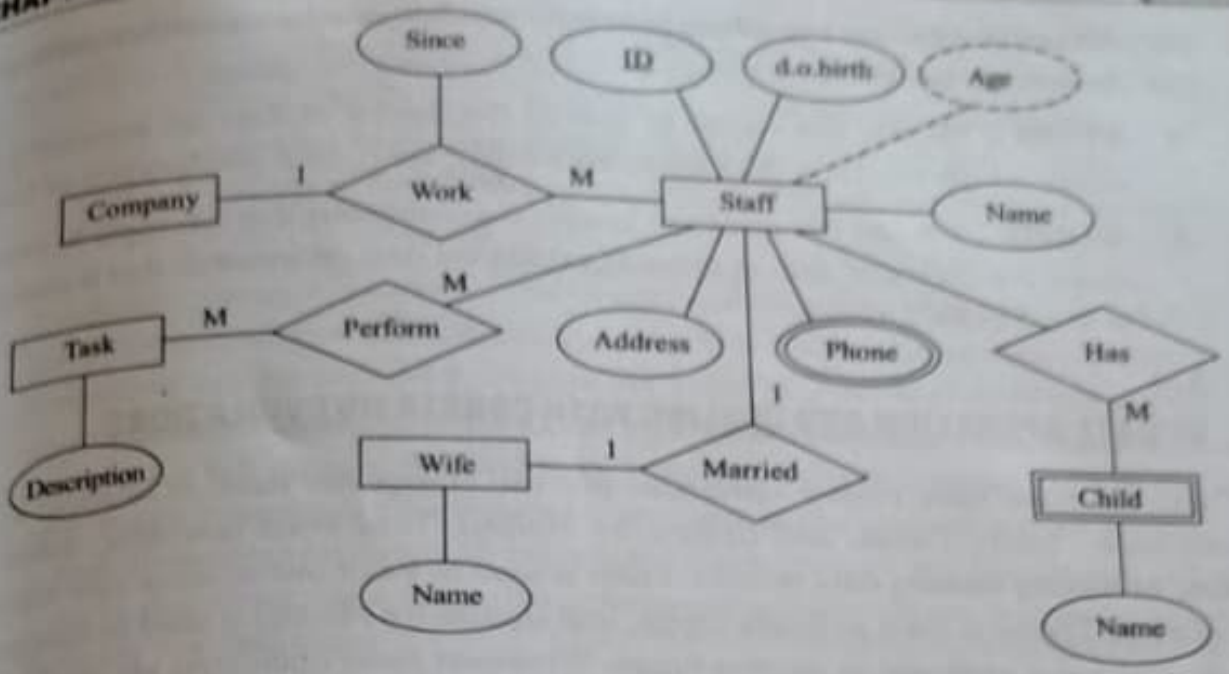


The relational schema for the ER Diagram is given below as:

Company(CompanyID, name, address)

Staff(StaffID, dob, address, ***WifeID***)

Child(ChildID, name, ***StaffID***)

Wife (WifeID, name)

Phone(PhoneID, phoneNumber, ***StaffID***)

Task (TaskID, description)

Work(WorkID, ***CompanyID, StaffID***, since)

Perform(PerformID, ***StaffID, TaskID***)

Interaction with Databases

The Structured Query Language or SQL makes it easy for the user to interact with the database. It is a comprehensive database language and contains statements for data definition, query, and update.

SQL also facilitates the migration of the users from one database application to another database application. Data Query Language (DQL) is a subset of SQL and is mostly in use to get the answer of most of the basic queries. The basic set of queries consists of :

Project

Pname	Pnumber	Plocation	Dnum

Works_On

Lssn	Pno	Hours

Dependent

Lssn	Dependent_name	Sex	Bdate	Relationship

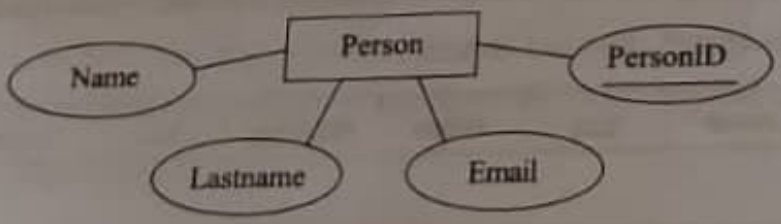
Convert ER Diagram to Relational Database schema/table

The guidelines or rules to design the relational database schema provide a step-by-step procedure. This procedure converts the ER design into a Relational Data model design in order to form the desired database. Following are the steps to convert the ER design into a Relational Data model design :

The ER Model is intended as a description of real-world entities. Although it is constructed in such a way as to allow easy translation to the relational schema model this is not an entirely trivial process. The ER diagram represents the conceptual level of database design meanwhile the relational schema is the logical level for the database design. We will be following the simple rules :

(i) **Entities and Simple Attributes :** An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters. Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.

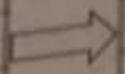
Taking the following simple ER diagram :



The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for

Person(personid , name, lastname, email)

Customer ID	Customer	Status Name	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3



Customer ID	Customer	Status Name	Dept ID
101	Amazon	Active	1
102	Google	Inactive	2
103	Flipkart	Active	3
104	Apple	Active	3

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of one relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

Operation :

Update the status of Customer tuple with CustomerName = 'Google' from 'Active' to 'Inactive'

Result : Aceptable.

Operation :

Update the DeptID of the Customer tuple with CustomerID=101 to 2.

Result : Acceptable.

Operation :

Update the DeptID of the Customer tuple with CustomerID = 102 to 5.

Result : Unacceptable, because it violates referential integrity.

Operation :

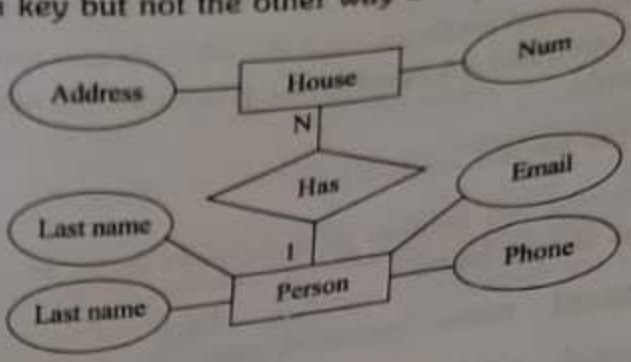
Update the CustomerID of the Customer tuple with CustomerID = 102 to 103.

Result : Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple.

Delete Operation :

To specify deletion, a condition on the attributes of the relation selects the tuple to be delete.

(iv) 1 : N Relationships : For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a House from zero to many, but a House can have only one Person. To represent such relationship the personid as the Parent node must be placed within the Child table as a foreign key but not the other way around as shown next :

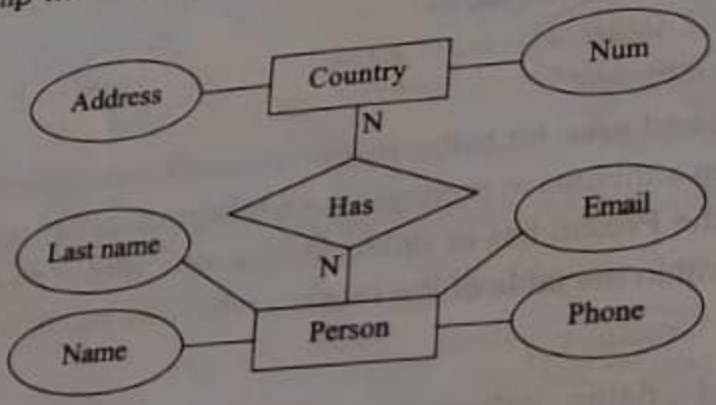


It should convert to:

Persons(personid, name, lastname, email)

House (houseid, num, address, ***personid***)

(v) N : N Relationships : We normally use tables to express such type of relationship. This is the same for N – ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below :



Person(personid, name, lastname, email)

Countrie(countryid, name, code)

*HasRelation (hasrelatid, **personid, countryid**)*

Example to convert ER-diagram to relational schema

(i) **Meaning of the relation attributes** : When the attributes are grouped to form a relation schema, it is assumed that attributes belonging to one relation have certain real-word meaning and a proper interpretation associated with them. This meaning (Semantics) specifies how the attribute values in a tuple relate to one another. The conceptual design should have a clear meaning, if it is done carefully, followed by a systematic mapping into relations and most of the semantics will have been accounted for. Thus the easier it is to explain the meaning of the relation, the better the relation schema design will be.

GUIDELINE 1 : Design a relation schema so that it is easy to explain its meaning. The attributes from multiple entity types and relationship types should not be combined into a single relation. Thus a relation schema that corresponds to one entity type or one relationship type has a straight forward meaning.

(ii) **Redundant information in tuples and update anomalies** : One major goal of schema design is to minimize the storage space needed by the base relations. A significant effect on storage space occurred, when we group attributes into relation schemas. The second major problem, when we use relations as base relations is the problem of update anomalies. There are mainly three types of update anomalies in a relation i.e., Insertion Anomalies, deletion anomalies and modification Anomalies. Various dependencies in relational database cause these anomalies.

Anomalies : Anomalies refer to the undesirable results because of modification of data. Consider the relation Employee with attributes EID, Name, Salary, Dept.No, and Dept.Name as shown in below Figure. The various anomalies are as follows :

Employee :

EID	Name	Salary	Dept.No	Dept.Name
1	Shivi Goyal	10,000	2	Accounts
2	Amit Chopra	9,000	2	Accounts
3	Deepak Gupta	11,000	1	Sales
4	Sandeep Sharma	8,500	5	Marketing
5	Vikas Malik	7,000	5	Marketing
6	Gaurav Jain	15,000	2	Accounts
7	Lalit Parmar	14,000	5	Marketing
	10	10	Finance	
8	Vishal Bamel	10,500	2	Accounts

Cannot be inserted

1. **Insertion Anomaly :** Suppose you want to add new information in any relation but cannot enter that data because of some constraints. This is known as Insertion anomaly. In relation Employee, you cannot add new department Finance unless there is an employee in Finance department. Addition of this information violates Entity Integrity Rule1. (Primary Key cannot be NULL). In other words, when you depend on any other information to add new information then it leads to insertion anomaly.

2. **Deletion Anomaly :** The deletion anomaly occurs when you try to delete any existing information from any relation and this causes deletion of any other undesirable information.

   In relation Employee, if you try to delete tuple containing Deepak this leads to the deletion of department "Sales" completely (there is only one employee in sales department).

3. **Updation Anomaly :** The updation anomaly occurs when you try to update any existing information in any relation and this causes inconsistency of data.

   In relation Employee, if you change the Dept.No. of department Accounts. We can update only one tuple at a time.

   This will cause inconsistency if you update Dept.No. of single employee only otherwise you have to search all employees working in Accounts department and update them individually.

GUIDELINE 2 : Design the base relation schema in such a way that no updation anomalies (insertion, deletion and modification) are present in the relations. If present, note them and make sure that the programs that update the database will operate correctly.

(iii) **Null values in tuples :** When many attributes are grouped together into a very big relation and many of the attributes do not apply to all tuples in the relation, then there exists many NULL's in those tuples. This wastes a lot of space. It is also not possible to understand the meaning of the attributes having NULL Values. Another problem occurs when specifying the join operation. One major and most important problem with Null's is how to consider them when aggregate functions (i.e., COUNT or SUM) are applied. The Null's can have multiple interpretations, like :

(a) The attribute does not apply to this rule.

(b) The attribute is unknown for this tuple.

(c) The value is known but not present i.e., cannot be recorded.

GUIDELINE 3 : Try to avoid, placing the attributes in a base relation whose value may usually be NULL. If Null's are unavoidable, make sure that apply in exceptional cases only and majority of the tuples must have some value which is not NULL.

(iv) **Generation of spurious tuples** : The Decomposition of a relation schema R into two relations R1 and R2 is undesirable, because if we join them back using NATURAL Join, we do not get the correct original information. The join operation generates spurious tuples that represent the invalid information.

GUIDELINE 4 : The relation Schemas are designed in such a way that they can be joined with equality conditions on attributes that are either primary key or foreign key. This guarantees that no spurious tuples will be generated. Matching attributes in relations that are not (foreign key, primary key) combinations must be avoided because joining on such attributes may produce spurious tuples.

4.2 FUNCTIONAL DEPENDENCIES

Functional dependencies are the result of interrelationship between attributes or in between tuples in any relation.

Definition : In relation R, X and Y are the two subsets of the set of attributes, Y is said to be functionally dependent on X if a given value of X (all attributes in X) uniquely determines the value of Y (all attributes in Y).

It is denoted by $X \rightarrow Y$(Y depends upon X).

Determinant : Here X is known as determinant of functional dependency.

Consider the example of Employee relation :

Employee :

EID	Name	Salary
1	Aditya	15,000
2	Manoj	16,000
3	Sandeep	9,000
4	Vikas	10,000
5	Manoj	9,000

In Employee relation, EID is primary key. Suppose you want to know the name and salary of any employee. If you have EID of that employee, then you can easily find

We can find all the closures by applying the following rules

1. **Reflexivity** : If $B \rightarrow A$ then $A \rightarrow B$
2. **Augmentation** : If $A \rightarrow B$ then $CA \rightarrow CB$
3. **Transitivity** : If $A \rightarrow B$, $B \rightarrow C$, then $A \rightarrow C$
4. **Union** : If $A \rightarrow B$ holds and $A \rightarrow C$ holds then $A \rightarrow BC$ holds
5. **Decompoisition** : If $A \rightarrow BC$ holds then $A \rightarrow B$, $A \rightarrow C$ holds
6. **Psuedotransitivity** : If $x \rightarrow y$, $wy \rightarrow z$ then $wx \rightarrow z$ then $wx \rightarrow z$

4.3 NORMAL FORMS BASED ON PRIMARY KEYS

The goal of a relational database design is to generate a set of relation schemas that allows us to store information without any redundant (repeated) data. It also allows us to retrieve information easily and more efficiently.

For this we use a approach normal form as the set of rules. These rules and regulations are known as Normalization.

Database normalization is data design and organization process applied to data structures based on their functional dependencies and primary keys that help build relational databases.

Normalization Helps :

- Minimizing data redundancy.
- Minimizing the insertion, deletion and update anomalies.
- Reduces input and output delays
- Reducing memory usage.
- Supports a single consistent version of the truth.
- It is an industry best method of tables or entity design.

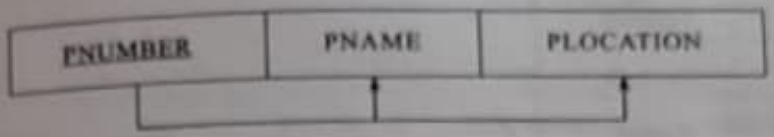
Uses : Database normalization is a useful tool for requirements analysis and data modeling process of software development. Thus

The normalization is the process to reduce the all undesirable problems by using the unctional dependencies and keys.

Since non key attribute ENAME is full FDS on primary key attribute SSN. Thus, it is in 2NF.

EP3 table:

$$SSN \rightarrow ENAME$$



$$PNUMBER \rightarrow (PNAME, PLOCATION)$$

Example - 2 : TEACHER

Course	Prof	Room	Room-Cap	Enrol-Unit
353	Vijay	A532	45	40
351	Vijay	C320	100	60
355	Santosh	H940	50	45
456	Santosh	B278	50	45
459	Gopal	D110	300	200

TEACHER Relation in Second Normal Form

T1 table :

Course	Prof	Enrol-Unt
353	Vijay	40
351	Vijay	60
355	Santosh	45
456	Santosh	45
459	Gopal	200

T2 table :

Course	Room
353	Vijay
351	Vijay
355	Santosh
456	Santosh
459	Gopal

Second Normal Form (2NF)

- 2NF is a normal form in database normalization. It requires that all data elements in a table are full functionally dependent on the table's primary key.
- If data element only dependent on part of primary key, then they are pulled out to separate tables.

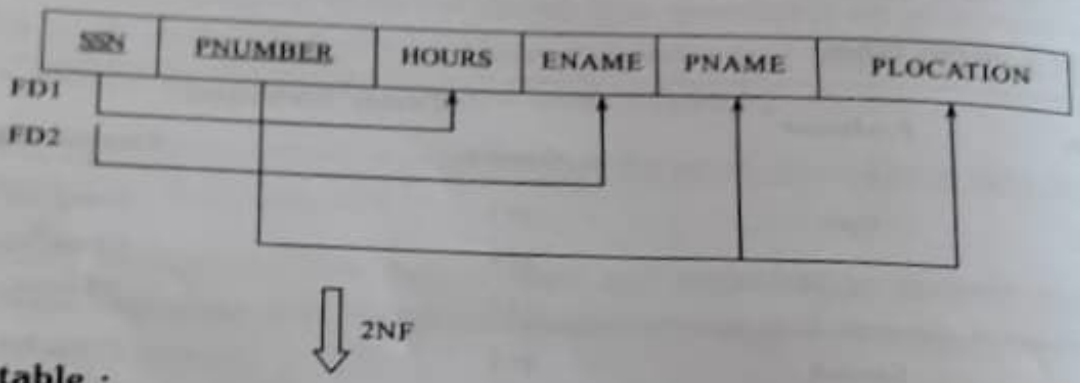
If the table has a single field as the primary key, it is automatically in 2NF.

A table is in 2NF if and only if

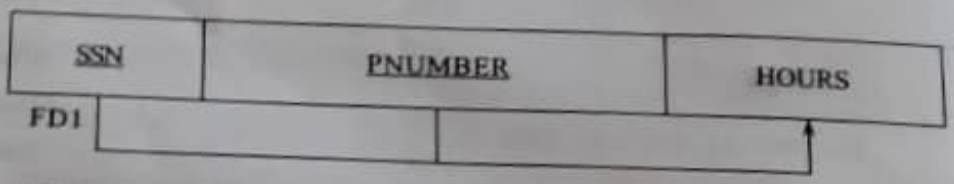
(i) It is in INF

(ii) Each non primary key attribute is full functionally dependent on the primary key

Example1: EMP-PROJ



EP1 table :



{SNN, PNUMBER} is a primary key and Hours is non key attribute.

(SSN, PNUMBER) 'HOURS

Thus HOURS is full FDS on primary key (SSN, PNUMBER)

EP2 table :

SSN is primary key and ENAME is a non key attribute.

There are two candidate keys (Stud_Id, Subject) and (SName, Subject)

In the above relation following FDS are exist:

SName, Subject → Grade

Stud_Id, Subject → Grade

Stud_Id → SName

Now BCNF decompose *R* into *R1* and *R2*.

R1

Stud_Id	Subject	Grade
10	Computer	A
10	Physics	B
10	Maths	B
20	Computer	A
20	Physics	A
20	Maths	C

R2

Stud_ID	SName
10	Vijay
20	Gopal

Ex 2: Normalize the relation professor so as it is in BCNF.

PROFESSOR

Prof. Code	Department	HOD	Present Time
P_1	Physics	Ghosh	50
P_1	Maths	Krishnan	50
P_2	Chemistry	Rao	25
P_2	Physics	Ghosh	75
P_3	Maths	Krishnan	100
P_4	Maths	Krishnan	30
P_4	Physics	Ghosh	70

T3 table :

Room	Room-Cap
353	Vijay
351	Vijay
355	Santosh
456	Santosh
459	Gopal

Third Normal Form (3NF)

The 3NF is a normal form used in database normalization to check if all the non key attributes of a relation depend only on the candidate keys of the relation.

This means that all non-key attributes are mutually independent or in other words that a non key attribute cannot be transitively dependent on another non-key attribute.

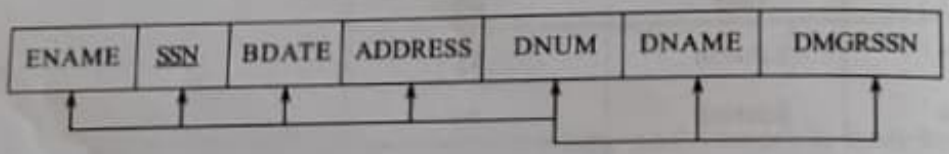
A relation schema R is in 3NF if every non prime attribute of R meets both of the following.

(i) It is in 2NF.

(ii) It is full functionally dependency on every key of *R*.

(iii) It is non transitively dependent on every key of *R*.

OR we can say : A relation schema R is in 3NF if, whenever a non trivial functional dependency



$X \rightarrow A$ holds in R.

Either

(a) X is a super key R. OR

(b) A is a prime attribute of R.

Example1: EMP-DEP

We can normalize EMP-DEP by decomposed into two 3NF said ED1 and ED2 respectively. Hence in 3NF:

The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

4.8 COMMIT, ROLLBACK, AND SAVE POINT

Transaction Control Language (TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command : COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT ;
```

ROLLBACK command : This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

SAVEPOINT command : SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

Now our class table will look like,

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

ROLLBACK TO A;

SELECT * FROM class;

Now the table will look like,

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit

4.9 CONCEPTS OF SERIALIZABILITY

A non serial schedule is said to be serializable, if it is conflict equivalent or view-equivalent to a serial schedule.

Consider a set of transactions (T1, T2, ..., Ti). S1 is the state of database after they are concurrently executed and successfully completed and S2 is the state of database after they are executed in any serial manner (one-by-one) and successfully completed. If S1 and S2 are same then the database maintains serializability.

Serializability is a widely accepted standard that ensures the consistency of a schedule. A schedule is consistent if and only if it is serializable. A schedule is said to be serializable if the interleaved transactions produces the result, which is equivalent to the result produced by executing individual transactions separately.

Example :

Transaction T_1	Transaction T_2		Transaction T_1	Transaction T_2
read (X)			read(X)	
write (X)			write(X)	
read (Y)				read(X)
write (Y)				write (X)
			read(Y)	
	read(X)		write (Y)	read (Y)
	write (X)			write (Y)
	read (Y)			
	write (Y)			

Serial Schedule Two interleaved transaction schedule

The above two schedules produce the same result, these schedules are said to be serializable. The transaction may be interleaved in any order and DBMS doesn't provide any guarantee about the order in which they are executed.

There two different types of Serializability. They are,

(i) Conflict Serializability

(ii) View Serializability

(i) **Conflict Serializability :** Consider a schedule S1, consisting of two successive instructions IA and IB belonging to transactions TA and TB refer to different data items then it is very easy to swap these instructions.

The result of swapping these instructions doesn't have any impact on the remaining instructions in the schedule. If IA and IB refers to same data item then the following four cases must be considered,

Case-1 : IA = read(x), IB = read(x),

Case-2 : IA = read(x), IB = write(x),

Case-3 : IA = write(x), IB = read(x),

Case-4 : IA = write(x), IB = write(x),

Case-1 : Here, both IA and IB are read instructions. In this case, the execution order of the instructions is not considered since the same data item x is read by both the transactions TA and TB.

Case-2 : Here, IA and IB are read and write instructions respectively. If the execution order of instructions is IA → IB, then transaction TA cannot read the value written by

S2 = <T1 T3 T2>

S3 = <T2 T3 T1>

S4 = <T2 T1 T3>

S5 = <T3 T1 T2>

S6 = <T3 T2 T1>

Taking first schedule S1 :

T1	T2	T3
Read(A)		
Write(A)		
	Write(A)	
		Write(A)

Step 1 : Final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2 : Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3 : Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

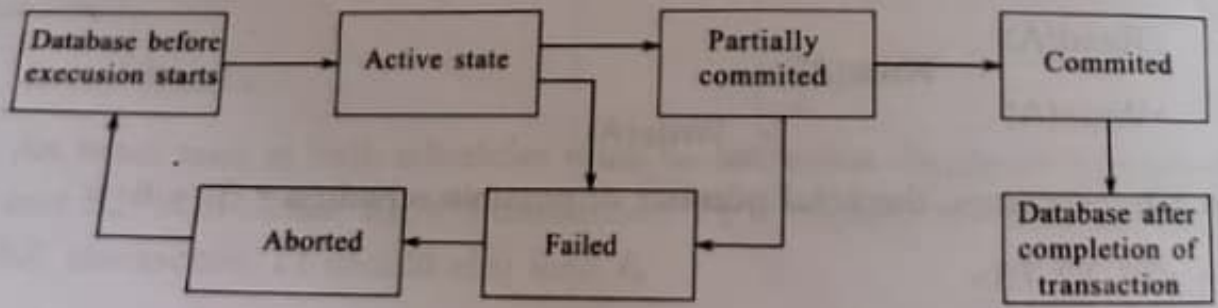
The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is :

T1 → T2 → T3

4.10 STATES OF TRANSACTIONS

A transaction must be in one of the following states as shown in Figure.



T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. **Updated Read :**

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. **Final Write :**

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example : Schedule S

T1	T2	T3
Read(A)	Write(A)	
Write(A)		Write(A)

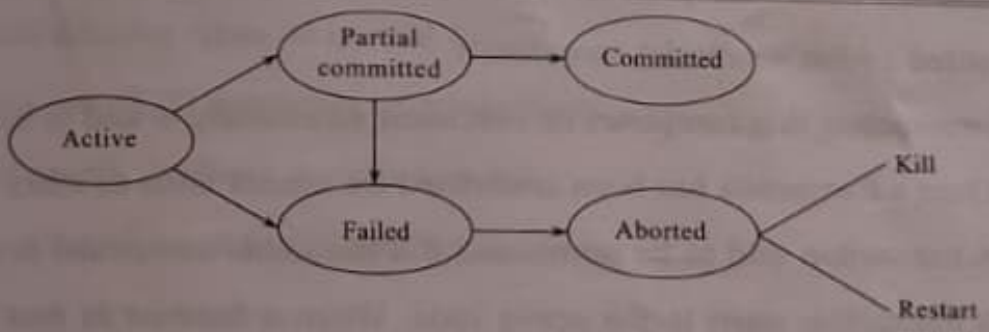
With 3 transactions, the total number of possible schedule = 3! = 6

S1 = <T1 T2 T3>

1. **Active state :** It is the initial state of transaction. During execution of statements, a transaction is in active state.
2. **Partially committed :** A transaction is in partially committed state, when all the statements within transaction are executed but transaction is not committed.
3. **Failed :** In any case, if transaction cannot be preceded further then transaction is in failed state.
4. **Committed :** After successful completion of transaction, it is in committed state.

The various states of a Database Transaction are listed below

State	Transaction types
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.



Let's study a state transition diagram that highlights how a transaction moves between these various states.

1. Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.

REVIEW QUESTIONS

One Mark Questions :

1. Define functional dependency ?
2. Define full functional dependency ?
3. Define transitive dependency.
4. Define normalization.
5. What is first Normal Form ?
6. What is second Normal Form ?
7. Define Boyce Codd Normal Form.
8. Define transaction.
9. Define Atomicity.
10. Define Consistency.
11. Define Isolation.
12. Define Durability.
13. Define serializability.

Three Mark Questions :

1. Describe the types of functional dependencies.
2. Write about different types of normal forms.
3. Write about the ACID properties of transactions.
4. Describe Commit, Rollback and Save point.
5. Write about the types of Serializabilities.
6. List the states of transaction.

Five Mark Questions :

1. Explain Informal design guidelines for relation schemas.
2. Explain 1NF with an example.
3. Illustrate 2NF with an example.
4. Explain 3NF with an example.

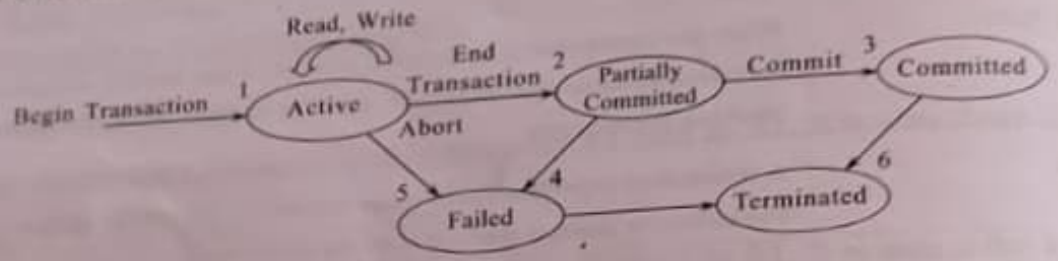
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the failed state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leaving the system.

A transaction must be in one of the following states :

Active : This state is the initial state, the transaction says in this state while it is executing.

Partial Committed : After the final statement has been executed.

Failed : Failed, after the discovery that normal execution can no longer proceed.



Aborted : Aborted, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed : After successful completion.

i.e., A transaction that completes its execution successfully is said to be committed.

- Once a transaction has been committed we cannot undo its effect by aborting it.
- A transaction said to be terminated if it has either committed or aborted.
- A transaction starts in the active state. When it finished its final statements, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted.

After swapping R(A) of T1 and W(B) of T2 we get:

T1	T2
	R(A)
	R(B)
R(A)	
	W(B)
R(B)	
W(A)	

T1	T2
	R(A)
	R(B)
	W(B)
R(A)	
R(B)	
W(A)	

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable.**

(ii) View Serializability :

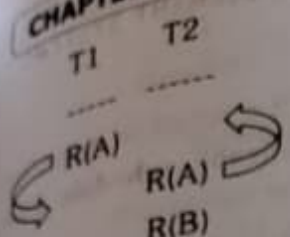
- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

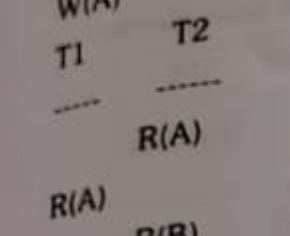
View Equivalent :

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions :

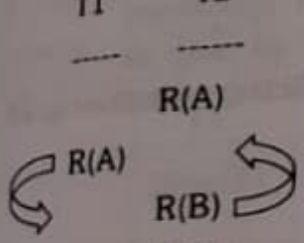
1. **Initial Read :**

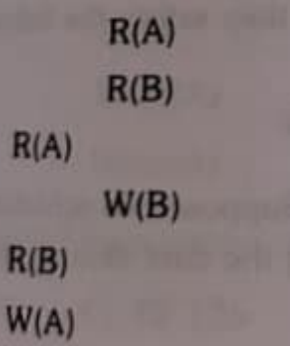
   An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.





After swapping R(A) of T1 and R(B) of T2 we get:





transaction TB in instruction IB. but order is IB → IA, then transaction TA can read the value written by transaction TB. Therefore in this case, the execution order of the instructions is important.

Case-3 : Here, IA and IB are write and read instructions respectively. If the execution order of instructions is IA → IB, then transaction TB can read the value written by transaction TA. but order is IB → IA, then transaction TB cannot read the value written by transaction TA. Therefore in this case, the execution order of the instructions is important.

Case-4 : Here, both IA and IB are write instructions. In this case, the execution order of the instructions doesn't matter. If a read operation is performed before the write operation, then the data item which was already stored in the database is read.

Conflicting operations :

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Let's see some examples to understand this :

Example-1 : Operation W(X) of transaction T1 and operation R(X) of transaction T2 are ***conflicting operations***, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operation in write operation.

Example-2 : Similarly Operations W(X) of T1 and W(X) of T2 are ***conflicting operations.***

Example-3 : Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.

Example-4 : Similarly R(X) of T1 and R(X) of T2 are ***non-conflicting operations*** because none of them is write operation.

Example-5 : Similarly W(X) of T1 and R(X) of T1 are ***non-conflicting operations*** because both the operations belong to same transaction T1.

Conflict Equivalent Schedules :

Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Conflict Serializable check :

Let's check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called **Conflict Serializable schedule**. Lets take few examples of schedules.

Example of Conflict Serializability :

Lets consider this schedule :

T1	T2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable.**

Lets take another example :

T1	T2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

Lets ***swap non-conflicting operations :***

After swapping R(A) of T1 and R(A) of T2 we get :

Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

Isolation : This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let X= 500, Y = 500.

Consider two transactions T and T".

T	T"
Read (X)	Read (X)
X: = X*100	Read (Y)
Write (X)	Z; = X + Y
Read (Y)	Write (Z)
Y : Y – 50 Write	

Suppose T has been executed till Read (Y) and then T" starts. As a result, interleaving of operations takes place due to which T" reads correct value of X but incorrect value of Y and sum computed by

T" : (X+Y = 50,000 + 500 = 50,500)

is thus not consistent with the sum at end of transaction :

T : (X+Y = 50,000 + 450 = 50,450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability : This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if system failure occurs. These updates now become permanent and are stored in a non-volatile memory. The effects of the transaction, thus, are never lost.

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using Savepoint and Rollback :

Following is the table class,

ID	Name
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');
COMMIT;
UPDATE class SET name = 'Abhijit' WHERE id = '5';
SAVEPOINT A;
INSERT INTO class VALUES(6, 'Chris');
SAVEPOINT B;
INSERT INTO class VALUES(7, 'Bravo');
SAVEPOINT C;
SELECT * FROM class;
```

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B

```
ROLLBACK TO B;
SELECT * FROM class;
```

(i) STUDI relation

Roll-No	Name	Dept.	Year
1784	Raman	Physics	1
1648	Krishnan	Chemistry	1
1768	Gopal	Maths	2
1848	Raja	Botany	2
1682	Maya	Geology	3
1485	Singh	Zoology	4

(ii) STUD2 relation

Year	Hostel_Name
1	Ganga
2	Kaveri
3	Krishna
4	Godavari

Boyce-Codd Normal Form (BCNF)

BCNF is a normal form used in database normalization. It is slightly stronger versi of the 3NF.

A table is in BCNF if and only if :

(a) It is in 3NF and

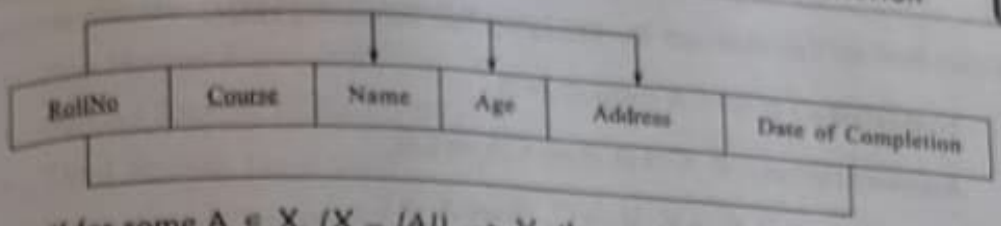
(b) For every of its nontrivial functional dependency X 'Y, X is a super key.

OR A relation schema R is in BCNF if whenever a nontrivial functional dependen X'A holds in R then

(1) X is a super key of R.

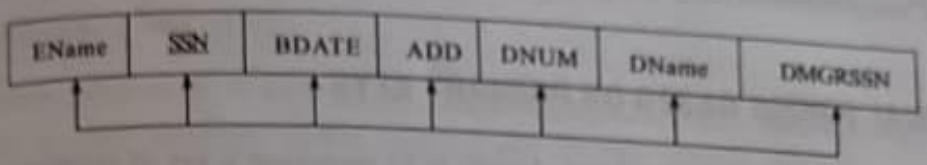
Ex1: Student

Stud_Id	SName	Subject	Grade
10	Vijay	Computer	A
10	Vijay	Physics	B
10	Vijay	Maths	B
20	Gopal	Computer	A
20	Gopal	Physics	A
20	Gopal	Maths	C



This is if for some A ∈ X, (X – {A}) → Y, then it is called partial dependency.

4. **Transitive Dependency :** A functional dependency X → Y in a relation scheme R is a transitive dependence if there is a set attributes Z that is neither a candidate key nor a subset of any key of



R and both X → Z and Z → Y hold.

Ex :

The dependency SSN → DMGRSSN is transitive through

SSN → DNUM and DNUM → DMGRSSN

5. **Multivalued Dependency :** Let R be a relation schema and let a ⊆ R and b ⊆ R. The multivalued dependency a->>b hold on R if any legal relation r (R), for all pairs of tuples *t1* and *t2* in r s.t. *t1 [a] = t2 [a], there exist tuples t3 and t4 in r s.t.*

Example : A table with schema (name, address, car)

Name	Address	Car
Vijay	Noida	Toyota
Vijay	G Noida	Honda
Vijay	Noida	Honda
Vijay	G Noida	Toyota

(name, address, car) where

name ->> address

name ->> car

Closure of functional Dependency :

The set of all functional dependency are logically implied by 'F' and the closure of 'F' is denoted by F^+

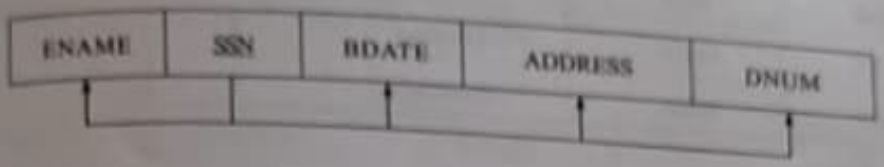
(i) ED1

(ii) ED2

The dependency SSN → DMGRSSN is transitive through the FDS

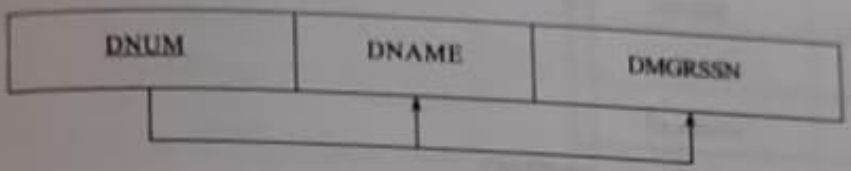
SSN → DNUM and DNUM → DMGRSSN

ED1:



Thus EMP-DEP is not in 3NF because of the transitive dependency of DMGRSSN on SSN Via DNUM.

ED2:



Example2: Student relation

Roll_No	Name	Dept.	Year	Hostel_Name
1784	Raman	Physics	1	Ganga
1648	Krishnan	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Maya	Geology	3	Krishna
1485	Singh	Zoology	4	Godavari

Here the dependency Roll-No → HOSTAL NAME is transitive through

ROLL-NO → YEAR AND YEAR → HOSTAL NAME

Thus the student Relation is not 3NF.

So we can normalize student Relation by decomposition into two 3NF, STUD1 AND STUD2 respectively.

Here we use the following normal forms :

- First Normal Form (INF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form

4.4 KNOW AND EXPLAIN GENERAL DEFINITION OF FIRST, SECOND AND THIRD NORMAL FORMS, BOYCE-CODD NORMAL FORM

First Normal Form (1NF)

A relation schema R is said to be in First Normal Form (INF) if the values in the domain of each attribute of the relation are atomic (single value).

For Ex: Course table

Fact-Dept	Professor	Course Prefrences	
		Course	Course-Dept
Comp-Sci	Vijay	353	Comp Sci
		370	Comp Sci
		310	Physics
	Santosh	353	Comp Sci
		320	Comp Sci
		370	Comp Sci
Chemistry	Gopal	456	Chemistry
		410	Mathematics
		370	Comp Sci

After converting to INF: Course table

Professor	Course	Fact-Dept	Course
Vijay	353	Comp Sci	Comp Sci
Vijay	370	Comp Sci	Comp Sci
Vijay	310	Comp Sci	Physics
Santosh	353	Comp Sci	Comp Sci
Santosh	320	Comp Sci	Comp Sci
Santosh	370	Comp Sci	Comp Sci
Gopal	456	Chemistry	Chemistry
Gopal	410	Chemistry	Mathematics
Gopal	370	Chemistry	Comp Sci

information of that employee. So, Name and Salary attributes depend upon EID attribute. Here, X is (EID) and Y is (Name, Salary)

X (EID) : Y (Name, Salary)

The determinant is EID

Suppose X has value 5 then Y has value (Mano), 9000)

A functional dependency exists when the value of one thing is fully determined by another.

For example "Given the relation

EMP (EMPNo, EmpName, Salary), attribute EmpName is functionally dependent on attribute EmpNo.

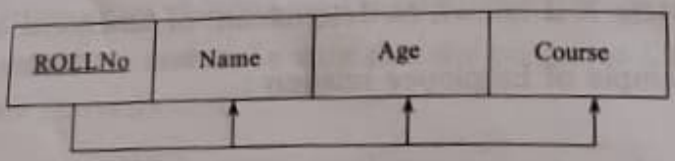
If we know EmpNo, we also know the EmpName. It is represented by

EMPNo → EmpName.

A Functional dependency is a constraint between two sets of attributes in a relation from a database.

Types of Functional Dependency :

1. **Trivial FD :** A functional dependency of the form $X \rightarrow Y$ is trivial if Y is trivial if $Y \subseteq X$.
2. **Full Functional Dependency :** A FD $X \rightarrow Y$ is a full functional dependency if removal of any



Attribute A from X means that the dependency does not hold any more. This is

Example Full/FD:

for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y.

$(X - \{A\}) \rightarrow Y$ is called full functional dependency.

3. **Partial FD :** A FD $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and then the dependency still hold.

CHAPTER

Four

NORMALIZATION AND FUNDAMENTALS OF DATA BASE TRANSACTION PROCESSING

CHAPTER OUTLINE

A single transaction may involve any number of retrieval operations and any numbe of update operations. These retrievals and updates will together form an atomic uni of work against the database. For example, a transaction to apply a bank withdrawa will typically read the user account record, check if there is a sufficient balance, an then update the record by the withdrawal amount.

A large number of commercial applications running against relational databases i **OnLine Transaction Processing (OLTP)** systems are executing transactions at rate that reach several hundred per second.

REVIEW QUESTIONS

One Mark Questions :

1. Define attribute.
2. Define relation schema.
3. Define tuple.
4. Define degree.
5. Define cardinality.
6. What is NULL value ?
7. Define relation instance.
8. Define update operation.
9. Define transaction.

Three Mark Questions :

1. Write the properties of relations.
2. List the relational model constraints.
3. Write about Referential Integrity rule.
4. Write about Domain constraints in detail.
5. Write an example to convert ER-diagram to relational schema.

Five Mark Questions :

1. Explain relational model constraints.
2. Explain relational database schema.
3. Give the steps to convert ER diagram to relational database schema/table.
4. Explain different update operation constraint violations.

4.1 INFORMAL DESIGN GUIDELINES FOR RELATION SCHEMAS

Introduction :

Normalization is based on the analysis of functional dependencies. A functional dependency is a constraint between two attributes or two sets of attributes. The purpose of the database design is to arrange the various data items into an organized structure so that it generates set of relationships and stores the information without repetition. A bad database design may result into redundant and spurious data information.

Normalization is a process for deciding which attributes should be grouped together in a relation. It is a tool to validate and improve a logical design, so that it satisfies certain constraints that avoid redundancy of data. Furthermore, Normalization is defined as the process of decomposing relations with anomalies to produce smaller well-organized relations. Thus, in normalization process, a relation with redundancy can be refined by decomposing it or replacing it with smaller relations that contain the same information, but without redundancy.

Normalization Helps :

- Minimizing data redundancy.
- Minimizing the insertion, deletion and update anomalies.
- Reduces input and output delays
- Reducing memory usage.
- Supports a single consistent version of the truth.
- It is an industry best method of tables or entity design.

Informal Design Guidelines for Relation Schemas :

There are four informal measures of quality for relation schema design. These measures are not always independent of one another. These Four informal measures are :

(i) Meaning (semantics) of the relation attributes.

(ii) Reducing the redundant (repetitive) values in tuples.

(iii) Reducing the null values in tuples.

(iv) Not allowing the possibility of generating spurious tuples.

Department :

DeptID	Location
1	Delhi
2	Hyderabad
3	Mumbai

The Insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Insert can violate any of the four types of constraints. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R). Entity integrity can be violated if any part of the primary key of the new tuple t is Null. Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation. Here are some examples to illustrate this discussion.

Operation :

Insert <NULL, 'TCS', 'Active',2> into Customer.

Result : This insertion violates the entity integrity constraint (NULL for the primary key CustomerID), so it is rejected.

Operation :

Insert <104, 'CTS', 'InActive',1> into Customer.

Result : This insertion violates the key constraint because another tuple with the same CustomerID value already exists in the Customer relation, and so it is rejected.

Operation :

Insert <106, 'Ebay','Active',4> into Customer.

Result : This insertion violates the referential integrity constraint specified on DeptID in Customer because no corresponding referenced tuple exists in Department with DeptID=4.

Operation :

Insert <106,'Ebay', 'Active',3> into Customer.

Result : This insertion satisfies all constraints, so it is acceptable.

Update Operation :

You can see that in the below-given relation table CustomerName= 'Google' is updated from Active to Inactive.

1. **SELECT :** We use this clause to specify the data or information that we requir to answer the query.
2. **FROM :** We use this clause to specify the source of data for answering th query which can be a data table, an existing query or a combination of both
3. **WHERE :** We use this clause to specify the conditions that we apply to narro down the choice of data in order to extract the data information that is desirab in the SELECT clause.

3.4 UPDATE OPERATION AND DEALING WITH CONSTRAINT VIOLATIONS

There are three basic update operations that can change the states of relations in t data-base : Insert, Delete, and Update (or Modify). They insert new data, delete o data, or modify existing data records. Insert is used to insert one or more new tuples a relation, Delete is used to delete tuples, and Update (or Modify) is used to change t values of some attributes in existing tuples. Whenever these operations are applied, t integrity constraints specified on the relational database schema should not be violat

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Update or Modify allows you to change the values of some attributes in exis tuples.

Whenever one of these operations are applied, integrity constraints specified on relational database schema must never be violated.

Insert Operation :

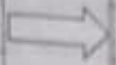
The insert operation gives values of the attribute for a new tuple which should inserted into a relation.

Consider the following relations Customer and Department

Customer :

CustomerID	Customer Name	Status	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3

Customer ID	Customer	Status Name	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3



Customer ID	Customer	Status Name	Dept ID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted. Here are some examples.

Operation :

Delete the Customer tuple with CustomerName='Apple'

Result : Acceptable.

Operation :

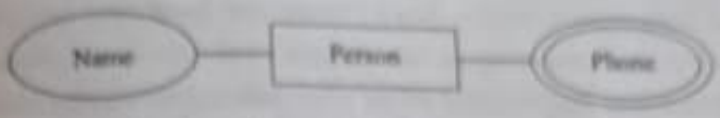
Delete the Department tuple with DeptID = 1.

Result : This deletion is not acceptable, because there are tuples in Customer relation that refer to this tuple. Hence, if the tuple in Departmet is deleted, referential integrity violations will result.

3.5 DEFINE TRANSACTION

A database application program running against a relational database typically executes one or more transactions. A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

(ii) **Multi-Valued Attributes :** A multi-valued attribute is usually represented with a double-line oval.



If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1:N relationship between the new entity and the existing one. In simple words.

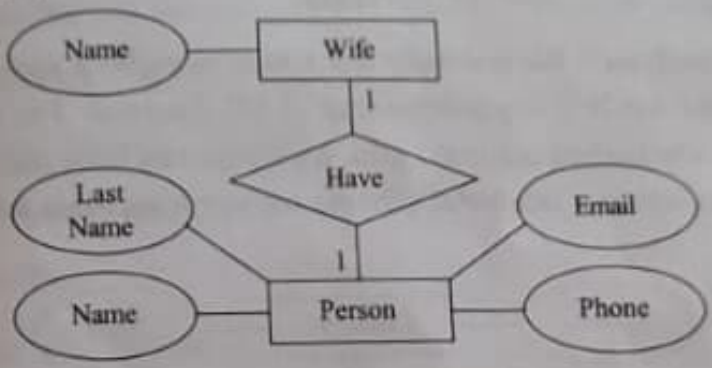
(a) Create a table for the attribute.

(b) Add the primary (id) column of the parent entity as a foreign key within the new table as shown below :

Person(personid , name, lastname, email)

Phone(phoneid , ***personid***, phone)

(iii) **1 : 1 Relationships :**



To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

Person(personid , name, lastname, email , ***wifeid***)

Wife (***wifeid*** , name)

Or vice versa to put the personid as a foreign key within the Wife table as shown below :

Person(personid , name, lastname, email)

Wife (wifeid , name, ***personid***)

Attributes have specific values in real-world scenario. For example, age can only be positive integer. The same constraints have been tried to employ on the attributes of relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Ex : create table employee

(Eid char (4),

Name char (20),

Age integer (2),

Salary integer,

primary key (Eid),

Check (age>18))

Employee :

EID	Name	Age	Salary
1	Aditya	22	10,000
2	Dinesh	18	5,600
3	Sumit	25	8,000
4	Lalit	20	ABC
5	Gaurav	23	11,000

Not allowed because age must be greater than 18

Not allowed because salary has integer data type

4. Key Constraints

In any relation R, if attribute A is primary key then A must have unique value or you can say that primary key attribute must have unique value.

Duplicate values in primary key are invalid.

Primary key or a part of it in any relation cannot be null. Suppose A be the attribute in relation R which is taken as primary key then A must not be null.

Key constraints force that -

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute cannot have NULL values.

Key constraints are also referred to as Entity Integrity Constraints.

Employee :

EID	Name	Age
1	Aditya	22
2	Sumit	19
3	Deepak	25
2	Manoj	24
4	Dheeraj	28

Invalid

5. **Tuple Uniqueness Constraints :** In any relation R, all tuples in relation R must have distinct values. In other words Duplicate tuples within a single relation are not allowed.

Employee :

EID	Name	Age
1	Aditya	22
2	Sumit	19
3	Deepak	25
2	Sumit	19

Not allowed (Duplicate tuple)

3.3 RELATIONAL DATABASE SCHEMA

A set S of relation schemas that belong to the same database is called relational database schema. S is the name of the whole database schema – S = {R1, R2... Rn} – R1, R2... Rn are the names of the individual relation schemas within the database S.

Example : Company database with 6 relation database schemas.

Employee :

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

Department :

Dname	Dnumber	Mgr_ssn	Mgr_str_date

Dept_Locations

Dnumber	Dlocation

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

These are the rules or constraints applied to the database to keep data stable, accurate or consistent. To keep database consistent we have to follow some rules known as integrity rules or integrity constraints.

Employee :

EID	Name	Salary	Dept-ID
1	Amit	6,000	1A
2	Sumit	10,000	2C
3	Lalit	15,000	4F
4	Deepak	9,000	3F
5	Sandeep	4,000	

Null value is allowed

This is not allowed because Dept-ID is a foreign key and the value 4F is not present in attribute Dept-ID of relation Department.

Department :

Dept-ID	Dept-Name
1A	Accounts
2C	Computer
3E	Electrical
4C	Civil

3. Domain Constraints

The restrictions which we applied on domain are known as domain constraints. These restrictions are applied to every value of attribute. By following these constraints you can keep consistency in database. These restrictions include data types (integer, varchar, char, time format, date format etc.), size of variable, checks (like value not null etc.) etc.

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value.
- Each attribute contains a distinct name.
- Attribute domain has no significance.
- tuple has no duplicate value.
- Order of tuple can have a different sequence.

3.2 RELATIONAL MODEL CONSTRAINTS

Every relation has some conditions that must hold for it to be a valid relation. Th conditions are called Relational Integrity or Relational Model Constraints.

1. **Entity Integrity Constraint (Integrity Rule 1)**

There must be at least one minimal subset of attributes in the relation, which identify a tuple uniquely. This minimal subset of attributes is called key for that relat If there are more than one such minimal subsets, these are called candidate keys

Employee

EID	Name	Salary	Department	
1	Amit	6,000	Accounts	
2	Sumit	10,000	Computer	
3	Lalit	15,000	Accounts →	This is not allowed because EID is a primary key
4	Deepak	9,000	Electrical	
5	Sandeep	4,000	Civil	

2. ***Referential Integrity Rule (Integrity Rule 2)***

A foreign key can be either null or it can have only those values which are present the primary key with which it is related.

Suppose A be the attribute in relation R1, which is also the primary key in relat R2, then value of A in R1 is either null or same as in relation R2.

2. **Tables** - In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

3. **Tuple** - Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as :

1	Ram	Delhi	9455123451	18

4. **Relation Schema** : A relation schema represents the name of the relation with its attributes. e.g.; Student (Roll– No, Name, Address, Phone and Age) is relation schema for Student. If a schema has more than 1 relation, it is called Relational Schema.

5. **Degree** : The number of attributes in the relation is known as degree of the relation. The Student relation defined above has degree 5.

6. **Cardinality** : The number of rows present in the table is known as cardinality of the relation. The student relation defined above has cardinality 4.

7. **Column** : Column represents the set of values for a particular attribute. The column Roll_No is extracted from relation Student.

Roll_No
1
2
3
4

8. **Relation Instance** - The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of Student at a particular time. It can change whenever there is insertion, deletion or updation in the database.

9. **Relation Key** - Every row has one, two or multiple attributes, which is called relation key.

10. **Attribute Domain** - Every attribute has some pre-defined value and scope which is known as **attribute domain**.

11. **NULL Values** - The value which is not known or unavailable is called Null value. It is represented by blank space. e.g.; Phone of Student having Roll_No 4 is Null.

3.1 RELATIONAL MODEL CONCEPTS

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some Popular Relational Database Management Systems are :

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB - Oracle
- SQL Server and Access - Microsoft

After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL etc.

Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation Student with attributes Roll_NO, Name, Address, Phone and Age shown in Table 1.

Student

Roll No	Name	Address	Phone	Age
1	Ram	Hyd	9455123451	18
2	Ramesh	Nalgonda	9652431543	18
3	Sujit	Warangal	9156253131	20
4	Suresh	Gadwal		18

Relational Model Concepts

1. **Attribute :** Each column in a Table. Attributes are the properties which define a relation. e.g.; ROLL_NO, NAME