|  |  |
| --- | --- |
|  | **WEEK—4 Lecture-3 hr** |
|  | **Functional dependency:** |
|  | **Functional dependency: overview** |
|  |  |
|  | * **Functional Dependency** is a relationship that exists between multiple attributes of a relation. * This concept is given by **E. F. Codd.** * It is a type of constraint existing between various attributes of a relation. * It is used to define various normal forms. * These dependencies are restrictions imposed on the data in database. * The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table. * It typically exists between the primary key and non-key attribute within a table.   X   →   Y  The left side of FD is known as a **determinant**, the right side of the production is known as a **dependent.**  **For example:**  Assume we have an employee table with attributes: **Emp\_Id, Emp\_Name, Emp\_Address.**  Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.  Functional dependency can be written as:   1. Emp\_Id → Emp\_Name   We can say that Emp\_Name is functionally dependent on Emp\_Id.  **Importance of functional dependency**   * Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database. * It helps you to maintain the quality of data in the database. * It helps you to defined meanings and constraints of databases. * It helps you to identify bad designs. * It helps you to find the facts regarding the database design. |
|  | **Rules of Functional Dependencies** |
|  | 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.   1. If X ⊇ Y then X  →    Y   **Example:**   1. X = {a, b, c, d, e} 2. Y = {a, b, c}  2. Augmentation Rule (IR2) The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.   1. If X  →  Y then XZ   →   YZ   **Example:**   1. For R(ABCD),  **if** A   →   B then AC  →   BC  3. Transitive Rule (IR3) In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.   1. If X  →   Y and Y →  Z then X →   Z  4. Union Rule (IR4) Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.   1. If X  →  Y and X  →  Z then X →  YZ   **Proof:**  1. X → Y (given) 2. X → Z (given) 3. X → XY (using IR2 on 1 by augmentation with X. Where XX = X) 4. XY → YZ (using IR2 on 2 by augmentation with Y) 5. X → YZ (using IR3 on 3 and 4) 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   **Proof:**  1. X → Y (given) 2. WY → Z (given) 3. WX → WY (using IR2 on 1 by augmenting with W) 4. WX → Z (using IR3 on 3 and 2) |
|  | **Types Functional Dependencies** |
|  | There are mainly four types of Functional Dependency in DBMS. Following are the types of Functional Dependencies in DBMS:   1. **Multivalued Dependency** 2. **Trivial Functional Dependency** 3. **Non-Trivial Functional Dependency** 4. **Transitive Dependency**   **Multivalued Dependency in DBMS**   * Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. * A multivalued dependency is a complete constraint between two sets of attributes in a relation. * It requires that certain tuples be present in a relation. * Consider the following Multivalued Dependency Example to understand.   **Example:**   |  |  |  | | --- | --- | --- | | Car\_model | Maf\_year | Color | | H001 | 2017 | Metallic | | H001 | 2017 | Green | | H005 | 2018 | Metallic | | H005 | 2018 | Blue | | H010 | 2015 | Metallic | | H033 | 2012 | Gray |   In this example, **Maf\_year** and **Color** are independent of each other but dependent on car\_model. In this example, these two columns are said to be multivalue dependent on car\_model.  This dependence can be represented like this:  car\_model -> **Maf\_year**  car\_model-> **Color**  **Trivial Functional Dependency in DBMS**  The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.  So, X -> Y is a trivial functional dependency if Y is a subset of X. Let’s understand with a Trivial Functional Dependency Example.  **For example:**   |  |  | | --- | --- | | Emp\_id | Emp\_name | | AS555 | Harry | | AS811 | George | | AS999 | Kevin |   Consider this table of with two columns Emp\_id and Emp\_name.  {Emp\_id, Emp\_name} -> Emp\_id is a trivial functional dependency as Emp\_id is a subset of {Emp\_id,Emp\_name}.  **Non Trivial Functional Dependency in DBMS**  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.   |  |  |  | | --- | --- | --- | | Company | CEO | Age | | Microsoft | Satya Nadella | 51 | | Google | Sundar Pichai | 46 | | Apple | Tim Cook | 57 |   **Example:**  (Company} -> {CEO} (if we know the Company, we knows the CEO name)  But CEO is not a subset of Company, and hence it’s non-trivial functional dependency.  **Transitive Dependency in DBMS**  A Transitive Dependency is a type of functional dependency which happens when “t” is indirectly formed by two functional dependencies.  Let’s understand with the following Transitive Dependency Example. Example:  |  |  |  | | --- | --- | --- | | Company | CEO | Age | | Microsoft | Satya Nadella | 51 | | Google | Sundar Pichai | 46 | | Alibaba | Jack Ma | 54 |   {Company} -> {CEO} (if we know the compay, we know its CEO’s name)  {CEO } -> {Age} If we know the CEO, we know the Age  Therefore according to the rule of rule of transitive dependency:  { Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.  **Note: transitive dependency can only occur in a relation of three or more attributes.** |
|  | **Normalization: normalization process** |
|  | **Importance of normalization** |
|  |  |
|  | Resolving the database anomalies The forms of Normalization i.e. 1NF, 2NF, 3NF, BCF, 4NF and 5NF remove all the Insert, Update and Delete anomalies.  **Insertion Anomaly**occurs when you try to insert data in a record that does not exist.  **Deletion Anomaly** is when a data is to be deleted and due to the poor deign of database, other record also deletes. Eliminate Redundancy of Data Storing same data item multiple times is known as Data Redundancy. A normalized table do not have the issue of redundancy of data. Data Dependency The data gets stored in the correct table and ensures normalization. Isolation of Data A good designed database states that the changes in one table or field do not affect other. This is achieved through Normalization. Data Consistency While updating if a record is left, it can led to inconsistent data, Normalization resolves it and ensures Data Consistency. |
|  | **1NF,2NF,3NF** |
|  | First Normal Form (1NF)  1. A relation will be 1NF if it contains an atomic value. 2. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute. 3. 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 |   We can observe that although a few values are getting repeated but values for the EMP\_PHONE column are now atomic for each record/row.  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)**  For a table to be in the Second Normal Form, it must satisfy two conditions:   1. The table should be in the First Normal Form. 2. There should be no Partial Dependency.   **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 |   **NOTE:**   1. For a table to be in the Second Normal form, it should be in the First Normal form and it should not have Partial Dependency. 2. Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key. 3. To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.  Third Normal Form (3NF)  1. A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency. 2. 3NF is used to reduce the data duplication. It is also used to achieve the data integrity. 3. 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:**  we have 3 tables, **Student**, **Subject** and **Score**.  **Student Table**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | student\_id | name | reg\_no | branch | address | | 10 | Akon | 07-WY | CSE | Kerala | | 11 | Akon | 08-WY | IT | Gujarat | | 12 | Bkon | 09-WY | IT | Rajasthan |   **Subject Table**   |  |  |  | | --- | --- | --- | | subject\_id | subject\_name | teacher | | 1 | Java | Java Teacher | | 2 | C++ | C++ Teacher | | 3 | Php | Php Teacher |   **Score Table**   |  |  |  |  | | --- | --- | --- | --- | | score\_id | student\_id | subject\_id | marks | | 1 | 10 | 1 | 70 | | 2 | 10 | 2 | 75 | | 3 | 11 | 1 | 80 |   In the Score table, we need to store some more information, which is the exam name and total marks.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | score\_id | student\_id | subject\_id | marks | exam\_name | total\_marks | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  |   With exam\_name and total\_marks added to Score table, it saves more data now.  Primary key for Score table is a composite key, which is made up of two attributes or columns → **student\_id + subject\_id**.  new column exam\_name depends on both student and subject.  For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Prctical exams and for some you don't. So we can say that exam\_name is dependent on both student\_id and subject\_id.  The column total\_marks depends on exam\_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.  But, exam\_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total\_marks depends on it.  This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.  We need to remove **Transitive Dependency** here.  the solution is very simple. Take out the columns exam\_name and total\_marks from Score table and put them in an **Exam** table and use the exam\_id wherever required.  **Score Table: In 3rd Normal Form**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | *score\_id* | *student\_id* | *subject\_id* | *marks* | *exam\_id* | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   **The new Exam table**   |  |  |  | | --- | --- | --- | | exam\_id | exam\_name | total\_marks | | 1 | Workshop | 200 | | 2 | Mains | 70 | | 3 | Practicals | 30 |   The advantage of removing transitive dependency is,   * Amount of data duplication is reduced. * Data integrity achieved.   Consider another 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)**   1. BCNF is the advance version of 3NF. It is stricter than 3NF. 2. A table is in BCNF if every functional dependency X → Y, X is the super key of the table. 3. 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. Fourth normal form (4NF)  1. A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency. 2. 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 |  Fifth normal form (5NF)  1. A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless. 2. 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy. 3. 5NF is also known as Project-join normal form (PJ/NF).  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 | |
|  |  |