**Definition of Super Key in DBMS**: A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.

## How candidate key is different from super key?

Answer is simple – Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That’s the reason they are also termed as minimal super key.

**Table: Employee**

Emp\_SSN Emp\_Number Emp\_Name

--------- ---------- --------

123456789 226 Steve

999999321 227 Ajeet

888997212 228 Chaitanya

777778888 229 Robert

**Super keys**: The above table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.

* {Emp\_SSN}
* {Emp\_Number}
* {Emp\_SSN, Emp\_Number}
* {Emp\_SSN, Emp\_Name}
* {Emp\_SSN, Emp\_Number, Emp\_Name}
* {Emp\_Number, Emp\_Name}

**Candidate Keys**: a candidate key is a minimal super key with no redundant attributes. The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets.

Lets select the candidate keys from the above set of super keys.

1. {Emp\_Id} – No redundant attributes  
2. {Emp\_Number} – No redundant attributes  
3. {Emp\_Id, Emp\_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.  
4. {Emp\_Id, Emp\_Name} – Redundant attribute Emp\_Name.  
5. {Emp\_Id, Emp\_Number, Emp\_Name} – Redundant attributes. Emp\_Id or Emp\_Number alone are sufficient enough to uniquely identify a row of Employee table.  
6. {Emp\_Number, Emp\_Name} – Redundant attribute Emp\_Name.

The **candidate keys** we have selected are:  
{Emp\_Id}  
{Emp\_Number}

Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification.

**Super key vs Candidate Key**

1. First you have to understand that all the candidate keys are super keys. This is because the candidate keys are chosen out of the super keys.  
2. How we choose candidate keys from the set of super keys? We look for those keys from which we cannot remove any fields. In the above example, we have not chosen {Emp\_SSN, Emp\_Name} as candidate key because {Emp\_SSN} alone can identify a unique row in the table and Emp\_Name is redundant.

[**Primary key**](https://beginnersbook.com/2015/04/primary-key-in-dbms/):  
A Primary key is selected from a set of candidate keys. This is done by database admin or database designer. We can say that either {Emp\_SSN} or {Emp\_Number} can be chosen as a primary key for the table Employee.

## What is a Primary Key?

**PRIMARY KEY** is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

### Rules for defining Primary key:

* Two rows can't have the same primary key value
* It must for every row to have a primary key value.
* The primary key field cannot be null.
* The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

## What is the Foreign key?

**FOREIGN KEY** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

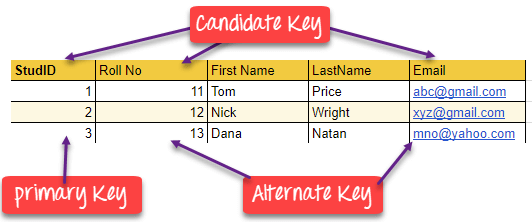
**COMPOSITE KEY** is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

**Note:** Any key such as [super key](https://beginnersbook.com/2015/04/super-key-in-dbms/), [primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/), [candidate key](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) etc. can be called composite key if it has more than one attributes.

Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudID** | Roll No | First Name | LastName | Email |
| 1 | 11 | Tom | Price | [abc@gmail.com](mailto:abc@gmail.com) |
| 2 | 12 | Nick | Wright | [xyz@gmail.com](mailto:xyz@gmail.com) |
| 3 | 13 | Dana | Natan | [mno@yahoo.com](mailto:mno@yahoo.com) |

[](https://www.guru99.com/images/1/100518_0517_DBMSKeysPri1.png)

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

**Anomalies in DBMS**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

**Example**:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | company | emp\_dept |
| 101 | Rick | TCS | D001 |
| 101 | Rick | TCS | D001 |
| 123 | Maggie | TCS | D001 |
| 166 | Glenn | TCS | D001 |
| 166 | Glenn | TCS | D001 |

The above table is not normalized. We will see the problems that we face when a table is not normalized.

**Update anomaly**:

**Insert anomaly**:

**Delete anomaly**:

To overcome these anomalies we need to normalize the data.

**Normalization**

Here are the most commonly used normal forms:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce & Codd normal form (BCNF)
* There are the four types of normal forms:
* 

|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless. |

**First normal form (1NF)**

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

**Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212  9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123  8123450987 |

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF**as the rule says “each attribute of a table must have atomic (single) values”, the emp\_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

**Second normal form (2NF)**

A table is said to be in 2NF if both the following conditions hold:

* Table is in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

Or

There should be no Partial Dependency.

**Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create 0a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

|  |  |  |
| --- | --- | --- |
| teacher\_id | Subject | teacher\_age |
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys**: {teacher\_id, subject}  
**Non prime attribute**: teacher\_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher\_age is dependent on teacher\_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:

|  |  |
| --- | --- |
| teacher\_id | subject |
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

**teacher\_details table: teacher\_subject table:**

|  |  |
| --- | --- |
| teacher\_id | teacher\_age |
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

Now the tables comply with Second normal form (2NF).

**Third Normal form (3NF)**

A table design is said to be in 3NF if both the following conditions hold:

* Table must be in 2NF
* [Transitive functional dependency](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [candidate key](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.

An attribute that is a part of one of the candidate keys is known as prime attribute.

**Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee\_details that looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_zip | emp\_state | emp\_city | emp\_district |
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

**Super keys**: {emp\_id}, {emp\_id, emp\_name}, {emp\_id, emp\_name, emp\_zip}…so on  
**Candidate Keys**: {emp\_id}  
**Non-prime attributes**: all attributes except emp\_id are non-prime as they are not part of any candidate keys.

Here, emp\_state, emp\_city & emp\_district dependent on emp\_zip. And, emp\_zip is dependent on emp\_id that makes non-prime attributes (emp\_state, emp\_city & emp\_district) transitively dependent on super key (emp\_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

**employee table:**

|  |  |  |
| --- | --- | --- |
| emp\_id | emp\_name | emp\_zip |
| 1001 | John | 282005 |
| 1002 | Ajeet | 222008 |
| 1006 | Lora | 282007 |
| 1101 | Lilly | 292008 |
| 1201 | Steve | 222999 |

**employee\_zip table:**

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_zip | emp\_state | emp\_city | emp\_district |
| 282005 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

## Boyce Codd normal form (BCNF)

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every [functional dependency](https://beginnersbook.com/2015/04/functional-dependency-in-dbms/) X->Y, X should be the super key of the table.

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| emp\_id | emp\_nationality | emp\_dept | dept\_type | dept\_no\_of\_emp |
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | Stores | D002 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | design and technical support | D134 | 600 |

**Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

**Functional dependencies in the table above**:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

**Candidate key**: {emp\_id, emp\_dept}

The table is not in BCNF as neither emp\_id nor emp\_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:  
**emp\_nationality table:**

|  |  |
| --- | --- |
| emp\_id | emp\_nationality |
| 1001 | Austrian |
| 1002 | American |

**emp\_dept table:**

|  |  |  |
| --- | --- | --- |
| emp\_dept | dept\_type | dept\_no\_of\_emp |
| Production and planning | D001 | 200 |
| Stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

|  |  |
| --- | --- |
| emp\_id | emp\_dept |
| 1001 | Production and planning |
| 1001 | Stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

**emp\_dept\_mapping table:**

**Functional dependencies**:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

**Candidate keys**:  
For first table: emp\_id  
For second table: emp\_dept  
For third table: {emp\_id, emp\_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

## What is Functional Dependency?

**Functional Dependency (FD)** is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.

The functional dependency is a relationship that exists between two attributes.

X   →   Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

## Types of Functional dependency

### DBMS Functional Dependency

### Trivial functional dependency

* A → B has trivial functional dependency if B is a subset of A.
* The following dependencies are also trivial like: A → A, B → B

**Ex**:Consider a table with two columns Employee\_Id and Employee\_Name.

Employee\_Id → Employee\_Id and Employee\_Name   →    Employee\_Name are trivial dependencies too

{Employee\_id, Employee\_Name}   →    Employee\_Id is a trivial functional dependency

Employee\_Name is a subset of {Employee\_Id, Employee\_Name}.valid

Employee\_Id is a subset of {Employee\_Id, Employee\_Name}. valid

**Non Trivial Functional Dependency**

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

**Inference Rule (IR):**

* The Armstrong's axioms are the basic inference rule.
* Armstrong's axioms are used to conclude functional dependencies on a relational database.
* The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
* Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

## 1. Reflexive Rule (IR1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

If X ⊇ Y then X  →    Y

## 2. Transitive Rule (IR3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

If X   →   Y and Y  →  Z then X  →   Z

## 3.Augmentation Rule

If X    →  Y then XZ   →   YZ

## 4. Union Rule (IR4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

If X    →  Y and X   →  Z then X  →    YZ

## 5. Decomposition Rule (IR5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. If X   →   YZ then X   →   Y and X  →    Z

**Proof:**

1. X → YZ (given)  
2. YZ → Y (using IR1 Rule)  
3. X → Y (using IR3 on 1 and 2

## 6. Pseudo transitive Rule (IR6)

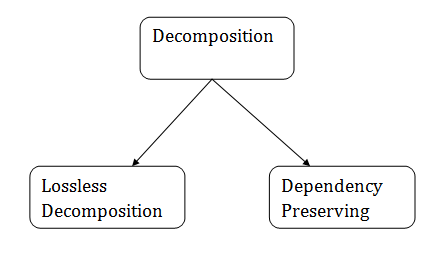
In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If X   →   Y and YZ   →   W then XZ   →   W

# Relational Decomposition

* When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
* In a database, it breaks the table into multiple tables.
* If the relation has no proper decomposition, then it may lead to problems like loss of information.
* Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

## Types of Decomposition



### Lossless Decomposition

* If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
* The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
* The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

**Example:**

**EMPLOYEE\_DEPARTMENT table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** |
| 22 | Denim | 28 | Mumbai |
| 33 | Alina | 25 | Delhi |
| 46 | Stephan | 30 | Bangalore |
| 52 | Katherine | 36 | Mumbai |
| 60 | Jack | 40 | Noida |

**0**

**D0EPARTMENT table**

|  |  |  |
| --- | --- | --- |
| **DE0PT\_ID** | **EMP\_ID** | **DEPT\_NAME** |
| 827 | 22 | Sales |
| 438 | 33 | Marketing |
| 869 | 46 | Finance |
| 575 | 52 | Production |
| 678 | 60 | Testing |

Now, when these two relations are joined on the common column "EMP\_ID", then the resultant relation will look like:

**Employee ⋈ Department**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

Hence, the decomposition is Lossless join decomposition.

**Dependency Preserving**

* It is an important constraint of the database.
* In the dependency preservation, at least one decomposed table must satisfy every dependency.
* If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
* For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

# Multivalued Dependency

* Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
* A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

**Example:** Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

|  |  |  |
| --- | --- | --- |
| **BIKE\_MODEL** | **MANUF\_YEAR** | **COLOR** |
| M2011 | 2008 | White |
| M2001 | 2008 | Black |
| M3001 | 2013 | White |
| M3001 | 2013 | Black |
| M4006 | 2017 | White |
| M4006 | 2017 | Black |

Here columns COLOR and MANUF\_YEAR are dependent on BIKE\_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE\_MODEL.

# Fourth normal form (4NF)

* A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
* For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

### Example

**STUDENT**

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **COURSE** | **HOBBY** |
| 21 | Computer | Dancing |
| 21 | Math | Singing |
| 34 | Chemistry | Dancing |
| 74 | Biology | Cricket |
| 59 | Physics | Hockey |

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

**STUDENT\_COURSE**

|  |  |
| --- | --- |
| **STU\_ID** | **COURSE** |
| 21 | Computer |
| 21 | Math |
| 34 | Chemistry |
| 74 | Biology |
| 59 | Physics |

**STUDENT\_HOBBY**

|  |  |
| --- | --- |
| **STU\_ID** | **HOBBY** |
| 21 | Dancing |
| 21 | Singing |
| 34 | Dancing |
| 74 | Cricket |
| 59 | Hockey |

**What is Join Dependency?**

If a table can be recreated by joining multiple tables and each of this table have a subset of the attributes of the table, then the table is in Join Dependency. It is a generalization of Multivalued Dependency

**Example**

**<Employee>**

|  |  |  |
| --- | --- | --- |
| **EmpName** | **EmpSkills** | **EmpJob**(Assigned Work) |
| Tom | Networking | EJ001 |
| Harry | Web Development | EJ002 |
| Katie | Programming | EJ002 |

The above table can be decomposed into the following three tables; therefore it is not in 5NF:  
  
**<EmployeeSkills>**

|  |  |
| --- | --- |
| **EmpName** | **EmpSkills** |
| Tom | Networking |
| Harry | Web Development |
| Katie | Programming |

**<EmployeeJob>**

|  |  |
| --- | --- |
| **EmpName** | **EmpJob** |
| Tom | EJ001 |
| Harry | EJ002 |
| Katie | EJ002 |

**<JobSkills>**

|  |  |
| --- | --- |
| **EmpSkills** | **EmpJob** |
| Networking | EJ001 |
| Web Development | EJ002 |
| Programming | EJ002 |

Our Join Dependency −

|  |
| --- |
| **{(EmpName, EmpSkills ), ( EmpName, EmpJob), (EmpSkills, EmpJob)}** |

The above relations have join dependency, so they are in 5NF. That would mean that a join relation of the above three relations is equal to our original relation **<Employee>**.

# Fifth normal form (5NF)

* A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data..
* 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
* 5NF is also known as Project-join normal form (PJ/NF).
* A relation decomposed into two relations must have loss-less join Property,

### Example

|  |  |  |
| --- | --- | --- |
| **SUBJECT** | **LECTURER** | **SEMESTER** |
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

**P1**

|  |  |
| --- | --- |
| **SEMESTER** | **SUBJECT** |
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

**P2**

|  |  |
| --- | --- |
| **SUBJECT** | **LECTURER** |
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

**P3**

|  |  |
| --- | --- |
| **SEMSTER** | **LECTURER** |
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

# Canonical Cover

In the case of updating the database, the responsibility of the system is to check whether the existing functional dependencies are getting violated during the process of updating. In case of a violation of functional dependencies in the new database state, the rollback of the system must take place.

A canonical cover or irreducible a set of functional dependencies FD is a simplified set of FD that has a similar closure as the original set FD.

### Extraneous attributes

An attribute of an FD is said to be extraneous if we can remove it without changing the closure of the set of FD.

**Example:** Given a relational Schema R( A, B, C, D) and set of Function Dependency FD = { B → A, AD → BC, C → ABD }. Find the canonical cover?

**Solution:** Given FD = { B → A, AD → BC, C → ABD }, now decompose the FD using decomposition rule( Armstrong Axiom ).

1. B → A
2. AD → B ( using decomposition inference rule on AD → BC)
3. AD → C ( using decomposition inference rule on AD → BC)
4. C → A ( using decomposition inference rule on C → ABD)
5. C → B ( using decomposition inference rule on C → ABD)
6. C → D ( using decomposition inference rule on C → ABD)

Now set of FD = { B → A, AD → B, AD → C, C → A, C → B, C → D }

The next step is to find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

**Calculating closure of all FD { B → A, AD → B, AD → C, C → A, C → B, C → D }**

1a. Closure B+ = BA using FD = { **B → A**, AD → B, AD → C, C → A, C → B, C → D }

1b. Closure B+ = B using FD = { AD → B, AD → C, C → A, C → B, C → D }

From 1 a and 1 b, we found that both the Closure( by including **B → A** and excluding **B → A** ) are not equivalent, hence FD B → A is important and cannot be removed from the set of FD.

2 a. Closure AD+ = ADBC using FD = { B →A, **AD → B**, AD → C, C → A, C → B, C → D }

2 b. Closure AD+ = ADCB using FD = { B → A, AD → C, C → A, C → B, C → D }

From 2 a and 2 b, we found that both the Closure (by including **AD → B** and excluding **AD → B**) are equivalent, hence FD **AD → B** is not important and can be removed from the set of FD.

**Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }**

3 a. Closure AD+ = ADCB using FD = { B →A, **AD → C**, C → A, C → B, C → D }

3 b. Closure AD+ = AD using FD = { B → A, C → A, C → B, C → D }

From 3 a and 3 b, we found that both the Closure (by including **AD → C** and excluding **AD → C** ) are not equivalent, hence FD AD → C is important and cannot be removed from the set of FD.

**Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }**

4 a. Closure C+ = CABD using FD = { B →A, AD → C, **C → A**, C → B, C → D }

4 b. Closure C+ = CBDA using FD = { B → A, AD → C, C → B, C → D }

From 4 a and 4 b, we found that both the Closure (by including **C → A** and excluding **C → A**) are equivalent, hence FD **C → A** is not important and can be removed from the set of FD.

**Hence resultant FD = { B → A, AD → C, C → B, C → D }**

5 a. Closure C+ = CBDA using FD = { B →A, AD → C, **C → B**, C → D }

5 b. Closure C+ = CD using FD = { B → A, AD → C, C → D }

From 5 a and 5 b, we found that both the Closure (by including **C → B** and excluding **C → B**) are not equivalent, hence FD **C → B** is important and cannot be removed from the set of FD.

**Hence resultant FD = { B → A, AD → C, C → B, C → D }**

6 a. Closure C+ = CDBA using FD = { B →A, AD → C, C → B, **C → D** }

6 b. Closure C+ = CBA using FD = { B → A, AD → C, C → B }

From 6 a and 6 b, we found that both the Closure( by including **C → D** and excluding **C → D**) are not equivalent, hence FD **C → D** is important and cannot be removed from the set of FD.

**Hence resultant FD = { B → A, AD → C, C → B, C → D }**

* Since FD = { B → A, AD → C, C → B, C → D } is resultant FD, now we have checked the redundancy of attribute, since the left side of FD AD → C has two attributes, let's check their importance, i.e. whether they both are important or only one.

Closure AD+ = ADCB using FD = { B →A, **AD → C**, C → B, C → D }

Closure A+ = A using FD = { B →A, **AD → C**, C → B, C → D }

Closure D+ = D using FD = { B →A, **AD → C**, C → B, C → D }

Since the closure of AD+, A+, D+ that we found are not all equivalent, hence in FD AD → C, both A and D are important attributes and cannot be removed.

Hence resultant FD = { B → A, AD → C, C → B, C → D } and we can rewrite as

**FD = { B → A, AD → C, C → BD } is Canonical Cover of FD = { B → A, AD → BC, C → ABD }.**

**Example 2:** Given a relational Schema R( W, X, Y, Z) and set of Function Dependency FD = { W → X, Y → X, Z → WXY, WY → Z }. Find the canonical cover?

A:Try it