BCA Part II
Paper-XI: DBMS Using MS-ACCESS
Topic: Normalisation

Prepared by: Dr. Kiran Pandey
(School of Computer Science)

Email-id: kiranpandey.nou@gmail.com

NORMALISATION CONCEPTS

Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

If a database design is not perfect, it may contain anomalies. Managing a database with anomalies is very difficult. Three basic anomalies are:

- **Update anomalies**: If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies**: We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

- **Insert anomalies**: We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

FUNCTIONAL DEPENDENCY

Functional dependency (FD) is a set of constraints between two attributes in a relation. It plays an important role in differentiating good database from bad database.

A functional dependency **A->B** in a relation holds if two tuples having same value of attribute **A** also have same value for attribute **B**. For Example, in relation **STUDENT** shown in table 1, Functional Dependencies:

STUD_NO->STUD_NAME,STUD_NO->STUD_ADDR hold but,

STUD_NAME->STUD_ADDR does not hold.

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in Table 1.

- We know that STUD_NO is unique for each student. So STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY and STUD_NO -> STUD_AGE all will be true.

- Similarly, STUD_STATE->STUD_COUNTRY will be true as if two records have same STUD_STATE, they will have same STUD_COUNTRY as well.

- For relation STUDENT_COURSE, COURSE_NO->COURSE_NAME will be true as two records with same COURSE_NO will have same COURSE_NAME.

Functional Dependency Set: Functional Dependency set or FD set of a relation is the set of all FDs present in the relation. For Example, FD set for relation STUDENT shown in table 1 is:

{STUD_NO->STUD_NAME, STUD_NO->STUD_PHONE, STUD_NO->STUD_STATE, STUD_NO->STUD_COUNTRY,

STUD_NO -> STUD_AGE, STUD_STATE->STUD_COUNT}

Armstrong's Axioms

If **F** is a set of functional dependencies then the closure of **F**, denoted as $\mathbf{F^+}$, is the set of all functional dependencies logically implied by **F**. Armstrong's Axioms are a set of rules that, when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule**: If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule**: If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule**: Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

Trivial Functional Dependency

- **Trivial**: If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial**: If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial**: If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

MULTIVALUED DEPENDENCY AND JOIN DEPENDENCIES

So far we have discussed the concept of functional dependency, which is by far the most important type of dependency in relational database design theory, and normal forms based on functional dependencies. However, in many cases relations have constraints that cannot be specified as functional dependencies. In this section, we discuss the concept of *multivalued dependency* (MVD). If we have two or more multivalued *independent* attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved.

Formal Definition of Multivalued Dependency

Definition. A multivalued dependency $X \rightarrow\rightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R:

If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R – (X \cup Y))$:

$t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

Whenever $X \rightarrow\rightarrow Y$ holds, we say that X **multi-determines** Y. Because of the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in R, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$, and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.

An **MVD** X →→ Y in R is called a **trivial MVD** if

(a) *Y* is a subset of *X*, or

(b) $X \cup Y = R$.

An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**. A trivial MVD will hold in *any* relation state *r* of *R*; it is called trivial because it does not specify any significant or meaningful constraint on *R*.

If we have a *nontrivial MVD* in a relation, we may have to repeat values redundantly in the tuples.

NORMAL FORMS

Normal forms are used to eliminate or reduce redundancy in database tables. Various types of normal forms are discussed below. Normalization concepts are used to divide relations into following normal forms:

1. **First Normal Form (1NF)**
2. **Second Normal Form (2NF)**
3. **Third Normal Form (3NF)**
4. **Boyce-Codd Normal Form (3.5NF)**
5. **Forth Normal Form (4NF)**
6. **Fifth Normal Form (5NF)**

Note: In real life scenarios, we really do not apply all the normal forms to get the database design done. We usually apply till 3NF normal form. 4th and 5th normal forms are ignored in most of the cases. It's all depends on the requirement of the system.

1. First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units. 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.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	Haryana	India
2	RAM	9898291281	Punjab	India
3	SURESH		Punjab	India

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
1	RAM	9871717178			
2	RAM	9898291281	Punjab	India	19
3	SURESH		Punjab	India	21

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

Example 3: Look at the table given below:

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form. Each attribute must contain only a single value from its predefined domain.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

2. Second Normal Form:

Before we learn about the second normal form, we need to understand the following:

- **Prime attribute**: An attribute, which is a part of the prime-key, is known as a prime attribute.

- **Non-prime attribute**: An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X for which Y → A also holds true.

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	Computer Networks
1	C2	Computer Networks

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.

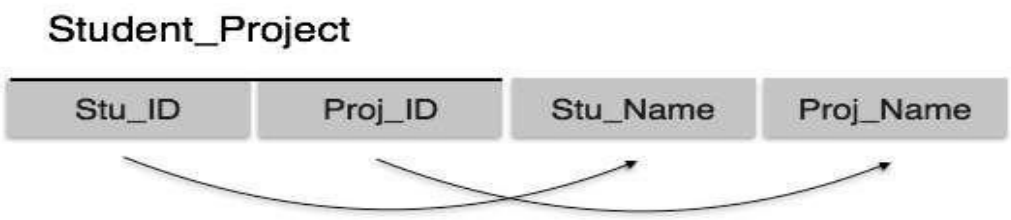
- **Example 2 –** Consider following functional dependencies in relation R (A, B , C, D)

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

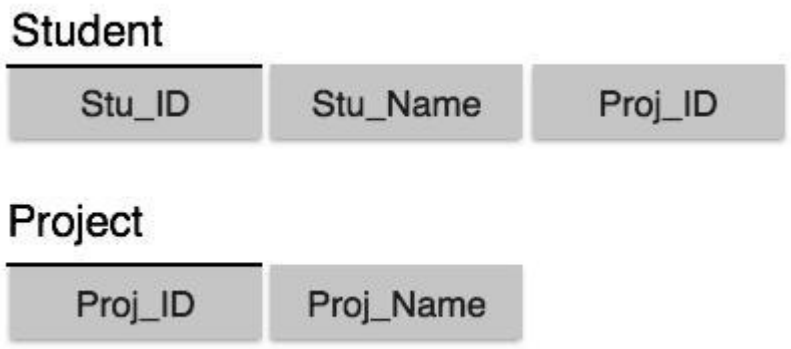
BC -> D [B and C together determine D]

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.

Example 3:



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e., Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

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. For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy:

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either:
  - X is a super key or,
  - A is prime attribute.

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

Table 4

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_NO -> 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

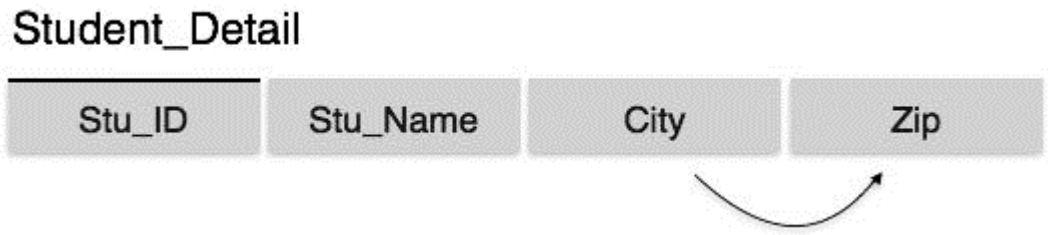
  STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

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

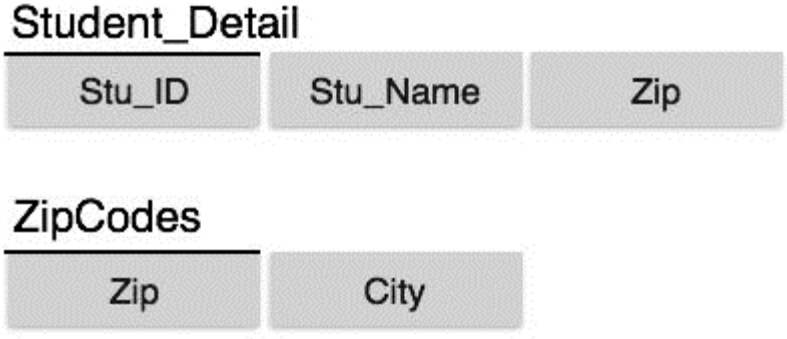
- **Example 2 –** Consider relation R(A, B, C, D, E)

  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.

  Example 3:



We find that in the above Student_detail relation, **Stu_ID** is the key and only prime key attribute. We find that City can be identified by **Stu_ID** as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally **Zip → City**, so there exists **transitive dependency**. To bring this relation into third normal form, we break the relation into two relations as follows:



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

Key Points
1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. 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).

7. 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.
8. 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.

4th Normal Form

In the fourth normal form,

- It should meet all the requirement of 3NF
- Attribute of one or more rows in the table should not result in more than one rows of the same table leading to multi-valued dependencies

To understand it clearly, consider a table with Subject, Lecturer who teaches each subject and recommended Books for each subject.

SUBJECT	**LECTURER**	**BOOKS**

Mathematics	Ram	Maths book1
Mathematics	Alam	Maths book2
Physics	John	Physics book
Chemistry	Mary	Chemistry Book

If we observe the data in the table above it satisfies 3NF. But LECTURER and BOOKS are two independent entities here. There is no relationship between Lecturer and Books. In the above example, either Ram or Alam can teach Mathematics. For Mathematics subject, student can refer either 'Maths Book1' or 'Maths Book2'. i.e.

SUBJECT --> LECTURER

SUBJECT-->BOOKS

This is a multivalued dependency on SUBJECT. If we need to select both lecturer and books recommended for any of the subject, it will show up (lecturer, books) combination, which implies lecturer who recommends which book. This is not correct.

To eliminate this dependency, we divide the table into two as below:

Now if we want to know the lecturer names and books recommended for any of the subject, we will fire two independent queries. Hence it removes the multi-valued dependency and confusion around the data. Thus the table is in 4NF.

5th Normal Form

A database is said to be in 5NF, if and only if,

- It's in 4NF
- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data

or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

Consider an example of different Subjects taught by different lecturers and the lecturers taking classes for different semesters.

In this case, combination of all these 3 fields is required to identify a valid data. Imagine we want to add a new class - Semester3 but do not know which Subject and who will be taking that subject. We would be simply inserting a new entry with Class as Semester3 and leaving Lecturer and subject as NULL. As we discussed above, it's not a good to have such entries. Moreover, all the three columns together act as a primary key, we cannot leave other two columns blank.

Hence we have to decompose the table in such a way that it satisfies all the rules till 4NF and when join them by using keys, it should yield correct record. Here, we can represent each lecturer's Subject area and their classes in a better way. We can divide above table into three - (SUBJECT, LECTURER), (LECTURER, CLASS), (SUBJECT, CLASS)

Now, each of combinations is in three different tables. If we need to identify who is teaching which subject to which semester, we need join the keys of each table and get the result.

For example, who teaches Physics to Semester 1, we would be selecting Physics and Semester1 from table 3 above, join with table1 using Subject to filter out the lecturer names. Then join with table2 using Lecturer to get correct lecturer name. That is we joined key columns of each table to get the correct data. Hence there is no lose or new data - satisfying 5NF condition.