Unit-2 Gaining Confidence-RDBMS

Structure

1. Introduction

2. Objectives

3. **Lesson-1**

    The Relational Model

    Domains, Attributes, Tuple and Relation

    Super keys Candidate keys and Primary keys

    Relational Constraints

    Domain Constraint

    Key Constraint

    Integrity Constraint

    Insert Operations and Dealing with Constraint Violations

    Delete Operations and Dealing with Constraint Violations

    Update Operations and Dealing with Constraint Violations

4. **Lesson-2**

    Entity Relationship (ER) Model

    Entities

    Attributes

    Relationships

Lossless-join Decomposition

Dependency Preservation

Lack of redundancy

Rules of Data Normalization

Eliminate Repeating Groups

Eliminate Redundant Data

Eliminate Columns Not Dependent on Key

8. Summary

9. Exercise

10. Suggested readings

Introduction:

In unit-1, we discussed about Database management System, its advantages, characteristics, architecture, etc. This unit is an attempt to provide you information about relational and E-R models. The relational model is a widely used model for DBMS implementation. In lesson-1we will discuss the terminology, operators and operations used in relational model. Relations must satisfy some properties, such as no duplicate tuples, no ordering of tuples, and atomic attributes, etc. Relations that satisfy these basic requirements may still have some undesirable characteristics such as data redundancy, and anomalies. What are these undesirable characteristics and how can we eliminate them from the database system? We will make an attempt to answer this question too. However most of these undesirable properties do not arise if the database modeling has been carried out very carefully using some technique like the Entity-Relationship Model that we will be discussing in the this unit. It is still important to use the Integrity constraints to check the database that has been obtained and ensure that no mistakes have been made in modeling. The central concept in these discussions is the concept of Database integrity (discussed in lesson-4), the notion of functional dependency, which depends on understanding the semantics of the data and which deals with what information in a relation is dependent on what other information in the relation.

We will also discuss, in this unit (in lesson-2), the E-R model, which primarily is a semantic model and is very useful in creating raw database design that can be further refined. We discuss Entity Relationship diagrams and their conversion to databases also. In lessons 3we will discusses how to convert ERDs into Relational database and in lesson 5, redundancy, its associated problems and Normalization are discussed.

Objectives

After going through this unit, you should be able to:

- Describe relational model and its advantages;
- Draw E-R diagrams for given problems;
- Normalize data and describe various Normal forms
- Convert an E-R diagram to a relational database and vice versa.
- Define the concept of entity integrity and measures to check the Integrity of data
- Describe relational database referential integrity constraints
- Define the concept of functional dependency
- Show how to use the dependency information to decompose relations

Lesson-1

The Relational Model

A model is a representation of something (A mathematical or graphical representation of some problem). In DBMS model basically defines the structure or organization of data and a set of operations on that data. Relational model is a simple model in which database is represented as a collection of two dimensional tables called "Relations". Thus, because of its simplicity, Relational model is most commonly used model.

Below is shown a relation named EMP showing simplicity of relational design:

E_ID	E_Name	E_Age	E_Address
1001	Mushtaq Ahmad	35	DDE, UOK, Sgr
1002	Tanveer Ahmad	34	DBFS, UOK, Sgr
1003	Tariq Ahmad	35	TBS, UOK, Sgr
1004	Rafi Ahmad	37	TBS, UOK, Sgr
1005	Muzaffar Rasool	30	DCS, IUST, Pul

Below are some of the advantages of relational model:

- **Ease of use:** The simple tabular representation of database helps the user define and query the database conveniently. For example, you can easily find out the age of the person whose first name is "Rafi".
- **Flexibility:** Since the database is a collection of tables, new data can be added and deleted easily. Also, manipulation of data from various tables can be done easily using various basic operations. For example, we can add a telephone number field in the above table.

- **Accuracy:** In relational databases the relational algebraic operations are used to manipulate data. These are mathematical operations and ensure accuracy as compared to other models.

Domain, Attribute, Tuple and Relation

Before we discuss the relational model in more detail, let us first define some very basic terms and concepts associated with RDBMS.

- **Tuple:** Each row in a table (a table generally represents a real world object or what is called an entity and a table in RDBMS is called a Relation) represents a record and is called a tuple. A table containing 'n' attributes in a record is called n-tuple.
- **Attribute:** The name of each column in a table is used as its identifier and is called an attribute (a characteristic feature of an entity).

For example, following table shows a relation named EMP(employee). The columns E_ID, E_Name, E_Age, E_Address and E_phone are the attributes of employee and each row in the table represents a separate tuple or record (viz 1001, Mushtaq Ahmad……).

Relation Name: EMP

E_ID	E_Name	E_Age	E_Address	E_phone
1001	Mushtaq Ahmad	35	DDE, UOK, Sgr	9419000101
1002	Tanveer Ahmad	34	DBFS, UOK, Sgr	9419000102
1003	Tariq Ahmad	35	TBS, UOK, Sgr	9906526072
1004	Rafi Ahmad	37	TBS, UOK, Sgr	9900000101
1005	Muzaffar Rasool	30	DCS, IUST, PUL	9900000102

- **Domain:** A domain is a set or range of permissible values that can be given to an attribute. So every attribute in a table has a specific domain. Values to these attributes cannot be assigned outside their domains. In the example above if domain of E_ID is a set of integer values from 1001 to 1099, then a value outside this range will not be valid. Some other common domains may be age between 18 and 37. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. However, the domains can also be non-contiguous. For example, the enrolment number of NET exam conducted by UGC has the first two digits as the centre number, thus the ten-digit enrolment numbers are non-continuous.
- **Relation:** A relation consists of:
  1) Relational Schema
  2) Relation instance

Relational Schema: A relational schema specifies the relation's name, its attributes and the domain of each attribute. If R is the name of a relation and A1, A2…An is a list of attributes representing R then R(A1, A2…An) is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called Domain (Ai). For example, the relational schema for relation EMP as in Figure 1 will be:

EMP (E_ID: number/integer, E_Name: character/string, E_Age: number/integer, E_Address: character/string)

Total number of attributes in a relation denotes the degree of a relation. Since the EMP relation contains four attributes, so this relation is of degree 4.

A relation schema is generally denoted by a Schema Diagram, which is a graphical representation of a schema. The schema Diagram of EMP relation is shown below:

EMP

E_ID	*E_Name*	*E_Age*	*E_Address*

Relation Instance or Relation State: A relation instance denoted as **r** is a collection of tuples for a given relational schema at a specific point of time. A relation state **r** of the relation schema R (A1,A2,…….AN), also denoted by **r**(R) is a set of n-tuples

$$r = \{t1,t2,\ldots\ldots\ldots tm\}$$

Where each n-tuple is an ordered list of n values

$$t = <v1,v2,\ldots\ldots.., vn>$$

Where each vi belongs to domain (Ai) or contains null values.

The relation schema is also called '**intension**' and relation state is also called '**extension**'.

Let us elaborate the definitions above with the help of examples:

Example 1:

RELATION SCHEMA for STUDENT entity or relation:

STUDENT (RollNo: number, Name: character, Course: character, Age: number)

RELATION INSTANCE of STUDENT relation:

STUDENT

	RollNo	Name	Course	Age
t1	101	Syed Tajali	IMBA	19
t2	102	Musadiq Noor	IMBA	19
t3				

103	Afsha Tariq	IMBA	18

Where t1 = (101, Syed Tajali, IMBA, 19) for this relation instance, m = 3 and n = 4.

Example 2:

RELATIONAL SCHEMA for EMP relation:

EMP (E_ID: number, NAME: character, SALARY: float, ADDRESS: character)

RELATION INSTANCE for EMP relation:

EMP

	E_ID	NAME	SALARY	ADDRESS
t1	1001	Dr. Mushtaq	40000	DDE, UOK, Sgr
t2	1002	Dr. Tanveer	30000	DDE, UOK, Sgr

In this instance, m = 2 and n = 4

Thus current relation state reflects only the valid tuples that represent a particular state of the real world. However, Null values can be assigned for the cases where the values are unknown or missing.

Ordering of tuples

In a relation, tuples are not inserted in any specific order. Ordering of tuples is not defined as a part of a relation definition. However, records may be organized later according to some attribute value in the storage systems. For example, records in EMP table may be organized according to E_ID. Such data or organization depends on the requirement of the underlying database application. However, for the purpose of display

we may get them displayed in the sorted order of age. The following table is sorted by age. It is also worth mentioning here that relational model does not allow duplicate tuples.

EMP

E_ID	E_Name	E_Age	E_Address	E_phone
1005	Muzaffar Rasool	30	DCS, IUST, PUL	990000102
1002	Tanveer Ahmad	34	DBFS, UOK, Sgr	941900103
1004	Rafi Ahmad	37	TBS, UOK, Sgr	990000101

Super Keys, Candidate Keys and Primary Keys for the Relations

As discussed in the previous section ordering of tuples in relations does not matter and all tuples in a relation are unique. However, can we uniquely identify a tuple in a relation? How? Let us discuss the concepts of keys that are primarily used for the above purpose.

Super Keys: A super key is an attribute or set of attributes used to identify the records uniquely in a relation. For Example, in the Relation EMP described earlier E_ID is a super key since E_ID is unique for each person. Similarly (E_ID, E_Age) and (E_ID, NAME) are also super keys of the relation EMP since their combination is also unique for each record.

Candidate keys: Super keys of a relation can contain extra attributes i.e. it can be a combination of many attributes or a composite key. Candidate keys are minimal super keys, i.e. such a key contains no extraneous attributes. An attribute is called extraneous if even after removing it from the key, the remaining attributes still have the properties of a key. The following properties must be satisfied by the candidate keys:

- A candidate key must be unique.
- A candidate key's value must exist. It cannot be null. (This is also called entity integrity rule)
- A candidate key is a minimal set of attributes.
- The value of a candidate key must be stable. Its value cannot change outside the control of the system.

Primary Keys: A relation can have more than one candidate keys and one of them can be chosen as a primary key (A key that uniquely identifies each tuple in the relation). For example, in the relation EMP the two possible candidate keys are E-ID and NAME (assuming names are unique in the table). E-ID may be chosen as the primary key.

RELATIONAL CONSTRAINTS

A constraint is a business rule or a real life limitation on data stored in the database. There are three types of constraints on relational database viz.:

- DOMAIN CONSTRAINTS
- PRIMARY KEY CONSTRAINTS
- INTEGRITY CONSTRAINTS

Domain Constraints: These specify that each attribute in a relation must contain an atomic value only from the corresponding domains. The data types associated with commercial RDBMS domains include:

1. Standard numeric data types for integer (such as short- integer, integer, long integer)
2. Real numbers (float, double precision floats)
3. Characters
4. Fixed length strings and variable length strings.

Thus, domain constraints specify the conditions that we want to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column. For example, consider the relation:

EMP

	E_ID	NAME	AGE	SALARY	ADDRESS
t1	1001	Dr. Mushtaq	35	40000	DDE, UOK, Sgr
t2	1002	Dr. Tanveer	34	30000	DDE, UOK, Sgr

In the relation above, AGE of an Employee always belongs to the integer domain within a specified range (e.g. 18<=age<=65), and not to strings or any other domain. Within a domain non-atomic values should be avoided. This sometimes cannot be checked by domain constraints. For example, a database which has area code and phone numbers as two different fields will take phone numbers as-

Country code	Phone No.
0191	9906526072

A non-atomic value in this case for a phone can be 01919906526072, however, this value can be accepted by the Phone field.

Key Constraint: This constraint states that the key attribute value in each tuple must be unique, i.e., no two tuples contain the same value for the key attribute. This is because the value of the primary key is used to uniquely identify the tuples in the relation.

Example 3:

If A is the key attribute in the following relation R then A1, A2 and A3 must be unique.

R

A	B
A1	B1
A3	B2
A2	B2

Example 4:

In relation EMP,E_ID is the primary key so E_ID cannot be given same value for two persons.

Integrity Constraints: There are two types of integrity constraints:

- Entity Integrity Constraint
- Referential Integrity Constraint

Entity Integrity Constraint:

It states that no primary key value can be null. This is because the primary key is used to uniquely identify individual tuples in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes. This constraint is specified on one individual relation.

Example 5:

Let R be the relation as shown below:

E_ID	**E_Name**	**E_Age**	**E_Address**	**E_phone**
NULL	Mushtaq Ahmad	35	DDE, UOK, Sgr	9419000101

1002	Tanveer Ahmad	34	DBFS, UOK, Sgr	9419000102
NULL	Mushtaq Ahmad	35	DDE, UOK, Sgr	9906526072
1004	Rafi Ahmad	37	TBS, UOK, Sgr	9900000101
1005	Muzaffar Rasool	30	DCS, IUST, PUL	9900000102
NULL	Asif Mohammad	25	DCS, UOK, Sgr	9858120001

Note: "XXX" identifies the Primary key of a relation.

In the relation R above, the primary key has null values in the tuples t1 & t3. NULL value in primary key is not permitted, thus, relation instance is an invalid instance.

Referential integrity constraint:

It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraint is generally specified on two relations (not necessarily distinct). It uses a concept of foreign key and has been dealt with in more detail in the next unit.

Employee

E_no	**E_name**	**E_sal**	***E_depno***
1001	Tariq Ahmad	40000	10
1002	Mushtaq Ahmad	39000	40
1003	Tanveer Ahmad	25000	40
1009	Kaiser Rashid	44000	20

Department

Depno	**Depname**	**Dep_head**	**D_location**
10	The Business School	Prof. Shabir	University of Kashmir
20	Distance Education	Prof. Neelofar	University of Kashmir
30	Islamic Studies	Prof. Hamid	University of Kashmir
40	Business & Financial Studies	Prof. Nazir	University of Kashmir
50	Computer Science	Dr. Khursheed	University of Kashmir

Note:

1) '<u>XXXX</u>' identifies the Primary key of a relation.

2) '*IIII*' (Italic) identifies the Foreign key of a relation.

In the example above, the value of ***E_depno*** in every **Employee** tuple is matching with the value of **Depno** in some **Department** tuple. If a tuple having values (60, 70) is added then it is invalid since referenced relation **Department** doesn't include 60 or 70. Thus, it will be a violation of referential integrity constraint.

Dealing with Constraint Violations: while performing operations on data, constraints may get violated. There are three basic operations that can be performed on relations:

- Insertion
- Deletion
- Update

INSERT Operation:

The insert operation allows us to insert a new tuple in a relation. When we try to insert a new record, then any of the following four types of constraints can be violated:

- Domain constraint: If the value given to an attribute lies outside the domain of that attribute.
- Key constraint: If the value of the key attribute in new tuple **t** is the same as in the existing tuple in relation **R**.
- Entity Integrity constraint: If the primary key attribute value of new tuple **t** is given as null.
- Referential Integrity constraint: If the value of the foreign key in **t** refers to a tuple that doesn't appear in the referenced relation.

Dealing with constraints violation during insertion:

If the insertion violates one or more constraints, then two options are available:

- Default option: - Insertion can be rejected and the reason of rejection can also be explained to the user by DBMS.
- Ask the user to correct the data and resubmit it.

Example 6:

Consider the Relations **Employee and Department**:

E_no	**E_name**	**E_sal**	***E_depno***
1001	Tariq Ahmad	40000	10
1002	Mushtaq Ahmad	39000	40
1003	Tanveer Ahmad	25000	40
1009	Kaiser Rashid	44000	20

Depno	**Depname**	**Dep_head**	**D_location**
10	The Business School	Prof. Shabir	University of Kashmir
20	Distance Education	Prof. Neelofar	University of Kashmir
30	Islamic Studies	Prof. Hamid	University of Kashmir
40	Business & Financial Studies	Prof. Nazir	University of Kashmir
50	Computer Science	Dr. Khursheed	University of Kashmir

(1) Insert into employee values (1001, 'jasim', 18000, 10);

Violated constraint: - Key constraint

Reason: - Primary key 1001 already exists.

Solution: - DBMS could ask the user to provide valid E_no value and accept the insertion if valid employee number is provided.

(2) Insert into employee values (NULL, 'jasim', 18000, 10);

Violated constraint: - Entity Integrity constraint

Reason: - Primary key is 'null'.

Solution: - DBMS could ask the user to provide valid E_no value and accept the insertion if valid employee number is provided.

(3) Insert into employee values (e101, 'jasim', 18000, 10);

Violated constraint: - Domain constraint

Reason: - value of E_no is alphanumeric, which is not valid.

Solution: - Provide a valid value from within the numeric domain

(4) Insert into employee values (1019, 'jasim', 18000, 60);

Violated constraint: - Referential Integrity Constraint

Reason: - Department number 60 does not exist.

Solution: - Either set the value to NULL or assign an existing department number to the employee or add a record to Department table with Depno 60 if possible.

(5) Insert into employee values (1019, 'jasim', 18000, 20);

Violated constraint: - None

The Deletion Operation:

Using the delete operation one or more records can be deleted from a relation. To delete some specific records from the database a condition is also specified based on which records can be selected for deletion.

Constraints that can be violated during deletion: Only one type of constraint can be violated during deletion that is the referential integrity constraint. It can occur when you want to delete a record in the table where it is referenced by the foreign key of another table.

Solution: - If the deletion operation violates referential integrity constraint, then three options are available:

- Default option: - Reject the deletion. It is the job of the DBMS to explain to the user why the deletion was rejected.
- Attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted.
- Change the value of referencing attribute that causes the violation.

The Update Operations:

Update operations are used for modifying database values. The constraint violations faced by this operation are logically the same as the problem faced by Insertion and Deletion Operations. Therefore, we will not discuss this operation here.

Lesson-2

ENTITY RELATIONSHIP (ER) MODEL

Let us first look into some of the main features of ER model.

- Entity relationship model is a high-level conceptual data model.
- It allows us to describe the real-world data in terms of objects and their relationships.
- It is widely used to develop an initial design of a database.
- It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system.

- It describes data as a collection of entities, relationships and attributes.

Let us explain all this with the help of an example application.

Here we will describe an example database application containing COLLEGE database and use it in illustrating various E-R modeling concepts. College database keeps track of Students, Staff, Departments and Courses organized by various departments. Following is the description of COLLEGE database:

College contains various departments like Department of English, Department of Hindi, and Department of Computer Science etc. Each department is assigned a unique ID and Name. Some staff members are also appointed in each department and one of them works as head of the department. There are various courses conducted by each department. Each course is assigned a unique ID, Name and Duration. Below is given a brief introduction of each of these entities:

- Staff/Faculty information contains faculty identification number, name, address, department, basic salary etc. A faculty member is assigned to only one department but can teach various courses of other department also.
- Student's information contain Roll no, Name, Address, course opted etc. A student can opt only for one course.
- Department information contains department name, department number, head of the department and location etc. One department will be headed by only one person.
- Course information contains Course number, title, duration etc.
- Parent (guardian) information is also kept along with each student. We keep each guardian's name, age, sex and address.

Entities: An entity is an object of concern used to represent the things in the real world having physical or logical existence e.g., car, table, book, software program, loan etc. It

represents a class of things, not any one instance, e.g., 'STUDENT' entity has instances of 'Rayees' and 'Muheet'.

Entity Set or Entity Type: A collection of similar kind of entities is called an Entity Set or entity type. E.g. for the COLLEGE database described above objects of concern are Students, Faculty, Course and departments. The collections of all the students' entities form entity set STUDENT. Similarly collection of all the courses form an entity set COURSE. Entity sets need not be disjoint. For example – an entity may be part of the entity set STUDENT, the entity set EMPLOYEE, and the entity set PERSON.

Entity identifier key attributes: An entity type usually has an attribute or a group of attributes whose values are distinct for each individual entity in the collection. Such an attribute or set of attributes is called a key attribute and its values can be used to identify each entity uniquely. E.g. E_ID, D_NO, SSN, ISBN

Strong entity set: The entity types containing a key attribute are called strong entity types or regular entity types. E.g. The EMPLOYEE entity has a key attribute E_ID which uniquely identifies it, hence is a strong entity.

Attributes: An attribute is a property or characteristic used to describe the specific feature of the entity. So to describe an entity entirely, a set of attributes is used that further describe the entity. For example, a student entity may be described by the student's name, age, address, course etc.

An entity will have a value for each of its attributes. For example for a particular employee the following values can be assigned:

E_ID: 1002

E_name: Tanveer

E_sal: 18000

E_depno: 10

Domain: Each attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain. E.g. for DEPARTMRNT entity D_NO has a specific domain, integer values say from the set {10, 20, 30, 40, 50, . .}.

Types of attributes: Attributes attached to an entity can be of various types. Below we discuss the different types of these attributes.

Simple: An atomic attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: The 'First name', 'Last name', age attributes of an employee entity represent simple attributes.

Composite: Attributes that can be further divided into smaller units and each individual unit contains a specific meaning. For example:-The NAME attribute of an employee entity can be sub-divided into First name, Last name and Middle name.

Single valued: Attributes having a single value for a particular entity. For Example, Age is a single valued attribute of an employee entity.

Multi valued: Attributes that have more than one values for a particular entity is called a multi valued attribute. Different entities may have different number of values for these kinds of attributes. For multi valued attributes we must also specify the minimum and maximum number of vales that can be attached. For Example phone number for employee entity is a multi valued attribute as an employee can have a home phone, an office phone and a mobile phone.

Stored: Attributes that are directly stored in the data base. For example, 'Date of Birth' or 'Date of Employment' attribute of an employee.

Derived: Attributes that are not stored directly but can be derived from stored attributes are called derived attributes. For Example, age of a student can be calculated from his 'DOB' or years of services of an employee entity can be determined from the current date

and the date of joining of the employee. Similarly, total salary of a person can be calculated from his 'basic salary' attribute.

Relationships: A relationship can be defined as:

- A connection or set of associations, or
- A rule for communication among entities:

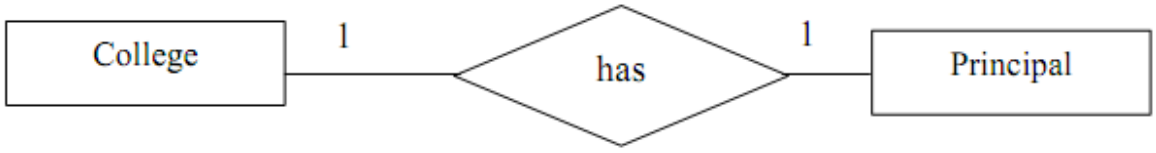
E.g. In College database, the association between student and course entity, i.e., "Student opts course" is an example of a relationship.

Relationship sets: A relationship set is a set of relationships of the same type. For example, consider the relationship between two entity sets student and course. Collection of all the instances of relationship sets forms a relationship set called relationship type.

Degree: The degree of a relationship type is the number of participating entity types. The relationship between two entities is called binary relationship. A relationship among three entities is called ternary relationship. Similarly relationship among n entities is called **n**-ary relationship. E.g. Relationship between employee and department entities is a binary relationship.

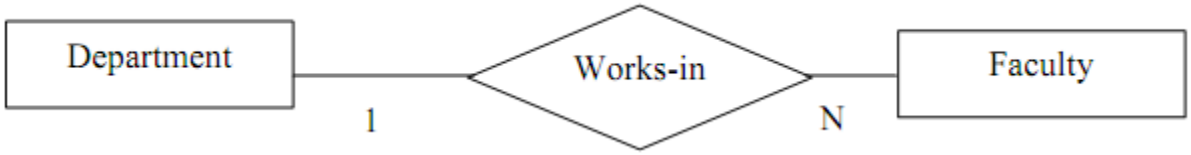
Relationship Cardinality: Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality, binary relationship can be further classified into the following categories:

- **One-to-one:** An instance in entity A is associated with at most one instance in entity B, and an instance in entity B is associated with at most one instance in entity A. E.g. Relationship between college and principal



One college can have at the most one principal and one principal can be assigned to only one college. Similarly we can define the relationship between a country and a prime minister.

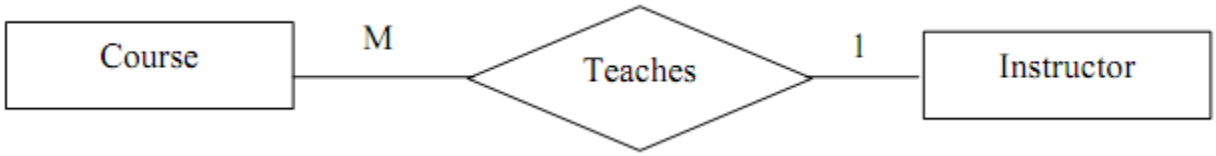
- **One-to-many:** An instance in entity A is associated with any number of instances of entity B. One instance in entity B is associated with at the most one instance in entity A. E.g. Relationship between department and employee (faculty).



One department can appoint any number of faculty members but a faculty member is assigned to only one department.

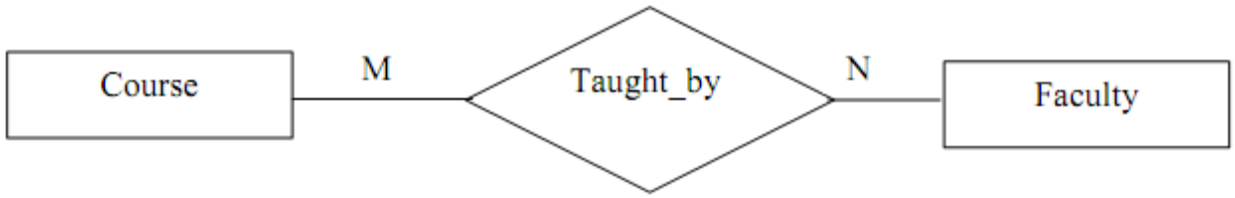
- **Many-to-one:** An instance in entity A is associated with at most one entity in B but an instance in entity B is associated with any number of instances in entity A. E.g. Relationship between course and instructor.

An instructor can teach various courses but a course can be taught only by one instructor. (assumed).



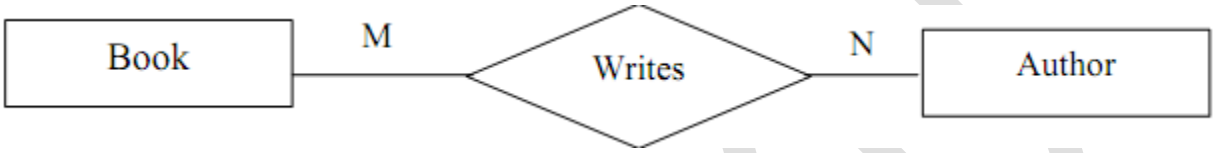
- **Many-to-many:** Any number of instances in entity A are associated with any number of instances in entity B and vice versa. E.g. Relationship between course and faculty.

One faculty member can be assigned to teach many courses and one course may be taught by many faculty members.



Relationship between book and author.

One author can write many books and one book can be written by more than one authors.



Recursive relationships

When the same entity type participates more than once in a relationship type, in different roles, the relationship types are called recursive relationships.

Participation constraints

The participation Constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are 2 types of participation constraints, Total and Partial.

Total Participation: When all the entities from an entity set participate in a relationship type, it is called total participation. For example, the participation of the entity set student in the relationship set must 'opt' is said to be total because every student enrolled must opt for a course.

Partial Participation: When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called partial participation. For example, the participation of the entity set faculty in 'manages' is partial, since every faculty member in a college does not head or manage a department`.

Weak Entity: Entity types that do not contain any key attribute, and hence cannot be identified independently, are called weak entity types. A weak entity can be identified uniquely only by considering some of its attributes in conjunction with the primary key attributes of another entity, which is called the identifying or strong or owner entity.

Generally a partial key is attached to a weak entity type that is used for unique identification of weak entities related to a particular owner entity type. The following restrictions must hold:

- The owner entity set and the weak entity set must participate in one-to-many relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

E.g. consider the entity type DEPENDENT related to EMPLOYEE entity, which is used to keep track of the dependents of each employee. The attributes of dependents are: name, birth date, sex and relationship. Each employee entity is said to own the dependent entities that are related to it. However, note that the 'dependent' entity does not exist of its own; it is dependent on the employee entity. In other words we can say that in case an employee leaves the organization all dependents related to him/her also leave along with him. Thus, the 'dependent' entity has no significance without the entity 'employee'. Thus, it is a weak entity and the employee entity is the strong or identifying entity for dependent entity.

Extended E-R model:

Although, the basic features of E-R diagrams are sufficient to design many database situations. However, with more complex relations and advanced database applications, it is required to move to enhanced features of E-R models. The three such features are:

- Generalization
- Specialization, and

- Aggregation

We will explain these concepts with the help of an example.

A car manufacturing company can manufacture different types of vehicles. These vehicles can be put into various categories like Car, Bus, and Truck etc. This categorization of vehicles represents a specialization/generalization hierarchy. It is shown as:

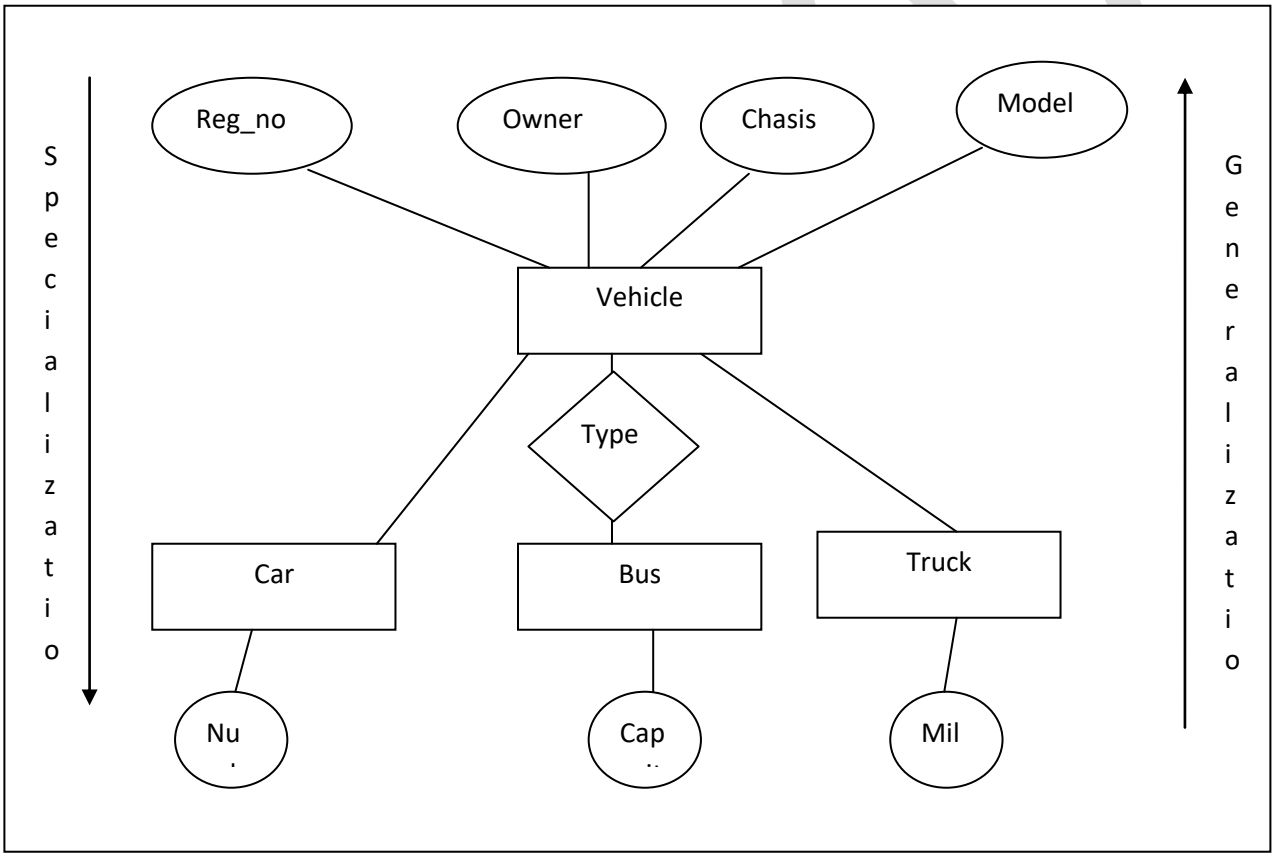


Figure: Generalization and Specialization hierarchy

But how are these diagrams converted to tables? This is shown in section 2.7.

Aggregation: One limitation of the E-R diagram is that they do not

allow representation of relationships among relationships. In such a case the relationship along with its entities are promoted (aggregated to form an aggregate entity which can be used for expressing the required relationships).

Defining Relationship for College Database

Using the concepts defined in above sections, we have identified that strong entities in COLLEGE database are STUDENT, FACULTY, COURSE and DEPARTMENT. This database also has one weak entity called GUARDIAN. We can specify the following relationships:

1. **Head of** is a 1:1 relationship between FACULTY and DEPARTMENT (some faculty member is head of the department). Participation of the entity FACULTY is partial since not all the faculty members participate in this relationship, while the participation from DEPARTMENT side is total, since every department has one head.
2. **Works in,** is a 1:N relationship between DEPARTMENT and FACULTY. Participation from both sides is total.
3. **Opts**, is a 1:N relationship between COURSE and STUDENT. Participation from student side is total because we are assuming that each student enrolled opts for a course. But the participation from the course side is partial, since there can be courses that no student has opted for.
4. **Taught by** is a M: N relationship between FACULTY and COURSE, as a faculty can teach many courses and a course can be taught by many faculty members.
5. **Enrolled**, is a 1:N relationship between STUDENT and DEPARTMENT as a student is allowed to enroll for only one department at a time.
6. **Has**, is a 1:N relationship between STUDENT and GUARDIAN as a student can have more than one local guardian and one local guardian is assumed to be related to one student only. The weak entity Guardian has total participation in the relation “Has”.

Next we will make an E-R diagram for the college database discussed in the previous sections.

E-R Diagram: Let us now make an Entity Relationship diagram for the student database as per the description given in the previous section. With E-R diagram we can express the overall logical structure of a database.

Notations used in E-R diagrams: Before constructing an E-R diagram let's describe the symbols used to construct an E-R diagram:

- ❖ Rectangle — Representing entity sets (strong).
- ❖ Rectangle — Representing weak entity sets.
- ❖ Eclipse — Representing attributes.
- ❖ Eclipse — Representing key attribute.
- ❖ Diamond — Representing relationship sets.
- ❖ ——— Representing flow of

Now let's construct E-R diagram for our COLLEGE database

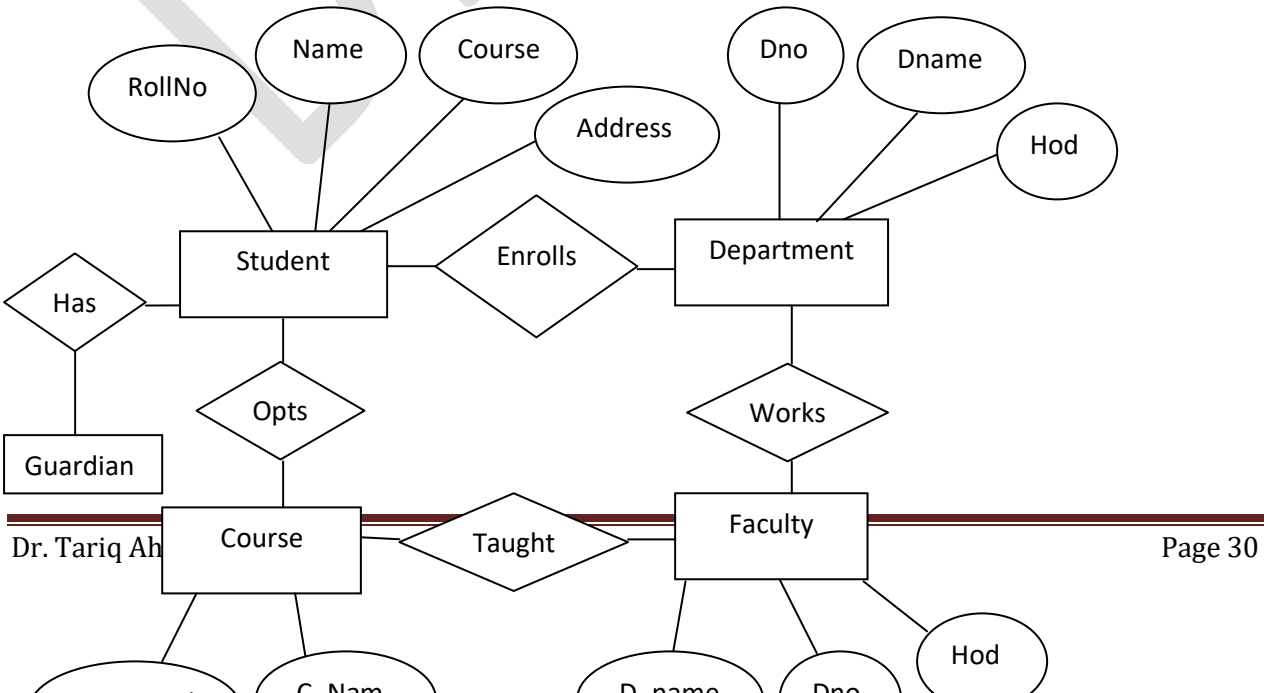


Figure 5: ER diagram of COLLEGE database

Lesson-3

Converting E-R diagram into Relational Database:

For every ER diagram we can construct a relational database which is a collection of tables. Following are the set of steps used for conversion of E-R diagram to a relational database.

Conversion of entity sets:

1. For each strong entity type **E** in the ER diagram, we create a relation **R** containing all the simple attributes of E. The primary key of the relation R will be one of the key attributes of R. For example, the STUDENT, FACULTY, COURSE and DEPARTMENT tables in the following figure.

STUDENT

RollNo	**Name**	**Course**	**Address**

FACULTY

F_id	F_Name	Address	Salary

COURSE

Course_id	Course_Name	Duration

DEPARTMENT

D_no	D_Name	HOD	Location

2. For each weak entity type **W** in the E R Diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also included in R. This key attribute of R is set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of weak entity type will form the key of the weak entity type. Figure below

shows the weak entity GUARDIAN, where the key field of student entity RollNo has been added.

GUARDIAN

RollNo	**Dep_Name**	**Address**	**Relationship**

Fig. Conversion of weak entity into a table

Conversion of relationship sets:

- **Binary Relationships:**
  - **One-to-one relationship:** For each 1:1 relationship type R in the ER diagram involving two entities E1 and E2 we choose one of entities (say E1) preferably with total participation and add primary key attribute of another entityE2 as a foreign key attribute in the table of entity (E1). We will also include all the simple attributes of relationship type R in E1 if any. For example, the DEPARTMENT relationship has been extended to include head-Id, an attribute of the relationship.

There is a 1:1 **Head_of** relationship between FACULTY and DEPARTMENT. We choose DEPARTMENT entity having total participation and add primary key attribute ID of FACULTY entity as a foreign key in DEPARTMENT entity named as **Head_ID**. Now the DEPARTMENT table will be as follows:

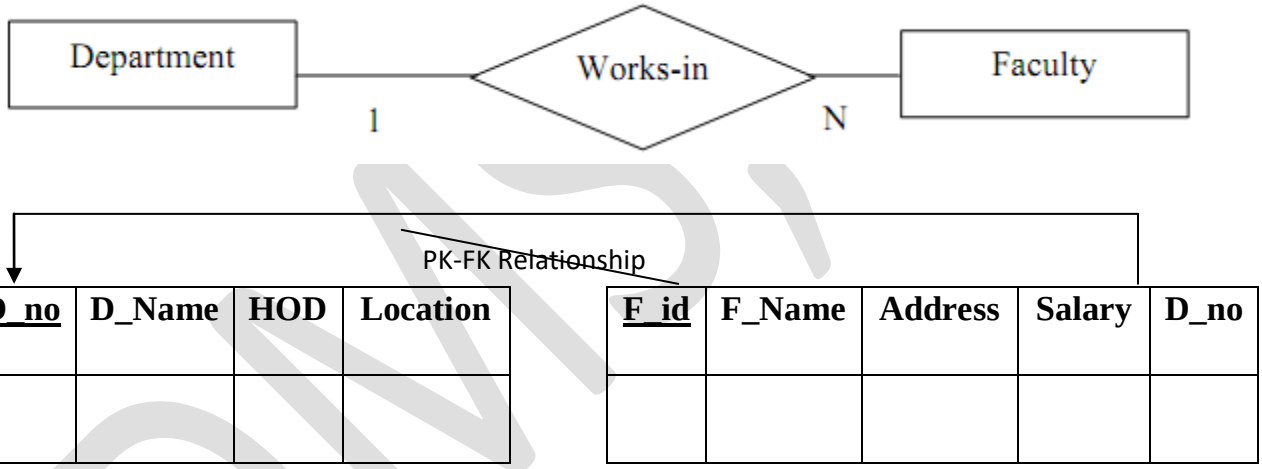
DEPARTMENT

D_NO	D_NAME	Head_ID	Date-from

Figure 9: Converting 1:1 relationship

- **One-to-many relationship:** For each 1: n relationship type **R** involving two entities **E1** and **E2**, we identify the entity type (say E1) at the n-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attributes (or simple components of a composite attributes of R (if any) in the table E1).

The **works_in** relationship between the DEPARTMENT and FACULTY entities is a 1: N relationship. For this relationship choose the entity at N side, i.e., FACULTY and add primary key attribute of another entity DEPARTMENT, i.e., DNO as a foreign key attribute in FACULTY entity.



D_no	D_Name	HOD	Location

F_id	F_Name	Address	Salary	D_no

Figure 10: Converting 1:N relationship

- **Many-to-many relationship:** For each m:n relationship type R, we create a new table (say S) to represent R. We also include the primary key attributes of both the participating entity types as a foreign key attribute in S. Any simple attributes of the m:n relationship type (or simple components of a composite attribute) is also included as attributes of S.

For example, the m: n relationship taught-by between entities COURSE and FACULTY should be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities. A new table TAUGHT-BY will be created as:

TAUGHT_BY

F_ID	Course_ID

Figure 11: Converting N:N relationship

- **n-ary Relationship:** For each n-ary relationship type R where n>2, we create a new table S to represent R. We include as foreign key attributes in s the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.

- **Multi-valued attributes:** For each multi-valued attribute '**A**', we create a new relation R that includes the multi-valued attribute and the primary key attribute **k** of the relation that represents the entity type or relationship type that has as an attribute. The primary key of R is then combination of A and k.

For example, if an EMPLOYEE entity has EMPNo, EName and EPhoneNo as its attributes where phone number is a multi-valued attribute, then we will create a table PHONE (EMPNo, EPhoneNo) where primary key is the combination OF EMPNo and EPhoneNo attributes. We need not have phone number as an attribute of EMPLOYEE relation then. It can have all attributes excluding phone number that will become a part of the new relation as shown in the figure below.

PHONE

EMPNO	EPhoneNo

- **Converting Generalization / Specialization hierarchy to tables:** A simple rule for conversion may be to decompose all the specialized entities into tables in case they are disjoint. For example, for the Figure shown under Generalization/Specialization section, we can create three tables as:

  Car (RegNo, ChasisNo, Model, Color).

  Bus (RegNo, ChasisNo, Model, Capacity).

  Truck (RegNo, ChasisNo, Model, Tonnage).

The other way is to create tables that are overlapping for example, assuming that in the example of Figure 4 if the accounts are not disjoint then we may create three tables:

account (account-no, holder-name, branch, balance)

saving (account-no, interest)

current (account-no, charges)

Lesson-4

DATABASE INTEGRITY:

A database is a collection of data. But, is the data stored in a database trustworthy? To answer this question we must first answer the question. What is integrity?

Integrity simply means to maintain the consistency of data. Thus, integrity constraints in a database ensure that changes made to the database by authorized users do not compromise data consistency. Thus, integrity constraints do not allow damage to the database. There are primarily three integrity constraints: the entity integrity constraint, Domain integrity constraint and the referential integrity constraint. In order to define these constraints, we need to understand the basic concept of Key with respect to a Database Management System.

The Keys

Candidate Key: In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following two properties:

- **Uniqueness**: No two distinct tuples in R have the same value for the candidate key
- **Irreducible**: No proper subset of the candidate key has the uniqueness property that the candidate key possesses.

Every relation must have at least one candidate key which cannot be reduced further. Duplicate tuples are not allowed in relations. Any candidate key can be a composite key also. For Example, (student_id + course_id) together can form the candidate key of a relation called marks (student-id, course-id, marks). Let us summarize the properties of a candidate key.

- A candidate key must be unique and irreducible
- A candidate may involve one or more than one attributes. A candidate key that involves more than one attribute is said to be composite.

But why are we interested in candidate keys? Candidate keys are important because they provide the basic tuple-level identification mechanism in a relational system. For example, if the enrolment number is the candidate key of a STUDENT relation, then the answer of the query: "Find student details from the STUDENT relation having enrolment number A0123" will output at most one tuple.

Primary Key: The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys, if any, are called alternate keys.

Foreign Keys: Let us first give you the basic definition of foreign key. Let R1 be a relation, then a foreign key in R2 is a subset of the set of attributes of R2, such that:

- There exists a relation R1 (R1 and R2 not necessarily distinct) with a candidate key, and
- For all time, each value of a foreign key in the current state or instance of R2 is identical to the value of Candidate Key in some tuple in the current state of R1.

Now, let us define the concept of foreign key in more practical terms with the help of an example. Assume that in an organization, an EMPLOYEE may always be assigned a department and no employee can be employed unless he/she is assigned a department. Therefore we will add an instance to an employee relation only if the department instance

exists or it will be rejected. Assume that the information is represented by the organization in two different relations named EMPLOYEE and DEPARTMENT. The DEPARTMENT relation describes the different departments in the organization. Assume that the relational schema for the above two relations are:

EMPLOYEE (E_id, E_name, D_no)

DEPARTMENT (D_no, D_name, D_head)

In the relations above E_ID and D_No are unique and not NULL, respectively. As we can clearly see, we can identify the complete instance of the entity set employee through the attribute E_ID. Thus E_ID is the primary key of the relation EMPLOYEE. Similarly D_No is the primary key for the DEPARTMENT relation. Figure below shows the E-R diagram for these entities and relationship.

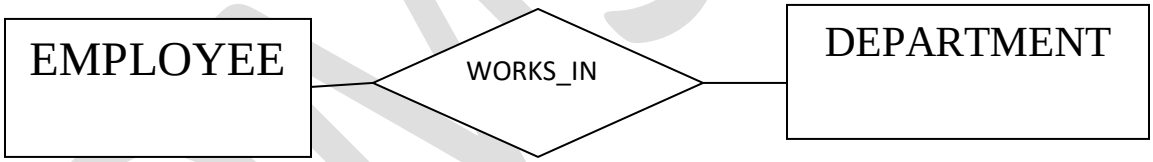


Figure 1: E-R diagram for employee role in department

Let us consider sample relation instances as:

E_id	E_name	D_no
1001	Tariq Ahmad	10
1002	Mushtaq Ahmad	40

EMPLOYEE

1003	Tanveer Ahmad	40
1004	Rafi Ahmad	10

DEPARTMENT

D_ no	**D_name**	**D_head**
10	The Business School	Prof. Shabir
40	Business & Financial Studies	Prof. Nazir
20	Distance Education	Prof. Nelofar

D_NO is the foreign key in EMPLOYEE relation; it references the relation DEPARTMENT where D_no is the primary key. This means that the D_no attribute of EMPLOYEE relation will derive its values from the values of primary key attribute of DEPARTMENT relation.

Database Integrity and Normalization: Now after defining the concept of foreign key, we can proceed to discuss the actual integrity constraints namely Referential Integrity and Entity Integrity.

Referential Integrity: It can be simply defined as:

The database must not contain any unmatched foreign key values. The term "unmatched foreign key value" means a foreign key value for which there does not exist a matching value of the relevant candidate key in the relevant target (referenced) relation. For example, any value existing in the D_no attribute in EMPLOYEE relation must exist in the DEPARTMENT relation. That is, the only D_nos that can exist in the EMPLOYEE relation are 10, 20 and 40 for the present state/ instance of the database given in tables above. If we want to add a tuple with D_no value 70 in the EMPLOYEE relation, it will cause violation of referential integrity constraint. Logically it is very obvious after all the department number 70 does not exist, so how can any employee be assigned to a nonexistent department. Database modifications can cause violations of referential integrity. We list here the test we must make for each type of database modification to preserve the referential-integrity constraint:

Delete: During the deletion of a tuple two cases can occur:

Deletion of tuple in relation having the foreign key: In such a case simply delete the desired tuple. For example, in EMPLOYEE relation we can easily delete any tuple.

Deletion of the target of a foreign key reference: For example, an attempt to delete an employee tuple in DEPARTMENT relation whose D_no is 40. This employee number appears not only in the DEPARTMENT relation but also in the EMPLOYEE relation. can this tuple be deleted? If we delete the tuple in DEPARTMENT relation then two unmatched tuples are left in the EMPLOYEE relation, thus causing violation of referential integrity constraint. Thus, the following three choices exist for such deletion:

RESTRICT: – The delete operation is "restricted" to only the case where there are no such matching tuples. For example, we can delete the DEPARTMENT record of D_N0 20 as no matching tuples are contained in the EMPLOYEE relation.

CASCADE: – The delete operation "cascades" or spreads over to delete those matching tuples also. For example, if the delete mode is CASCADE then deleting department having D_no as 10 from DEPARTMENT relation will also cause deletion of one tuple from EMPLOYEE relation.

SET NULL: – The delete operation in the referred relation causes the values of the referring attribute to be set to NULL. For example, if the delete mode is SET NULL, then deleting department having D_no as 10 from DEPARTMENT relation will set D_no value of employee with E_id 10 to NULL in EMPLOYEE relation.

Insert: The insertion of a tuple in the target of reference does not cause any violation. However, insertion of a tuple in the relation in which, we have the foreign key, for example, in EMPLOYEE relation it needs to be ensured that all matching target candidate key exist; otherwise the insert operation can be rejected. For example, one of the possible EMPLOYEE insert operations would be (104, Rafi Ahmad, 30) which will lead to violation of referential integrity constraint and will therefore be rejected.

Modify: Modify or update operation changes the existing values. If these operations change the value that is the foreign key also, the only check required is the same as that of the Insert operation.

What should happen to an attempt to update a candidate key that is the target of a foreign key reference? For example, an attempt to update the D_no of department named "The Business School" for which there exists at least one matching EMPLOYEE tuple? In general there are the same possibilities as for DELETE operation:

RESTRICT: The update operation is "restricted" to the case where there are no matching department tuples. (it is rejected otherwise) .

CASCADE: The update operation "cascades" to update the foreign key in those matching EMPLOYEE tuples also.

Entity Integrity

Before describing the second type of integrity constraint, viz., Entity Integrity, we should be familiar with the concept of NULL value. Basically, NULL is intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records sometimes have entries such as "Date of birth unknown", or police records may include the entry "Present whereabouts unknown." Hence it is necessary to have some way of dealing with such situations in database systems. Thus Codd proposed an approach to this issue that makes use of special markers called NULL values to represent such missing or unknown information.

A given attribute in the relation might or might not be allowed to contain NULL. But, can the Primary key or any of its components (in case of the primary key is a composite key) contain a NULL? To answer this question an Entity Integrity Rule states: No component of the primary key of a relation is allowed to accept NULL. In other words, the definition of every attribute involved in the primary key of any basic relation must explicitly or implicitly include the specifications of NULL NOT ALLOWED.

Foreign Keys and NULL: Let us consider the relations:

EMPLOYEE			
E_no	**E_name**	**D_no**	**Salary**
1001	Tariq Ahmad	10	40000
1002	Mushtaq Ahmad	20	39000
1003	Tanveer Ahmad	20	25000

1004	Rafi Ahmad	10	44000

DEPARTMENT

D_no	D_name	Budget
10	The business School	5000000
20	Distance Education	7000000
30	Islamic Studies	2500000

Suppose that "Kaiser Rashid" is not assigned any Department. In the EMPLOYEE tuple corresponding to "Kaiser Rashid", therefore, there is no genuine department number that can serve as the appropriate value for the D_no foreign key. Thus, one cannot determine D_name and Budget for "Kaiser Rashid's" department as those values are NULL. This may be a real situation where the person has newly joined and is undergoing training and will be allocated to a department only on completion of the training. Thus, NULL in foreign key values may not be a logical error. So, the foreign key definition may be redefined to include NULL as an acceptable value in the foreign key for which there is no need to find a matching tuple. Are there any other constraints that may be applicable on the attribute values of the entities? Yes, these constraints are basically related to the domain and termed as the domain constraints.

Domain Constraints: Domain constraints are primarily created for defining the logically correct values for an attribute of a relation. The relation allows attributes of a relation to be confined to a range of values, for example, values of an attribute age can be restricted as 18 to 36 or a specific type such as positive integers, etc.

Lesson-5

REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following relation.

EMP_DEPT						
ENO	**ENAME**	**ESAL**	**EADDRESS**	**DNO**	**DNAME**	**DHEAD**
E101	Tariq Ahmad	40000	Pattan	10	TBS	Dr. Shabir
E102	M. Asif	35000	Soura	50	BIOTECH	Dr. Andrabi
E103	Rafi Ahmad	45000	Dalgate	10	TBS	Dr. Shabir
E104	Ayub Shah	43000	Baramulla	20	DDE	Dr. Nelofar
E105	Bashir Ahmad	81000	Hawal	10	TBS	Dr. Shabir
E106	Ajaz Khaki	25000	Khanyar	20	DDE	Dr. Nelofar

Fig. A state of EMP_DEPT relation

The above relation satisfies all the properties of a relation. Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database. But the relation above suffers from the following undesirable features:

Data Redundancy:-A lot of information is being repeated in the EMP_DEPT relation. For example, the information that "DNO 10" is named "TBS" and its head is "Dr. Shabir" is repeated three records. Same is the case with other records also and this will increase with each new record being added to the relation (Please find the other duplicate information in the relation yourself). So we find that the department number, department name, department head are being repeated often, thus, the table has Data Redundancy.

Also note that every time we wish to insert an employee record, we must insert the department number, department name, and head of the department that the employee belongs to. This repetition of information results in problems in addition to the wastage of space. Look for these problems in the EMP_DEPT relation. What are these problems? Let us define them.

All these problems together, are called database anomalies. There are three anomalies in database systems:

- **Update Anomaly:** This anomaly is caused due to data redundancy. Redundant information makes updates more difficult since, for example, changing the name of the head of department number 10 would require that all employee tuples containing DNO 10 be updated. If for some reason, all tuples are not updated, we might have a database that gives two names for head of the department of DNO 10, which is inconsistent information. This problem is called update anomaly. An update anomaly results in data inconsistency.
- **Insertion Anomaly:** Inability to represent (insert, add) certain information in the database is termed as insertion anomaly. The primary key of the above relation would be (ENO, DNO). Any new tuple to be inserted in the relation must have a value for the primary key. Since entity integrity constraint requires that a key may

not be totally or partially NULL. However, in the given relation if one wanted to insert the number and name of a new department in the database, it would not be possible until an employee is assigned to that department. Similarly information about a new employee cannot be inserted in the database until the employee is assigned to a department. These problems are called insertion anomalies.

- **Deletion Anomaly:** Loss of Useful Information from the database is called deletion anomaly. In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to employee E102 assigned to DNO 50, we will lose relevant information about the department viz. department number, name and head of the department. This is called deletion anomaly.

The anomalies arise primarily because the relation EMP_DEPT is badly designed and has information about employees as well as departments. One solution to these problems is to decompose the relation into two or more smaller relations. But what should be the basis of this decomposition? To answer such questions, let us try to formulate how data is related in the relation with the help of the following Figure:

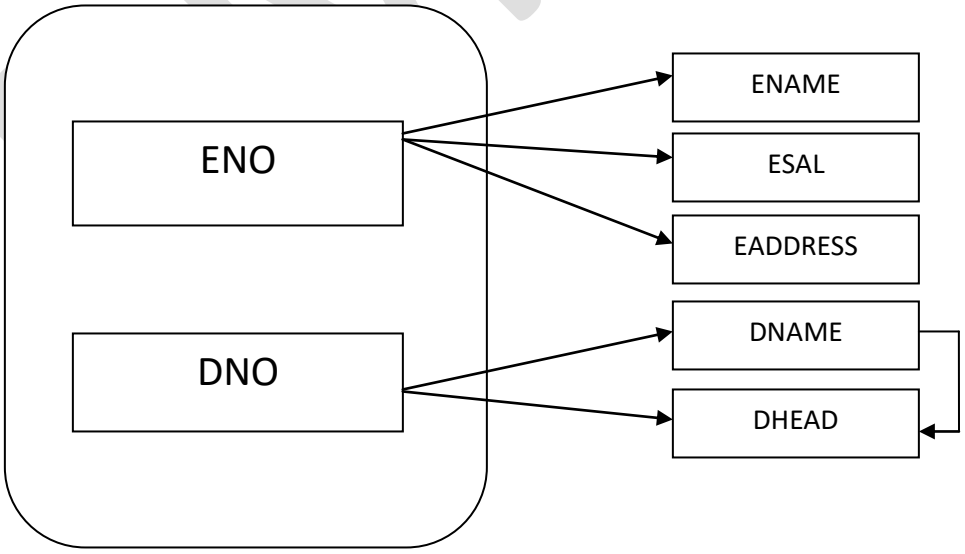


Figure: Dependency diagram of EMP_DEPT relation

Please note that the arrows in Figure are describing data inter-relationship. For example, ENO column is unique for every employee so if we know the ENO of an employee, we can uniquely determine his/her name, address and salary. Similarly, DNO uniquely determines department name (DNAME) and department head (DHEAD). We also note one important interrelationship in the above Figure, that is, the DHEAD is dependent on DNAME. The root cause of the presence of anomalies in a relation is determination of data by the components of the key and non-key attributes.

The question arises that how do we eliminate all these problems and design good relations or refine badly designed relations so that all these anomalies are removed and we have non-redundant and consistent data in our database. The answer to all these questions is Normalization.

Functional Dependencies, Normalization and its different forms:

Normalization involves decomposition of a relation into smaller relations based on the concept of functional dependence to overcome the above mentioned undesirable anomalies. Normalization sometimes can affect performance, as it results in decomposition of tables, so some queries desire to join these tables to produce the data once again. But such performance overheads are minimal as Normalization results in minimization of data redundancy and may result in smaller relation sizes. Also DBMSs implement optimized algorithms for joining of relations and many indexing schemes that reduce the load on joining of relations. In any case the advantages of normalization normally overweigh the performance constraints. Normalization does lead to more efficient updates since an update that may have required several tuples to be updated, whereas normalized relations, in general, require the information updating at only one place. A relation that needs to be normalized may have a very large number of attributes. In such relations, it is almost impossible for a person to conceptualize all the information and suggest a suitable decomposition to overcome the problems. Such relations need an

algorithmic approach of finding if there are problems in a proposed database design and how to eliminate them if they exist. Here we will first introduce the basic concept that supports the process of Normalization of large databases. So let us first define the concept of functional dependence and subsequently the concepts of normalization will follow.

SINGLE-VALUED DEPENDENCIES

A database is a collection of related information and it is therefore inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multi-valued. The name of a person or his date of birth are single-valued facts; qualifications of a person or subjects that an instructor teaches are multi-valued facts. We will deal only with single-valued facts and discuss the concept of functional dependency.

Functional Dependency

Consider a relation R that has two attributes A and B. The attribute B of the relation is functionally dependent on the attribute A if and only if for each value of A, no more than one value of B is associated. In other words, the value of attribute A uniquely determines the value of attribute B and if there were several tuples that had the same value of A then all these tuples will have an identical value of attribute B. That is, if t1 and t2 are two tuples in the relation R where t1(A) = t2(A), then we must have t1(B) = t2(B).

Both, A and B need not be single attributes. They could be any subsets of the attributes of a relation R. The functional dependency (FD) between the attributes can be written as: R.A R.B or simply A B, if B is functionally dependent on A (or A functionally determines B). Please note that functional dependency does not imply a one-to-one relationship between A and B. For example, the EMP_DEPT relation whose dependencies are shown in dependency diagram above can be written as:

ENO ⟶ ENAME

ENO → ESAL

ENO → EADDRESS

DNO → DNAME

DNO → DHEAD

These functional dependencies imply that there can be only one employee name for each ENO, only one address for each employee and only one department name for each DNO. It is of course possible that several employees may have the same name and several employees may live at the same address like same residential quarters for university employees.

In the example above, you may be wondering if the following FDs hold:

ENAME → ENO (1)

DNAME → DNO (2)

Certainly there is nothing in the given instance of the database relation presented that contradicts the functional dependencies as above. However, whether these FDs hold or not would depend on whether the organization whose database we are considering allows duplicate employee names and department names. If it was the enterprise policy to have unique department names then (2) holds. If duplicate employee names are possible, and one would think there always is the possibility of two employees having exactly the same name, then (1) does not hold otherwise it holds too.

A simple example of the functional dependency above is when A is a primary key of an entity (e.g., ENO) and B is some single-valued property or attribute of the entity (e.g., Salary). A → B then must always hold.

Functional dependencies also arise in relationships. Let C be the primary key of an entity and D be the primary key of another entity. Let the two entities have a relationship. If the

relationship is one-to-one, we must have both C D and D C. If the relationship is many-to-one, we would have C D and not D C. For many-to-many relationships, no functional dependencies hold.

For example, consider the following E-R diagram:

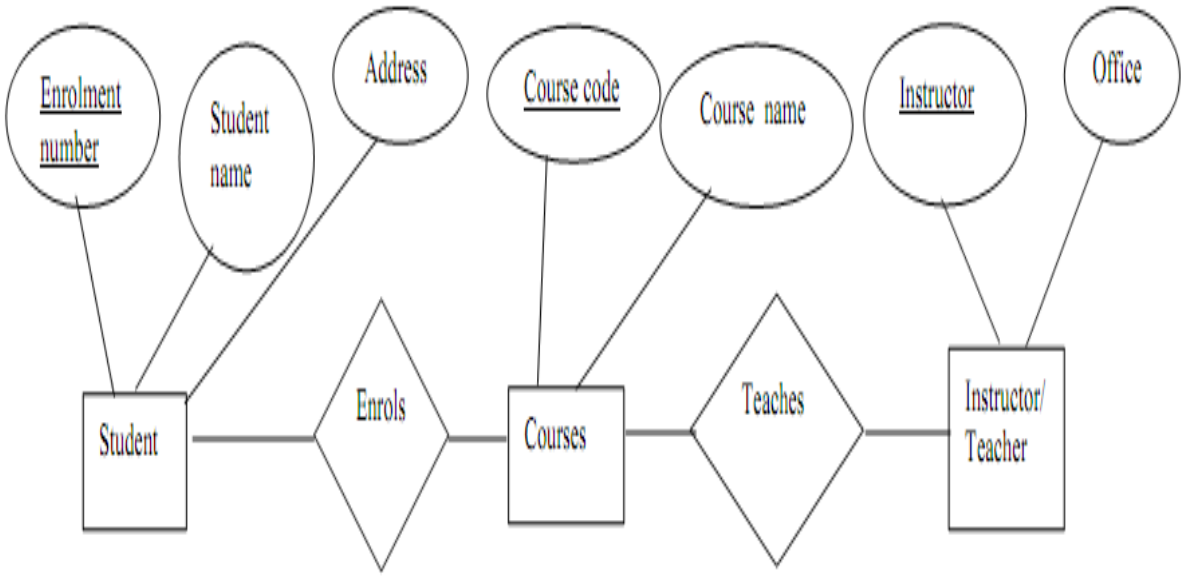


Figure: E-R diagram for student course Teacher

In the ER diagram above, the following FDs exist:

FDs in Entities:

Student entity:

Enrolment number ⟶ Student name, Address

Course Entity:

Course code ⟶ Course name

Teacher Entity:

Teacher ⟶ Office

FDs in Relationships:

Enrols Relationship:

None as it is many to many

Teaches Relationship:

Course code ⟶ Instructor

Instructor ⟶ Course code

The next question is: How do we identify the functional dependence in a database model?

Functional dependencies arise from the nature of the real world that the database models. Often A and B are facts about an entity where A might be some identifier for the entity and B some characteristic. Functional dependencies cannot be automatically determined by studying one or more instances of a database. They can be determined only by a careful study of the real world and a clear understanding of what each attribute means. There are no thumb rules for determining FDs.

SINGLE VALUED NORMALIZATION

E.F Codd in the year 1972 presented three normal forms (1NF, 2NF, and 3NF). These were based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multi-valued and join dependencies and were proposed later. In this script we will cover normal forms till BCNF only. Fourth and fifth normal forms are beyond the scope of this script. For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed for most common situations. It should be clearly understood that there is no obligation to

normalize relations to the highest possible level. Performance should be taken into account and sometimes an organization may take a decision not to normalize, say, beyond third normal form. But, it should be noted that such designs should be careful enough to take care of anomalies that would result because of the above decision.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). A sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 2NF or 3NF. Normalization results in decomposition of the original relation. It should be noted that decomposition of relation has to be always based on principles, such as functional dependence, that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. Careless decomposition of a relation can result in loss of information.

Let us now define normal forms in more detail with examples so that we understand the concept very well.

The First Normal Form (1NF)

A relation is said to be first normalized or in 1NF if

A. There are no duplicate rows or tuples in the relation.
B. Each data value stored in the relation is single-valued or atomic.
C. Entries in a column (attribute) are of the same kind (type).

Kindly note that in a 1NF relation, the order of the tuples (rows) and attributes (columns) does not matter. The first requirement above means that the relation must have a key that uniquely identifies each record in the relation. The key may be single attribute or composite key. It may even, possibly, contain all the columns. The first normal form defines only the basic structure of the relation and does not resolve the anomalies discussed earlier.

The relation **EMP_DEPT** (**ENO, ENAME, ESAL, EADDRESS, DNO, DNAME, DHEAD**) shown below is in 1NF, since it satisfies all the above mentioned criteria. The primary key of the relation is (ENO, DNO).

EMP_DEPT						
ENO	**ENAME**	**ESAL**	**EADDRESS**	**DNO**	**DNAME**	**DHEAD**
E101	Tariq	40000	Pattan	10	TBS	Dr. Shabir
E102	Asif	35000	Soura	50	BIOTECH	Dr. Andrabi
E103	Rafi	45000	Dalgate	10	TBS	Dr. Shabir
E104	Ayub	43000	Baramulla	20	DDE	Dr. Nelofar
E105	Bashir	81000	Hawal	10	TBS	Dr. Shabir
E106	Ajaz	25000	Khanyar	20	DDE	Dr. Nelofar

The Second Normal Form (2NF)

The EMP_DEPT relation shown above is in 1NF, yet it suffers from all anomalies as discussed earlier, so we need to define the next normal form. A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on the primary key of the relation. Some of the points that should be noted here are:

- A relation having a single attribute key has to be in 2NF.
- In case of composite key, partial dependency on key that is part of the key is not allowed.
- 2NF tries to ensure that information in one relation is about one thing
- Non-key attributes are those that are not part of the primary key.

Let us now reconsider the dependency diagram shown on page 71, which defines the FDs of the relation EMP_DEPT.

These FDs can also be written as:

ENO ⟶ ENAME, ESAL, EADDRESS (1)

DNO⟶ DNAME, DHEAD (2)

DNAME ⟶ DHEAD (3)

The key attributes of the relation are (ENO +DNO). All other attributes are non-key attributes. For the 2NF decomposition, we are concerned with the FDs (1) and (2) above as they relate to partial dependence on the key that is (ENO +DNO). As these dependencies (also refer to dependency diagram on page 71) show that the relation is not in 2NF and hence suffer from all the three anomalies and redundancy problems as many non-key attributes can be derived from partial key attribute. To convert the relation into 2NF, let us use FDs. As per FD (1) the ENO uniquely determines ENAME, EADDRESS and ESAL, so one relation should be:

EMPLOYEE (ENO, ENAME, EADDRESS, ESAL)

Now as per FD (2), DNO uniquely identifies DNAME and DHEAD, although head of the department can be known if we know the Name of the department, which in turn can be known if we know the department number. This is called transitive dependency. We find in FD (2) that DNO attribute uniquely determines the name of the department. In FD (3) we see that the name of the department uniquely determines the head of the department. This can be written as:

DNO⟶DNAME

DNAME⟶ DHEAD

⇒ DNO⟶ DHEAD (Transitive dependency)

Thus, FD (2) can now be rewritten as:

DNO ⟶ DNAME, DHEAD

This FD now gives us the second decomposed relation:

DEPARTMENT (DNO, DNAME, DHEAD)

Thus, the relation EMP_DEPT has been decomposed into following two relations:

EMPLOYEE (ENO, ENAME, EADDRESS, ESAL)

DEPARTMENT (DNO, DNAME, DHEAD)

Is the decomposition into 2NF complete now?

No, how would you join the two relations created above any way? To achieve this we can either create a joining relation comprising of the two key attributes forming the composite key and any other attribute that is not covered in the decomposition, thus forming a third relation as EMP_DEP (ENO,DNO). The other way out is to make some attribute a joining attribute for primary key- foreign key relationship between the two relations. This can be done in this case by adding DNO to EMPLOYEE relation and making it a foreign key referring DNO of DEPARTMENT relation.

So, the relation EMP_DEPT in 2NF form would be:

EMPLOYEE (ENO, ENAME, EADDRESS, ESAL)

DEPARTMENT (DNO, DNAME, DHEAD)

EMP_DEP (ENO, DNO)

Or

EMPLOYEE (ENO, ENAME, EADDRESS, ESAL, DNO)

DEPARTMENT (DNO, DNAME, DHEAD)

EMPLOYEE				
ENO	**ENAME**	**ESAL**	**EADDRESS**	**DNO**
E101	Tariq	40000	Pattan	10
E102	Asif	35000	Soura	50
E103	Rafi	45000	Dalgate	10
E104	Ayub	43000	Baramulla	20
E105	Bashir	81000	Hawal	10
E106	Ajaz	25000	Khanyar	20

DEPARTMENT		
DNO	**DNAME**	**DHEAD**
10	TBS	Dr. Shabir
50	BIOTECH	Dr. Andrabi
20	DDE	Dr. Nelofar

The Third Normal Form (3NF)

Although, transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies, it does not necessarily remove all anomalies. Thus, further Normalization is sometimes needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the candidate keys of the relation.

"A relation is in third normal form, if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on the primary key of the relation". Note that 3NF is concerned with transitive dependencies which do not involve candidate keys. A 3NF relation with more than one candidate key will clearly have transitive dependencies of the form: primary_key → other_candidate_key → any_non-key_column.

Another definition of 3NF where in we have only one candidate key can be given as:

A relation R having just one candidate key is in third normal form (3NF) if and only if the non-key attributes of R (if any) are:

1) Mutually independent, and

2) Fully dependent on the primary key of R.

A non-key attribute is any column which is not part of the primary key. Two or more attributes are mutually independent if none of the attributes is functionally dependent on any of the others. Attribute Y is fully functionally dependent on attribute X if X → Y, but Y is not functionally dependent on any proper subset of the (possibly composite) attribute X

But what is transitive dependence?

Let A, B and C be three attributes of a relation R such that A → B and B → C. From these FDs, we may derive A → C. This dependence A → C is transitive.

Now, let us consider the following relation:

STUD_HOSTEL				
RollNo	**SName**	**Deptt.**	**Semester**	**H_name**
1084	Reyaz	TBS	1	Sheikhul Alam
1048	Khalid	BFS	1	Sheikhul Alam

1068	Gaffar	Math	2	Mehboobul Alam
1038	Rahim	CS	2	Mehboobul Alam
1082	Maria	TBS	2	Mehboobul Alam
1045	Asif	Zoo	4	Gani Kashmiri

In the above relation **RollNo** is the key and all other attributes are functionally dependent on it, thus it is in 2NF. If all the 1st semester students are accommodated in Sheikhul Alam hostel, all 2nd semester students in Mehboobul Alam, all 3rd semester students in Anwar Shah and all 4th semester students in Gani Kashmiri hostel, then the non-key attribute **H_name** is dependent on the non-key attribute **Semester.**

Please observe that given the semester of a student, his/her hostel is known and vice-versa. The dependency of hostel on semester leads to duplication of data as is evident from the relation.

Now let's assume that all 1st semester students are asked to move to Mehboobul Alam hostel and all 2nd semester students are asked to move to Sheikhul Alam hostel, this change should be made at many places in the STUD_HOSTEL relation. Again when a student's semester of study changes, his/her hostel change should be made at many places in the relation, which is simply undesirable. Thus the relation STUD_HOSTEL is not in 3NF.

To transform it into 3NF, we should decompose this relation into two relations as:

STUDENT (RollNo, Sname, Deptt., Semester) and

HOSTEL (Semester, H_name)

This decomposition removes data redundancy and making changes as mentioned above becomes easier. This is shown below.

STUDENT			
RollNo	**Sname**	**Deptt.**	**Semester**
1084	Reyaz	TBS	1
1048	Khalid	BFS	1
1068	Gaffar	Math	2
1038	Rahim	CS	2
1082	Maria	TBS	2
1045	Asif	Zoo	4

HOSTEL	
Semester	**H_name**
1	Sheikhul Alam
2	Mehboobul Alam
3	Anwar Shah
4	Gani Kashmiri

Note here that, in case hostel allocated to students don't depend on their semester of study then the relation STUD_HOSTEL is already in 3NF.

The 3NF is usually quite adequate for most relational database designs. There are however some situations where a relation may be in 3 NF, but have the anomalies. In such cases further normalization is needed.

Boyce-Codd Normal Form (BCNF)

BCNF was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF because every relation in BCNF is also in 3NF, however a relation in 3NF is not necessarily in BCNF.

The definition of BCNF addresses certain situations which 3NF does not handle. The characteristics of a relation which distinguish 3NF from BCNF are given below. Since it is so unlikely that a relation would have these characteristics, in practical real-life design it is usually the case that relations in 3NF are also in BCNF. Thus many authors make a vague distinction between 3NF and BCNF when it comes to giving advice on "how far" to normalize a design. Since relations in 3NF but not in BCNF are slightly unusual, it is a bit more difficult to come up with meaningful examples. To be precise, the definition of 3NF does not deal with a relation that:

1. has multiple candidate keys, where
2. those candidate keys are composite, and
3. the candidate keys overlap (i.e., have at least one common attribute)

If an attribute of a composite key is dependent on an attribute of another composite key, BCNF is needed.

Consider the following relation:

PROFESSOR (P_code, Deptt., HOD, Percent_time)

PROFESSOR			
P_code	**Deptt.**	**HOD**	**Percent_time**
P1	TBS	Shabir	50
P1	BFS	Nazir	50
P2	CHM	Khaleeq	25
P2	PHS	Nissar	75
P3	BFS	Nazir	100

Following assumptions are made:

1. A Professor can work in more than one department
2. The percentage of time he spends in each department is given
3. Each department has only one HOD

The two possible composite keys are:

(P_code, Deptt.)

(P_code, HOD)

Observe that DEPTT. And HOD are not non-key attributes, they are part of composite keys. The relation PROFESSOR is in 3NF. However the naes of department and HOD are duplicated. Further if Professor P2 resigns, rows 3 and 4 are deleted and we lose the information that Kahleeq and Nissar are the HODs of the CHM and PHS departments.

So to remove these problems, we need to decompose the relation to normalize it to BCNF as is shown below:

PROFESSOR		
P_code	**Deptt.**	**Percent_time**
P1	TBS	50
P1	BFS	50
P2	CHM	25
P2	PHS	75
P3	BFS	100

DEPTT.	
Deptt.	**HOD**
TBS	Shabir
BFS	Nazir

CHM	Khaleeq
PHS	Nissar

The above two relations are BCNF normalized.

Higher Order Normal Forms:

There are more normal forms beyond BCNF. However, these normal forms are not based on the concept of functional dependence. Further normalization is needed if the relation has Multi-valued, join dependencies, or template dependencies. These topics are not covered here because these are beyond the scope of this script and further it has been observed that normalization up to BCNF is enough to achieve the required results in most of the databases.

DECOMPOSITION AND ITS PROPERTIES

We have used normalization to decompose relations in the previous section. But what does decomposition formally mean? Decomposition is a process of splitting a relation into its projections that will not be disjoint. This means that we vertically divide a relational into two or smaller normalized relations.

Desirable properties of decomposition are:

- Attribute preservation
- Lossless-join decomposition
- Dependency preservation
- Lack of redundancy

Attribute Preservation

This is a simple and obvious requirement that involves preserving all the attributes that are there in the relation that is being decomposed. If we join the decomposed relations, we get back the original relation that was decomposed.

Lossless-Join Decomposition

Let us show an intuitive decomposition of a relation. We need a better basis for deciding decompositions since intuition may not always be correct. We illustrate how a careless decomposition may lead to problems including loss of information. Consider the following relation

ENROL (sno, cno, date-enrolled, room-no, instructor)

Suppose we decompose the above relation into two relations ENROLl and ENROL2 as follows:

ENROL1 (sno, cno, date-enrolled)

ENROL2 (date-enrolled, room-no, instructor)

Let an instance of the relation ENROL be:

ENROLL				
SNO	**CNO**	**DATE_ENROLLED**	**ROOM_NO**	**INSTRUCTOR**
1011	COM101	20-08-2011	1	NAZIR
1012	COM102	20-08-2011	2	MUSHTAQ
1013	COM103	20-08-2011	1	RIYAZ
1014	COM104	20-08-2011	5	SARTAJ
1015	COM105	20-08-2011	6	KAISER

Figure: A sample relation for decomposition

Then on decomposition the relations ENROL1 and ENROL2 would be:

ENROL1		
SNO	**CNO**	**DATE_ENROLLED**
1011	COM101	20-08-2011
1012	COM102	20-08-2011
1013	COM103	20-08-2011
1014	COM104	20-08-2011
1015	COM105	20-08-2011

ENROL2		
DATE_ENROLLED	**ROOM_NO**	**INSTRUCTOR**
20-08-2011	1	NAZIR AHMAD
20-08-2011	2	MUSHTAQ BHAT
20-08-2011	1	RIYAZ AHMAD
20-08-2011	5	SARTAJ AZIZ
20-08-2011	6	KAISER AHMAD

All the information that was in the relation ENROL appears to be still available in ENROL1 and ENROL2 but this is not so. Suppose, we wanted to retrieve the student numbers of all students taking a course from Riyaz Ahmad, we would need to join

ENROL1 and ENROL2. For joining the only common attribute is Date-enrolled. Thus, the resulting relation obtained will not be the same as that of first Figure. (We will learn join operation in the next unit)

The join will contain a number of spurious (or unwanted) tuples that were not in the original relation. Because of these additional tuples, we have lost the right information about which students take courses from Riyaz Ahmad. (Yes, we have more tuples but less information because we are unable to say with certainty who is taking courses from Riyaz Ahmad). Such decompositions are called lossy decompositions. A non-lossy or lossless decomposition is that which guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not.

We need to analyze why the decomposition is lossy. The common attribute in the above decompositions was Date-enrolled. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. If the common attribute would have been the primary key of at least one of the two decomposed relations, the problem of losing information would not have existed. The problem arises because several enrolments may take place on the same date.

The dependency based decomposition scheme as discussed earlier creates lossless decomposition.

Dependency Preservation

It is clear that the decomposition must be lossless so that we do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a dependency is a constraint on the database. If all the attributes appearing on the left and the right side of a dependency appear in the same relation, then a dependency is considered to be preserved. Thus, dependency preservation can be checked easily. Dependency preservation is important, because as stated earlier,

dependency is a constraint on a relation. Thus, if a constraint is split over more than one relation (dependency is not preserved), the constraint would be difficult to meet. Please refer to suggested readings for more details. But remember that "A decomposition into 3NF is lossless and dependency preserving whereas a decomposition into BCNF is lossless but may or may not be dependency preserving."

Lack of Redundancy

We have discussed the problems of repetition of information in a database. Such repetition should be avoided as much as possible. Let us state once again that redundancy may lead to inconsistency. On the other hand controlled redundancy sometimes is important for the recovery in database system. Decomposition reduces redundancy while we normalize the relations.

RULES OF DATA NORMALIZATION

Let us now summarize Normalization process with the help of several clear and clean rules. The following are the basic rules for the Normalization process:

- ***Eliminate Repeating Groups:*** Make a separate relation for each set of related attributes, and give each relation a primary key.
- ***Eliminate Redundant Data:*** If an attribute depends on only part of a multi-attribute key, remove it to a separate relation.
- ***Eliminate Columns Not Dependent On Key:*** If attributes do not contribute to a description of the key, remove them to a separate relation.
- ***Isolate Independent Multiple Relationships:*** No relation may contain two or more **1:n or n:m** relationships that are not directly related.

Isolate Semantically Related Multiple Relationships: There may be practical constraints on information that justify separating logically related many-to-many relationships.

SUMMARY

This unit covered the details of Database integrity in detail. It covered aspects of keys, entity integrity and referential integrity. The role of integrity constraints is to make data more consistent. A functional dependency (FD) is a many-to-one relationship between two sets of attributes of a given relation. Given a relation R, the FD A → B (where A and B are subsets of the attributes of R) is said to hold in R if and only if, whenever two tuples of R have the same value for A, and they also have the same value for B.

We discussed Single valued Normalization and the concepts of first, second, third, and Boyce/Codd normal forms. The purpose of Normalization is to avoid redundancy, and hence to avoid certain update insertion and detection anomalies. We have also discussed the desirable properties of a decomposition of a relation to its normal forms. The decomposition should be attribute preserving, dependency preserving and lossless.

We have also discussed various rules of data Normalization, which help to normalize the relations cleanly. Those rules are eliminating repeating groups, eliminate redundant data, and eliminate columns not dependent on key, isolate independent multiple relationship and isolate semantically related multiple relationships.

Exercise:

I. Consider supplier-part-project database given below:

SUPPLIERS		
S_NO	**S_NAME**	**CITY**

S001	Sahil Enterprises	Srinagar
S002	Metro Sales Corp.	Sopore
S003	Danish Enterprises	Pulwama
S004	Bharat Sanchar Nigam Ltd	Delhi
S005	Solar India	Mumbai

PARTS

P_NO	P_NAME	COLOR	CITY
P2001	Nut	Red	Srinagar
P2002	Bolt	Blue	Srinagar
P2003	Screw	White	Sopore
P2004	Screw	Blue	Delhi
P2005	Cam	Brown	Pulwama
P2006	Cog	Grey	Delhi

PROJECTS

PR_NO	PR_NAME	CITY
PR101	Gate Simulator	Srinagar
PR102	Remote Controller	Jammu

PR103	Door Opener	Doda
PR104	GIS	Sopore
PR105	SIS	Srinagar
PR106	Grade Calculator	Baramulla
PR107	Salary Manager	Badgam

SUP_PAR_PROJ

S_NO	P_NO	PR_NO	QUANTITY
S001	P2001	PR101	200
S001	P2001	PR104	700
S002	P2003	PR102	400
S002	P2002	PR107	200
S002	P2003	PR103	500
S003	P2005	PR105	500
S003	P2003	PR103	700
S003	P2004	PR103	200
S003	P2005	PR104	300
S004	P2006	PR102	400
S004	P2006	PR101	500

S004	P2006	PR103	600
S004	P2001	PR102	800
S004	P2003	PR104	900
S005	P2004	PR103	100

1) What are the Candidate keys and PRIMARY key to the relations?

2) What are the entity integrity constraints? Are there any domain constraints?

3) What are the referential integrity constraints in these relations?

4) What are referential actions you would suggest for these referential integrity constraints?

II. Consider the following relation

LIBRARY (member_id, member_name, book_code, book_name, issue_date, return_date)

The relation stores information about the issue and return of books in a Library to its members. A member can be issued many books. What are the anomalies in the relation above?

3) What are the functional dependencies in the relation above? Are there any constraints, especially domain constraint, in the relation?
4) Normalize the above relation to 3NF.

III. What is the need of dependency preservation?

IV. What is a lossless decomposition?

V. What are the steps for Normalization till BCNF?

VI. Fill in the blanks:

1. A database system is fully relational if it supports _____ and _____
2. A Relation resembles a _____, a tuple resembles a _____ and an attribute resembles a _____
3. A candidate key which is not a primary key is known as a _____key.
4. Which one of the following is not a traditional set operator defined on relational algebra?

   a. Union b. Intersection

   c. Difference d. Join

VII. Consider the relation and answer the questions given

EMP (ecode, ename, pcode, timespent)

a). Identify the candidate keys

b). Are these keys composite. If yes, which is the overlapping sttribute

c). Normalize the relation into BCNF, if needed

VIII. Consider the relation and answer the questions given

EMPLOYEE (ecode, ename, deptt., salary, project_no, p_termination_date)

a). Draw a dependency diagram for the relation

b). Identify the anomalies in the relation

c). What are the causes of these anomalies

d). Normalize the relation into 3NF and BCNF (if required)

Further/Suggested Readings

1. Elmasri R., Navathe B., "Fundamentals of Database Systems", Pearson Education.

2. Desai Bipin C., "An Introduction to Database Systems", Galgotia Publications Pvt. Ltd.

3. Silberschatz A., Korth Henry, Sudarshan S., "Database System Concepts", McGraw Hill Publications.