Unit 3- Normalization

Keys in DBMS

The different types of keys in DBMS are –

- **Candidate Key -** The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.

- **Primary Key -** The primary key is selected from one of the candidate keys and becomes the identifying key of a table. It can uniquely identify any data row of the table.

- **Super Key -** Super Key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in the table.

- **Composite Key -** If any single attribute of a table is not capable of being the key i.e it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.

- **Secondary Key -** Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.

- **Foreign Key -** A foreign key is an attribute value in a table that acts as the primary key in another table. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.

Functional Dependency

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

If F is a set of functional dependencies then the closure of F, denoted as 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.

Attribute Closure

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_PHONE **hold**
but

STUD_NAME->STUD_ADDR **do not hold**

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_COUNTRY }

Attribute Closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
How to find attribute closure of an attribute set?
To find attribute closure of an attribute set:

- Add elements of attribute set to the result set.
- Recursively add elements to the result set which can be functionally determined from the elements of the result set.

Using FD set of table 1, attribute closure can be determined as:

(STUD_NO)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_STATE)+ = {STUD_STATE, STUD_COUNTRY}

How to find Candidate Keys and Super Keys using Attribute Closure?

- If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation.
- If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well. For Example, using FD set of table 1,

(STUD_NO, STUD_NAME)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_NO)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_NO, STUD_NAME) will be super key but not candidate key because its subset (STUD_NO)+ is equal to all attributes of the relation. So, STUD_NO will be a candidate key.

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.

Lossless Decomposition in DBMS

Lossless join decomposition is a decomposition of a relation R into relations R1,R2 such that if we perform natural join of relation R1 and R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data..

In other words by lossless decomposition it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition we select the common attribute and the criteria for selecting common attribute is that the common attribute must be a candidate key or super key in either of relation R1,R2 or both.

Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

R1 ∩ R2 → R1

OR

R1 ∩ R2 → R2

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **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.

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.

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

We re-arrange the relation (table) as below, to convert it to First Normal Form.

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

Each attribute must contain only a single value from its pre-defined domain.

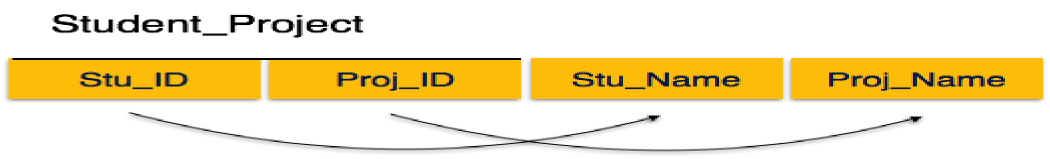
Second Normal Form

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

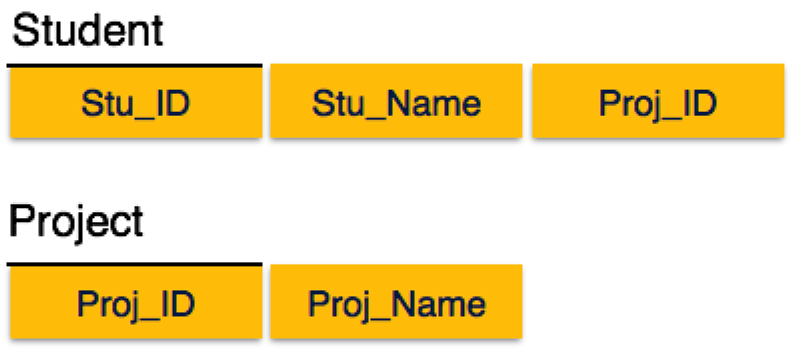
- **Prime attribute** – An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** – An attribute, which is not a part of the prime-key, is said to

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.



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.



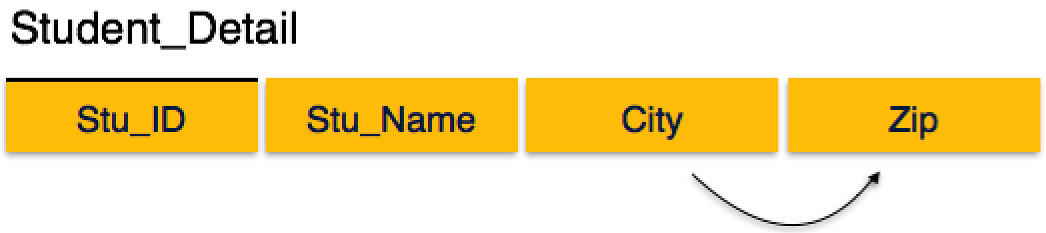
We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

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

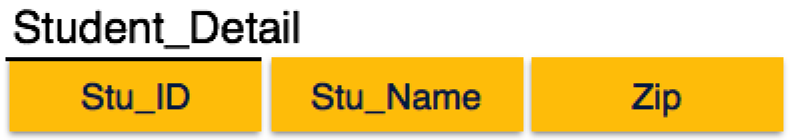
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
    - X is a superkey or,

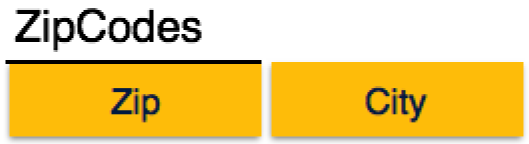
- A is prime attribute



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, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows −





Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −

- For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF.

DBMS – Transaction

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

ACID Properties

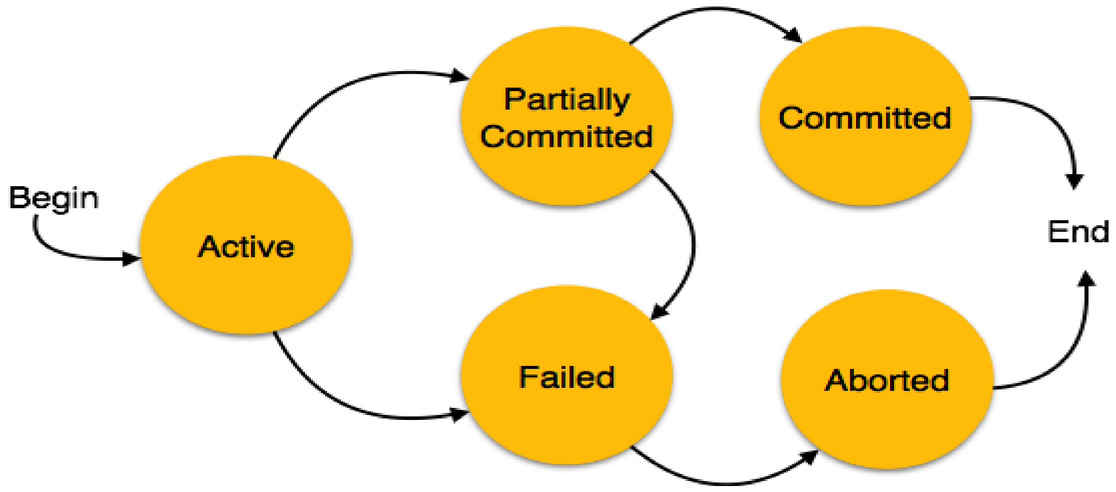
A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all

the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

States of Transactions

A transaction in a database can be in one of the following states –



- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –
    - Re-start the transaction
    - Kill the transaction
- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.
- **Question 1:**
  Let R (A, B, C, D) be a relational schema with the following functional

dependencies:

- A → B, B → C,
- C → D and D → B.
-
- The decomposition of R into
- (A, B), (B, C), (B, D)

(A) gives a lossless join, and is dependency preserving
(B) gives a lossless join, but is not dependency preserving
(C) does not give a lossless join, but is dependency preserving
(D) does not give a lossless join and is not dependency preserving

Solution:-

Lossless-JoinDecomposition:
Decomposition of R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

- R1 ∩ R2 → R1
- OR
- R1 ∩ R2 → R2
-
- **Dependency Preserving Decomposition:**
  Decomposition of R into R1 and R2 is a dependency preserving decomposition if closure of functional dependencies after decomposition is same as closure of of FDs before decomposition.
  A simple way is to just check whether we can derive all the original FDs from the FDs present after decomposition.

Question: Consider the relation scheme R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R. What is the key for R?
A. {E, F}
B. {E, F, H}
C. {E, F, H, K, L}
D. {E}
Answer: Finding attribute closure of all given options, we get:
{E,F}+ = {EFGIJ}
{E,F,H}+ = {EFHGIJKLMN}
{E,F,H,K,L}+ = {{EFHGIJKLMN}
{E}+ = {E}

{EFH}+ and {EFHKL}+ results in set of all attributes, but EFH is minimal. So it will be candidate key. So correct option is (B).

Question: In a schema with attributes A, B, C, D and E following set of functional dependencies are given

{A -> B, A -> C, CD -> E, B -> D, E -> A}

Which of the following functional dependencies is NOT implied by the above set? (GATE IT 2005)

A.CD->AC

B.BD->CD

C.BC->CD

D. AC -> BC

Using FD set given in question,

(CD)+ = {CDEAB} which means CD -> AC also holds true.

(BD)+ = {BD} which means BD -> CD can't hold true. So this FD is no implied in FD set. So (B) is the required option.

Others can be checked in the same way.

Question: Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A–>B, BC–> D, E–>C, D–>A}. What are the candidate keys of R?

(a)AE,BE

(b)AE,BE,DE

(c)AEH,BEH,BCH

(d) AEH, BEH, DEH

(AE)+ = {ABECD} which is not set of all attributes. So AE is not a candidate key. Hence option A and B are wrong.

(AEH)+ = {ABCDEH}

(BEH)+ = {BEHCDA}

(BCH)+ = {BCHDA} which is not set of all attributes. So BCH is not a candidate key. Hence option C is wrong.

So correct answer is D.

Question: Given a relation R(A, B, C, D) and Functional Dependency set FD = { AB → CD, B → C }, determine whether the given R is in 2NF?

Question : Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. F = {CH -> G, A -> BC, B -> CFH, E -> A, F -> EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R. How many

candidate keys does the relation R have?
Ans: 4
Question: Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. F = {CH -> G, A -> BC, B -> CFH, E -> A, F -> EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R. How many candidate keys does the relation R have also find the normal form?

Ans: 4; 1NF