UNIT-7

Relational Database Design

⊛. Relational Database Design Using ER-to-Relational Mapping:

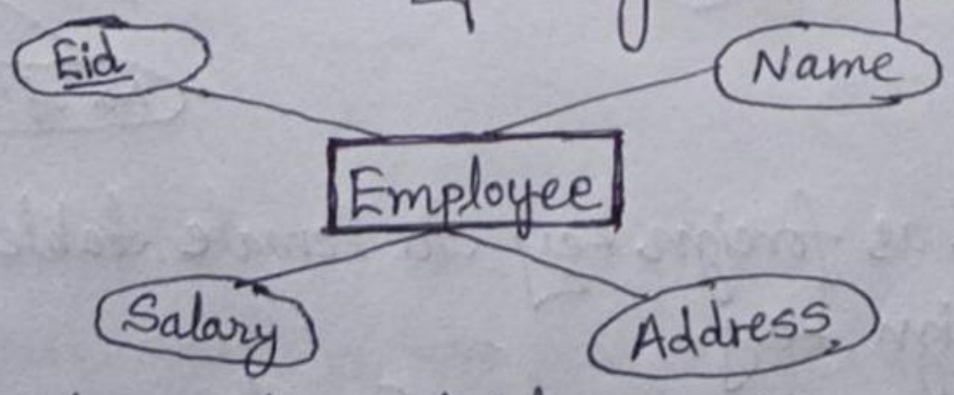
After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc. To reduce given ER diagram into tables normally we divide ER diagram into following sections:

1. Mapping Strong entity sets to ER
2. Mapping Weak entity sets to ER
3. Mapping Relation sets to ER
   - i) Mapping of Binary 1:1 relation types to ER
   - ii) Mapping of Binary 1:N relationship types to ER
   - iii) Mapping of Binary M:N relationship types to ER.
4. Mapping of multivalue attributes to ER
5. Mapping composite attributes to ER
6. Mapping of N-ary relationship types to ER
7. Mapping specialization/generalization to ER
8. Mapping Aggregation to ER.

1. Mapping Strong entity sets to ER:

→ Create table for each of the strong entity.
→ Entity's attributes should become fields of table with their respective data types.
→ Declare key attribute of strong entity set as primary key of the table.

Example: Let's take a strong entity set Employee with following attributes;



Here, we simply create a table "Employee" with fields Eid, Name, Salary and Address and make Eid as primary key of the created table as below;

Employee

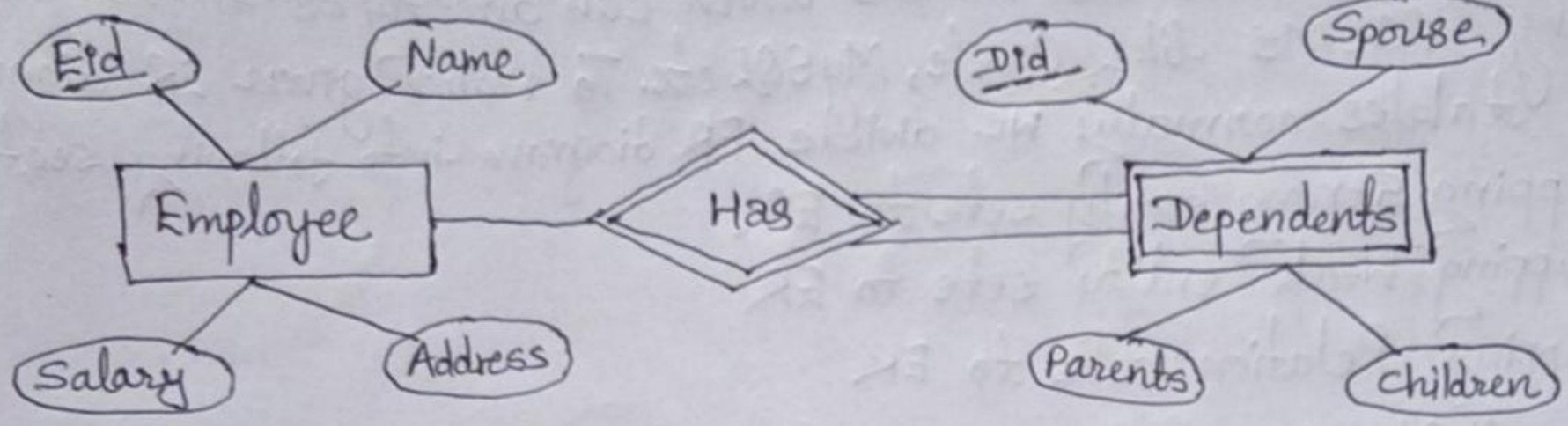
Eid	Name	Salary	Address

Scanned by TapScanner

2. Mapping Weak entity sets to ER:

→ Create table for weak entity set.
→ Add all it's attributes to table as field.
→ Add the primary key of identifying entity set.
→ Declare all foreign key constraints.

Example: Let's take weak entity set Dependents as shown in ER diagram below:



Here, to draw table of weak entity set 'Dependents' we simply set all their attributes and also set primary key of Employee to the Dependents table as below:

Employee

Eid	Name	Salary	Address

Dependents

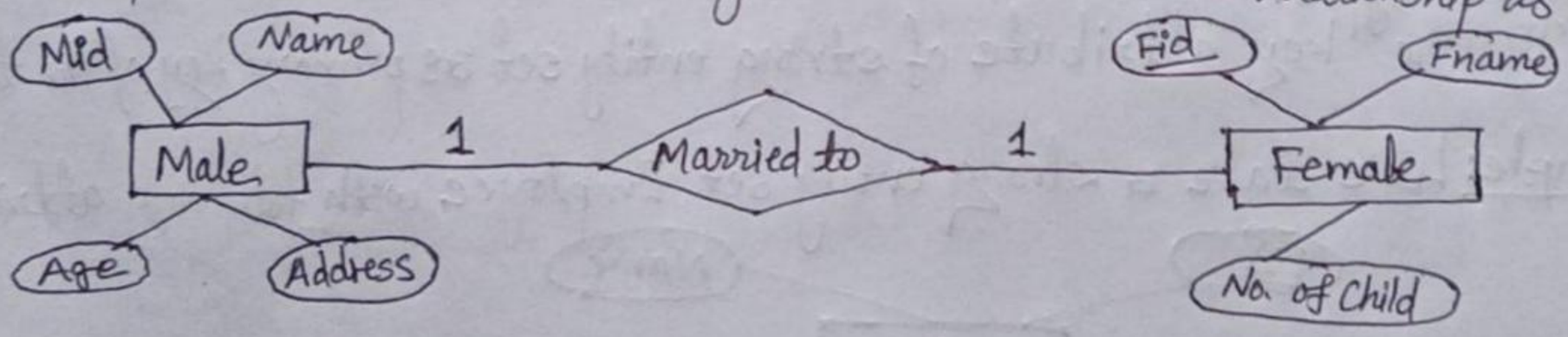
Did	Did	Parents	Spouse	Children

3. Mapping Relation sets to ER

i) Mapping of Binary 1:1 relation types to ER:

For constructing table from a binary one to one relationship we set primary key of any one of the entity set as foreign key.

Example: Let's take ER diagram with one to one relationship as below:



Here we can set Mid as foreign key to Female table or Fid to Male table as their foreign key.

Male

Mid	Name	Age	Address

Female

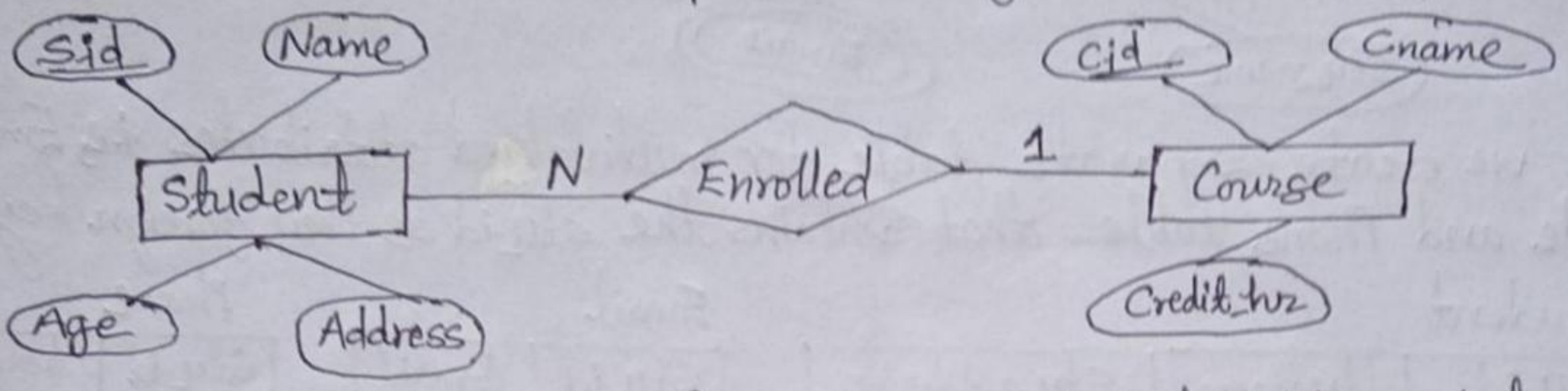
Fid	Mid	Fname	No of child

Scanned by TapScanner

ii) Mapping of Binary 1:N relationship types to ER:

For binary one-to-many relationship identify the relation that represent the participating entity type at the N-side of the relationship type and then include primary key of one side entity set into many side entity set as foreign key. Seperate relation is created for the relationship set only when the relationship set has it's own attributes.

Example: Let's take ER diagram with many to one relationship as below;



Here we can set Cid as foreign key to Student table as below;

Student

Sid	Name	Age	Address	Cid

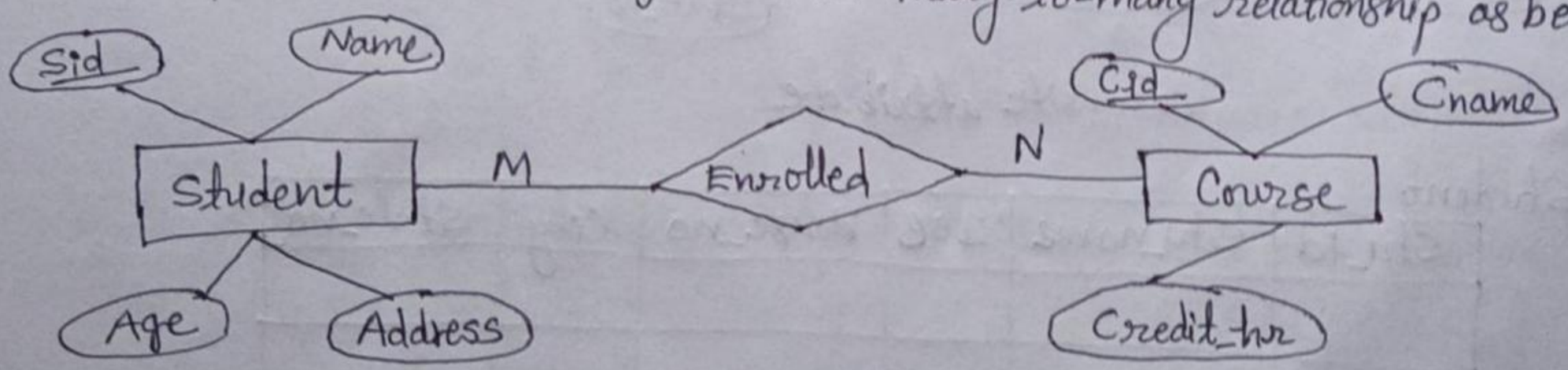
Course

Cid	Cname	Credit_hrs

iii) Mapping of Binary M:N relationship types to ER:

For a binary many-to-many relationship type, seperate relation is created for the relationship type. Primary key for each participating entity set is included as foreign key in the relation and their combination will form the primary key of the relation.

Example:- Let's take ER diagram with many-to-many relationship as below;



Here, we can create a new table Enrolled that contains the primary keys of entities student and course entities as below;

Student

Sid	Name	Age	Address

Course

Cid	Cname	Credit_hr

Enrolled

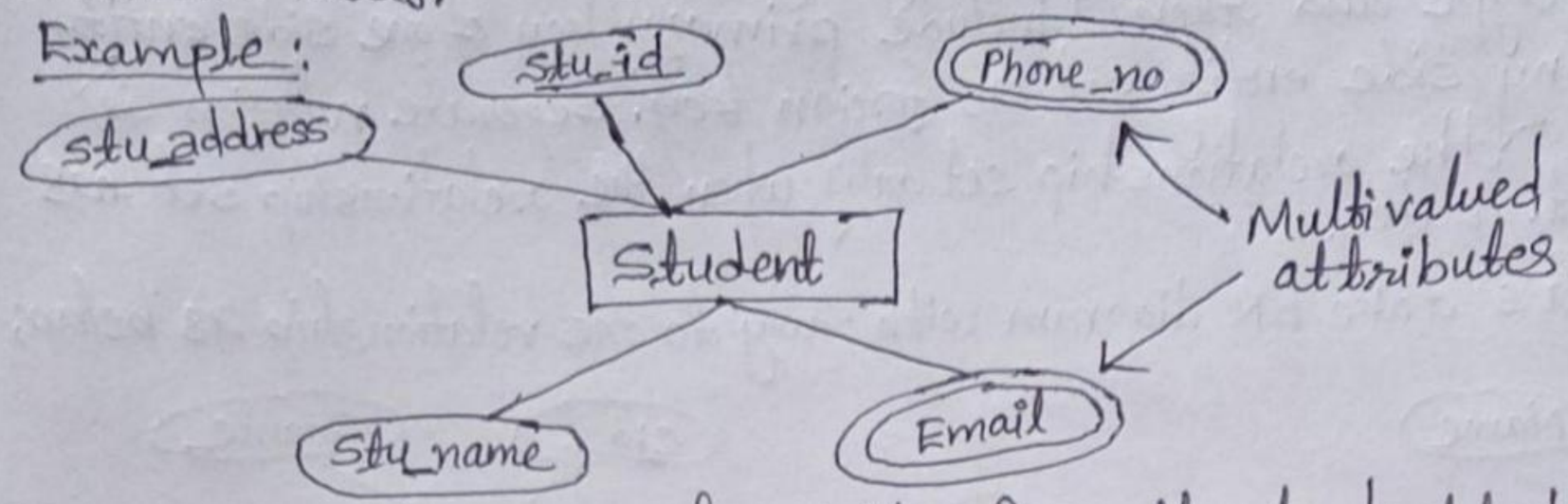
Sid	Cid

Scanned by TapScanner

4. Mapping of multivalue attributes to ER:

If an entity has multivalued attributes, separate relation is created with primary key of the entity set and multivalued attribute itself.

Example:



Here we create separate table for multivalued attributes i.e, Email table and Phone table that contains the stu_id as their foreign key.

Student

Stu_id	Stu_name	Stu_address

Email

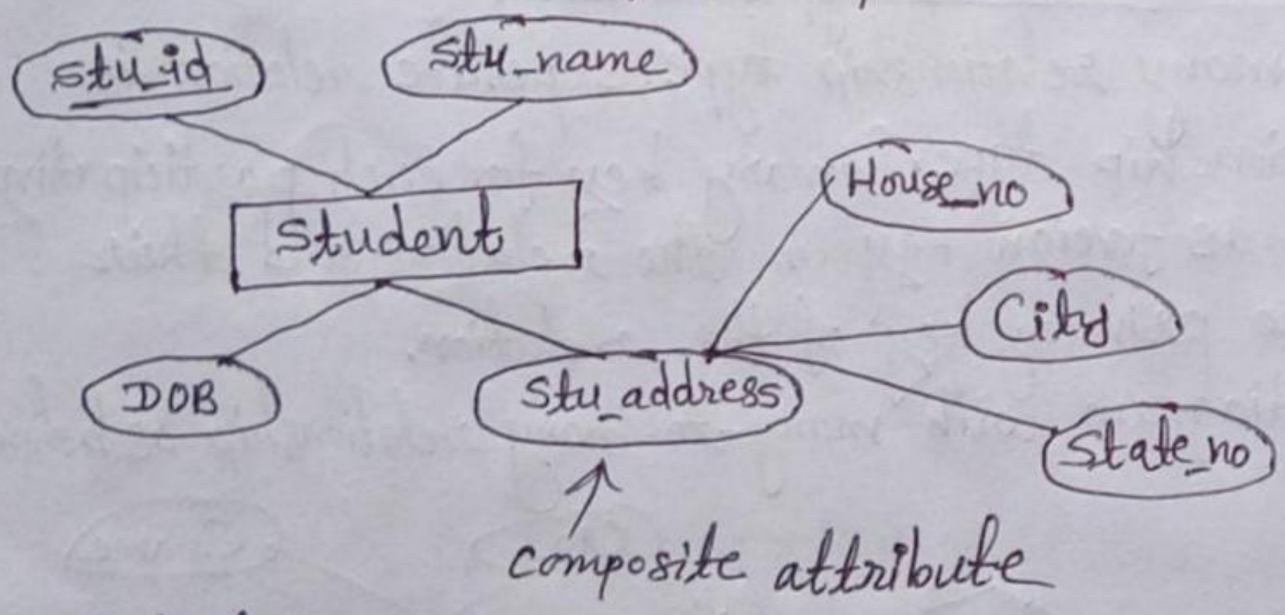
Stu_id	Email

Phone

Stu_id	Phone_no

5. Mapping composite attributes to ER:

If an entity has composite attributes, no seperate attribute (column) is created for composite attribute.



Student

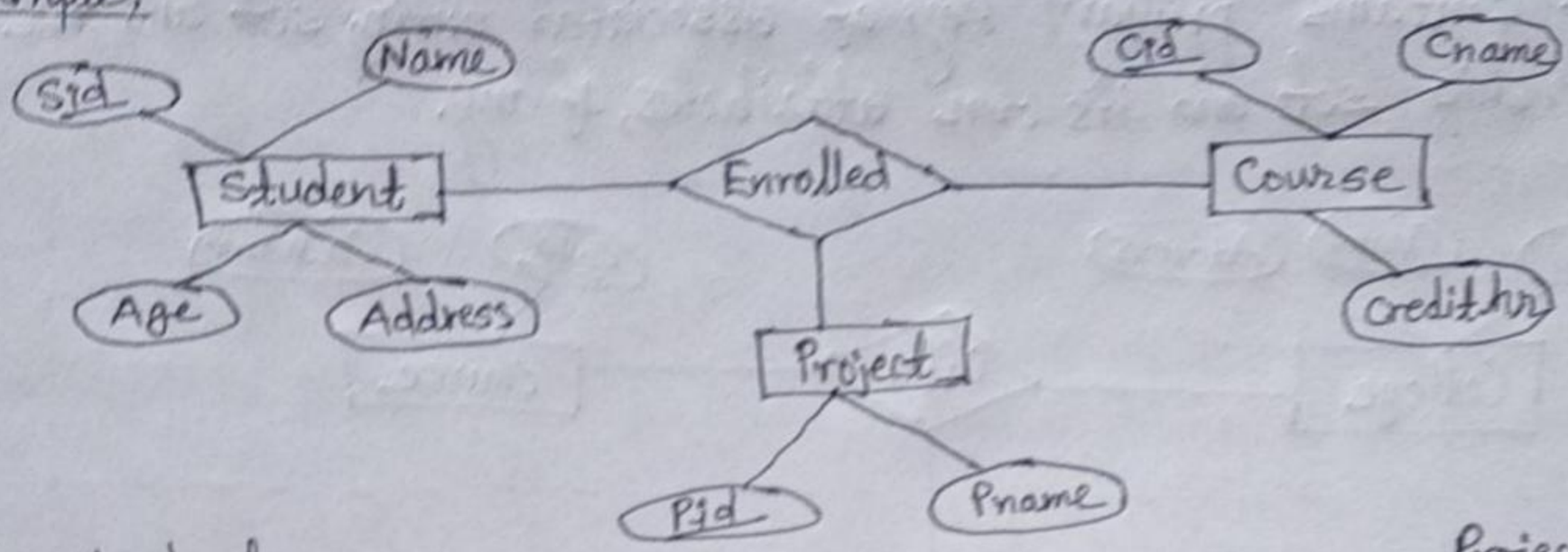
Stu_id	Stu_name	DOB	House_no	City	State_no

6. Mapping of N-ary relationship types to ER:-

For each n-ary relationship set for $n>2$, a new relation is created. Primary keys of all participating entity sets are included in the relation as foreign key attributes. Besides this all simple attributes are included as attributes of the relation.

Scanned by TapScanner

Example:



Student

Sid	Name	Age	Address

Course

Cid	Cname	credithr

Project

Pid	Pname

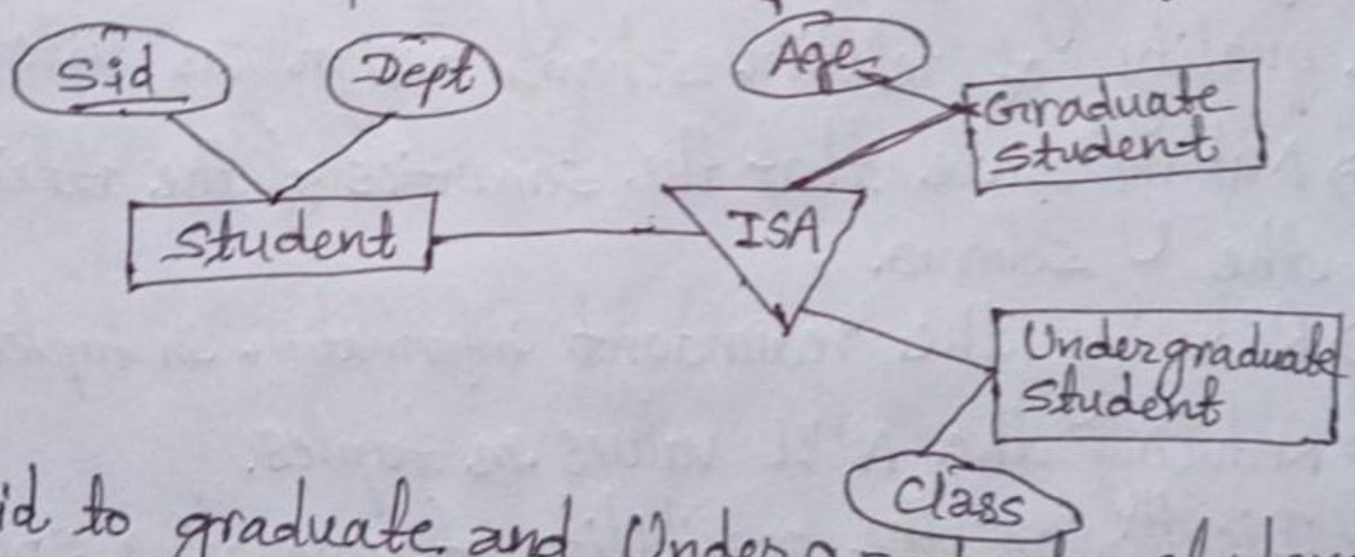
Enrolled

Sid	Cid	Pid

7. Mapping Specialization/generalization to ER:-

To construct relational table for this, we set primary key of the supper class to their sub classes as their foreign key. If subclasses are disjoint and complete then relation for a subclass entity set includes all attributes of superclass entity set and all of its own attributes.

Example:



Here we set sid to graduate and Undergraduate student tables as below;

Student

Sid	Dept

Graduate student

Sid	Age

Undergraduate students.

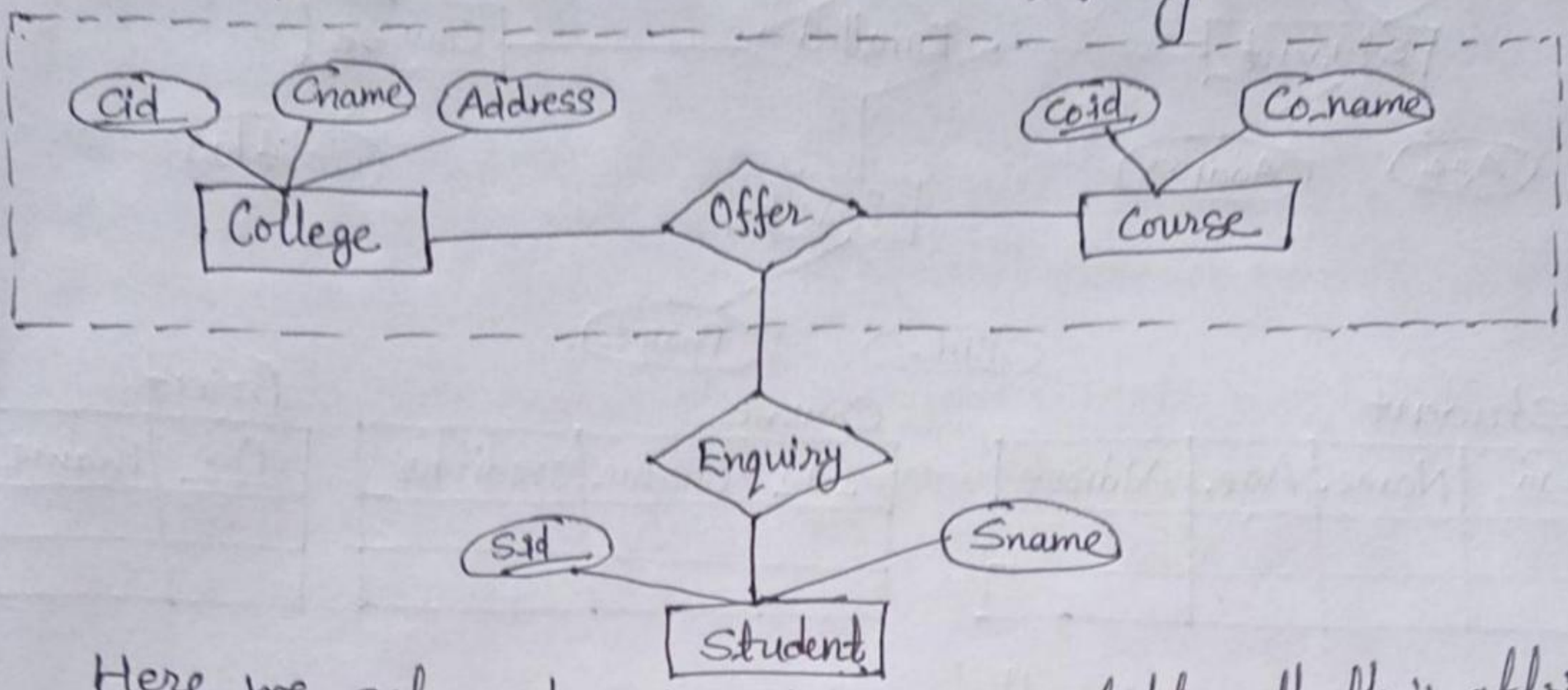
Sid	class

8. Mapping Aggregation to ER:-

In this there is no distinction between entity sets and relationship sets. in relational model. In relational model seperate relation is created for this relationship and the

Scanned by TapScanner

relation contains primary key of associated entity set and the relationship set and its own attributes, if any.



Here we set cid and coid to enquiry table with their attributes;

College

Cid	Cname	Address

Course

Coid	Co_name

Enquiry

Sid	Sname	Cid	Coid

#Informal Design Guidlines for Relational Schemas:

Informal guidlines that may be used as measures to determine the quality of relation schema design as listed below:

→ Making sure that the semantics of the attributes is clear in the schema.
→ Reducing the redundant information in tuples.
→ Reducing the NULL values in tuples.
→ Disallowing the possibility of generating spurious tuples.

⊛. Imparting clear semantics to attributes in relations;

The semantics of a relation refers to it's meaning resulting from the interpretation of attribute values in a tuple. The relational schema design should have a clear meaning. We can thus formulate the following informal design guidline.

Guidline 1: Design a relation schema so that it is easy to explain it's meaning. Do not combine attributes from multiple

Scanned by TapScanner

entity types into a single relation.

If a relation schema corresponds to one entity type or one relationship type, it is straight forward to explain it's meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

⊛ Redundant information in tuples and update anomalies:

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has a significant effect on storage space.

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These are: insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies happen;
→ When insertion of a new tuple is not done properly and will therefore can make the database become inconsistent.
→ When the insertion of a new tuple introduces a NULL value.

Deletion Anomalies happen;
→ When the insertion of a new tuple introduces a NULL value.

Modification Anomalies happen;
→ When we fail to update all tuples as a result in the change in a single one.

Guidline 2: Design the base relation schemas so that no insertion, deletion or modification anomalies are present in the relations.

If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

Scanned by TapScanner

⊛. NULL values in Tuples;

If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.

Guidline 3: As much as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only.

⊛. Generation of Spurious Tuples;

A relation in which too many attributes are grouped is called fat relation. Often, we may elect to split a "fat" relation into two relations, with the intention of joining them together if needed. However, applying a NATURAL JOIN may not yield the desired effect. On the contrary, it will generate many more tuples and we cannot recover the original table.

Guidline 4: Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated.

Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Scanned by TapScanner

#Functional Dependencies

meaning limitation बाधा

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, ..., An. If we think of the whole database being described by a single universal relation schema $R = \{A1, A2, ..., An\}$.

Definition:- A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R, such that any two tuples t1 and t2 in r that have $t1[X] = t2[X]$, they must also have $t1[Y] = t2[Y]$. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component. We say that Y is functionally dependent on X.

Functional dependency is represented as FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

Example:

Employee number	Employee Name	Salary	City
1	Bipin	50000	Butwal
2	Nisha	40000	Pokhara
3	Ram	38000	Kathmandu

In this example, if we know the value of Employee number, we can obtain Employee Name, City, Salery. By this, we can say that the City, Employee Name, and Salary are functionally dependent on Employee number.

Conditions:

→ If a constraint on R states that there can not be more than one tuple with a given X-value in any relation-instance r(R) – that is, X is a candiate key of R. If X is a candidate key of R, then $X \rightarrow R$.

→ If $X \rightarrow Y$ in R, this does not say whether or not $Y \rightarrow X$ in R.

Scanned by TapScanner

#Normal Forms Based on Primary Keys:

⊛ Normalization:

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. It can be considered as a "filtering" or "purification" process to make the design have successively better quality.

Hence normalization of data is a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of:-

→ minimizing redundancy and
→ minimizing the insertion, deletion and update anomalies.

We assume that a set of functional dependencies is given for each relation, and that each relation has a designated primary key. Each relation is then evaluated for adequacy and decomposed further as needed to achieve higher normal forms, using the normalization theory.

⊛. Normal Forms

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized. Normal forms, when considered in isolation from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is in a given normal form. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

→ nonadditive join or lossless join property
→ dependency preservation property.

Scanned by TapScanner

⊛ Practical Use of Normal Forms;

Most practical design projects acquire existing designs of databases from previous designs, designs in legacy models, or from existing files. Although several higher normal forms have been defined, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

The database designers need not normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF, for performance reason.

Denormalization → It is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

⊛ Definitions of Keys and Attributes Participating in Keys:

Superkey → A superkey of a relation schema $R=\{A1, A2, \ldots, An\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples $t1$ and $t2$ in any legal relation state r of R will have $t1[S] = t2[S]$.

Key → A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore. The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key

$K=\{A1, A2, \ldots, Ak\}$ of R, then $K-\{Ai\}$ is not a key of R for any Ai, $1 \leq i \leq k$.

Example: $\{Ssn\}$ is a key for EMPLOYEE, where $\{Ssn\}$, $\{Ssn, Ename\}$, $\{Ssn, Ename, Bdate\}$, and any set of attributes that includes Ssn are all superkeys.

Candidate key → If a relation schema has more than one key each.

Scanned by TapScanner

One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondry keys. In practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey. If ~~tabet~~ table EMPLOYEE contains {Ssn} as the only candidate key for EMPLOYEE, then it is also the primary key.

Prime attribute → An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.

Non-Prime attribute → An attribute is called nonprime if it is not a member of some candidate key of R.

⊛. First Normal Form (1NF):-

A relation is in first normal form if it does not contain any composite or multi-valued attribute. If a relation contain composite or multi-valued attribute, it violates first normal form. Simply we can say that, A relation is in first normal form if every attribute in that relation is single valued attribute.

Example: Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STU_NAME	STUD_PHONE	STUD_ADDRESS
1	RAM	9816271721 9871712717	Kathmandu
2	SURESH	9848162117	Butwal

Table 1

Conversion from table 1 to 1NF in table 2.

STUD_NO	STU_NAME	STUD_PHONE	STUD_ADDRESS
1	RAM	9816271721	Kathmandu
1	RAM	9871712717	Kathmandu
2	SURESH	9848162117	Butwal

Table 2

Scanned by TapScanner

Partial Dependency = If Proper subset of candidate key determines non-prime attribute

Second Normal Form (2NF):

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency. i.e, non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency → If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example: Consider table as below;

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	10000
2	C2	15000
1	C4	20000
4	C3	10000
4	C1	10000
2	C5	20000

Note that ~~many~~ there are many courses having the same course fee

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE-FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate {STUD_NO, COURSE_NO};
But, COURSE_NO → COURSE_FEE i.e, COURSE_FEE is dependent on COURSE_NO, which is a proper subset of candidate key.
Non-prime attribute COURSE_FEE is dependent on a proper subset of the candiate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF:

we need to split the table into two tables such as;
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1
2	C5

Table 2

COURSE_NO	COURSE_FEE
C1	10000
C2	15000
C3	10000
C4	20000
C5	20000

Note: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we do not need to store it's fee 10000 for all the 100 records, instead we can store it in the second table only once.

Third Normal Form (3NF):

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form. A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency $X \rightarrow Y$.

i) X is a super key.
ii) Y is a prime attribute (each element of Y is part of some candidate key).

Transitive dependency → If $A \rightarrow B$ and $B \rightarrow C$ are two FDs then $A \rightarrow C$ is called transitive dependency.

Example:

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

In above table;
FD set: {STUD_NO → STUD_NAME, STUD_NO → STUD_STATE, STUD_STATE → STUD_COUNTRY, STUD_NO → STUD_AGE}
Candidate key: {STUD_NO}

For this relation, STUD_NO → STUD_STATE and STUD_STATE → STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT as:

STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY).

Boyce-Coded Normal Form (BCNF):

BCNF was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.

Definition: A relation schema R is in BCNF if whenever a nontrivial function dependency $X \rightarrow A$ holds in R, then X is a superkey of R. In practice, most relation schemas that are in 3NF are also in BCNF, if for every FD, LHS is a super key.

BCNF is free from redundancy. If a relation is in BCNF, then 3NF is also satisfied. Every Binary Relation (a relation with only 2 attributes) is always in BCNF. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE Table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table functional dependencies are as follows:

EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP_ID, EMP_DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables: EMP_COUNTRY, EMP_DEPT and EMP_DEPT_MAPPING as follows:

EMP_COUNTRY Table:

EMP_ID	EMP_COUNTRY
264	India
264	India

Scanned by TapScanner

EMP_DEPT Table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional Dependencies:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now this is in BCNF because left side part of both the functional dependencies is a key.

#Properties of Relational Decomposition

1) Dependency Preservation:

If each functional dependency X→Y specified in FD appears directly in one of the relation schemas Ri in the decomposition or could be inferred from the dependencies that appear in some Ri. This is the Dependency Preservation. If a decomposition is not dependency preserving some dependency is lost in decomposition. To check this condition, take the JOIN of 2 or more relations in the decomposition.

For Example:-

R=(A,B,C)

FD={A→B, B→C}

Key={A}

R is not in BCNF.

Decomposition R1=(A,B), R2=(B,C).

R1 and R2 are in BCNF, lossless-join decomposition, Dependency preserving. Each functional dependency specified in FD either

Scanned by TapScanner

appears directly in one of the relations in the decomposition. It is not necessary that all dependencies from the relation R appear in some relation Ri.

2) Lossless Join:

Lossless join is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.

For Example:

Let R: relation, FD: set of functional dependencies on R,
X,Y: decomposition of R,

A decomposition $\{R_1, R_2, \ldots, R_n\}$ of a relation R is called a lossless decomposition for R if the natural join of R1, R2, ..., Rn produces exactly the relation R.

A decomposition is lossless if we can recover:

R(A,B,C) → Decompose → R1(A,B) R2(A,C) → Recover → R'(A,B,C)
Thus R' = R.

Decomposition is lossless if:

X intersection Y → X (i.e, all attributes common to both X and Y functionally determine all the attributes in X.)

X intersection Y → Y (i.e, all attributes common to both X and Y functionall determine all the attributes in Y.)

If X intersection Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

Multivalued Dependency

Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on third attribute. A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Scanned by TapScanner

Example: Suppose there is a bike manufacturer company which produces two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M 3001	2013	Black
M 4006	2017	White
M 4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.
In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below;

BIKE_MODEL →→ MANUF_YEAR
BIKE_MODEL →→ COLOR.

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

⊛. Concept of Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties: A relation R is in 4NF if and only if the following conditions are satisfied:

i) It should be in the Boyce-Codd Normal Form (BCNF).
ii) The table should not have any Multi-valued dependency.

A table with a multivalued dependency violates the normalization standard of 4NF, because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this upto 4NF, it is necessary to break information into two tables.

Scanned by TapScanner