LET'S START WITH DBMS :)

Intension and Extension in DataBase

Intension in Database :

The intension defines what kind of data can be stored and the relationships between them. This is basically the blueprint or definition of the database structure. It doesn't change frequently and its the permanent definition of the database structure.

It includes:

- Table definitions(name of tables, their columns, and the data types allowed in each column)
- Constraints(Rules that govern the data, such as primary keys, foreign keys, and data validation rules)
- Relationships between tables(how tables are connected through shared columns)

LET'S START WITH DBMS :)

Intension and Extension in DataBase

Intension in Database

Example:

```
customer(
id INT PRIMARY KEY,
name VARCHAR(50))
```

LET'S START WITH DBMS :)

Intension and Extension in DataBase

Extension in Database :

The extension is the actual data stored in the database at a given instance in time.
Basically the data which is stored in tuples/rows at a given instance of time.
When there are more tuples added the data can change .

Employee

id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'

Data at instance t1

Employee

id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'
4	Aditya	'Marketing'

Data at instance t2

Employee

id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'
4	Aditya	'Marketing'
5	Raj	'Finance'

Data at instance t3

LET'S START WITH DBMS :)

RDBMS

What is RDBMS ?

Database : Collection of data is called as database.
DBMS: A software application to manage our data.

LET'S START WITH DBMS :)

RDBMS

What is RDBMS ?

Relational(Use tables to store data)

MySQL
Oracle
MariaDB,

Non-relational(Data is not stored in tables)

MongoDb

LET'S START WITH DBMS :)

RDBMS

What is RDBMS ?

These databases structure data into organized tables that have predefined connections between them.

Data manipulation and querying are performed using SQL (Structured Query Language).

LET'S START WITH DBMS :)

Normalisation and its types

Normalisation

Normalization is a process in which we organize data to reduce redundancy(duplicacy) and improve data consistency. It involves dividing a database into two or more tables

What is data redundancy and consistency and why its important

When there is same set of data repeated each and every time it results in duplicacy of data (either in row or column)
Now row level duplicacy can be remove by using primary key for unique values.

LET'S START WITH DBMS :)

Normalisation and its types

Now when we have same data for some set of columns , it leads to different anomalies (inconsistencies or errors that occur when manipulating or querying data in a database)

1. Insertion Anomaly
2. Updation Anomaly
3. Deletion Anomaly

Also it also increases the size of database with the same data.

LET'S START WITH DBMS :)

Normalisation and its types

Insertion Anomaly:

It occurs when it is difficult to insert data into the database due to the absence of other required data.

Consider you want to add a new department but there is no employee in that dept yet.

Employee

id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

LET'S START WITH DBMS :)

Normalisation and its types

Deletion Anomaly:

It occurs when deleting data removes other valuable data.

Consider if you delete all the record in the table, you will loose the track of dept, their manager and salaries.

Employee

id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

LET'S START WITH DBMS :)

Normalisation and its types

Updation Anomaly:

It occurs when changes to data require multiple updates

Consider you want to change the salary for people working in HR department, you need to update it at 3 place .

Employee

id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

LET'S START WITH DBMS :)

Normalisation and its types

How normalisation helps here?

Using normalisation we can divide the employee table in two tables

1. Employee
2. Department

Employee

id	name	age	department
1	Rahul	25	IT
2	Afsara	26	HR
3	Abhimanyu	27	IT
4	Aditya	25	HR
5	Raj	24	HR

Department

department	Manager	salary
IT	Raj	1500
HR	Avinash	1000

LET'S START WITH DBMS :)

Normalisation and its types

Types of Normalisation

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form(BCNF)

LET'S START WITH DBMS :)

Denormalization

This is the opposite of normalization. It involves intentionally introducing some redundancy into a well-normalized database schema to improve query performance.

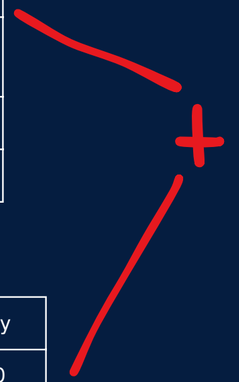
Consider if you wish to find the salary of Rahul
so first you have to make a query in employee table to find department of Rahul and then in department table to find the salary of Rahul

Employee

id	name	age	department
1	Rahul	25	IT
2	Afsara	26	HR
3	Abhimanyu	27	IT
4	Aditya	25	HR
5	Raj	24	HR

Department

department	Manager	salary
IT	Raj	1500
HR	Avinash	1000



Employee

id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

LET'S START WITH DBMS :)

Denormalization

Benifits

- **Faster Queries** : It can reduce the need for complex joins between tables during queries which can eventually improve the speed of retrieving frequently accessed data.
- **Simpler Queries**: It can simplify queries by allowing them to be executed on a single table instead of requiring joins across multiple tables.

Disadvantages

- Increased Data Redundancy
- Less Data Consistency
- Denormalization can make the database schema less flexible for future changes. like adding/modifying new data elements

LET'S START WITH DBMS :)

Functional Dependecy

Functional dependency describes the relationship between attributes in a relation.

A FD is a constraint between two sets of attributes in a relation from a database

For a Relation(table) R, if there are two attributes X and Y then

FD : X(determinant) -> Y(dependent)

Attribute Y is functionally dependent on attribute X.

R

X	Y

LET'S START WITH DBMS :)

Functional Dependecy

If x=1, we can find the value of y.
F.D : X->Y (X,Y is a subset of R)

EmpID	**EmpFirstName**	**EmpLastNmae**
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

What is subset?

Let A={1,2,3}
Let B={1,2,3,4,5}
A is a subset of B because every element of A is also an element of B.

FD: **EmpId-> EmpFirstName** (EmpFirstName is functionally dependent on EmpId)
EmpId->EmpLastNmae

LET'S START WITH DBMS :)

Functional Dependecy

Properties of Functional Dependencies:

1. Reflexivity: If Y is a subset of X, then X -> Y . (X->X)
2. Augmentation: If X -> Y, then XZ -> YZ for any Z.
3. Transitivity: If X -> Y and Y -> Z, then X -> Z.
4. Union: If X -> Y and X -> Z, then X -> YZ.
5. Decomposition: If X -> YZ, then X -> Y and X -> Z.

LET'S START WITH DBMS :)

Functional Dependecy

Types of Functional Dependency

1. Trivial dependency
2. Non-trivial dependency

EmpID	**EmpFirstName**	**EmpLastNmae**
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

LET'S START WITH DBMS :)

Functional Dependecy

Trivial dependency

A functional dependency X –> Y is trivial if Y is a subset of X

We can also say it as X–>X .

{EmpID, EmpFirstName} –> {EmpID}

is trivial because {EmpID} is a subset of {EmpID, EmpFirstName}.

X ∩ Y = Y

EmpID	**EmpFirstName**	**EmpLastNmae**
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

LET'S START WITH DBMS :)

Functional Dependecy

Non-Trivial dependency

A functional dependency X -> Y is non-trivial if Y is not a subset of X i.e X Y=
{EmpID} -> {EmpFirstName}
is trivial because {EmpFirstName} is not a subset of {EmpID}.

X ∩ Y = empty

EmpID	**EmpFirstName**	**EmpLastNmae**
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

LET'S START WITH DBMS :)

Attribute closure/closure set

Attribute closure helps us for identifying candidate keys, checking for functional dependencies, and in normalisation.

X^+ where x is an attribute or set of attribute which have all the attributes in a relation which can determine X.

Ques : Consider we have a relation R with atrributes A,B,C,D,E and FD are
A–> B, B–> C ,C–> D, D–> E

1. Now according to the rule of Reflexivity all the attributes can determine theirself.

A–>A, B–>B , C–>C , D–>D, E–>E

LET'S START WITH DBMS :)

Attribute closure/closure set

Ques : **Consider we have a relation R with atrributes A,B,C,D,E and FD are**
A–> B, B–> C ,C–> D, D–> E

1. Now according to the rule of Reflexivity all the attributes can determine theirself.
A–>A, B–>B , C–>C , D–>D, E–>E

2. Now according to the rule of transivity if A determines B and B determines C, then A can also determine C and same for all other attributes.
A–>C, A–> D, A–>E
B–>D , B–>E
C–>E

LET'S START WITH DBMS :)

Attribute closure

Ques : Consider we have a relation R with atrributes A,B,C,D,E and FD are
A–> B, B–> C, C–> D, D–> E

3. Now according to the rule of UNION if as the determinant is same we can combine dependent

For A
A–> B , A–>C, A–> D, A–>E , A–>A
A–>ABCDE

For B
B–> C , B–>D , B–>E , B–>B
B–>CDEB

For C
C–> D, C–>E , C–>C
C–>DEC

For D
D–> E , D–>D
D–>ED

For E
E–>E

LET'S START WITH DBMS :)

Attribute closure

Ques : Consider we have a relation R with atrributes A,B,C,D,E and FD are
A–> B, B–> C, C–> D, D–> E

4. Now lets find the closure set of attributes

A– {A,B,C,D,E}
B– {B,C,D,E}
C–{C,D,E}
D– {D,E}
E– {E}
AB – {A,B,C,D,E}

LET'S START WITH DBMS :)

Attribute closure

5. Now lets find candidate key, super key , prime and non-prime attributes

Super key : Set of attributes whose closure contains all the attributes given in a relation Super set of any candidate key is super key. A key(combination of all possible attributes) which can uniquely identify two tuples.

How to find Super Key?

- Identify All Attributes
- 2. Analyze the functional dependencies and find closure.
- Generate Power Set : If A has n attributes, the power set will have 2^n subsets.
- Check for Super Key Property : For each subset in the power set, check if it can uniquely identify each tuple in the relation.

LET'S START WITH DBMS :)

Attribute closure

Q. Find all the superkeys for the relation R(A, B, C) and FD : A → B, B → C.

1.Identify All Attributes:

A={A,B,C}

2. Analyze the functional dependencies and find closure.

From A → B and B → C, we can infer A → C

A+={A,B,C}

LET'S START WITH DBMS :)

Attribute closure

B+ = {B,C} because B → C, but B does not determine A.

C+ ={C}

Clsoure of A gives or determins all the attributes in the table so we can say its s auper key.

3. How to find all the super keys?

Find the power subset

The power set of {A,B,C} is 2^3=8

{}, {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}

LET'S START WITH DBMS :)

Attribute closure

A power set is the set of all subsets of a given set, including the empty set and the set itself. If you have a set X, the power set of X is denoted as 2^X

Example:

Let's take a simple set X={a,b}
The power set of X would include the following subsets:

1. The empty set: ∅
2. The single-element subsets: {a} & {b}
3. The full set itself: {a,b}

So, the power set P(X) would be: P(X)={∅,{a},{b},{a,b}}

LET'S START WITH DBMS :)

Attribute closure

4. Verify each subset to see if it is a super key

1. {}: Not a super key.
2. {A}: Super key (as its closure determines all the attributes in relation).
3. {B}: Not a super key (does not determine A).
4. {C}: Not a super key (does not determine A or B).
5. {A, B}: Super key (A is already a super key, so adding B still keeps it a super key).
6. {A, C}: Super key (A is already a super key, so adding C still keeps it a super key).
7. {B, C}: Not a super key (B determines C, but does not determine A).
8. {A, B, C}: Super key (A is already a super key, so adding B and C keeps it a super key).

LET'S START WITH DBMS :)

Attribute closure

5. Super keys are : {A}, {A, B}, {A, C}, {A, B, C}

We can also say to find max number of super keys we can use the formula

2^n-k

where
k- candidate key with k attributes (k < n) in a relation
n- total no of attributes in a relation

When 1 C.K is there = 2^n-1
When 2 C.K is there = 2^n-1(for 1st ck)+2^n-1(for 2nd ck)-2^n-2(for combination of both)

LET'S START WITH DBMS :)

Attribute closure

5. Now lets find candidate key, super key , prime and non-prime attributes

Candidate key : A superkey whose proper subset is not a super key

A set A is considered a proper subset of set B

- All elements of A are also elements of B
- Set B has at least one element that is NOT in A.

Prime Attribute: An attribute that is part of any candidate key.
Non-Prime Attribute: An attribute that is not part of any candidate key.

LET'S START WITH DBMS :)

Attribute closure

How to find the number of candidate keys?

1. Take the closure of the entire attribute set
2. Discard the dependents if their deteminants are present
3. After discarding the final combination you get is a candidate key if its subset doesnt have a super key and mention them in prime attribute
4. Now check all the functional dependencies and see if in the right hand side is there a attribute which has a determinant present in the prime attribute, if yes mention that as well in the prime attribute and replace the dependent attribute with detrminant and check if its candidate key.
5. If no prime attribute is available in teh right hand side of the functional dependency we can say there are no more candidate key and the one key we made after discarding dependent and checking for super key, that is the only candidate key.

LET'S START WITH DBMS :)

Attribute closure

Q. Find no of candidate and super key for the given relation R (A,B,C,D) and functional dependency A->B, B->C, C->D , B->A

1. Lets find the minimal superkeys
2. We will get the value of k and n ,where k- candidate key with k attributes (k < n) in a relation , n- total no of attributes in a relation
3. Use this to get the super keys in the relation 2^n-k

LET'S START WITH DBMS :)

Normalisation and its types

First Normal Form (1NF)

First normal form is the first step in the normalisation process which helps us to reduce data redudancy. Every table should have atomic values i.e there shouldn't be any multivalued attributes

It ensures the following set of rules is followed in a table:

- Atomicity(Attributes should have single valuse)
- Uniqueness of rows (Each row should be uniquely identifiable)

LET'S START WITH DBMS :)

Normalisation and its types

First Normal Form (1NF)

- Atomicity: Each column contains only indivisible (atomic) values, meaning each attribute holds a single value.

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

Here, Order is a multivalued attribute(having more than on evalue)

LET'S START WITH DBMS :)

Normalisation and its types

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

First Normal Form (1NF)

How to achieve atomicity?

1. Repeat the values in id and PersonName column twice to store single value of multivauled attribute order

ID	PersonName	Order
1	Raj	Muffin
1	Raj	Sugar
2	Riti	Muffin
3	Rahul	Sugar
3	Rahul	Egg

PK – Order+ ID

LET'S START WITH DBMS :)

Normalisation and its types

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

First Normal Form (1NF)

How to achieve atomicity?

2. Make new columns for each multivalue present.

ID	PersonName	Order1	Order2
1	Raj	Muffin	Sugar
2	Riti	Muffin	Null
3	Rahul	Sugar	Egg

PK – ID

LET'S START WITH DBMS :)

Normalisation and its types

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

First Normal Form (1NF)

How to achieve atomicity?

3. Divide the table into student(base) and order(referencing) table based on the multivalued attribute order.

pk

ID	PersonName
1	Raj
2	Riti
3	Rahul

fk

ID	Order
1	Muffin
1	Sugar
2	Muffin
3	Sugar
3	Egg

LET'S START WITH DBMS :)

Normalisation and its types

Second Normal Form (2NF)

A relation is in 2NF if it satisfies the following conditions:

1. It is in First Normal Form (1NF).
2. It has no partial dependency, which means no non-prime attribute is dependent on a part of any candidate key.

When partial dependency is there in a table?
(LHS is a proper subset of Candidate key AND RHS is a non-prime attribute)

non-prime : an attribute that is not part of any candidate key

LET'S START WITH DBMS :)

Normalisation and its types

Second Normal Form (2NF)

CustomerId	OrderId	OrderName
1	1	Muffin
2	1	Muffin
1	2	Sugar
4	2	Sugar

Candidate key : CustomerId+OrderId
Prime attribute :{CustomerId,OrderId}
Non-prime attribute : {OrderName}

In this relation OrderName is dependent on OrderId only, according to OrderId we provide the OrderName

OrderName is determined by only OrderId.

LET'S START WITH DBMS :)

Normalisation and its types

2NF

CustomerId	OrderId	OrderName
1	1	Muffin
2	1	Muffin
1	2	Sugar
4	2	Sugar

CustomerId	OrderId
1	1
2	1
1	2
4	2

OrderId	OrderName
1	Muffin
1	Muffin
2	Sugar
2	Sugar

LET'S START WITH DBMS :)

Normalisation and its types

Second Normal Form (2NF)

Consider there is a relation R(A,B,C,D) with FD : AB->C, AB->D, B->C. Find if this is in 2NF?

1.Identify the Candidate Key
A+={A}
B+={B,C}
C+={C}
D+={D}
AB+={A,B,C,D}

So, AB is a candidate key here.

LET'S START WITH DBMS :)

Normalisation and its types

Second Normal Form (2NF)

2. Check for Partial Dependencies
    1. LHS is a proper subset of Candidate key AND
    2. RHS is a non-prime attribute

FD: AB->C, AB->D, B->C
CK : AB
prime attribute : {A,B} non-prime : {C,D}

a. AB->C (fully dependent, AB is not a proper subset of candidate key)
b.AB->D (fully dependent, AB is not a proper subset of candidate key
c.B->C (partial dependency as B is a proper subset of CK and C is non-prime)

Not in 2NF.

LET'S START WITH DBMS :)

Normalisation and its types

Third Normal Form (3NF)

1. It is in Second Normal Form (2NF).
2. It has no transitive dependency, which means no non-prime attribute is transitively dependent on a candidate key.

Transitive dependent: no non-prime attribute should be dependent on another non-prime attribute

For any functional dependency X→Y, one of the following conditions must be true to be in 3rd normal form.
X is a superkey or candidate key(LHS)
Y is a prime attribute (i.e., part of some candidate key). (RHS)

LET'S START WITH DBMS :)

Normalisation and its types

Third Normal Form (3NF)

Consider there is a relation R(A,B,C,D) with FD : AB->C, C->D. Find if this is in 3NF?

1.Identify the Candidate Key

A+={A}

B+={B}

C+={C,D}

D+={D}

AB+={A,B,C,D}

So, AB is a candidate key here.

LET'S START WITH DBMS :)

Normalisation and its types

Third Normal Form (3NF)

2. Check for transitive dependency
-> no non-prime attribute should be dependent on another non-prime attribute

FD: AB->C, C->D
CK : AB
prime attribute : {A,B} non-prime : {C,D}

a. AB->C (no transitive as AB is a C.K)
b.C->D (transitive as C is not a superkey or candidate key or D is a prime attribute)

Not in 3NF.

LET'S START WITH DBMS :)

Normalisation and its types

BCNF(Boyce Codd Normal Form)

A relation is in BCNF if it satisfies the following conditions:

1. It is in Third Normal Form (3NF).
2. For a given FD X->Y should always have CK or SK, and should only determine non-prime attributes

LET'S START WITH DBMS :)

Normalisation and its types

BCNF(Boyce Codd Normal Form)

Consider there is a relation R(A,B,C,D) with FD : AB->C, AB->D. Find if this is in BCNF?

1.Identify the Candidate Key
A+={A}
B+={B}
C+={C}
D+={D}
AB+={A,B,C,D}

So, AB is a candidate key here.

LET'S START WITH DBMS :)

Normalisation and its types

BCNF(Boyce Codd Normal Form)

2. Check for C.K or S.K in LHS

FD: AB->C, AB->D
CK : AB

a. AB->C (LHS i.e AB is a CK)
b.AB->D (LHS i.e AB is a CK)

It is in BCNF.

LET'S START WITH DBMS :)

Dependency Preserving decomposition

Dependency preserving decomposition ensures that the functional dependencies are preserved/maintained after decomposing a relation into two or more smaller relations.

Consider a relation R(A, B, C) with FD : A->B, B->C, find if its dependency preserving when divided into R1(AB) and R2(BC)

- R1(AB) : A → B
- R2(BC) : B → C

The decomposition is dependency preserving because the functional dependencies A → B and B → C are preserved in R1 and R2

LET'S START WITH DBMS :)

Dependency Preserving decomposition

Consider a relation R(A, B, C, D) with FD : A → B, A → C, C → D, find if its dependency preserving when divided into R1(A, B, C) and R2(C,D)

R1(A, B, C): A → B, A → C(Functional dependency preserved)
R2(C, D): C → D(Functional dependency preserved)

The decomposition is dependency preserving because the functional dependencies A → B and B → C are preserved in R1 and R2

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

Lossy Decomposition

In normalisation we generally break/decompose the table into 2 or more tables.

Consider there is a relation R,
Lossy decomposition occurs when a relation R is decomposed or broken into two or more relations, but data is lost, and the original relation R can't be reconstructed by joining these decomposed relations.

There are 2 rules for a decomposition to be lossy

1. Some data from the original relation R is lost after decomposition
2. Join of the decomposed relations(R1, R2..Rn) is not equal to the original relation R

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

A	B	C
1	2	3
4	2	6

R

Lossy decomposition

There is a relation R(A,B,C)

Step 1: Lets decompose the relation based on any attribute and keep that attribute as common, for now lets use B as common attribute
Decomposed relations: R1(A,B) and R2(B,C)

A	B
1	2
4	2

R1

B	C
2	3
2	6

R2

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

A	B	C
1	2	3
4	5	6

R

Lossy decomposition

Step 2: Lets perform a natural join between R1 and R2
When we do R1 natural join R2 we won't get the original relation back(lossy)

A	B	C
1	2	3
1	2	6
4	2	3
4	2	6

We can see some additional tuples that were not in the original relation R (lossy decomposition)

R1 natural join R2

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

Lossy decomposition

How to ensure a decomposition is lossless

1. Divide or decompose the table on basis of CK or SK present in the relation so that there is no duplicacy
2. For a decomposition to be lossless
    a. R1 U R2= R
    b. R1$\cap$R2 = common attribute
3. To ensure that a decomposition is lossless, a common approach is to use the dependency preservation property

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

Lossless Decomposition

In normalisation we generally break/decompose the table into 2 or more tables.

Consider there is a relation R,
Lossless decomposition ensures that when a relation R is decomposed/breaked into two or more relations, no data is lost, and the original relation R can be again reconstructed by joining these decomposed relations.

There are 2 rules for a decomposition to be lossless

1. All data in the original relation R should be preserved after decomposition
2. Join of the decomposed relations(R1, R2..Rn)= original relation R

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

Lossless Decomposition

So if the table is decomposed and we want to query the attributes present in both the tables we will use the join operation.

Natural Join :

- The natural join operation combines tuples(rows) from two relations based on common attributes.
- It only includes those combinations of tuples that have the same values for the common attributes.

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

A	B	C
1	2	3
4	5	6

R

Lossless decomposition

There is a relation R(A,B,C) with CK as A

Step 1: Lets decompose the relation based on the CK or SK of the given R
Decomposed relations: R1(A,B) and R2(A,C)

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

LET'S START WITH DBMS :)

Lossy and Lossless decomposition

A	B	C
1	2	3
4	5	6

R

Lossless decomposition

There is a relation R(A,B,C) with CK as A

Step 2: Lets perform a natural join between R1 and R2
When we do R1 natural join R2 we get the original relation back(lossless)

A	B	C
1	2	3
4	5	6

R1 natural join R2

LET'S START WITH DBMS :)

Normalisation and its types

4NF(Fourth Normal Form)

A relation is in 4NF if it satisfies the following conditions:

1. It is in BCNF
2. It has no multi-valued dependencies

Multivalued dependency :

A multi-valued dependency $X \rightarrow\rightarrow Y$ $X \rightarrow\rightarrow Z$ in a relation R(X,Y,Z) implies that for each value of X, there is a set of values for Y and a set of values for Z that are independent of each other.

LET'S START WITH DBMS :)

Normalisation and its types

FourthNormal Form (4NF)

Student (StudentID, Course, PhoneNbr)

- StudentID->->Course (A student can take multiple courses)
- StudentID->->PhoneNbr (A student can have multiple PhoneNbr)

StudentId	Course	PhoneNbr
1	Math	123
1	Sci	123
1	Math	345
1	Sci	345
2	Hin	678
2	Eng	678
2	Hin	910
2	Eng	910

LET'S START WITH DBMS :)

StudentId	Course	PhoneNbr
1	Math	123
1	Sci	123
1	Math	345
1	Sci	345
2	Hin	678
2	Eng	678
2	Hin	910
2	Eng	910

4NF

StudentId	Course
1	Math
1	Sci
2	Hin
2	Eng

StudentId	PhoneNbr
1	123
1	345
2	678
2	910

LET'S START WITH DBMS :)

Normalisation and its types

5NF(Fifth Normal Form) or Project-Join Normal Form (PJNF)

A relation is in 5NF if it satisfies the following conditions:

1. It is in 4NF
2. The decomposition should be lossless.

Lossless decomposition : It ensures that when a relation R is decomposed/breaked into two or more relations, no data is lost, and the original relation R can be again reconstructed by joining these decomposed relations.

LET'S START WITH DBMS :)

Normalisation and its types

5NF(Fifth Normal Form)

A	B	C
1	2	3
4	5	6

R

There is a relation R(A,B,C) with CK as A

Step 1: Lets decompose the relation based on the CK or SK of the given R
Decomposed relations: R1(A,B) and R2(A,C)

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

LET'S START WITH DBMS :)

Normalisation and its types

5NF(Fifth Normal Form)

A	B	C
1	2	3
4	5	6

R

There is a relation R(A,B,C) with CK as A

Step 2: Lets perform a natural join between R1 and R2
When we do R1 natural join R2 we get the original relation back(lossless)

A	B	C
1	2	3
4	5	6

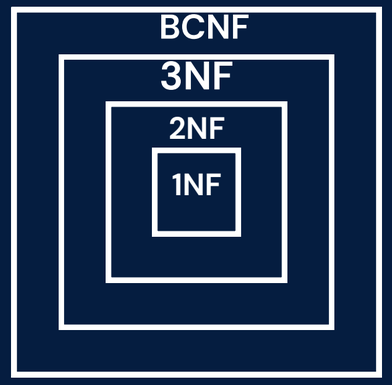
R1 natural join R2

LET'S START WITH DBMS :)

Normalisation and its types

How to find the highest Normal form of a given relation?

Step 1: Identify the candidate key for the given relation using FD and closure method.
Step 2: Find the prime and non-prime attributes.
Step 3: Start checking for normal forms one by one according to their rule



LET'S START WITH DBMS :)

Normalisation and its types

For a given relation R(A,B,C) with the following functional dependencies:
A→BC, B→C ,A→B, AB→C, B→A , find the highest normal form.

1. Find the CK for the given relation

C.K : A,B

2. Find prime and non-prime attributes

P.A={A,B}

N.P.A={C}

3. Checking for normal forms one by one according to their rule

LET'S START WITH DBMS :)

1. **First Normal Form (1NF)**

A relation is in 1NF if it contains only atomic values (no multivalued attributes). Since we are assuming our relation R is in a standard relational model, it is **already in 1NF**.

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

A relation is in 2NF if it is in 1NF and every non-prime attribute is fully functionally dependent on every candidate key of the relation(P.D ->LHS is a proper subset of Candidate key AND RHS is a non-prime attribute).

A→BC = no partial dependency (A is a CK)
B→C = no partial dependency (B is a CK)
A→B= no partial dependency (A is a CK)
AB→C= no partial dependency (AB is a combination of candidate keys, Its SK)
B→A= no partial dependency (B is a CK) , **R is in 2NF**.

LET'S START WITH DBMS :)

3. Third Normal Form (3NF)

A relation is in 3NF if it is in 2NF and no transitive dependency exists.

X->Y (X is a superkey OR Y is a prime attribute if true no transitive dependency)

A→BC = no transitive dependency (A is a CK)

B→C = no transitive dependency (B is a CK)

A→B= no transitive dependency (A is a CK)

AB→C= no transitive dependency (AB is a combination of candidate keys, Its SK)

B→A= no transitive dependency (B is a CK) , **R is in 3NF**.

LET'S START WITH DBMS :)

4. BCNF

A relation is in BCNF if it is in 3NF and for every functional dependency X→Y, X is a superkey.

A→BC = A is a CK
B→C = B is a CK
A→B= A is a CK
AB→C= AB is a combination of candidate keys, Its SK
B→A= B is a CK , **R is in BCNF**.

The highest normal form for the given relation R(A,B,C,D) is BCNF.

LET'S START WITH DBMS :)

How to normalise table

In normalisation we generally break/decompose the table into 2 or more tables.

Steps to normalize a table

1.Write down all the attributes of table, CK, Prime and non-prime attributes and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3.For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

LET'S START WITH DBMS :)

1.Write down all the attributes of table, CK, PA,NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3.For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

How to normalise table

R(A,B,C,D) and assume we have the following functional dependencies:
A→B, B→C, C->D

Step 1 : ABCD , CK-> A , Prime Attribute={A} , Non-Prime Attribute ={B,C,D}

Step 2: ABCDE, Since we are assuming our relation R is in a standard relational model, it is already in 1NF

Step 3 : Check for 2NF
A→B=(no pd as A is not a proper subset of CK and B is non prime(False and True=false))
B→C=(no pd as B is not a proper subset of CK and C is non prime (False and True=false))
C->D=(no pd as C is not a proper subset of CK and D is non prime (False and True=false))

LET'S START WITH DBMS :)

How to normalise table

1.Write down all the attributes of table, CK, PA,NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3.For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

ABCD , CK-> A , Prime Attribute={A} , Non-Prime Attribute ={B,C,D}

Step 4 : Check for 3NF

A→B=(no td as LHS is a CK)

B→C=(td is there as LHS is not CK and RHS non-prime)

C->D=(td is there as LHS is not CK and RHS non-prime)

So lets decompose the table

R1(A,B), R2(B,C), R3(C,D)

LET'S START WITH DBMS :)

How to normalise table

1.Write down all the attributes of table, CK, PA,NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3.For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

ABCD , CK-> A , Prime Attribute={A} , Non-Prime Attribute ={B,C,D}

Step 4 : Check for BCNF

R1(AB) A→B=(A is a candidate key OR a super key, so R1 is in BCNF)

R2(BC) B→C=(B is a candidate key OR a super key, so R2 is in BCNF)

R3(CD) C->D=(C is a candidate key OR a super key, so R3 is in BCNF)

Now, all decomposed relations R1, R2, and R3 are in BCNF

LET'S START WITH DBMS :)

Minimal cover of Functional Dependency

Why Do We Need to Find Minimal Cover?

It is a simplified version of the original set of functional dependencies

1. It helps to remove redundant functional dependencies.
2. It reduces the complexity of the functional dependencies.
3. It ensures that there are no unnecessary dependencies, which can lead to anomalies in database operations (insertion, deletion, and update).

LET'S START WITH DBMS :)

Minimal cover of Functional Dependency

How to Find Minimal Cover?

Step 1: Decompose FDs (RHS) i.e X->AB can be written as X->A, X->B

Step 2: Remove Redudant FD.
a. Make a new FD set excluding the one you feel is redudant
b. Now find the closure of LHS from the rest of the FD and see if it determines all the attributes of a table, if yes you can remove that, if no jump to the next one.

Step 3: Remove unnecessary attributes from LHS, if the determinant is a super key, it can be reduced to CK (minimal super key)

LET'S START WITH DBMS :)

Find Minimal Cover FD: A→BC , B→C, A→B, AB→C

Step 1 : A->B, A->C, B->C, A->B, AB->C
FD: A->B, A->C, B->C, AB->C

Step 2 :

1. For **A->B**

FD: A->C, B->C, AB->C
A+={A,C} since A+ doesn't have all the attributes we shouldn't discard this

2. For **A->C**

FD: A->B, B->C, AB->C
A+={A,B,C}, since A+ have all the attributes we can discard this

Step 1: Decompose FDs (RHS) i.e X->AB can be written as X->A, X->B

Step 2: Remove Redudant FD.
a. Make a new FD set excluding the one you feel is redudant
b. Now find the closure of LHS from the rest of the FD and see if it determines all the attributes of a table, if yes you can remove that, if no jump to the next one.

Step 3: Remove unnecessary attributes from LHS, if the determinant is a super key, it can be reduced to CK (minimal super key)

3. For **B->C**

FD: A->B, AB->C
B+={B},since B+ doesn't have all the attributes we shouldn't discard this

4. For **AB->C**

FD: A->B, B->C
AB+={A,B,C} since AB+ have all the attributes we can discard this

Therefore, the minimal cover of the given functional dependencies is:
{A→B,B→C}

LET'S START WITH DBMS :)

Equivalence of Functional Dependencies

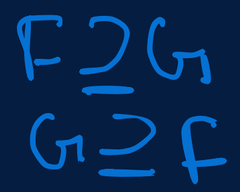
Equivalence of Functional Dependencies

Functional Dependency: A functional dependency $X \rightarrow Y$ holds on a relation R if, for any two tuples t1 and t2 in R, whenever $t1[X]=t2[X]$, then $t1[Y]=t2[Y]$

Two sets of functional dependencies, F and G, are equivalent if the following conditions hold:

1. F implies G : Every functional dependency in G can be derived from F.
2. G implies F: Every functional dependency in F can be derived from G.

If both conditions are true, then we say that F and G are equivalent.



LET'S START WITH DBMS :)

Equivalence of Functional Dependencies

How to find if two FD are equivalent

Step 1: Compute the Closure of both the sets
Step 2: Ensure that every functional dependency in set1 is in set2 closure
Step 3: Ensure that every functional dependency in set2 is in set1 closure
Step 4: If both subset checks pass, then set1 and set2 are equivalent.

LET'S START WITH DBMS :)

Step 1: Compute the Closure of both the sets
Step 2: Ensure that every functional dependency in set1 is in set2 closure
Step 3: Ensure that every functional dependency in set2 is in set1 closure
Step 4: If both subset checks pass, then set1 and set2 are equivalent.

Equivalence of Functional Dependencies

Consider two sets of functional dependencies: F={A→B,B→C} , G={A→C,A→B}
Check if F and G are equivalent

F={A→B,B→C}
Closure of F, attributes -> A,B,C
A+={A,B,C}
B+={B,C}
C+={C}

G={A→C,A→B}
Closure of G, attributes -> A,B,C
A+={A,C,B}
B+={B}
C+={C}

F and G are not equivalent because B→C is not implied by G