**Unit – 6 Relational Database Design**

**Introduction :**

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.* Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies. It can be considered as a “filtering” or “purification” process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions—the normal form tests—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties. Thus, the normalization procedure provides database designers with the following:

■ A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes

■ A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree

**Definition**. The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Redundancies

**Emp\_dept**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ename** | **Ssn** | **Bdate** | **Address** | **Dnumber** | **Dname** | **Dmgr\_ssn** |
| Smith, John B. | 123456789 | 1/9/1965 | 731 Fondren, Houston, TX | 5 | Research | 333445555 |
| Wong, Franklin T. | 333445555 | 12/8/1955 | 638 Voss, Houston, TX | 5 | Research | 333445555 |
| Zelaya, Alicia J. | 999887777 | 7/19/1968 | 3321 Castle, Spring, TX | 4 | Administration | 987654321 |
| Wallace, Jennifer S. | 987654321 | 6/20/1941 | 291 Berry, Bellaire, TX | 4 | Administration | 987654321 |
| Narayan, Ramesh K. | 666884444 | 9/15/1962 | 975 FireOak, Humble, TX | 5 | Research | 333445555 |
| English, Joyce A. | 453453453 | 7/31/1972 | 5631 Rice, Houston, TX | 5 | Research | 333445555 |
| Jabbar, Ahmad V. | 987987987 | 3/29/1969 | 980 Dallas, Houston, TX | 4 | Administration | 987654321 |
| Borg, James E. | 888665555 | 11/10/1937 | 450 Stone, Houston, TX | 1 | Headquarters | 888665555 |

**Modification Anomalies :-**

**Insertion Anomalies**. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP\_DEPT relation:

To insert a new employee tuple into EMP\_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP\_DEPT. In the design of schema, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation. It is difficult to insert a new department that has no employees as yet in the EMP\_DEPT relation. The only way to do this is to place NULL values in the table.

**Deletion Anomalies**. The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. If we delete from EMP\_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database. This problem does not occur in the database if DEPARTMENT tuples are stored separately.

**Modification Anomalies.** In EMP\_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong, It is easy to see that these three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates that can be avoided.

**Result of above anomalies**.

● Anomalies that cause redundant work to be done during insertion into and modification of a relation, and that may cause accidental loss of information during a deletion from a relation.

● Waste of storage space due to NULLs and the difficulty of performing selections, aggregation operations, and joins due to NULL values

● Generation of invalid and spurious data during joins on base relations with matched attributes that may not represent a proper (foreign key, primary key) relationship.

**Functional Dependencies**

• Functional dependencies are constraints on the set of legal relations. It defines attributes

of relation, how they are related to each other.• It determines unique value for a certain set of attributes to the value for another set of attributes that is functional dependency is a generalization of the notation of key. • Functional dependencies are interrelationship among attributes of a relation.

**Definition:**

For a given relation R with attribute X and Y, Y is said to be functionally dependent on X, if given

value for each X uniquely determines the value of the attribute in Y. X is called determinant of

the functional dependency (FD) and functional dependency denoted by X→ Y.

Example 1: consider a relation supplier

Supplier(supplier\_id#, sname, status, city)

Here, sname, status and city are functionally dependent on supplier\_id. Meaning is that each

supplier id uniquely determines the value of attributes supplier name,supplier status and city This can be express by

Supplier.supplier\_id→supplier.sname

Supplier.supplier\_id→supplier.status

Supplier.supplier\_id→supplier.city

Or simply,

supplier\_id→ sname

supplier\_id→ status

supplier\_id→city

Question: is following functional dependency is valid ?

sname→status

sname→city

**Types of functional dependencies**

**Fully functionally dependency**

For a given relation schema R, FD X→Y, Y is said to be fully functionally dependent on X if there

is no Z (where Z is a proper subset of X) such that Z→Y.

**Example:**

Let us consider relational schema R=(A,B,C,D,E,H) with the FDs

F={A→BC,CD→E,C→E)

Here, the FD A→BC is left reduced, so clearly, BC is fully functionally dependent on A

(because there is no possible proper subset of only element A)

● Here, the FDs CD→E, C→E where E is functionally dependent on CD and again E is

functionally dependent on subset of C. That is C (i.e. C→E). Hence E is not fully

functionally dependent on CD.

Example: Consider a relation sales

**Sales (product\_id#,sales\_date#,quantity,product\_name)**

With the following functional dependencies

F={product\_id,sales\_date→quantity, product\_id→quantity, product\_id→product\_name}

**●** Here,. FDs product\_id,sales\_date→quantity, product\_id→quantity, quantity is not fully

functional dependent on product\_is,sales\_date.

**●**  Here, functional dependency product\_id→product\_name, product\_name is fully functional

dependent on product\_id.

**Partial functional dependency**

For a given relation schema R with set of functional dependency F on attribute of R. Let K as a

candidate key in R. if X is a proper subset of K and X→A then A is said to be partially dependent on K.

Example: Consider a relation schema ‘student\_course\_info’

student\_course\_info(name#,course#,grade,phone\_no,major,course\_department)

with the following FDs

{

name→phone\_no,major

course→course\_department,

name,course→grade

}

Here {name,course} is a candidate key. Here grade is fully functionally dependent on

{name,course}. If there is a possible FD name→grade then we can not say grade is fully functionally dependent on {name,course}. Here phone\_no, major and course\_department are

partially dependent on {name,course}

**Transitive dependency**

For a given relational schema R with set of functional dependency F. Let X and Y be the subset of

r and Let A be the attribute of R s.t. X ⊄ Y, A ⊄ XY. If the functional dependencies {X→Y, Y→A}

implies by F (i.e. X→Y→A) then A is said to be transitively dependent on X.

**Advantages or importance of Normalization**

Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:

* Minimize data redundancy i.e. no unnecessarily duplication of data.
* To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
* Data should be consistent throughout the database i.e. it should not suffer from following anomalies.
* **Insert Anomaly** - Due to lack of data i.e., all the data available for insertion such that null values in keys should be avoided. This kind of anomaly can seriously damage a database
* **Update Anomaly** - It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.
* Deletion Anomaly - It leads to loss of data for rows that are not stored else where. It could result in loss of vital data.
* Complex queries required by the user should be easy to handle.
* On decomposition of a relation into smaller relations with fewer attributes on normalization the resulting relations whenever joined must result in the same relation without any extra rows. The join operations can be performed in any order. This is known as Lossless Join decomposition.
* The resulting relations (tables) obtained on normalization should possess the properties such as each row must be identified by a unique key, no repeating groups, homogenous columns, each column is assigned a unique name etc.

**DISADVANTAGES OF NORMALIZATION**

The following are disadvantages of normalization.

* You cannot start building the database before you know what the user needs.
* On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the

performance degrades.

* It is very time consuming and difficult process in normalizing relations of

higher degree.

* Careless decomposition may leads to bad design of database which may

leads to serious problems.

**First Normal Form**

**First normal form (1NF**) is now considered to be part of the formal definition of a relation in the basic (flat) relational model; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a *single tuple.* In other words, 1NF disallows *relations within relations* or *relations as attribute values within tuples*. The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.

Consider the DEPARTMENT relation schema shown in Figure (a) whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute as shown in Figure (b). We assume that each department can have *a number of* locations. The DEPARTMENT schema and a sample relation state are shown in Figure . As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure (b).

|  |  |  |
| --- | --- | --- |
| Dname | Dnumber | Dmsr\_ssn |

Figure(a)

|  |  |  |  |
| --- | --- | --- | --- |
| Dname | Dnumber | Dmsr\_ssn | location |
| Research | 5 | 333445555 | {Bellaire,Sugarland, Houston} |
| Administration | 4 | 987654321 | {Stafford} |
| Headquarters | 1 | 888665555 | {Houston} |

Figure(b)

In this case, the DEPARTMENT relation in Figure is not in 1NF; in fact, it does

not even qualify as a relation according to our definition of relation

**solution**

Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT\_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation}, as shown in Figure . A distinct tuple in DEPT\_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.

|  |  |  |
| --- | --- | --- |
| Dname | Dnumber | Dmsr\_ssn |
| Research | 5 | 333445555 |
| Administration | 4 | 987654321 |
| Headquarters | 1 | 888665555 |

Department

|  |  |
| --- | --- |
| Dnumber | Loacation |
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |
|  |  |

Department location

**Second normal form (2NF)** is based on the concept of *full functional dependency.* A

functional dependency *X* *Y* is a **full functional dependency** if removal of any

attribute *A* from *X* means that the dependency does not hold any more; that is, for

any attribute *A* *X*, (*X* – {*A*}) does *not* functionally determine *Y*. A functional

dependency *X**Y* is a **partial dependency** if some attribute *A* *X* can be removed

from *X* and the dependency still holds; that is, for some *A* *X*, (*X* – {*A*}) *Y*. In

Figure below(a) , {Ssn, Pnumber} Hours is a full dependency (neither Ssn Hours

nor Pnumber****Hours holds). However, the dependency {Ssn, Pnumber}Ename is

partial because SsnEname holds.

**Definition.** A relation schema *R* is in 2NF if every nonprime attribute *A* in *R* is *fully functionally dependent* on the primary key of *R*.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. The EMP\_PROJ relation in Figure below (a) is in 1NF but is not in 2NF. The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3. The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key {Ssn, Pnumber} of EMP\_PROJ, thus violating the 2NF test.

|  |
| --- |
| SSN Pnumber Hours Ename Pname plocation  FD1  FD2  FD3  FD2  FD3 |
| SSN pnumber Hourse  SSN pnumber Hourse  EP1  Ssn ename  Pnumber pname plocatioin |
|  |

**Third Normal Form [3FN] -:** According to Codd’s original definition, a relation schema *R* is in **3NF** if it satisfies 2NF *and* no nonprime attribute of *R* is transitively dependent on the primary key. The relation schema EMP\_DEPT in Figure given below is in 2NF, since no partial dependencies on a key exist. However, EMP\_DEPT is not in 3NF because of the transitive dependency of Dmgr\_ssn (and also Dname) on Ssn via Dnumber. We can normalize EMP\_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Figure given below. Intuitively, we see that ED1 and ED2 represent independent entity facts about employees and departments. A NATURAL JOIN operation on ED1 and ED2

Definition. According to Codd’s original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

**EMP-DEPT**

|  |
| --- |
|  |

**ED1**

|  |
| --- |
|  |

**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 |

**Lossless Decomposition :-**

Lossless Join and Dependency Preserving Decomposition

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

**Lossless Join Decomposition**

If we decompose a relation R into relations R1 and R2,

* Decomposition is lossy if R1 ⋈ R2 ⊃ R
* Decomposition is lossless if R1 ⋈ R2 = R

**To check for lossless join decomposition using FD set, following conditions must hold:**

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

Att(R1) U Att(R2) = Att(R)

1. Intersection of Attributes of R1 and R2 must not be NULL.

Att(R1) ∩ Att(R2) ≠ Φ

1. Common attribute must be a key for at least one relation (R1 or R2)

Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.