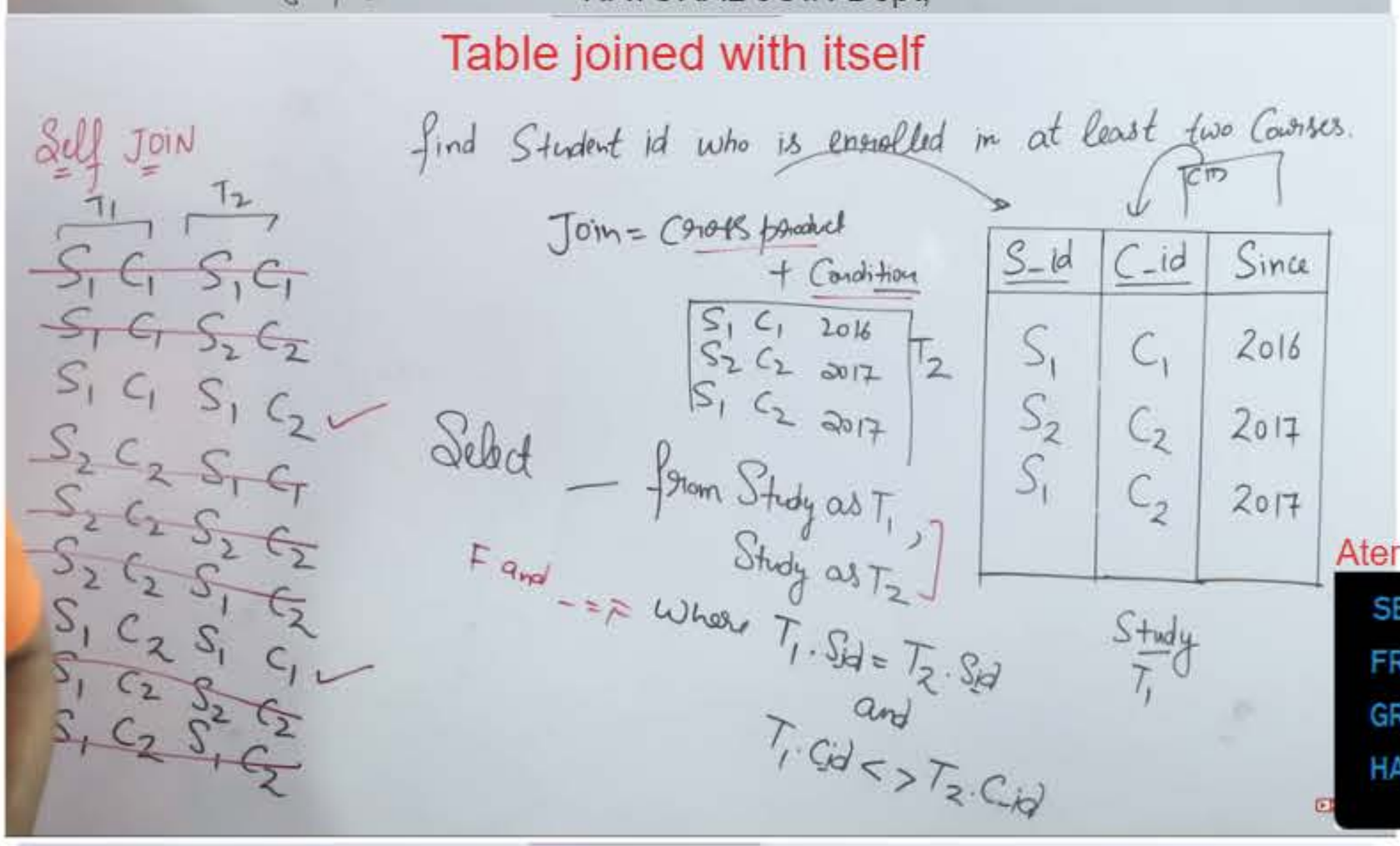
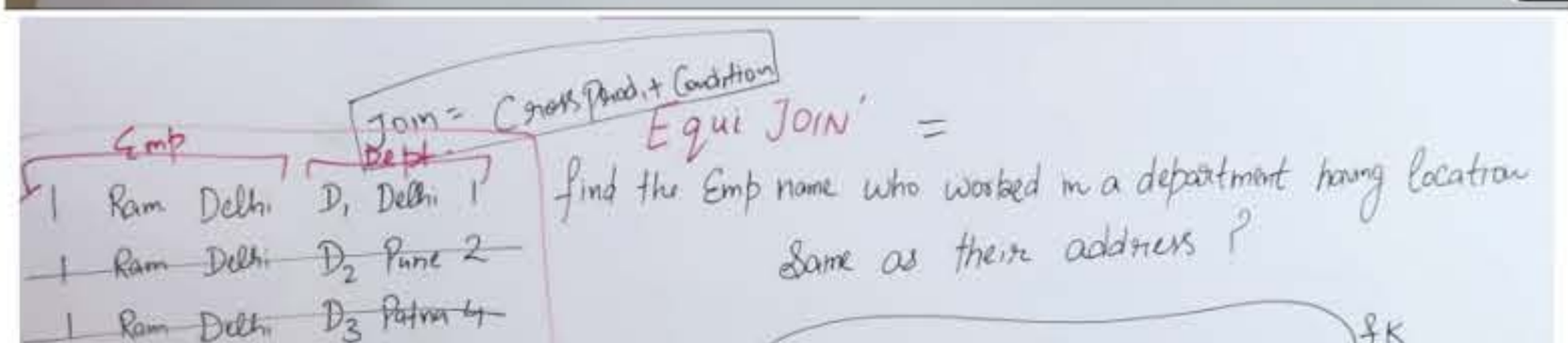


Table joined with itself



Aternate approcah

```
SELECT sid
FROM study
GROUP BY sid
HAVING COUNT(cid) >= 2;
```



An equi join, also known as inner join, is a type of join operation in relational database systems that combines rows from two or more tables based on matching values in specified columns.

1. Left outer join

Keeps every tuple in first or left relation R, if no matching tuple is found in S, then attributes of S in the join result are filled or "padded1' with null values.

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

R ⋈ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
B	2	NULL	NULL
D	3	D	3
E	4	E	4
F	5	NULL	NULL

2.Right outer join

Keeps every tuple in second or right relation S in the result,

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

R ⋈ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
NULL	NULL	C	2
D	3	D	3
E	4	E	4

3. Full outer join

Keeps all tuples in both left and right relations when no matching tuples are found, padding them with null values as needed

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

R ⋈ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
B	2	NULL	NULL
D	3	D	3
E	4	E	4
F	5	NULL	NULL
NULL	NULL	C	2

What is Cursor in SQL ?

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

SQL QUERIES

write an sql query to get the third maximum salary of an employee from employee table

Assuming you have an "Employee" table with columns "EmployeeID" and "Salary", you can use the following SQL query to get the third maximum salary:

```sql
SELECT Salary
FROM Employee
ORDER BY Salary DESC
LIMIT 1 OFFSET 2;
```

Explanation:

- The `ORDER BY Salary DESC` clause sorts the salaries in descending order.
- The `LIMIT 1 OFFSET 2` clause selects only the third row (salary) from the sorted result set. The `OFFSET 2` skips the first two rows (salaries), and the `LIMIT 1` selects the third row (salary).

How can you create an empty tabel from an existing table

You can create an empty table from an existing table in MySQL by using the `CREATE TABLE` statement with the `LIKE` keyword to copy the structure of the existing table. Here's an example:

```sql
CREATE TABLE new_table LIKE existing_table;
```

This statement creates a new table called `new_table` with the same structure as `existing_table`, but without any data.

You can also add additional table options to the `CREATE TABLE` statement, such as indexes, primary keys, and foreign keys, to further customize the new table.

Once the new table has been created, you can insert data into it using the `INSERT INTO` statement or by using other SQL commands to manipulate the data as needed.

1. `SELECT INTO` for Creating a Table:

```sql
SELECT *
INTO temp_table
FROM base_table;
```

In this command, the `SELECT INTO` syntax creates a new table called `temp_table` and inserts all rows and columns from the `base_table` into the newly created table. This is a common way to quickly create a new table that mirrors the structure and data of an existing table.

2. `INSERT INTO` for Inserting Data:

```sql
INSERT INTO #temp_table
SELECT * FROM base_table;
```

In this command, the `INSERT INTO` statement inserts data into an existing table called `#temp_table`. This is typically used when you already have a temporary table created and you want to populate it with data from the `base_table`.

How to fetch common records from two tables

To fetch common records from two tables, you can use the `INNER JOIN` clause in your SQL query. Here's an example:

Suppose you have two tables called "Table1" and "Table2" with a common column called "ID", and you want to fetch records that exist in both tables. You can use the following SQL query:

```sql
SELECT Table1.*, Table2.*
FROM Table1
INNER JOIN Table2
ON Table1.ID = Table2.ID;
```

How to fetch alternate records from a table

To fetch alternate records from a table, you can use the `ROW_NUMBER()` function along with a modulo operation in your SQL query to number the rows in the table, and then filter by even or odd row numbers. Here's an example:

Suppose you have a table called "Employee" with a column called "EmployeeID", and you want to fetch alternate rows from the table. You can use the following SQL query:

```vbnet
SELECT *
FROM (
  SELECT *, ROW_NUMBER() OVER (ORDER BY EmployeeID) AS rownum
  FROM Employee
) AS numbered_rows
WHERE numbered_rows.rownum % 2 = 0;
```

Explanation:

- The `ROW_NUMBER()` function assigns a sequential number to each row in the table, based on the order specified in the `ORDER BY` clause.
- The `AS rownum` alias gives a name to the row number column.
- The `WHERE` clause filters the result set to only include rows with even row numbers, using the modulo operator `%` to check for divisibility by 2.

Note that you can customize the ordering of the rows by changing the `ORDER BY` clause,

How to fetch unique records from a table

To fetch unique records from a table, you can use the `DISTINCT` keyword in your SQL query

```sql
SELECT DISTINCT CustomerID
FROM Orders;
```

What is the command used to fetch first five characters of the string

To fetch the first five characters of a string, you can use the `LEFT` function in your SQL query. Here's an example:

```sql
SELECT LEFT(ProductName, 5) AS ShortName
FROM Products;
```

Explanation:

- The `LEFT` function returns the specified number of characters from the beginning of a string.

BY SHIVKARAMAN