Unit III

Pitfalls in Relational Database design :-

A bad design of a relation may have :
1) Repetition of information
2) Inability to represent certain information.

Consider the following eg. of a bad design of a database

Consider the following loan relation to store the info concerning loans :

Loan = (br_name, br_city, assets, cust_nm, loan_no, amt.)

fig. 4.1 The loan relation

Br_name	Br_city	Assets	cust_name	loan_no.	Amount
BOH	Aurangabad	20000	Sachine	L-17	10000
BOM	pune	32000	Rahul	L-23	20000
BOI	Mumbai	100000	Ajay	L-15	1500
ICICI	pune	3700000	Rahul	L-14	1500
SBI	Mumbai	400000	sachin	L-93	5000
HDFC	Delhi	490000	Raj	L-23	20000
city Bank	Delhi	2500000	Anil	L-25	25000
HDFC	Delhi	490000	Ramesh	L-10	22000

Insertion anomalies :

Suppose we wish to add a new loan to the database say that the loan is made by ICICI to Ramesh for an amount of Rs. 10000/- and the loan is L-30.

Thus we will add the following tuple

(ICICI, pune, 3700000, Ramesh, L-30, 10000)

to the Loan relation.

In the above case we have repeated the info assets and city for the branch ICICI ie.

deletion anomalies:
If we delete a tuple from the loan rel^n that represents customer located at a branch the details about that branch are also lost from the database.

we have 2 tuples having the following information:

(ICICI, pune, 3700000, Rahul, L-14, 1500)
(ICICI, pune, 3700000, Ramesh, L-30, 10000)

Here the assets and city information is repeating.
Thus, for insertion of each loan information the info^n concerning the assets and city are being repeated.

This repetation wastes space.

This redundancy of data also complicates the updation opeeations made on the database
eg:- if ICICI branch moves from pune to chennai.
we will have to update all the redundant entries as well.

relation anomalies:
Thus updates are more costly bcoz while updating as in this case we have to ensure that every tuple pertaining to ICICI has to be updated or else our database will show two different cities for the same branch name (ICICI) thereby resulting in an inconsistent state of the database.

Moreover suppose that there is a branch ICICI which doesn't have any loan at that branch then we cannot represent the required branch information in the loan relation.

Tuples in the loan relation require values for loanno, amount, and customer name, otherwise null values can be inserted for these attributes, but the null values are difficult to handle.

Decomposition :-

The above mentioned problems can be solved by

using decomposition.

Thus the loan relation of above fig. 1 can be decomposed into the following two relational schemas.

Branch-cust-schema (branch_name, branchcity, assets, customername)

customer_loan_schema (customer_name, loanno, amount).

Thus 2 relations will have the following contents:

Branch_cust relation:

branch_name	branch_city	assets	customer_name
BOH	Aurangabad	20000	Sachin
BOM	pune	32000	Rahul
BOI	Mumbai	100000	Ajay
ICICI	pune	3700000	Rahul
SBI	mumbai	400000	Sachin
HDFC	Delhi	490000	Raj
City Bank	Delhi	2500000	Anil
HDFC	Delhi	490000	Ramesh

The customer_loan relation:

customer_name / branch_name	loanno	amount
Sachin	L-17	10000
Rahul	L-23	20000
Ajay	L-15	1500
Rahul	L-14	1500
Sachin	L-93	5000
Raj	L-23	22000
Anil	L-25	25000
Ramesh	L-10	22000

We can reconstruct the original loan relation as follows:

branch_cust ⋈ customer_loan.

The result of this is as shown in the table below.

Table 4.2

branch_name	branch_city	assets	cust_name	loan_no name	amount
BOH	Aurangabad	20000	sachin	L-17	10000
BOH	Aurangabad	20000	sachin	L-93	5000
BOM	pune	32000	Rahul	L-23	20000
BOM.	pune	32000	Rahul	L-14	1500
BOI	mumbai	100000	Ajay	L-15	1500
ICICI	pune	3700000	Rahul	L-23	20000
ICICI	pune	3700000	Rahul	L-14	1500
SBI	mumbai	400000	sachin	L-17	20000
SBI	mumbai	400000	sachin	L-93	5000
HDFC	Delhi	490000	Raj	L-23	2200
city Bank	Delhi	250000	Anil	L-25	25000
HDFC	Delhi	490000	Ramesh	L-10	22000

Consider the query:

"Find the branch name of all branches that have made loan in amount less than Rs. 15000".

The By refering to the above table which shows the loan relation. The result of the above query are the following branch names: BOH, BOI, ICICI, SBI.

The result of the same query using the Branch_cust ⋈ Customer_loan relation is: BOH, BOM, BOI, ICICI, SBI. Here in this case there is one additional branch name ie. BOM.

Moreover, if a customer happens to have several loans from different branches, we cannot tell which loan belongs to which branch for eg. from table 4.1 We see that, sachin is having 2 loan's (L-93 and L-17).

However, when we decompose the relation of table 4.1 into the two relations - (customer_loan and Branch_cust) and after taking their join we obtain an relation of Table 4.2. In this relation we cannot make out that the loan numbered L-17 belongs to which branch as there are two entries pertaining to it ie. BOM and SBI.

Thus, when we join Branch_cust and customer_loan We obtain not only the tuples that we had originally in loan but also several additional tuples.

Athough we have more tuples in Branch_cust ⋈ customer_loan we actually have less information.

Thus we are not able to represent in the database the information about which customers are borrowers from which branch because of this loss of information we call the decomposition of the loan relation schema as a lossy decomposition or a lossy-join decomposition.

Thus a lossy-join decomposition is a bad database design

Functional Dependencies :-

Functional dependencies are constraints on the set of legal relations. It is a relationship between the attributes.

In a particular relation an attribute 'Y' is functional dependant on an attribute 'X' if every value of X in the relation has exactly one value of 'Y' in the given relation. This functional dependancy is represented as follows:

$$X \longrightarrow Y$$

which says that the attribute Y is functionally dependant on the attribute X. or we say that X attribute functionally determines the attribute Y.

Consider the relation r as shown below :-

Consider the dependency $A \rightarrow C$.

Table 4.3 Relation R

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Every value of A has exactly one value of C in the given relation

This dependancy is satisfied as there are 2 tuples which have the value a_1 for the attribute A and these tuples have the same c value i.e. c_1 for the attribute C.

Similarly, there are 2 tuples with the value a_2 for the attribute A but have the same value i.e. c_2 for the attribute C. There are no other pairs of distinct tuples that have the same A value.

The functional dependency $C \rightarrow A$ is not satisfied - consider the tuples $t_1 = (a_2, b_3, c_2, d_3)$ and $t_2 = (a_3, b_3, c_2, d_4)$. These two tuples have the same C values, c_2, but they have different A values i.e. a_2 and a_3 respectively. Hence we say that the relation r does not satisfy the functional dependancy $C \rightarrow A$. Thus, we have found a pair of tuples $t_1 \neq t_2$ such that $t_1[C] = t_2[C]$, but $t_1[A] \neq t_2[A]$.

Thus functional dependency can be formally defined as follows:

The functional dependency $\alpha \rightarrow \beta$ holds on R if, in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$ we require that $t_1[\beta]$ should also be equal to $t_2[\beta]$.

Page No:
Date: \ \

Full functional dependency:

Given a relation R and an FD $X \rightarrow Y$. Y is fully functionally dependent on X if there is no proper subset of X' of X such that X' is functionally dependent on Y.

Prime attributes and Non prime attributes:-

An attribute A in a relation schema R is a prime attribute if A is a part of any candidate key of the relation. If A is not a part of any candidate key of R, A is called a non prime attribute.

Normalization using Functional dependency.

Desirable properties of decomposition:-

decomposition must be
- Loss-less join decomposition
- Dependency preserving
- without any repetition of information.

1. loss less join decomposition:-

To have a loss less join decomposition it is necessary to determine if a decomposition is lossy or not. This can be done by using the following rule.

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R_1 and R_2 form a decomposition of R. This decomposition is a loss-less join decomposition of R if at least one of the following functional dependencies are in F^+

$$R_1 \cap R_2 \rightarrow R_1$$
$$R_1 \cap R_2 \rightarrow R_2$$

Consider again the loan schema = (br_name, br_city, assets, customer_name, loan_no, amount)

The set of functional dependencies that we require to hold on the loan schema are:

br_name ⟶ br_name, br_city, assets.
loan_no ⟶ amount, br_name.

The Loan schema as seen earlier is a bad database design. Assume we decompose it into the following two relations:

R_1 Branch_schema = (br_name, assets, br_city)
R_2 loan_info_schema = (br_name, customer_name, loan_no, amount)

since we have the F.D.

branch_name ⟶ br_city, assets,

the augmentation rule of functional dependencies implies that

br_name ⟶ br_name, br_city, assets.

Since, branch schema ∩ loan schema = {br_name}
it follows that our initial decompusition is a loss-less join decomposition.

II. Dependency preservation :-

There is another goal in relational database design - dependency preservation. When an update is made to the database, the system should be able to check that the update will not create an illegal relation - i.e. an relation which doesnot satisfy all the given functional dependencies.

Given a relation scheme R and set of functional dependencies (F) associated with it. R is decomposed into the relation schemas $R_1, R_2, \ldots R_n$ with the functional dependencies $F_1, F_2, \ldots, F_n$. Then the decomposition is dependency preserving if the closure of F' (where $F' = F_1 \cup F_2 \ldots \cup F_n$) is identical to F^+ i.e.

$$F'^+ = F^+$$

is a dependency preserving decomposition.

Eg: Let R (A, B, C, D) and

$$F = \{A \to B, A \to C, C \to D\}$$

R is decomposed into

$R_1 = (A, B, C)$ with the FD's

$F_1 = \{A \to B, A \to C\}$

$R_2 = (C, D)$ with the FD's

$F_2 = \{C \to D\}$

$$F' = F_1 \cup F_2 = \{A \to B, A \to C, C \to D\}$$

Hence $F'^+ = F^+$.

Thus the decomposition is dependency preserving and also loss-less.

eg: We consider each member of the set F of functional dependencies that we require to hold on loan schema and show that each one can be tested in at least one relation in the decomposition.

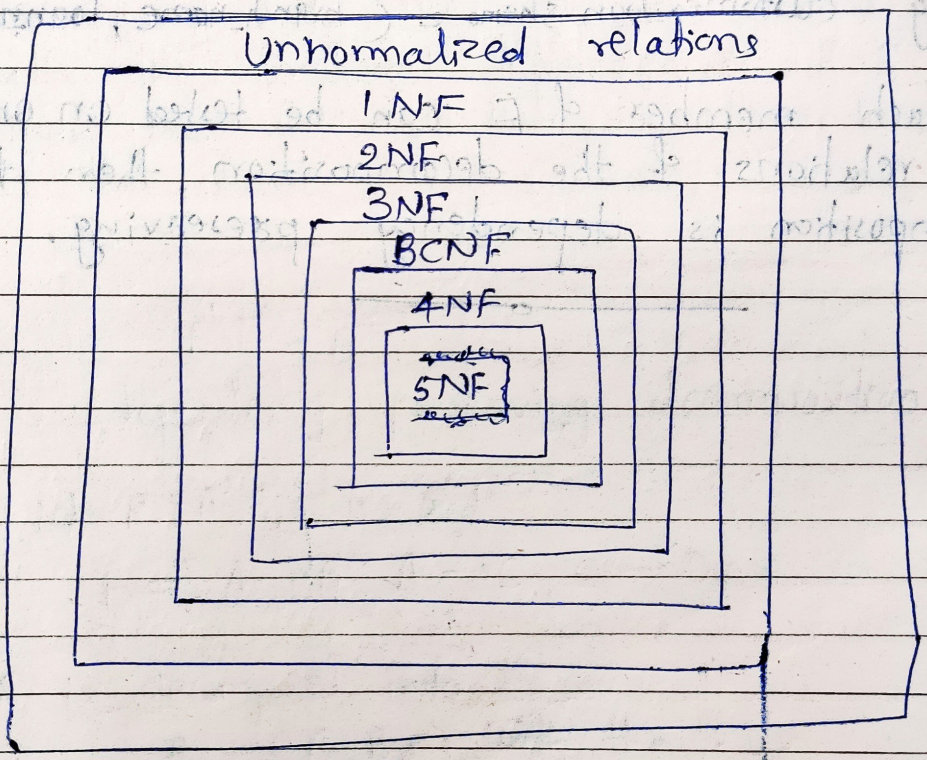
- We can test the F.D. : br_name → br_city assets using branch_cust br customer_branch = (branch_name, branch_city, assets, customer_name).
- We can test the F.D. : loan_number → amount, branch_name using customer_loan_schema = (branch_name, loan_no, amount)

If each member of F can be tested on one of the relations of the decomposition, then the decomposition is dependency preserving.

Normalization :-

The theory of normalization is developed around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints or conditions.

Many normal forms are defined on relations First normal form (1NF), 2NF, 3NF, Boyce-Codd normal form (BCNF), fourth normal form (4NF) and (5NF) are some of the normal forms defined as below. indicated in the following :-



Objectives of Normalization :-

(1) To develop "good" database design
(2) Each normal form has a number of constraints (conditions) associated with it which ensures that various types of anomalies (problems) and inconsistencies are not introduced in the database.

G.H.Raisoni College of Engineering & Management, Pune
Unit Test No.:
Name of Student:
Subject:
Class: Branch: Date: / /

Question No.	1	2	3	4	5	6	Total Marks	Signature of Examiner
Marks								

First normal form :

Def^n: "[A relation is said to be in 1NF if the values in the domain of each attribute of the relation are atomic. ie. only one value associated with each attribute.] A database system is in 1NF if every relation in the system is in 1NF."

Consider the following unnormalised relation as shown in Table a.

supplier relation

supplier_no	states	city	product information	
			product_id	qty
S_1	20	Latur	P_1	300
			P_2	200
			P_3	400
			P_4	200
			P_5	100
			P_6	100
S_2	10	pune	P_1	300
			P_2	400
S_3	10	pune		
S_4	20	Latur	P_2	200
			P_4	300
			P_5	400

This relation is unnormalized and is not in INF some of its attributes do not have atomic values.

The above table can be represented in INF as shown in table below: (b).

supplier relation

supplierno.	status	city	productno.	qty
S_1	20	Latur	P_1	300
S_1	20	Latur	P_2	200
S_1	20	Latur	P_3	400
S_1	20	Latur	P_4	200
S_1	20	Latur	P_5	100
S_1	20	Latur	P_6	100
S_2	10	pune	P_1	300
S_2	10	pune	P_2	400
S_3	10	Latur	P_2	200
S_4	20	Latur	P_2	200
S_4	20	Latur	P_4	300
S_4	20	Latur	P_5	400

The redundancies in the supplier relation lead to a variety of update anomalies ie. difficulties with update operation such as insert, delete and update operations.

(i) insert anomaly: It is difficult to enter that a particular supplier is located in a particular city until that supplier supplies at least one product. The table does not include indicate that the supplier S_5 is located at Agra because S_5 has not supplied any product. Moreover to make an insert into the relation each attribute must have at least one atomic value to satisfy the constraint in

(ii) Deletion anomaly :

If we delete a tuple for a particular supplier all the information regarding the supplier will be lost.

If a tuple of the supplies relation is deleted it deletes the product information regarding a supplier as well as the information regarding the supplies's ~~relation~~ location.

(iii) Updation :

Redundancy occues bcoz of duplication of the same data in the table. This leads to update problems.

Second Normal Form (2NF):

Def" :- "A relation R is in 2NF if and only if it is in 1NF and every non-key attribute (non prime attribute) is fully functionally dependent on the primary key"

An attribute is a non key or non prime if it does not participate in the primary key.

By splitting the supplier relation of table into supplier details and product relation we can eliminate the insertion, deletion and updation problems encountered in 1NF relation.

The decomposition of the supplier relation is as shown in following table c.

Sid, Sname → Sid Valid ~~Sid~~ is

supplier_details

supplierno	status	city
S_1	20	Latur
S_2	10	pune
S_3	10	pune
S_4	20	Latur
S_5	30	Agra

product

supplierno	productno	qty
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_2	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

delete

functional dependencies for the relations supplier details and products are as shown in fig i.e.

Thus the problems encountered in INF supplier relation are solved as follows:

(1) Insertion: Information regarding the supplier S_5 can be added into the table supplier details even if he is not supplying any product

(2) Deleting: The tuple (S_3, P_2) in the product table can be deleted without losing the information that S_3 is located in pune.

(3) Updating: Since redundancy is eliminated the problem during updation is less.

However the relations supplier_details and product causes certain problems in the 3 operations.

G.H.Raisoni College of Engineering & Management, Pune
RAISONI GROUP a vision beyond
Unit Test No.:
Name of Student:
Roll No.: Subject:
Class: Branch: Date: / /

Question No.	1	2	3	4	5	6	Total Marks	Signature of Examiner
Marks								

(1) Insertion Anomaly :
The information that a particular city has a particular status value cannot be inserted until some supplier is located in that city, because until a supplier exists there is no appropriate primary key value.

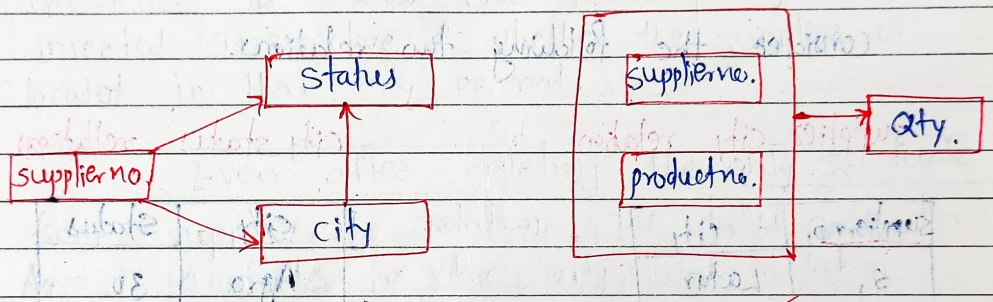


fig. 4.2

(2) Deletion Anomaly :
If a tuple of the suppliers details is deleted, then information regarding the supplier and the city is lost, i.e. if the tuple pertaining to S_5 is deleted then the status of Agra is also lost.

(3) Updation anomaly :
Redundancy still exists in the supplier details relation bcoz status for a city is still repeated in the supplier details relation. This leads to the same updation problem as in 1NF.

Third normal form (3NF)

A relation is in 3NF if and only if it is in 2NF and every nonkey attribute (nonprime attribute) is non-transitively dependent on the primary key. i.e. no attribute of table should be transitively dependent on the primary key.

Transitivity means if $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

Transitive dependency causes problems in updation. non transitive dependencies implies no mutual dependencies i.e. none of the attributes are functionally dependent on any combination of the other attributes. such independence implies each can be updated independently of all the

Consider the following two relations.

supplier_city relation

supplierno.	city
S_1	Latur
S_2	pune
S_3	pune
S_4	Latur
S_5	Agra

city_status relation

city	status
Agra	30
Latur	20
pune	10

They have the following functional dependencies

supplierno. → city

city → status

The above two relations supplier_city and city_status are both in 3NF. The primary keys for supplier_city and city_status is supplier_No. and city respectively. A relation that is in 2NF and not in 3NF can always be reduced

A relation schema R is in 3NF w.r.t. a set F of functional dependencies if for all FDs in F^+ of the form $\alpha \rightarrow \beta$ where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following ~~table~~ holds:

(1) $\alpha \rightarrow \beta$ is a trivial F.D. i.e. ($\beta \subseteq \alpha$)
(2) α is a superkey for R.
(3) Each attribute A in $\beta - \alpha$ is a candidate key for R.

Third cond^n does not say that single candidate key should contain all the attributes in $\beta - \alpha$. i.e. each attribute A in $\beta - \alpha$ may contained in a different Candidate key.

The problems that were encountered in 2NF can be solved as follows:

Insertion: The status code for any city can be inserted irrespective of whether the supplier is located in that city or not.

Deletion: Even after deleting the only S_5 tuple from supplier-city relation, the status code for Agra is available in the city-status relation.

Updating: Redundancy is completely removed and hence no updation problems.

BCNF

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it is much more strick than 3NF. i.e. every relation in BCNF is also in 3NF, however, a relation in 3NF is not necessarily in BCNF.

refer page No.

4NF.

Multivalued Dependencies – this normal form called 4NF.

The definition of fourth normal form makes use of a new kind of dependency called a multivalued dependency (MVD).

Consider an relation example to explain this, consider a relation containing information about courses teachers and text in which the attributes are course, teacher and text.

The tuples in this relation mean that the specified course can be taught by any of the specified teachers and uses all of the specified texts as references.

CTX

Course	Teacher	Text
Maths	prof. Brown	Algebra
Maths	prof. White	Algebra
Maths	prof. Brown	Geometrics
Computer	prof. Green	C
Computer	prof. Green	C++

The meaning of relation CTX is a tuple (CTX) appears in CTX, iff a course C can be taught by teacher A and uses text X. The above relation envolves a good deal of redundencies, leading to update anomalies.

For eg: to add the information that the Computer course can be taught by a new teacher, it is necessary to insert two new tuples, one for each of the two texts. We do so, by decomposing the CTX relation into 2

4NF:→ even though some relation schemas, are in BCNF, do not seem to be sufficiently normalized, i.e. they still suffer from the problem of repetition of info^n.

To deal with this problem we must define a new form of constraint called a multivalued dependency. This normal form also called as 4NF.

"Every 4NF schema is also in BCNF, but there are BCNF schemas that are not in 4NF."

G.H.Raisoni College of Engineering & Management, Pune
Unit Test No.:
Name of Student:
Roll No.: Subject:
Class: Branch: Date: / /

Question No.	1	2	3	4	5	6	Tot Mar	Signature of Examiner
Marks								

2 projections CT (Course, Teacher) and CX (Course, text) because teachers and text are completely independent of one another.

CT

Course	Teacher
Maths	prof. Braun
Maths	prof. White
Computer	prof. Green

CX

Course	Text
Maths	Algebra
Maths	Geometrics
Computer	C
Computer	C++

To add the information that the Computer can be taught by a new teacher all we have to do is insert a single tuple into relation CT.

Obviously the design of CTX is lossy and the decomposition into CT and CX is better.
CTX satisfies no FDs at all CTX is in BCNF.

Now we'll come back to our discussion on MVDs, MVDs are a generalization of FDs, i.e. every FD is an MVD but converse is false.

There are 2 MVD's in the above eg.

Course → Teacher
Course → Text

This means that attributes Teacher / Text is multi-dependent on Course attribute or the attribute Course multidetermines Teacher/Text attributes.

Sid, Sname → <.1 empty.

A course does not have a single corresponding teacher it has a well defined set of corresponding teachers. i.e for a given course C and a given text X, the set of teachers A matching the pair (C, X) in CTX depends on the value C alone.

The formal Definition of MVD is:

Let R be a relation and A, B and C be subsets of attributes of R. Then we say B is multidependent on A in symbols as

$A \rightarrow\!\!\rightarrow B$

iff in every possible legal value of R, the set of B values matching a given pair (A, C) depends only on the A value and is independent of the C value.

If we have a Relation R{A, B, C}, the MVD $A \rightarrow\!\!\rightarrow B$ holds iff the MVD $A \rightarrow\!\!\rightarrow C$ and also holds. It can be represented in one statement as,

$A \rightarrow\!\!\rightarrow B/C$.

Eg: Course $\rightarrow\!\!\rightarrow$ Teacher / Text.

The two projections (tables) CT & CX donot involve any such MVDs.

Fifth Normal form (5NF): / PJNF

Project join Normal Fo

1) The normal forms upto 4NF so far require that the given relation R if not in the normal form be decomposed in two relation
2) In some rare cases, a relation can have

problems like redundent info^{n} and update anomalies b'coz of it but cannot be decomposed in two relations to remove the problems.

3) In such cases it may be possible to decompose the relation in three or more relations using the 5NF.

4) The fifth normal form deals with join-dependencies which is a generalisation of the MVD.

5) The aim of fifth normal form is to have relations that cannot be decomposed further.

A relation in 5NF cannot be constructed from several smaller relations.

6) A relation R satisfies join dependency $(R_1, R_2, \ldots R_n)$ if and only if R is equal to the join of $R_1, R_2, \ldots R_n$ where R_i are subsets of the set of attributes of R.

7) A relation R is in 5NF (or PJNF project Join Normal Form) if for all join dependencies at least one of the following holds:

(a) $(R_1, R_2, \ldots R_n)$ is a trivial join-dependency (i.e. one of R_i is R).

(b) every R_i is a candidate key for R.

Example :- A 5NF can be provided by the eg. below that deals with departments, subjects, and students.

dept	subject	student
comp. sci.	CP1000	John
mathematics	MA1000	John
comp. sci.	CP2000	Arun
comp. sci.	CP3000	Reena
physics	PH1000	Raymond
chemistry	CH 2000	Albert

i) The above relation says that comp. sci offers subjects cp1000, cp2000 and cp3000 which are taken by a variety of students.

ii) No student takes all the subjects and no subj has all students enrolled in it and therefore all three fields are needed to represent the information.

The above relation does not show MVDs since the attributes subject and student are not independ they are related to each other and the pairing have significant info in them.

The relation can therefore not be decompos in two relations (dept, subject) and (dept, without loosing some important info.

The relation can however be decomposed in the following three relations :

(dept, subject), and
(dept, student)
(subject, student)

and now it can be shown that this decomposition is lossless.

Domain key Normal form (DKNF):-

1) DKNF is a normal form used in database normalization which requires that the database contains no constraints other than domain constraints and key constraints.

2) A domain constraint specifies the permissible values for a given attribute, while a key constraint specifies the attributes that uniquely identify a row in a given table.

3) A violation of DKNF occurs in the following table:

Wealthy person		
Wealthy person	Wealthy person type	Net worth in dollars
Steve	Eccentric Millionaire	124,543,621
Roderick	Evil Billionaire	6,553,228,893
Katrina	Eccentric Billionaire	8,829,462,998
Gary	Evil Millionaire	495,565,211

Assume that the domain for wealthy person consists of the names of all wealthy people in a pre-defined sample of wealthy people, the domain for wealthy person type consists of the values 'Eccentric Billionaire', 'Evil Millionaire', and 'Evil Billionaire', and the domain for Net worth in dollars consists of all integers greater than or equal to 1,000,000.

The constraint is neither a domain constraint nor a key constraint, therefore we can not rely on domain constraints and key constraints to guarantee that an inconsistent wealthy person type / Net worth in dollars combination does not make its way into the database.

The DKNF violation could be eliminated by altering the wealthy person Type domain to make it consist of just two values, 'Evil' and 'Eccentric'
(the wealthy person's ~~type~~ ~~domain~~ status of a millionaire or billionaire is implicit in their ~~net~~ Net worth in Dollars, so no useful information is lost.

DKNF is frequently difficult to achive in practice.

Q Define key & Explain different types of keys?
A key is the data item that exclusively identifies a record.
eg:- Account number, product code, empno and customer number are used as key fields b'coz they identify a record stored in a database

1) Super key:
A super key for an entity is a set of one or more attributes whose combined value uniquely identifies the entities in the entity set.
for eg: for an ~~attribute~~ entity set Employees, the set of attributes (emp-name, address) can be considered to be a super key, if we assume that there are no two employees with the same name emp-name and same address.

2) primary key:
The primary key of a relation can be said to be a minimal super key.
The field or group of fields which forms the unique identifier for a table is called the table's primary key

The primary key uniquely identifies each record in the table and must never be the same for two records.
for eg: emp_code can be primary key for the entity set employee.

The primary key should be chosen such that its attributes are never or very rarely changed.
eg: address field of a person should not be part of the primary key, since it is likely to change.
Emp_code, on the other hand, is guaranteed to never change, till he is in the organization.

3) Candidate key:

There is only one primary key in a table.
But there can be multiple Candidate keys.
A candidate key is an attribute or set of attributes that uniquely identifies a record.
These attributes or combination of attributes are called candidate key.

In such a case, one of the candidate key is chosen to be a primary key. The remaining candidate keys are called Alternate keys.

4) Composite key:

When a record cannot be uniquely identified by a single field, in such cases a Composite key is used.
A composite key is a group of fields that uniquely identify a record.

5) secondary key:

A secondary key is an attribute or combination of attributes that many not be a candidate key but classifies the entity set on a particular characteristics.

for eg:
In a EMPLOYEE table having the attribute department, which identifies by its value which ~~more than one~~ means all instances of EMPLOYEE who belong to a given department.

more than one employee may belong to a department attribute, so the Dept attribute is not a candidate key for the relation EMPLOYEE since it cannot uniquely identify an individual employee. However the Department attribute does identify all employees belonging to a given department.

6) Foreign key
In a relation, column whose data values correspond to the values of a key column in another relation is called a Foreign key.

In a relational database, the foreign key of a relation would be the primary key of another relation.

Functional Dependancy

$X \rightarrow Y$

X: Determinant, Y: dependant attribute

X determines Y

or

Y is determined by X.

eg: Sid → Sname

1 → Ranjit

2 → Ranjit

where Sid is student id.

are these different Ranjit we will determine from Sid

Case 1 Valid

Sid	Sname
1	Ranjit
1	Ranjit

Case 2 Valid

Sid → Sname

1 → Ranjit

2 → Ranjit

Case 3 Valid

Sid	Sname
1	Ranjit
2	Varun

Case 4 Invalid case

1 Ranjit

1 Varun

Sid → Sname

Two types of F.D.:

Trivial F.D. always valid/true. because Y is subset of X. RHS.

$X \rightarrow Y$ then Y is subset of X. RHS

eg: Sid → Sid

X Y

$X \rightarrow Y$

$LHS \cap RHS \neq \phi$ empty.

RHS subset of LHS.

Sid, Sname → Sid valid ~~Sid~~ is

ii) Nontrivial F.D.

$X \rightarrow Y \qquad X \cap Y = \phi$

Sid → Sname
Sid → Semester
Sid → phoneno
RHS ∩ Y Intesection always empty.

Properties of Functional Dependency :-

1) Reflexivity : if Y is subset of X then X → Y
Sid → Sid.

2) Augmentation : if X → Y, then XZ → YZ
extra add
Sid → Sname
Sid, phoneno → Sname, phone no.

3) Transitive : if X → Y and Y → Z then X → Z

4) Union : if X → Y and X → Z then X → YZ
X → YZ

5) Decomposition :
if X → YZ then X → Y and X → Z
XY → Z X → Z, Y → Z Cannot decom

6) pseudotransitivity :
If X → Y and WY → Z then WX → Z
extra add

7) Composition : If X → Y and Z → W then
XZ → YW.
XZ → YZ

PAGE No.
DATE / /

Finding closure method.

R(ABCD) and following FD given

FD {A → B, B → C, C → D}

A^+ = BCDA

Closure

B^+ = BCD

C^+ = CD

D^+ = D

A → A A determines A

Transitive

A → B

B → C

A → C

Ck = {A}

Candidate key

$(AB)^+$ = ABCD

AB = ck ✗

↑ superkey

eg. ② R(ABCD)

FD = {A → B, B → C, C → D, D → A}

A^+ = ABCD

B^+ = BCDA

C^+ = CDAB

D^+ = DABC

Ck = {A, B, C, D}

prime attribute = {A, B, C, D}

Non prime attribute

PAGE No.
DATE / /

X = set of attributes

X^+ → Contains set of attributes determined by X.

$A^+ = \{A, B, C, D, E\}$

$AD^+ = \{A, D, B, C, E\}$

augmentation property

$A \rightarrow ABCDE$

$AD \rightarrow BD$

$AD \rightarrow B$

$AD \rightarrow D$

$B^+ = \{B, C, D, E\}$

$CD^+ = \{C, D, E\}$

Candidate key is

Keys in DBMS.

sid	name	marks	Dept	Course
1	a	78	CS	C_1
2	b	60	EE	C_2
3	a	78	CS	C_3
4	b	60	EE	C_3
5	c	80	IT	C_2

same name then identify student with sid.
key.

{dept, Course} = key

{CS, C_1}

{name, marks} x not valid key

{name, marks, Dept, Course} x not valid key

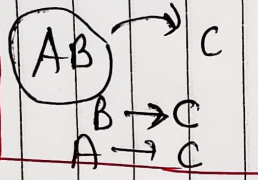
an attribute or set of attributes that uniquely identify each record of.

1st Normal form

- No multivalued attribute
- only single valued

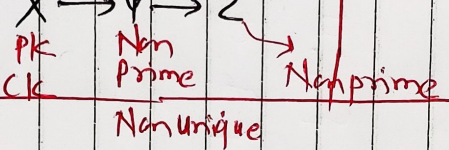
2nd Normal form

- In 1st NF
+
* No partial Dependency
* only full Dependency



3rd Normal form

* In 2nd NF
+
* No Transitive Dependancy
* No non-prime should determine non-prime



BCNF

* In 3rd NF
+
* LHS must be Ck or Sk

4th Normal form

In BCNF
+
No multivalued Dependancy
X →→ Y

5th Normal form

* In 4th NF
+
* Lossless DeCompositin

PAGE No.
DATE