**Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database**

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let’s discuss about anomalies first then we will discuss normal forms with examples.

**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**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp\_id for storing employee’s id, emp\_name for storing employee’s name, emp\_address for storing employee’s address and emp\_dept for storing the department details in which the employee works. At some point of time the table looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_dept |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

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

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

**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)

**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.

**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 |

**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.

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 a 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\_details table:**

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

**teacher\_subject table:**

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

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.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

* X is a [super key](https://beginnersbook.com/2015/04/super-key-in-dbms/) of table
* Y is a prime attribute of table

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.

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

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

**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\_dept\_mapping table:**

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

**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.

**Normalization**

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
* Normalization divides the larger table into the smaller table and links them using relationship.
* The normal form is used to reduce redundancy from the database table.

**Types of Normal Forms**

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)**

* A relation will be 1NF if it contains an atomic value.
* It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

**Second Normal Form (2NF)**

* In the 2NF, relational must be in 1NF.
* In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

**TEACHER\_DETAIL table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **TEACHER\_AGE** |
| 25 | 30 |
| 47 | 35 |
| 83 | 38 |

**TEACHER\_SUBJECT table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |
| **Third Normal Form (3NF)**   * A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency. * 3NF is used to reduce the data duplication. It is also used to achieve the data integrity. * If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.   A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.   1. X is a super key. 2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.   **Example:**  **EMPLOYEE\_DETAIL table:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** | | 222 | Harry | 201010 | UP | Noida | | 333 | Stephan | 02228 | US | Boston | | 444 | Lan | 60007 | US | Chicago | | 555 | Katharine | 06389 | UK | Norwich | | 666 | John | 462007 | MP | Bhopal |   **Super key in the table above:**   * 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on   **Candidate key:** {EMP\_ID}  **Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.  Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.  That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.  **EMPLOYEE table:**   |  |  |  | | --- | --- | --- | | **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | | 222 | Harry | 201010 | | 333 | Stephan | 02228 | | 444 | Lan | 60007 | | 555 | Katharine | 06389 | | 666 | John | 462007 |   **EMPLOYEE\_ZIP table:**   |  |  |  | | --- | --- | --- | | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** | | 201010 | UP | Noida | | 02228 | US | Boston | | 60007 | US | Chicago | | 06389 | UK | Norwich | | 462007 | MP | Bhopal | | | |

**Boyce Codd normal form (BCNF)**

* BCNF is the advance version of 3NF. It is stricter than 3NF.
* A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
* For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** | **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** |
| 264 | India |
| 264 | India |

**EMP\_DEPT table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP\_DEPT\_MAPPING table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_DEPT** |
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

**Functional dependencies:**

1. EMP\_ID   →    EMP\_COUNTRY
2. EMP\_DEPT   →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate keys:**

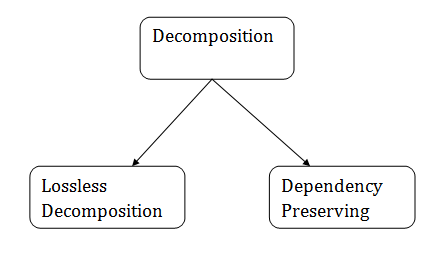
**For the first table:** EMP\_ID   
**For the second table:** EMP\_DEPT  
**For the third table:** {EMP\_ID, EMP\_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

# 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 |

**DEPARTMENT table**

|  |  |  |
| --- | --- | --- |
| **DEPT\_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. The representation of these dependencies is shown below:

1. BIKE\_MODEL   →  →  MANUF\_YEAR
2. BIKE\_MODEL   →  →  COLOR

This can be read as "BIKE\_MODEL multidetermined MANUF\_YEAR" and "BIKE\_MODEL multidetermined COLOR".

**join 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 decompositions 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.

**Inclusion Dependency**

* Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
* Inclusion dependencies are quite common. They typically show little influence on designing of the database.
* The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
* The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
* Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.
* Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
* In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.
* In practice, most inclusion dependencies are key-based that is involved only keys.

# 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?

**Solution:** Given FD = { W → X, Y → X, Z → WXY, WY → Z }, now decompose the FD using decomposition rule( Armstrong Axiom ).

1. W → X
2. Y → X
3. Z → W ( using decomposition inference rule on Z → WXY )
4. Z → X ( using decomposition inference rule on Z → WXY )
5. Z → Y ( using decomposition inference rule on Z → WXY )
6. WY → Z

Now set of FD = { W → X, Y → X, WY → Z, Z → W, Z → X, Z → Y }

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 { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }**

**1 a.** Closure W+ = WX using FD = { **W → X**, Y → X, Z → W, Z → X, Z → Y, WY → Z }

**1 b.** Closure W+ = W using FD = { Y → X, Z → W, Z → X, Z → Y, WY → Z }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }**

**2 a.** Closure Y+ = YX using FD = { W → X, **Y → X**, Z → W, Z → X, Z → Y, WY → Z }

**2 b.** Closure Y+ = Y using FD = { W → X, Z → W, Z → X, Z → Y, WY → Z }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }**

**3 a.** Closure Z+ = ZWXY using FD = { W → X, Y → X, **Z → W**, Z → X, Z → Y, WY → Z }

**3 b.** Closure Z+ = ZXY using FD = { W → X, Y → X, Z → X, Z → Y, WY → Z }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }**

**4 a**. Closure Z+ = ZXWY using FD = { W → X, Y → X, Z → W, **Z → X**, Z → Y, WY → Z }

**4 b.** Closure Z+ = ZWYX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }**

**5 a.** Closure Z+ = ZYWX using FD = { W → X, Y → X, Z → W, **Z → Y**, WY → Z }

**5 b.** Closure Z+ = ZWX using FD = { W → X, Y → X, Z → W, WY → Z }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }**

**6 a.** Closure WY+ = WYZX using FD = { W → X, Y → X, Z → W, Z → Y, **WY → Z** }

**6 b.** Closure WY+ = WYX using FD = { W → X, Y → X, Z → W, Z → Y }

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

**Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }**

Since FD = { W → X, Y → X, Z → W, Z → Y, WY → Z } is resultant FD now, we have checked the redundancy of attribute, since the left side of FD WY → Z has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

Closure WY+ = WYZX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Closure W+ = WX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Closure Y+ = YX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Since the closure of WY+, W+, Y+ that we found are not all equivalent, hence in FD WY → Z, both W and Y are important attributes and cannot be removed.

Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z } and we can rewrite as:

**FD = { W → X, Y → X, Z → WY, WY → Z } is Canonical Cover of FD = { W → X, Y → X, Z → WXY, WY → Z }.**

**Example 3:** Given a relational Schema R( V, W, X, Y, Z) and set of Function Dependency FD = { V → W, VW → X, Y → VXZ }. Find the canonical cover?

**Solution:** Given FD = { V → W, VW → X, Y → VXZ }. now decompose the FD using decomposition rule (Armstrong Axiom).

1. V → W
2. VW → X
3. Y → V ( using decomposition inference rule on Y → VXZ )
4. Y → X ( using decomposition inference rule on Y → VXZ )
5. Y → Z ( using decomposition inference rule on Y → VXZ )

Now set of FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.

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 { V → W, VW → X, Y → V, Y → X, Y → Z }.**

**1 a.** Closure V+ = VWX using FD = {**V → W**, VW → X, Y → V, Y → X, Y → Z}

**1 b.** Closure V+ = V using FD = {VW → X, Y → V, Y → X, Y → Z }

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

**Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.**

**2 a**. Closure VW+ = VWX using FD = { V → W, **VW → X**, Y → V, Y → X, Y → Z }

**2 b.** Closure VW+ = VW using FD = { V → W, Y → V, Y → X, Y → Z }

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

**Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.**

**3 a.** Closure Y+ = YVXZW using FD = { V → W, VW → X, **Y → V**, Y → X, Y → Z }

**3 b.** Closure Y+ = YXZ using FD = { V → W, VW → X, Y → X, Y → Z }

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

**Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.**

**4 a**. Closure Y+ = YXVZW using FD = { V → W, VW → X, Y → V, **Y → X**, Y → Z }

**4 b.** Closure Y+ = YVZWX using FD = { V → W, VW → X, Y → V, Y → Z }

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

**Hence resultant FD = { V → W, VW → X, Y → V, Y → Z }.**

**5 a.** Closure Y+ = YZVWX using FD = { V → W, VW → X, Y → V, **Y → Z** }

**5 b.** Closure Y+ = YVWX using FD = { V → W, VW → X, Y → V }

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

**Hence resultant FD = { V → W, VW → X, Y → V, Y → Z }.**

Since FD = { V → W, VW → X, Y → V, Y → Z } is resultant FD now, we have checked the redundancy of attribute, since the left side of FD VW → X has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

Closure VW+ = VWX using FD = { V → W, VW → X, Y → V, Y → Z }

Closure V+ = VWX using FD = { V → W, VW → X, Y → V, Y → Z }

Closure W+ = W using FD = { V → W, VW → X, Y → V, Y → Z }

Since the closure of VW+, V+, W+ we found that all the Closures of VW and V are equivalent, hence in FD VW → X, W is not at all an important attribute and can be removed.

Hence resultant FD = { V → W, V → X, Y → V, Y → Z } and we can rewrite as

**FD = { V → WX, Y → VZ } is Canonical Cover of FD = { V → W, VW → X, Y → VXZ }.**

**CONCLUSION:** From the above three examples we conclude that canonical cover / irreducible set of functional dependency follows the following steps, which we need to follow while calculating Canonical Cover.

**STEP 1:** For a given set of FD, decompose each FD using decomposition rule (Armstrong Axiom) if the right side of any FD has more than one attribute.

**STEP 2:** Now make a new set of FD having all decomposed FD.

**STEP 3:** 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.

**STEP 4:** Repeat step 4 till all the FDs in FD set are complete.

**STEP 5:** After STEP 4, find resultant FD = { B → A, AD → C, C → B, C → D } which are not redundant.

**STEP 6:** Check redundancy of attribute, by selecting those FD's from FD sets which are having more than one attribute on its left, let's an FD AD → C has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

**STEP 6 a:** Find Closure AD+

**STEP 6 b:** Find Closure A+

**STEP 6 c:** Find Closure D+

Compare Closure of STEP (6a, 6b, 6c) if the closure of AD+, A+, D+ are not equivalent, hence in FD AD → C, both A and D are important attributes and cannot be removed, otherwise, we remove the redundant attribute.