Database Normalization | Introduction

Database normalization is the process of organizing the attributes of the database to reduce or eliminate **data redundancy (having the same data but at different places).**

Problems because of data redundancy

Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete and update operations.

Functional Dependency

Functional Dependency is a constraint between two sets of attributes in a relation from a database. A functional dependency is denoted by arrow (→). If an attributed A functionally determines B, then it is written as A → B.

For example, employee_id → name means employee_id functionally determines the name of the employee. As another example in a time table database, {student_id, time} → {lecture_room}, student ID and time determine the lecture room where the student should be.

What does functionally dependent mean?

A function dependency A → B means for all instances of a particular value of A, there is the same value of B.

For example in the below table A → B is true, but B → A is not true as there are different values of A for B = 3.

A B

---

1 3

2 3

4 0

1 3

4 0

Trivial Functional Dependency

X → Y is trivial only when Y is subset of X. Examples

ABC → AB

ABC → A

ABC → ABC

Non Trivial Functional Dependencies

X → Y is a non trivial functional dependencies when Y is not a subset of X. X → Y is called completely non-trivial when X intersect Y is NULL. Examples:

Id → Name,

Name → DOB

Semi Non Trivial Functional Dependencies

X → Y is called semi non-trivial when X intersect Y is not NULL. Examples:

AB → BC,

AD → DC

Database Normalization | Normal Forms

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

1. First Normal Form –

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- **Example 1 –** 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.

Example 2 –

ID Name Courses

---

1 A c1, c2

2 E c3

3 M C2, c3

In the above table Course is a multi valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

ID Name Course

---

1 A c1

1 A c2

2 E c3

3 M c1

3 M c2

2. Second Normal Form –

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 iff it has **No Partial Dependency,** i.e.**,** no 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.

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

Table 3

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

- **Example 1 –** In relation STUDENT_COURSE given in Table 3,
- FD set: {COURSE_NO->COURSE_NAME}
- Candidate Key: {STUD_NO, COURSE_NO}

In FD COURSE_NO->COURSE_NAME, COURSE_NO (proper subset of candidate key) is determining COURSE_NAME (non-prime attribute). Hence, it is partial dependency and relation is not in second normal form.

To convert it to second normal form, we will decompose the relation STUDENT_COURSE (STUD_NO, COURSE_NO, COURSE_NAME) as :

STUDENT_COURSE (STUD_NO, COURSE_NO)

COURSE (COURSE_NO, COURSE_NAME)

Note – This decomposition will be lossless join decomposition as well as dependency preserving.

- **Example 2 –** Consider following functional dependencies in relation R (A, B , C, D)

  AB -> C [A and B together determine C]

  BC -> D [B and C together determine D]

  In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form –

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes is it is in second normal form.

A relation is in 3NF iff **at least one of the following condition holds** in every non-trivial function dependency X –> Y

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

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 –** In relation STUDENT given in Table 4,
  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE, STUD_STATE -> STUD_COUNTRY}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as: STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 –** Consider relation R(A, B, C, D, E)
  A -> BC,

CD -> E,
B -> D,
E -> A

All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) –

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

- **Example 1 –** Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}

  Step 1. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

  Step 2. Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

  Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

  The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

  The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

  So the highest normal form of relation will be 2nd Normal form.

- **Example 2 –**For example consider relation R(A, B, C)

  A -> BC,

  B ->

  A and B both are super keys so above relation is in BCNF.

  Key Points –

3. BCNF is free from redundancy.
4. If a relation is in BCNF, then 3NF is also also satisfied.
5. If all attributes of relation are prime attribute, then the relation is always in 3NF.
6. A relation in a Relational Database is always and at least in 1NF form.
7. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF. 8. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible). 9. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
10. 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.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC --> D

CD --> AE

Important Points for solving above type of question. **1)** It is always a good idea to start checking from BCNF, then 3 NF and so on. **2)** If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.

Candidate keys in given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.
3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so relation is not in 3NF. **2NF:** In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2 NF. So, the highest normal form is 1 NF.

DBMS | Minimum relations satisfying 1NF

The design of database proceeds in a following way:

1. Talk to the stakeholder for which we are designing the database. Get all the requirements, what attributes need to be stored and establish functional dependencies over the given set of attributes.

2. Draw an Entity-Relationship diagram on the basis of requirements analysis. 3. Convert the ER-diagram into relational model and finally create these relations into our database with appropriate constraints.

We pretty much know, how to draw an ER-diagram. But, when we ask questions like how many minimum relations would be required satisfying first normal form (1NF), it is little confusing sometimes. We establish certain simple rules which are formed after deep analysis of each case and hence, could be used directly.

Note – It is not encouraged to mug up the rules, rather understanding the logic behind each case would help you remember them easily and for long time.

Algorithm –

1. If there is total participation on both sides;
   Merge the two entities involved and the relationship into 1 table.
2. Else if, one side is total participation and one side is partial;
   - **M:N** Merge the relationship on total participation side.
   - **1:N** Merge the relationship on total participation side.
   - **1:1** Merge the two entities involved and the relationship into 1 table.
3. else if, both sides are partial participation;
   - **M:N** Separate table for each entity as well as relationship. Hence, 3 tables.
   - **1:N** Merge the relationship on N-side using foreign key referencing 1-side.
   - **1:1** Merge the relationship and one entity into 1 table using foreign key and 1 table for other entity.

Now, you would definitely have a question in your mind, how we form such rules? This is very easy and logical. Let's understand the logic behind it for one case and you can similarly establish the results for other cases too.

We have been given a scenario of 1:N relationship with two entities E1(ABC) and E2(DEF),

where A and D are primary keys, respectively. E1 has a partial participation while E2 has total participation in the relationship R.

Based on above scenario, we create certain tuples in E1:

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

Similarly, create certain tuples for E2:

D	E	F
d1	e1	f1
d2	e2	f2
d3	e3	f3

Now, create a relationship R satisfying above conditions, i.e. E1 is partial pariticipation and E2 is total participation and E1 to E2 is 1:N relationship.

A	D
a1	d1
a1	d2
a2	d3

Think about possibilities, how can we merge?

- **Way-1:** Merge the two entities and relationship into a single table. This is not correct as (AD) will become primary key for this table, but primary key can never have a NULL value.

A	B	C	D	E	F
a1	b1	c1	d1	e1	f1
a1	b1	c1	d2	e2	f2
a2	b2	c2	d3	e3	f3
a3	b3	c3	NULL	NULL	NULL

- **Way-2:** Merge relationship on 1-side. This is not correct as (AD) will become primary key for this table, but primary key can never have a NULL value.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c2	d3
a3	b3	c3	NULL

- **Way-3:** Merge relationship on N-side. This is correct.

D	E	F	A
d1	e1	f1	a1
d2	e2	f2	a1
d3	e3	f3	a2

- On the same grounds, could you think why we allow merging the two entities as well as relationship into 1 table, when it is a 1:1 relationship? Simply, we would not have a composite primary key there, so we will definitely have a primary key with no NULL values present in

it. Stress some more, why we allow merging the entities and relationship with both sides total participation? The reason is even if we have a composite primary key for such merged table, we are sure that it will never have any NULL values for primary key.

DBMS | Introduction of 4th and 5th Normal form

If two or more independent relation are kept in a single relation or we can say **multivalue dependency** occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A **multivalued dependency** always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

For a dependency A -> B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for A ->> B multivalued dependency. For example,

Person Mobile Food_Likes

Mahesh 9893/9424 Burger / pizza

Ramesh 9191 Pizza

Person->-> mobile,

Person ->-> food_likes

This is read as “person multidetermines mobile” and “person multidetermines food_likes.”

Note that a functional dependency is a special case of multivalued dependency. In a functional dependency X -> Y, every x determines exactly one y, never more than one.

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: 1. It should be in the Boyce-Codd Normal Form (BCNF).
2. the table should not have any Multi-valued Dependency.

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

Example – Consider the database table of a class whaich has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table – R1(SID, SNAME)

SID	SNAME
S1	A
S2	B

Table – R2(CID, CNAME)

CID	CNAME
C1	C
C2	D

When there cross product is done it resulted in multivalued dependencies:

Table – R1 X R2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Multivalued dependencies (MVD) are:

SID->->CID; SID->->CNAME; SNAME->->CNAME

Joint dependency – Join decomposition is a further generalization of Multivalued dependencies. If the join of R1 and R2 over C is equal to relation R then we can say that a join dependency (JD) exists, where R1 and R2 are the decomposition R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D). Alternatively, R1 and R2 are a lossless decomposition of R. A JD ⋈ {R1, R2, …, Rn} is said to hold over a relation R if R1, R2, ….., Rn is a lossless-join decomposition. The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to
the relation R. Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Let R is a relation schema R1, R2, R3……..Rn be the decomposition of R. r(R) is said to satisfy join dependency if and only if $\bowtie_{i=1}^{n} \Pi_{R_i}(r) = r$

Example –

Table – R1

Company	Product
C1	pendrive
C1	mic
C2	speaker

Company	Product
C2	speaker

Company->->Product

Table – R2

Agent	Company
Aman	C1
Aman	C2
Mohan	C1

Agent->->Company

Table – R3

Agent	Product
Aman	pendrive
Aman	mic
Aman	speaker
Mohan	speaker

Agent->->Product

Table – R1⋈R2⋈R3

Company	Product	Agent
C1	pendrive	Aman
C1	mic	Aman
C2	speaker	speaker
C1	speaker	Aman

Agent->->Product

Fifth Normal Form / Projected Normal Form (5NF):

A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

Properties – A relation R is in 5NF if and only if it satisfies following conditions: 1. R should be already in 4NF.
2. It cannot be further non loss decomposed (join dependency)

Example – Consider the above schema, with a case as "if a company makes a product and an agent is an agent for that company, then he always sells that product for the company". Under these circumstances, the ACP table is shown as:

Table – ACP

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decompose into 3 relations. Now, the natural Join of all the three relations will be shown as:

Table – R1

Agent Company
A1 PQR
A1 XYZ
A2 PQR

Table – R2

Agent Product
A1 Nut
A1 Bolt
A2 Nut

Table – R3

Company Product
PQR Nut
PQR Bolt
XYZ Nut
XYZ Bolt

Result of Natural Join of R1 and R3 over 'Company' and then Natural Join of R13 and R2 over 'Agent'and 'Product' will be table **ACP**.

Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation is in 5NF as it does not violate the property of lossless join.

Dr Rahul Shajan