**UNIT – 4**

**DATABASE DEPENDENCIES AND NORMALIZATION**

**FUNCTIONAL DEPENDENCIES**

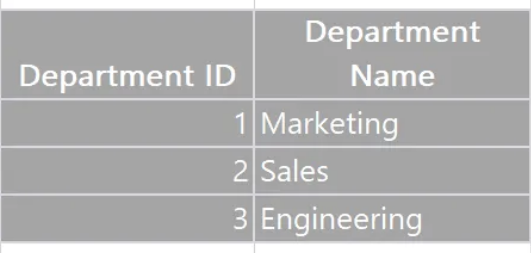
It is the relationship between the attributes of a table dependent on each other. it helps in preventing data redundancy and bad design of the database can be recognized.

To understand the functional dependency let us consider P is a relation (table) with attributes A and B. Functional dependency is represented by ( ) arrow sign i.e. A B.

**EXAMPLE-**

Let us consider a relation department, where department\_id is the primary key which uniquely identifies the department\_name attribute.

**DEPARTMENT TABLE**

****

Here, functional dependencies can be determined as department\_id is functionally dependent upon department\_name as follows –

department\_id department\_name

**TYPES OF FUNCTIONAL DEPENDENCIES**

1. **TRIVAL –** This functional dependency occurs when B is a subset of A.

For example, department\_id department\_name

1. **NON-TRIVIAL** occurs when B is not a subset of A.

For example, department\_id department\_name

This is a non-trivial functional dependency since department\_name is not a subset of department\_id.

1. **COMPLETELY NON-TRIVAL –** It occurs when A and B intersections are null.

**DECOMPOSITION**

It is the process of breaking down the functions of an organization into progressively greater levels of detail. In decomposition, one function of the organization is described in greater detail by a set of other supporting functions.

Therefore, the decomposition of a relation R consists of replacing relational schema with two or more relational schemas, each of which contains a subset of attributes R. Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistency etc. There are two types of decomposition –

1. Lossy decomposition
2. Non-lossy decomposition

Decomposition of a relation R into R1 and R2 is lossy when the join operation on R1 and R2 does not yield the same relation as in R. One of the main disadvantages of the lossy decomposition is some info is lost during the retrieval of the original relation.

For example – Let us consider a relation student having attributes roll\_no, s\_name, and branch.

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **s\_name** | **Branch** |
| 1001 | Abhinash | CSE |
| 1002 | Vinay | IT |

This relation can be decomposed into R1 and R2 as follows:

**R1 TABLE**

|  |  |
| --- | --- |
| **roll\_no** | **S\_name** |
| 1001 | Abhinash |
| 1002 | Vinay |

**R2 TABLE**

|  |  |
| --- | --- |
| **S\_name** | **Branch** |
| Abhinash | CSE |
| Vinay | IT |

In lossy decomposition output after joining will be as follows –

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **S\_name** | **Branch** |
| 1001 | Abhinash | CSE |
| 1001 | Abhinash | CSE |
| 1002 | Vinay | IT |
| 1002 | Vinay | IT |

When join is applied to R1 and R2 copies of tuple are generated. This decomposition is bad decomposition or lossy decomposition.

The decomposition of a relation in R1 and R2 is lossless or non-loss when the join operation on R1 and R2 produces the same relation as R.

A relation is to be decomposed into two or more tables in such a way that the designer can capture the precise content of the original table by joining the decomposed table.

**NORMALIZATION**

It is a process of organizing the data in the database to avoid data redundancy, insertion anomaly, update anomaly and deletion anomaly in DBMS. Three types of anomalies can occur when the database is not normalized.

For example, let us consider an organization storing its employee details in a table named “Employee”, which has four attributes i.e. E\_id, E\_name, E\_address, and E\_department. At this point, the table will look as follows –

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_id** | **E\_name** | **E\_address** | **E\_department** |
| 101 | Rohit | Delhi | D001 |
| 101 | Rohit | Delhi | D002 |
| 103 | Shivani | Agra | D003 |
| 166 | Subham | Chennai | D004 |
| 166 | Subham | Chennai | D005 |

This table is not normalized, due to which various problems can be faced as follows –

1. **Insertion Anomaly -** Suppose, when a new employee joins the organization, who is under training and not assigned to any department, then it is not possible to insert the data of that employee into the table if the E\_department field does not allow NULL value.
2. **Update Anomaly –** In our table, we have two rows for employee “Rohit” as he belongs to two departments. If we must update the address of “Rohit”, then we must update the same in two rows, otherwise, data will become inconsistent.
3. **Deletion Anomaly -** Suppose, if an organization closes the department “D003” then, deleting the row which has E\_department as “D003”, will also delete the info of the employee “Shivani” since she is assigned only to this department.

To overcome these anomalies, data must be normalized. Various forms of normalization are –

1. **FIRST NORMAL FORM (1 NF)**
2. **SECOND NORMAL FORM (2 NF)**
3. **THIRD NORMAL FORM (3 NF)**
4. **BCNF (BOYCEE-CODD NORMAL FORM)**
5. **FIRST NORMAL FORM (1 NF) -** As per the rule of the **First normal form,** an attribute or column of a table cannot hold multiple values, it should hold only atomic values.

For example – Suppose, a company must store employee details, it will create a table as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_id** | **E\_name** | **E\_address** | **E\_mobile** |
| 101 | Abhi | HAMIRPUR | 7876876523 |
| 102 | Bishnu | CHAMBA | 8863625432  9852433321 |
| 103 | Sahil | KANGRA | 8217786756 |
| 104 | Subham | DELHI | 7876554545  8823633565 |

Here. Two employees, Bishnu and Subham have two mobile numbers, so the company has to store them in the same fields as in the table. This table is not in 1 NF because E\_mobile is holding multiple values, which violates the rule of 1NF.

To make this table in 1NF, data should be stored as follows-

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_id** | **E\_name** | **E\_address** | **E\_mobile** |
| 101 | Abhi | HAMIRPUR | 7876876523 |
| 102 | Bishnu | CHAMBA | 8863625432 |
| 102 | Bishnu | CHAMBA | 9852433321 |
| 103 | Sahil | KANGRA | 8217786756 |
| 104 | Subham | DELHI | 7876554545 |
| 102 | Subham | DELHI | 8823633565 |

1. **SECOND NORMAL FORM (2 NF) –** A table is said to be in 2 NF if the following conditions are satisfied:
2. Table must be in 1NF.
3. No, the non-prime attribute is dependent upon the proper subset of any candidate key of the table.

For example, suppose a school must store data on its teachers and the subjects they teach. The following table is created:

|  |  |  |
| --- | --- | --- |
| **T\_id** | **subject** | **T\_age** |
| 111 | Math | 30 |
| 111 | Physics | 30 |
| 222 | Bio | 25 |
| 333 | IT | 26 |
| 333 | Chemistry | 26 |

A teacher can teach more than one subject so, the table has multiple rows for the same teacher.

Here candidate key is (T\_id, subject) and the non-prime attribute is T\_age.

This table is in 1 NF but not in 2 NF because the non-prime attribute T\_age is dependent upon T\_id which is a proper subset of the candidate key.

To make this table 2 NF comply, we must break this table into two tables as follows:

**Teacher table**

|  |  |
| --- | --- |
| **T\_id** | **T\_age** |
| 111 | 30 |
| 222 | 25 |
| 333 | 26 |

**SUBJECT TABLE**

|  |  |
| --- | --- |
| **T\_id** | **Subject** |
| 111 | Math |
| 111 | Physics |
| 222 | Bio |
| 333 | IT |
| 333 | Chemistry |

1. **THIRD NORMAL FORM (3 NF) –** A table is said to be in 3 NF if the following conditions are satisfied:
2. A table must be in 2 NF.
3. Transitive functional dependencies of non-prime attributes on any super key must be removed.

In other words, 3 NF can also be explained as follows –

A table is said to be in 3 NF if it is in 2 NF and for each functional dependency X Y, at least one of the following conditions must be there:

1. X is the super key of the table.
2. Y is the prime attribute of the table.

For example, A company must store the complete address of each employee. They can create a table “Employee Details” as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **E\_id** | **E\_name** | **E\_zip** | **E\_state** | **E\_city** | **E\_district** |
| 1001 | A | 176311 | UP | AGRA | DAYAL BAGH |
| 1002 | B | 182322 | HP | SHIMLA | SHIMLA |
| 1006 | C | 178221 | HP | SHIMLA | SHIMLA |
| 1101 | D | 172192 | UK | RISHIKESH | RISHIKESH |
| 1201 | E | 172713 | MP | GWALIOR | ASHOK NAGAR |

Here, the candidate key is E\_id and the super key are {E\_id, E\_name}, {E\_id, E\_name, E\_zip}, {E\_id, E\_name, E\_zip, E\_state} and so on.

All attributes accept E\_id are non-prime attributes because they are not part of the candidate key.

Here, E\_state, E\_city, and E\_district is dependent on E\_zip code and E\_zip is dependent on E-id which makes non-prime attributes E\_state, E\_city, and E\_district transitively dependent upon super key i.e. E\_id, which is violating the rule of 3 NF.

To make this table 3NF comply, we must break this table into two tables to remove transitive dependencies.

**EMPLOYEE TABLE**

|  |  |  |
| --- | --- | --- |
| **E\_id** | **E\_name** | **E\_zip** |
| 1001 | A | 176311 |
| 1002 | B | 182322 |
| 1006 | C | 178221 |
| 1101 | D | 172192 |
| 1201 | E | 172713 |

**EMPLOYEE ZIP**

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_zip** | **E\_state** | **E\_city** | **E\_district** |
| 176311 | UP | AGRA | DAYAL BAGH |
| 182322 | HP | SHIMLA | SHIMLA |
| 178221 | HP | SHIMLA | SHIMLA |
| 172192 | UK | RISHIKESH | RISHIKESH |
| 172713 | MP | GWALIOR | ASHOK NAGAR |

1. **BOYCE-CODD NORMAL FORM –** It is the advanced version of 3 NF and is also referred to as 3.5 NF.

A table is in BOYCE-CODD NF if it is in 3 NF and for every functional dependency X Y, X should be the super key of the table.

For example, suppose there is a company where employees work for more than one department then data will be stored as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **E\_id** | **E\_nationality** | **E\_department** | **Department\_type** | **No\_of\_employees** |
| 1001 | INDIAN | P & P | D001 | 200 |
| 1001 | INDIAN | Stores | D001 | 250 |
| 1002 | AMERICAN | Technical support | D134 | 100 |
| 1002 | AMERICAN | Purchasing | D134 | 600 |

Here, functional dependencies are E\_id E\_nationality, E\_department Department\_type, No\_of\_employees.

Here, candidate keys are E\_department, {E\_id, E\_department}.

This table is not in BOYCE-CODD NF because neither E\_id nor E\_department alone are keys.

To make this table in BOYCE-CODD NF complied, we must break this table into three tables.

**EMPLOYEE TABLE**

|  |  |
| --- | --- |
| **E\_id** | **E\_nationality** |
| 1001 | INDIAN |
| 1002 | AMERICAN |

**DEPARTMENT TABLE**

|  |  |  |
| --- | --- | --- |
| **E\_department** | **Department\_type** | **No\_of\_employees** |
| P & P | D001 | 200 |
| Stores | D001 | 250 |
| Technical support | D134 | 100 |
| Purchasing | D134 | 60 |

**DEPARTMENT MAPPING TABLE**

|  |  |
| --- | --- |
| **E\_id** | **E\_department** |
| 1001 | P & P |
| 1001 | Stores |
| 1002 | Technical support |
| 1002 | Purchasing |

Here, functional dependencies are E\_id E\_nationality, E\_department Department\_type, No\_of\_employees and

Candidate keys are E\_id, E\_department, and {E\_id, E\_department}

Now it is in BOYCE-CODD NORMAL FORM.