Uttar Pradesh Rajarshi Tondon open University

Master of Computer Science

MCS - 109

Database Management System

Database Management System

[2]

Uttar Pradesh Rajarshi Tondon open University

Master of Computer Science

MCS - 109

Database Management System

Block 1

Basic concepts of DBMS

Course Design Committee

Prof. Ashutosh Gupta **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Prayagraj

Prof. Suneeta Agarwal Member
Department of CSE
MNNIT, Prayagraj

Dr. Upendra Nath Tripathi Member
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare Member
Associate Professor, Department of Computer Science
University of , Prayagraj

Ms. Marisha **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Mr. Manoj Kumar Balwant **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Course Preparation Committee

Mr. Manoj Kumar Balwant **Author(Block 1: Unit 1, 2, 3)**
Assistant Professor (computer science)
School of Sciences, UPRTOU Prayagraj

Dr. Abhay Sexena **Editor**
Professor and Head, Department of Computer Science
Dev Sanskriti Vishwavidyalya, Haridwar, Uttrakhand

Prof. Ashutosh Gupta
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU Prayagraj

Mr. Manoj Kumar Balwant **Course Coordinator**
Assistant Professor (computer science)
School of Sciences, UPRTOU, Prayagraj

UPRTOU, Prayagraj-2021
MCS - 109 **, Database Management System**
ISBN –

Printed and Published by Vinay Kumar Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2024.

Printed By : Chandrakala Universal Pvt. Ltd. 42/7 Jawahar Lal Neharu Road, Prayagraj.

PEPE-149 / ETH-119

BLOCK INTRODUCTION

This block is organised into 3 units. The first unit introduces basic concepts of database and database management system. It starts with the need of DBMS over traditional file processing systems. After that, it presents the architecture and the data models on which they are designed. Then it briefly describes database language to create and maintain databases on the computer. Finally, it explains various components of DBMS such as query optimizer, DDL compiler, DML compiler, runtime database processor and stored data manager. The second unit presents entity-relationship diagram to represent a database. This ER diagram is then used to synthesis relational schemas and identify their attributes along with primary key and foreign key. The third unit illustrates the basic building blocks of relational data model which includes attributes, tuples and relations. It explains various types of constraints in a relational database table such as domain constraints, key constraints entity integrity constraints and referential integrity constraints. It gives understanding of what are the constraints that violate during insertion, deletion and updation operation. It describes how DBMS handles when these constraints violate.

UNIT-1 Introduction

Structure

1.1 Introduction

A Database is a collection of interrelated data. The data here refers to some known facts with implicit meaning. For example, consider you want to maintain the names, phone numbers, and addresses of your friends and relatives. You may store these data in computer software such as MS Excel or MS access. This collection of similar data with implicit meaning is called as database. Large commercial companies such as Flipkart and Amazon maintain their own large databases that contain data of over millions of books, movies, games, DVDs, electronics and other items. For example, the Flipkart database takes over many terabytes of spaces which is stored on varied computers (called servers). Around millions ofpeople daily visit Flipkart and use the database to make their purchase. The database is updated every time whenever a new item is purchased to reflect the updated stocks in the database. A softwarewhich store interrelated data into the database and access themwhen required is called database management system (DBMS). The primary goal of a DBMS is to conveniently store data and efficiently access it. The datais most important for many organizations, the DBMS system must ensure the safety of stored data against system crash or unauthorized access to the data.

1.2 Objectives

After studying this unit you will able to

- Differentiate between database and database management system.
- Explain advantages of DBMS over traditional file processing systems.
- Describe the architecture upon which DBMS is designed.
- Explain various data models on which database is designed.
- Learn Database language to create and maintain databases on the computer.
- Describe different interfaces to DBMS.

1.3 Examples of database approach

Consider a University database which maintains data related to its students, their courses and their grades. The database organises student data into several tables, each of which stores data as data records. Figure 1.1 shows database structure and a few data records for each database table.

SID	S_name	Contact_no	Date_of_Birth	Branch
S003	Dinesh	9915633342	03-03-1996	IT
S004	Anil	9915633987	07-03-1993	EC
. .	. . .	. . .	. . .	. . .
S005	Pankaj	9915633325	05-03-1994	CS

a. Student Table

CID	Course_name	SID	Credit
C01	Algorithm	S003	4
C02	DBMS	S004	3
. . .	. . .	. . .	
C03	Java	S005	4

b. Course Table

CID	SID	GRADE
C01	S003	B
C02	S004	A
. . .	. . .	. . .
C03	S005	A

c. Grade Table.

Figure 1.1: A university database to store student's details, enrolled courses and faculties information of a university.

Here, the student table stores student details. The course table stores details of each subject being taught and its associated department. The grade table grades of each student. Each record in a table stores different types of data elements. For example, the student table's record contains student name, student id and branch (such as IT for information technology, CS for computer science or EC for electronics &

communication). Each data record in the course table contains course name, course id, credit and student ID(a student which enrols this course). Each data record in the grade table contains student id, course id, and grade (from a set {'A','B','C','D','F','I'}.). You notice that the records in various files are interrelated. For example, data records of Dinesh and Saukat are related to their corresponding data records in the grade table. Similarly, the course table contains data records corresponding to CS and IT branches. The database can be manipulated whenever is required. The database manipulation involves querying and updating which include:

Querying

- Retrieve a list of all courses and grades.
- Retrieve names of students who have taken data structures courses.

Updating

- Add a new student to the student table.
- Enter grade A to Dinesh in data structure course.
- Delete record of Sandeep from student table.
- Change the grade of Shaukat from B to A.
- These are informal queries which must be specified precisely in a query language (which we will discuss later) so that the DBMS can process it.

1.4 Characteristics of Database Approach

The main characteristics of database approach over traditional file processing approach are as follows:

1 **Supports multiple views of the data:**A database presents different views of the database for different types of users. A view is a subset of a database or may be derived from the database and it is not separately stored in the database. For example, staffs working in the account department require different views (parts) of data than the staffs working in the sales department.

2 **Multi user and concurrent access:** DBMS allows several users to access a database at the same time. DBMS supports concurrency control whichfacilitate simultaneous updates on the same data item by several users in a controlled manner. For example, when multiple users are booking a seat in a train, it should not be allotted to more than one person.

3 **Self-describing nature of a database system:**A database system contains a database but along with descriptions of the database structure and its constraints in DBMS catalog. The information stored in the database catalog is called metadata.

4 **Programs and data independency:** DBMS approach allows changing data storage structures without changing DBMS access programs.

1.5 Advantage of using database approach

In earlier days, the traditional file processing system was an alternative to DBMS. In traditional file processing systems, all the data and information related to an organization are stored in different files. For example, the university's admission department maintains details of its students in one file while the examination department maintains results of students in another file. Similarly, the library unit maintains books issued to different students in different files. This way, each file carries information specific to a department or unit. This needs different application programs to extract records from appropriate files. The DBMS software has a number of major advantages over traditional file processing systems.

1. **Data redundancy:** In a traditional file processing system, each user group such as admission, examination and library maintain different files for students as per their needs. This causes data redundancy because the same information may be placed at several files. For example, the admission department stores the name, student ID, age, date of registration, which are also present in the file maintained by the examination department. This redundancy of data leads to higher storage and access cost. In DBMS, all user groups store information in only one single place.

2. **Data inconsistency:** In a file processing system, since there are various copies of the same data present over different files, a change in the data in one file will not reflect the change in all other places. For example, a change in student address in one file maintained by the admission section will not reflect the change in the address of the same student in another file maintained by the library department. This causes data inconsistency which is eliminated in DBMS.

3. **Data dependency:** The file processing system requires the user group to know the physical details of a file such as name of the file, the format of the file and location of the file in order to access data. DBMS hides these low level physical details from the user group and offers a more convenient approach to access data from the database.

4. **Integrity constraints:** DBMS offers integrity constraints to a database where the user group may specify a type of data or condition on data to be entered into the database. For example, in a banking database, each account number must be positive integers and should be exactly 12 digits. Such integrity constraints are not possible in a file processing system.

5. **Concurrency control:** To improve overall system performance and faster response time, multiple users are allowed to simultaneously access and update data. But, this concurrent update results in inconsistent data. The file processing systems may overcome this problem by locking a whole file even if only a small piece of data is needed. DBMS provides locks at record level to allow concurrent updates without resulting inconsistent data.

6. **Atomicity problem:** Atomicity means either executes an operation entirely or not at all. For example, if a program is transferring Rs 5000 from account A to account B, but meanwhile during the execution of the program a system failure occurs. Due to this, Rs 5000 was debited from account A but it was not credited to account B. So, we must ensure either both debit and credit occur or none of them occur. It is difficult to provide atomicity in a file processing system.

Check your progress

1. Why do the big commercial companies like Flipkart need database?
2. What are the capabilities provided by a DBMS?
3. What do you mean by data inconsistency and data redundancy?

1.6 Schemas and Instances

A database changes over time as we insert and delete information but its data types and constraints are generally fixed. A collection of records stored in a relation (or table) at any particular time is known as an instance of the relation. The description of the overall design of the relation which includes its name, data type of each attribute and constraints is called schema. The schema is considered as a template for describing data to be stored in the relation and it changes infrequently. For example, Figure 1.2 shows the schema for a student database which specifies the structure of each relation in the database but not the actual records. When we store student records in the schema, it will be referred to as its instances.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
StudentNumber	31	4
Class	35	4
Major	39	4

Figure 1.2- Internal storage format for student record.

1.7 DBMS architecture and database independency

Now, we will discuss three-schema architecture of DBMS which achieves three of the four important characteristic as discussed in section 1.4. Most DBMS including the modern DBMS are based on this architecture. The major goal of this architecture is to separate user applications from physical databases. This architecture describes data stored in a database at three levels of abstractions as shown in Figure 1.1. The data description at conceptual and external are defined by data definition language (DDL) of SQL. The three-schema architecture contains following are three levels:

1. **Logical schema or Conceptual level:**It describes stored data in a database in terms of the data model of a DBMS. For example in a relational model, it describes all relations in a database in terms of their data types, constraints and user operations. It describes relationships between different relations as separate relations. The Logical schema hides physical storage structure details of the database from the user and provides physical data independence. The process of designing a good logical schema is a part of conceptual database design.

2. **Physical schemaor internal level:**Physical schemadescribes how the relations described at conceptual level are actually stored on storage devices (e.g. disk). It specifies what file organisations and data structures (indexes) are used to speed up data access from the database. The process of designing a good physical schema is a part of physical database design.

3. **External schema or View level:**It is the highest level of abstraction which provides only a part of a database as per the need of the external user because many users require some parts of information from the database.Each database has only one logical and physical schema because it has only one set of relations in the database. But, it has many external schemas and each is designed for a particular user. Each external schema is not actually stored in the database, rather it is computed on demand for each type of usersusing existing relations in the database.

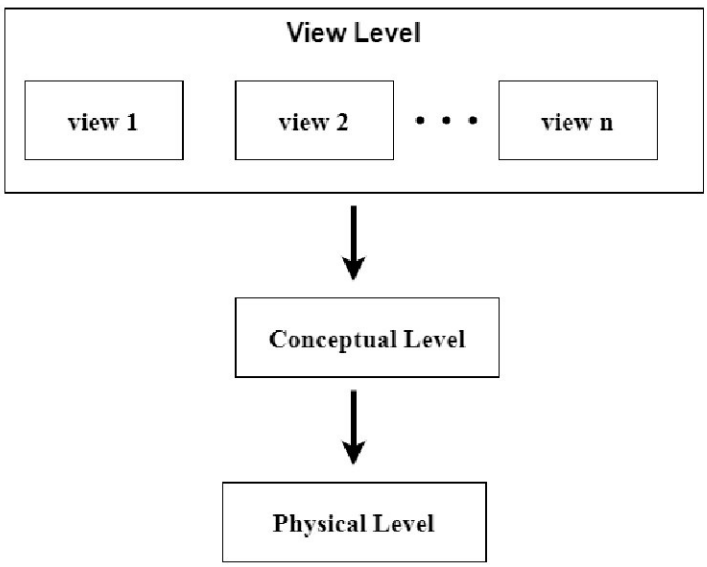


Figure 1.1- Three schema architecture[1]

The important benefit offered by three-schema architecture of DBMS is **data independency** which provides the ability for DBMS to modify schemas at one level without modifying schemas at the next higher level. This architectureprovides two types of data independency:

1. **Logical data independency:**It provides the ability for DBMS to modify logical schemas of a database without modifying its external schemas. The logical schemas are generally modified to incorporate new data types and constraints to a database.For example,changes in logical schemas include: adding a record type or data item, reduce the database size by removing record type or data item or changing constraints. The change in logical schemasonly results in change in view definitions and mappings. After the reorganization of the logical schemas, the external schemas that refers to the user group works as before.

2. **Physical data independency:**It provides ability for DBMSto modify physical schemas of a database without modifying its logical schemas.Changes to physical schemas such as reorganization of files are required to improve

retrieval and update performance. For example, reorganization of files helps in improving retrieval performance of some sections of records, while query to access these records remain unchanged. However, the query will execute faster in this scenario.

1.8 Data model

One fundamental characteristic of a DBMS approach is to hide data organization and storagefor providing an improved view of data. This is referred to as data abstraction. Data model is a high-level description of the overall structure of a database to achieve this data abstraction and provide how to store and access the data. The structure of the database refers to the data types, relationships and constraints that apply to the data to be stored in its relation. Today, several data models exist which are categorized below based on the type of concepts used for describing thestructure of the database.

Relational data model: Relational data model is the most popular data model which stores data in tables consisting of rows and columns. Each column has an attribute such as student ID and student name. Each row which is also called a **tuple**contains data related to attributes. Majority of current database systems are based on relational data models. Throughout this book, we will discuss relational data models. The relational data model was givenby E.F. Codd in 1970.

Object oriented data model: It is developed to store audio, video, graphics files in a relational database. This data model uses tables to store data, but it is not only limited to table. This data model is sometimes also called a hybrid data model.

Hierarchical data model: organizes data in a tree like structure where each record has a single parent. This model was good for describing many real world entities. This data model was initially used by IBM in the 60s. But due to the operational efficiency, it is rarely used now.

Networked data model: Network data model is built on the top of the hierarchical data model which allows many to many relationships between linked records. This allows multiple parents to record. This data model consists of sets of related records. Each set has one patent record and one or more member or child records. A record can be a member or child in multiple sets. It was popular in the 70s, but rarely used now.

1.9 Database languages and interfaces

Database language provides ability to users to design and maintain databases on computers. There exist a number of database languages which includes Oracle, MySQL, MS Access, dBase and FoxPro. However, SQL is the most widely used database language for both commercial and experimental purposes. We will study SQL in more detail in unit V. A database language broadly consists of two parts: Data definition language (DDL) and Data manipulation language (DML). DDL is used to specify schema of a database. Once the database schema is specified, the database is stored with data. DML offers insertions, deletions, retrievals and modifications of the data stored in the database.

Data Definition Language: Data definition language is used to create modify and remove the structure of database tables.

Data Manipulation Language: Data Manipulation Language provides access and manipulation of data organised in the database. DML offers four types of operations on the data:

- Insertion of new information in the database
- Deletion of existing information from the database.
- Modification of stored information in the database.
- Retrieval of stored information from the database.

DML is further classified as procedural and non-procedural DML.

Procedural DML: The procedural DML requires a programming language construct such as looping and conditional statements to retrieve and process each record from a set of records. A user needs to specify bothwhat type of data is needed and how to get those data?

Non procedural DML or high level DML: This can be used to specify complex database operations concisely. The non-procedural DML can be issued from monitor or terminal. A user is only required to specify what data is needed and there is no need to specify how to get those data.

DBMS provides a number of user friendly interfaces which are the following:

1 **Menu-Based Interfaces for Web Clients or Browsing:**These interfaces offer a list of options called menu to users that help them in formation of requests. A query is formed by a step by step process of picking options from the menu. Here, users do not need to memorize the command and Syntax of the query language.

2 **Forms-Based Interfaces:**These interfaces offer a form to users where they can fill all entries to add new data or only some entries to retrieve matching data. These interfaces are designed especially for naive users.

3 **Graphical User Interfaces:**They display database schema in diagrammatic form on screen where users specify a query by manipulating the schema diagram. Users use pointing devices like a mouse to select a part of the schema diagram.

4 **Interfaces for the DBA:**Most DBMS provide special commands such as creating an account, granting account authorization, changing schema and changing system parameters. These commands are only used by DBA staff.

Check your progress

1. How a database schema differs from a databaseinstance?
2. What is the purpose of three schema architecture of DBMS?
3. Which database language is commercially and widely used for DBMS?

1.10 Database system environment

DBMS is a complex software which consists of several software components. Figure 1.3 shows these software components in simple form. The top portion of the figure shows various users and interfaces with which they interact with DBMS. The DBA

staff designs database schema using DDL and other special commands. The casual users use interactive interfaces such as menu based and form based interaction to generate queries automatically. These queries are parsed by the query compiler to check their correctness as per the query syntax. These queries are then optimized by query optimizer which rearrange the possible operations of the queries and eliminates the redundancies. It consults with system catalog for physical information about stored data and generates executable queries. The parametric users who are responsible for data entry supply parameters through predefined transactions. The lower part of the figure shows the internal components that are responsible for data storage and transactions processing. The database is usually stored on disk which is controlled by the operating system through disk read/write operation. A stored data module of the DBMS controls access to information stored in the database. The DDL compiler is responsible for processing of schema definitions (as specified by DDL language) and storing schema descriptions into DBMS catalog. The DBMS catalog also holds other information such as names and datatypes of data items, constraints, mapping information among schema, names and size of files. Application programmers write application programs in high level languages such as c, c++ and Java. A precompiler then extracts DML commands from these application programs. Finally, these commands are converted into object code by the DML compiler. In the lower part of the figure, the runtime database processor is responsible for the execution of special commands, executable queries, and transactions with runtime parameters. The runtime database processor during its working interacts with the system catalog and stored data manager. The stored data manager uses basic operating system services to perform read/write operation between disk and main memory. In the Figure 1.3, Concurrency and backup and recovery are integrated into a single module which helps in transaction management by runtime database processor.

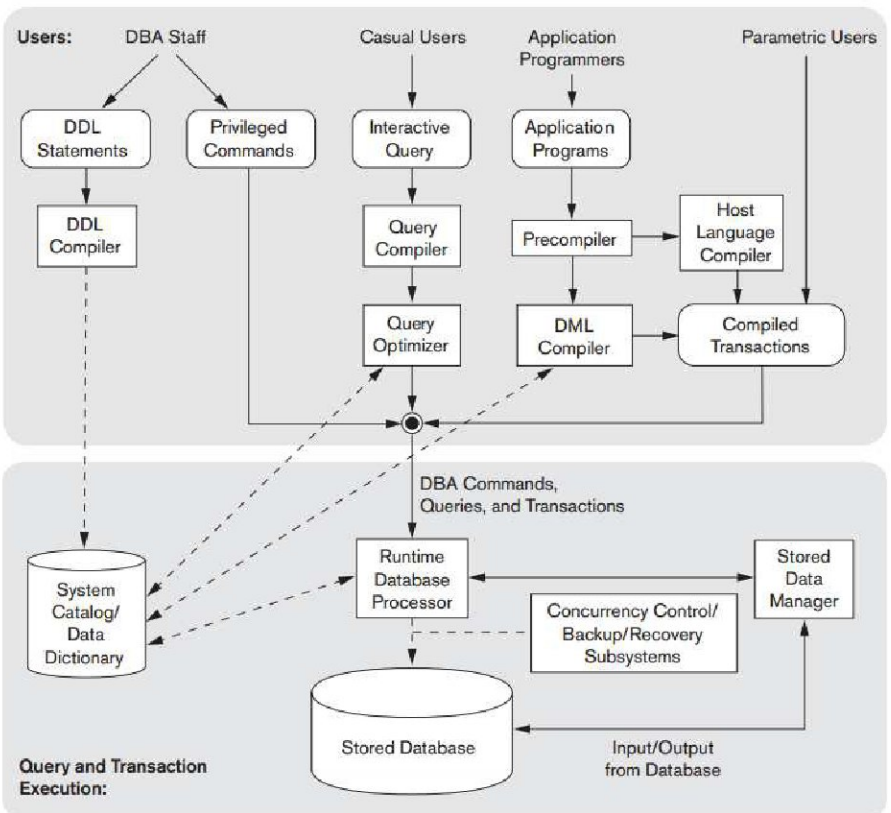


Figure 1.3- Component modules of DBMS and their interactions[1]

1.11 Summary

In summary, you have learned

- About database and database management system.
- Advantages of DBMS over traditional file processing systems.
- Different interfaces of DBMS which varies from menu and form based interfaces to commands and query based interfaces.
- Various software components of DBMS such as query optimizer, DDL compiler, DML compiler, runtime database processor and stored data manager.

1.12 Terminal Questions

1. What are various advantages of DBMS over traditional file processing systems?
2. What do you understand by database and database management system?
3. What do you mean by atomicity?
4. Explain the Three-Schema Architecture.
5. What are the purposes of physical data independence and logical data independence?
6. What do you understand about data models?
7. Briefly explain various types of data models?
8. How does database definition language differs from database manipulation language?
9. What is the purpose of database language? Explain types of database Language.
10. Describe the database system environment with appropriate diagrams.
11. What are the characteristics of the DBMS approach?

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

UNIT-2 Relational Data Model

Structure

2.1 Introduction

Earlier,network and hierarchical data models were quiet popular in 70's decade.However, today relational data model is used as the primary data model in almost all commercial data processing applications. The relational data model represents a database as a group of one or more relations where each relation is a table with rows and columns. This simple database representation allows even naïve person to understand the content of the database and allows us to easily express even complex data access operations in a high level language. This chapter will give you basic concepts of a relational data model. In this chapter, we will discuss how to represent the relational data model and what are relational constraints. Later, we will describe how the relational data model handles the violation of integrity constraints.

2.2 Objectives

After studying this unit you will able to:

- Understand basic concepts of relational data model which includes attributes, tuples and relations, domain of attributes.
- Analyse various types of constraints in a relational data model: domain constraints, key constraints, entity Integrity constraints and referential integrity constraints.
- Know which constraints violate during insertion, deletion and updation operation. And how does DBMS handle when one or more constraints violate.

2.3 Basic Relational data model Concepts

Relational data model represents a database as a collection of tables. Each table is given a unique name and each row of a table represents a collection of related data values. The row represents a real world entity or relationship. A table name and column name help in interpreting meaning of data values of a row. For example, consider a STUDENT table which contains attributes: SID, Age, Branch and Marks. The data values of any row can be interpreted with its corresponding column names.

SID	Age	Branch	Marks
S01	19	IT	68
S02	20	EC	66
S03	18	EC	67
S04	22	CS	73
S05	23	IT	74
S06	19	CS	76

Figure 2.1- A STUDENT table.

In the relational data model terminology, a table is called a relation, a row is called a tuple, and a column is called an attribute. Each attribute can take values from its domain. A domain D is a set of allowed individual values by its attribute and it can be specified by a data type. For example, a domain of an attribute Age is a set of all possible ages of students and each age can take integer value between 18 and 30.

2.4 Relational Databases and Relational Database Schemas

In the previous section, we have learnt about a single relation and its characteristics. A relational database consists of many such relations and each relation contains many tuples. A relational database schema S={R1,R2...Rn} is a set of relational schema R1,R2...Rn and a set of integrity constraints IC. A relational database state DB={r1,r2...rn} of a relational database schema S is a set of relational states r1,r2...rn of relational schemas R1,R2...Rn respectively such that each ri satisfies integrity constraints specified in IC. For example, figure 2.2 shows a relational database schema UNIVERSITY={STUDENT, COURSE, TEACHERS} of a university. The primary keys are underlined attributes. The STUDENT schema contains attributes SID which is unique to each student, S_name refers to names of students, Contact_no refers to telephone number of students, Date_of_Birth symbolizes the birth date of students and Branch refers to the name of the department. Attribute SID is the primary key of the relation. The SID uniquely identifies each tuple of the STUDENT table. The TEACHERS schema consists of attributes FID which contains faculty identifiers of each faculty, F_name refers to name of faculty, Course_id refers to the course taught by a faculty, Salary refers to monthly salary of a facility. In this relational schema attributes {FID,Course_id} is a candidate key of the relational schema.

SID	S_name	Contact_no	Date_of_Birth	Branch

a) STUDENT relational schema

CID	SID	Course_name	FID

b) COURSE relational schema

FID	F_name	Course_id	Salary

c) Teachers relational schema

Figure 2.2- A database schema UNIVERSITY={STUDENT, COURSE, TEACHERS} of a university.

Generally, when we refer to a relational database, we mean both its database schema and its current database state. The figure 2.3 shows a database state of the UNIVERSITY database which contains relational states corresponding to each relational schema STUDENT, COURSE and TEACHERS. Most DBMS provides SQL to define database schema and various integrity constraints.

SID	S_name	Contact_no	Date_of_Birth	Branch
S002	Sharad	9915633565	07-04-1994	IT
S003	Dinesh	9915633342	03-03-1996	EC
S004	Anil	9915633987	07-03-1993	EC
. .	. . .	. . .	. . .	. . .
S005	Nikhil	9915633325	05-03-1994	CS
S006	Vikash	9915633127	23-08-1993	IT
S007	Pankaj	9915633675	27-09-1993	CS
S008	Krishna	9915633872	07-12-1995	IT
S009	Sanjeev	9915633971	17-04-1993	EC

a) Relational states of STUDENT schema

CID	SID	Course_name	FID
C01	S003	Algorithm	F01
C02	S004	DBMS	F02
. . .	. . .	. . .	. . .

C03	S005	Java	F03
C04	S006	Basic Electronics	F04
C05	S003	Mathematics	F01

b) Relational states of COURSE schema.

FID	F_name	Course_id	Salary
F01	Arun	C01	70,000
F02	Anupma	C02	80,000
F03	Pradeep	C03	75,000
F04	Denesh	C04	90,000

c) Relational states of Teacher schema

Figure 2.3- A database state for database schema UNIVERSITY={STUDENT, COURSE, TEACHERS} of a university.

2.5 Relational Model Constraints

During the database design, we put several restrictions on the values to be inserted and what types of modification and deletion to be allowed. Constraints are a set of rules that ensures accuracy and reliability of data stored in a database table. Generally there are four types of constraints in a relational database: Domain Constraints, Key constraints, Entity Integrity Constraints and Referential Integrity Constraints.

1. **Domain Constraints**

It states that each attribute's value of a relation should be individual (not multiple values) and should belong to its valid set of values. A domain constraint of an attribute is specified by its data type. The data type can be numeric (such as integer, long integer), real numbers (such as float), characters, Booleans, date, time etc.

2. **Key constraints**

The key constraints include super key, candidate key and primary key.

Super key: A relation consists of many tuples and no two tuples can contain the same combination of values for all attributes.A set of attributes of a relation which ensures no two tuples can contain the same combination of values corresponding to these attributes is called a super key. For example, consider two tuples t1 and t2 of a relation R with a super key SK= {A1, A2…An}as some set of attributes, then

$t1\ [A1, A2…An] \neq t2\ [A1, A2…An]$

There may exist multiple sets of attributes (or subset of attributes) which hold this property. For example, {SID, Age}, {SID, Branch} are two super keys of the Student relation shown in Figure 2.1.

Candidate key:A set of minimum number of attributes of a relation that uniquely identifies each tuple of the relation is called candidate key.The values of these attributes of the candidate key are different for their corresponding tuples of the relation and there exists no subset of these attributes such that their values are different for all the tuples of the relation.It is one of the super keys of a relation that has a minimum set of attributes. In any relation, every candidate key is a super key. For example, {SID} and {Age, Branch} are two candidate keys of Student relation shown in Figure 2.1.

Primary key: A primary key is one of the candidate keys of a relation which is chosen by the database administrator. Similar to the candidate key, the values of attributes of a primary key are different for all tuples of a relation and hence it uniquely identifies each tuple of the relation.The primary key is generally shown underlined in ER diagram and database schema. It is chosen from any one of the candidate keys but generally it is one of the candidate keys with single attribute or minimum attributes. The other remaining candidate keys are called unique keys or alternate keys.

3. Entity Integrity Constraints

It ensures that a value of a primary key of a relation cannot be NULL. If two or more tuples of a relation contain NULL values in their primary key, then it cannot uniquely identify these tuples. And it is not possible to access these tuples.

.

4. Referential Integrity Constraints

So far we have seen constraints such as key and integrity constraints that are applicable on a single relation. A set of attributes FK of a relation R1 is called a foreign key of R1 which references another relation R2 (or same relation R1) if it satisfies two properties:

1. The foreign key FK must reference the primary key PK of relation R2 and both keys have the same domains.
2. Each value of FK in a tuple t1 of R1 either occurs as a value of PK in a tuple t2 of R2 (i.e. t1[FK]=t2[PK]) or is NULL.

In the above definition of foreign key, the relation R1 is called referencing relation and the relation R2 is called referenced relation. A referential integrity constraint from R1 to R2 holds if the above two properties. For example, consider the SUBJECT and STUDENT relation shown in Figure 2.4,2.1. The primary key for SUBJECT relation is {SID,Subject_code}.

SID	Subject_code	Subject_name
S02	C01	Operating System
S02	C02	DBMS
S04	C03	Algorithm
S04	C02	DBMS
S05	C01	Operating System

Figure 2.4- A SUBJECT relation.

The attribute SID of SUBJECT relation refers to the student in STUDENT relation who opted subject in SUBJECT relation. The SID attribute in SUBJECT relation serves as foreign key which references the primary key SID of STUDENT relation. A referential integrity constraint from SUBJECT to STUDENT hold if the values of foreign key SID of SUBJECT for each tuple in SUBJECT should have a matching value in primary key SID of STUDENT relation for some tuple. Otherwise, the value of the foreign key SID of SUBJECT relation should be NULL.

Illustrative Question: Given a relation R(ABCDE) with A and BC are its candidate keys. How many super keys are possible for the relation?

Solution:

Total number of super keys = number of super keys with candidate key A

+ number of super keys with candidate key BC

- number of super keys with both A and BC.

= number of subsets of attributes of R which contain A

+ number of subsets of attributes of R which contain BC

- number of subsets of attributes of R which contain ABC

$= 2^{(5-1)} + 2^{(5-2)} - 2^{(5-3)}$

$= 16 + 8 - 4$

$= 20$

Check your progress

1. How many super keys are possible for a relation R with n attributes?

2. Which of the following is NOT a superkey in a relational schema with attributes V,W,X,Y,Z and primary key VY?
   a) VXYZ
   b) VWXZ
   c) VWXY
   d) VWXYZ

3. Which one is correct with respect to RDBMS?
   a) primary key ⊆ super key ⊆ candidate key
   b) primary key ⊆ candidate key ⊆ super key
   c) super key ⊆ candidate key ⊆ primary key
   d) super key ⊆ primary key ⊆ candidate key

2.6 Dealing With Constraint Violations in Relational Database

There are three basic operations i.e. insert, delete and update that cause constraints violation in a database. Whenever these operations are performed, DBMS must ensure that they should not lead to constraints violation. In case of constraints violation, DBMS must perform the necessary actions.

On Insert Operation: When we insert a new tuple into a relation, we provide a list of values for each attribute. An insert operation can violate one or more constraints i.e. domain constraints, key constraints, entity integrity constraints and referential integrity constraints. If an insert operation violates one or more constraints, a default action is to reject the insert operation. The domain constraint violates if an attribute value provided during insert operation does not belong to its domain. The key constraint violates if an attribute value given during an insert operation already exists in another tuple for the same attribute. The entity Integrity constraintviolates if a NULL value is given to a primary key during a tuple insertion. The referential integrity constraint violates during a tuple insertion if the value of foreign key of a relation does not refers to any value of the primary key of another relation.

On Delete Operation: A delete operation removes a one or more tuple from a relation. The delete operation can violate only referential integrity constraints when a deleted tuple is referenced by some value of foreign key. If a referential integrity constraint violates with delete operation, we can choose several options. The first option is to reject the delete operation. The second option is to perform cascade (propagate) delete operation which removes all those tuples also that reference to a deleted tuple. For example, consider again STUDENT and SUBJECT relation shown in Figure 2.1 and 2.4. If we delete a tuple containing SID S04 from the STUDENT table, this violates the referential integrity constraints because this tuple is referenced by other two tuples of SUBJECT relation which contain foreign key values S04. So we need to delete these two tuples also from SUBJECT relation which contain 04 value in its foreign key.

On Update Operation: An update operation changes values of one or more attributes of a relation. We handle constraints violation during update operation by following ways:

- When the value of an attribute which is neither primary key nor foreign key modifies, the DBMS only ensures that the new value belongs to the domain of the attribute.
- If an attribute is being modified is primary key of a relation, then DBMS ensures that the new value is not NULL and unique.
- If we modify a value of foreign key, then DBMS ensures that the new value refers to some tuple in the referenced relation or it is NULL.
- If the update operation violates referential integrity constraints, then DBMS has all those options which we discussed in the delete operation.

Illustrative Question: The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade.

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

What are the tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted.

Solution: We are given that C is a foreign key which references to primary key A within the same table with delete on cascade. Therefore when the tuple (2,4) is deleted, all those tuples which contain 2 as the value of attribute C are also be deleted. Hence tuples (5,2) and (7,2) are deleted. The tuple (7,2) do not result to any other tuple which contain 7 as the attribute value of C, so this does not cause deletion of any other tuple. But when the tuple (5,2) is deleted, only tuple (9,5) is deleted because it is only tuple which contain value 5 for the attribute C. Therefore, tuples (5,2), (7,2) and (9,5) must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted.

Check your progress

1. Let R1 (a, b, c) and R2 (x, y, z) be two relations in which a is the foreign key of R1 that refers to the primary key of R2. Consider following four operations.
   a) Insert into R 1
   b) Insert into R 2
   c) Delete from R 1
   d) Delete from R 2

   Explain which of the above operations violate the referential integrity constraint.

2. Match the following with respect to RDBMS:

(a)	Entity integrity	(i)	enforces some specific business rule that do not fall into entity or domain
(b)	Domain integrity	(ii)	Rows can't be deleted which are used by other records
(c)	Referential integrity	(iii)	enforces valid entries for a column
(d)	Userdefined integrity	(iv)	No duplicate rows in a table

2.7 Summary

- We understand basic building blocks of a database which includes attributes, tuples and relations, domain of attributes.
- We explained key constraints: primary key, candidate key, super key and foreign key.

- We discussed various types of constraints in a database relation such as domain constraints, key constraints entity Integrity constraints and referential integrity constraints.
- We illustrated which constraints violate during insertion, deletion and updation operation. And how does DBMS handle when one or more constraints violate.

2.8 Terminal Questions

1. What are the constraints, whichviolate when a tuple is inserted in a relation?
2. How does DBMS deal when a deletion of a tuple causes violation of referential integrity constraints?
3. What are the constraints violated during the update operation and how does DBMS handle it?
4. Explain referential integrity constraints with a suitable example.
5. Explain the various constraints of database relations.
6. Briefly explain the following terms:
    a. Super key
    b. Candidate key
    c. Primary key
    d. Foreign key

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

UNIT-3Entity Relationship model

Structure

3.1 Introduction

In the last unit, you studied database schema. Now, in this unit you will learn how to design database schema. We will discuss Entity Relationship (ER) Model which helps in identifying entities to represent a database and relationships among these entities. ER model is a tool to model conceptual design of an organisation's database in terms of objects and their relationships. It allows us to develop the initial database design of an organisation and describe what the user wants from his database in a formal and precise way. In this unit, we will discuss relational database design and associated set of constraints. We will see how we can transform an entity relationship design to a set of relational schemas along with an associated set of constraints.

3.2 Objectives

After studying this unit you will able to:

- Understand each component of an ER diagram for any application.
- Draw an ER diagram for any database application.
- Transform any ER diagram to relational schemas and identify their attributes, primary key and foreign key.

3.3 Basic ER Model Concepts

Consider the database of NTPC limited company to illustrate basic concepts of ER model. Assume the database designers after collecting requirements and performing detailed analysis, provide the following description of the database:

1. The company has several departments and each department is identified by its unique name and number. There is an employee in each department who manages it and we need to keep track of when the employee starts managing the department.

2. A department manages several projects and each project is identified by a unique name and number and has a single location.

3. We need to store each employee's name,sex, date of birth, social security number, address and salary and. An employee is appointed to only one department but may work for several projects. In addition to this, there is a need to keep track of each employee's supervisor and number of hours per week an employee works on each project.

4. There is also a need to maintain dependant of each employee for insurance purposes. Each dependent includes name, sex, date of birth and relationship with the employee.

In the subsequent sections we will discuss the step by step process to derive schemas from above requirements. The Figure 3.1 shows final schemas for the above company database through .ER Diagram.Each component of the ER diagram will be explained gradually as we introduce the concepts of the ER model.

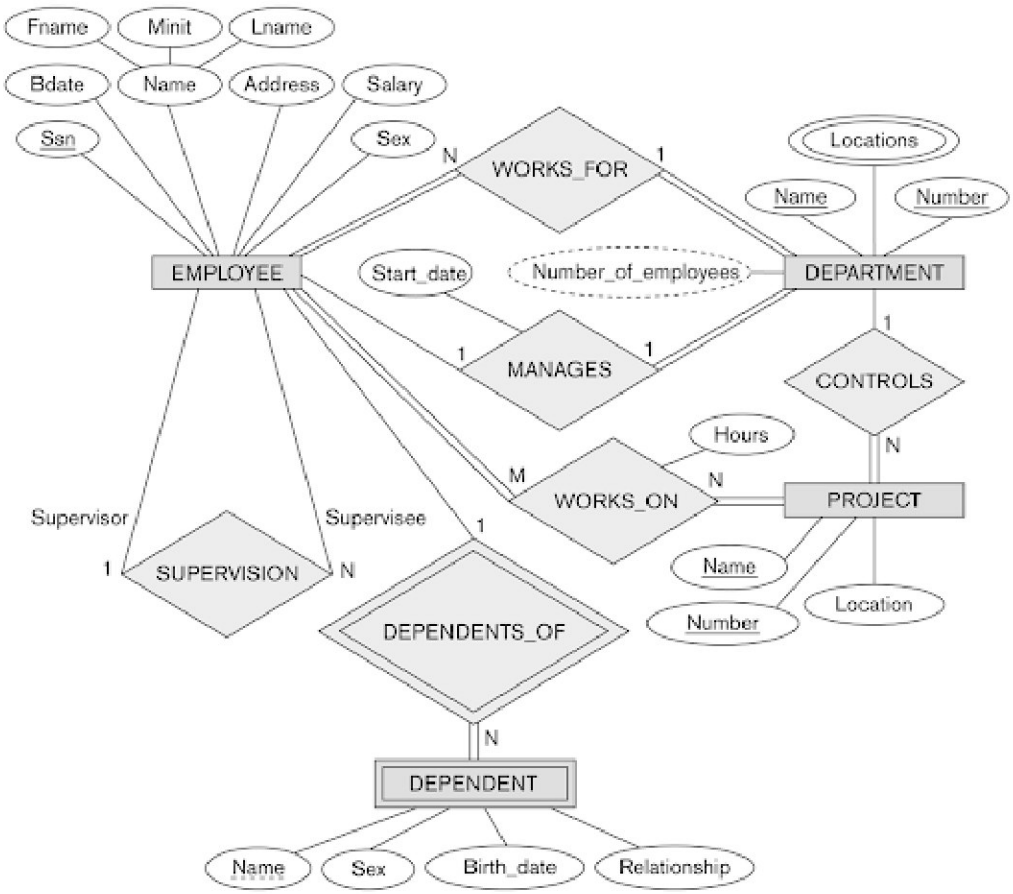


Figure 3.1 - An ER schema for the COMPANY database[1].

Entity:An entity is a real world object or thing that is different from other objects. For example, each employee in accompany is an entity. Attributes:Each entity is described by asset of properties called attributes. A set of values (one for each attribute) corresponds to an entity. Or example,an employee entity e1can be described by values of its attributes name, sex, date of birth, mobile as Raghav ,male,

06/08/1986,6765436785. While, another entitye 2 can be described with its attribute values Vishal,male, 02/04/1985, 5643345678. An attribute is represented in the ER diagram with an oval containing the name of the attribute.The values of the seen tities corresponding to its attributes become part of data in the database.

There are several types of attributes present in ER model:
Simple and Composite attributes: So far we have discussed simple attributes in our examples such as sex, date of birth and mobile. But, some attributes can further be divided into sub parts (or two or more attributes). For example, the attribute name can be further divided into three attributes: first_name, middle_name, last_name. Similarly, the address attribute can be divided into four subparts or attributes: Street, city, state, postal_code. In the ER diagram, a composite attribute is attached to its component attributes with straight lines as shown in Figure 3.2.

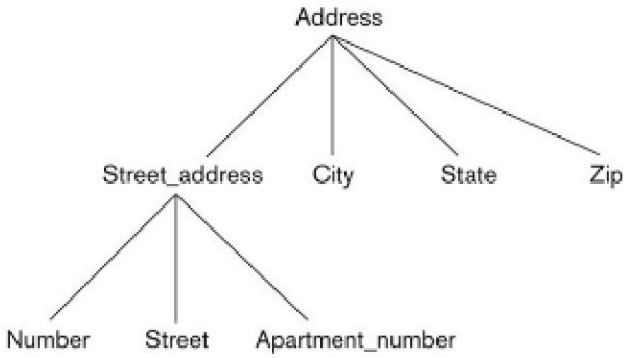


Figure 3.2- Address is a composite attribute of employee entity set[1].

Single valued and multivalued attributes: We have seen that each attribute has single value for its entity. For example, the attribute name and sex can have only single values. Such attributes are known as single valued attributes. But, there are many cases where an attribute can have several values possible for a specific entity. For example, an employee entity e1 may have one or more than one phone numbers. Such types of attributes are known as multivalued attributes. As another example, an attribute colour for a car entity c1 may have one or more colours. In an ER diagram a multivalued attribute is shown by a double oval containing the name of the multivalued attribute.

Derived attributes: The value of this attribute can be derived from the value of another attribute. For example, an attribute age for employee entity e1 can also be determined from its date_of_birth attribute. The date_of_birth attribute is referred to as base attribute or stored attribute.

An attribute may take null value, if an entity does not have a value for it. The null value may be treated as "not applicable", as the value does not exist for some entity. For example, some employees may not have middle_name. A null value may also indicate "unknown value" or "missing value" for its attribute. For example, if the date_of_birth attribute of an employee is null, then it is treated as missing value because every employee must have a date of birth.

Entity Types, Entity Sets: A collection of entities with the same attributes is called entity type. For example, an "employee" entity type will have the same attributes for each of its employee entities but have different attribute's values. Each entity type in a database is described by its name and attributes. In the ER

diagram, an entity type is represented by a rectangular box contain in gentity type name as shown in figure 3.3.An entity set is a set of the same type of entities that have the same attributes.In other words,an entity set is a collection of all entities that correspond to the same entity type in a database. For example,a set of all departments in a company is known as an entity set department. Similarly,a set of all employees in a company form an entity set employee.

EMPLOYEE
Name, Age, Salary

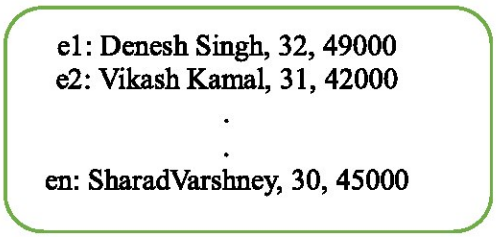


Figure 3.3- An EMPLOYEE entity type has an entity set with entities e1, e2…en as elements of the entity set.

Key attributes of an entity type: An attribute in an entity type that has district values for each entity in an entity set is called key attribute. There can be one or more key attributes in an entity type. In an ER diagram, a key attribute is denoted by its underlined name inside an oval.The key attribute is capable of uniquely identifying each entity in an entity set.For example, department name is a key attribute for entity type departments since not we department shave the same name.Similarly,social security number attributes no is a key attribute for employee entity type.Some times,acombination of several attributes serveasa key attribute.This type of key attribute is called composite key.The composite key is also capable of uniquely identifying each entity in an entity set.

3.4 Initial conceptual design of the company database

We can now start ER diagram with initial design of the company database which will be further refined after introduction of relationship concept. Based on the requirements described in in section 3.3, we have identified four entities for the company database:

1. Department: The entity type department has several attributes. These attributes are Name, Number, Locations, Manager, and Manager_start_date. Here, the attribute name and number are key attributes while, Locations is a multivalued attribute as shown in Figure 3.4.

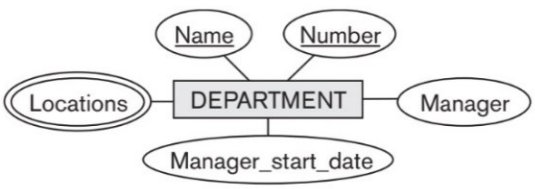


Figure 3.4- A Department entity type[1].

2. Project: The entity type Project has attributes: Name, Number, Location, and Controlling_department. Here, both Name and Number are key attributes as shown in Figure3.5.

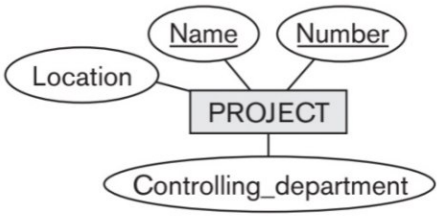


Figure 3.5- A Project entity type[1].

3. Employee: The entity type Employee has Name, Ssn, Sex, Address, Salary,Birth_date, Department, and Supervisor as attributes (as shown in Figure3.6). Here, the Name attribute is a composite attribute. While, the attribute Ssn is the only key attribute in it.

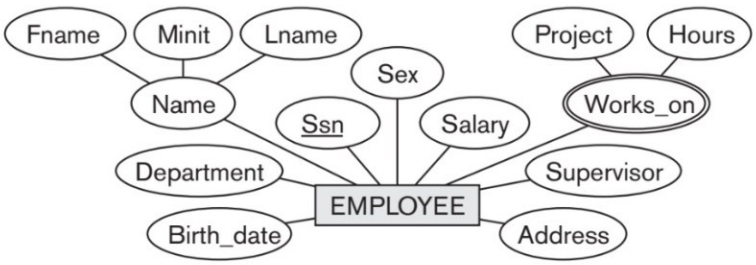


Figure 3.6- An Employee entity type[1].

4. Dependent: The entity type Dependent has attributes: Employee, Dependent_name, Sex, Birth_date, and Relationship as shown in Figure 3.7.

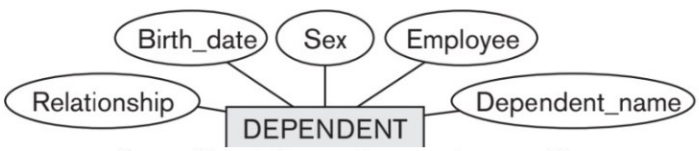


Figure 3.7- A Dependent entity type[1].

Check your progress

1. Given the basic ER and relational models, which of the following is INCORRECT?
   a) An attribute of an entity can have more than one value
   b) An attribute of an entity can be composite
   c) In a row of a relational table, an attribute can have more than one value
   d) In a row of a relational table, an attribute can have exactly one value or a NULL value

3.5 Relationship Types, Relationship Sets,Roles, and Structural Constraints

In the previous section, we noticed several relationships which exist among different entity types.These relationships exist whenever one entity type is associated with another entity type in some way. For example, there exists a relationship that associates an employee Vikash Kamal as the manager of the HR Department. Similarly, we can specify a relationship that associates a project "e-training" that is

controlled by the IT department. In the ER model these associations among different entity types are represented through relationships. Each relationship is formally called a relationship type R that shows association among in entity types E1, E2, ... En. A relationship set is a set of the same type of relationships. Formally, a relationship set R consists of a set of tuples where each tuple consists of entities (e1, e2, ..., en) representing a relationship such that e1,e2.....en ∈E1,E2,....En respectively .In other words, the relationship set R is a subset of the Cartesian product of entity set E1×E2,....×En. Each tuple in the relationship contains exactly one entity from each participating entity type and they all are related to each other in some way.

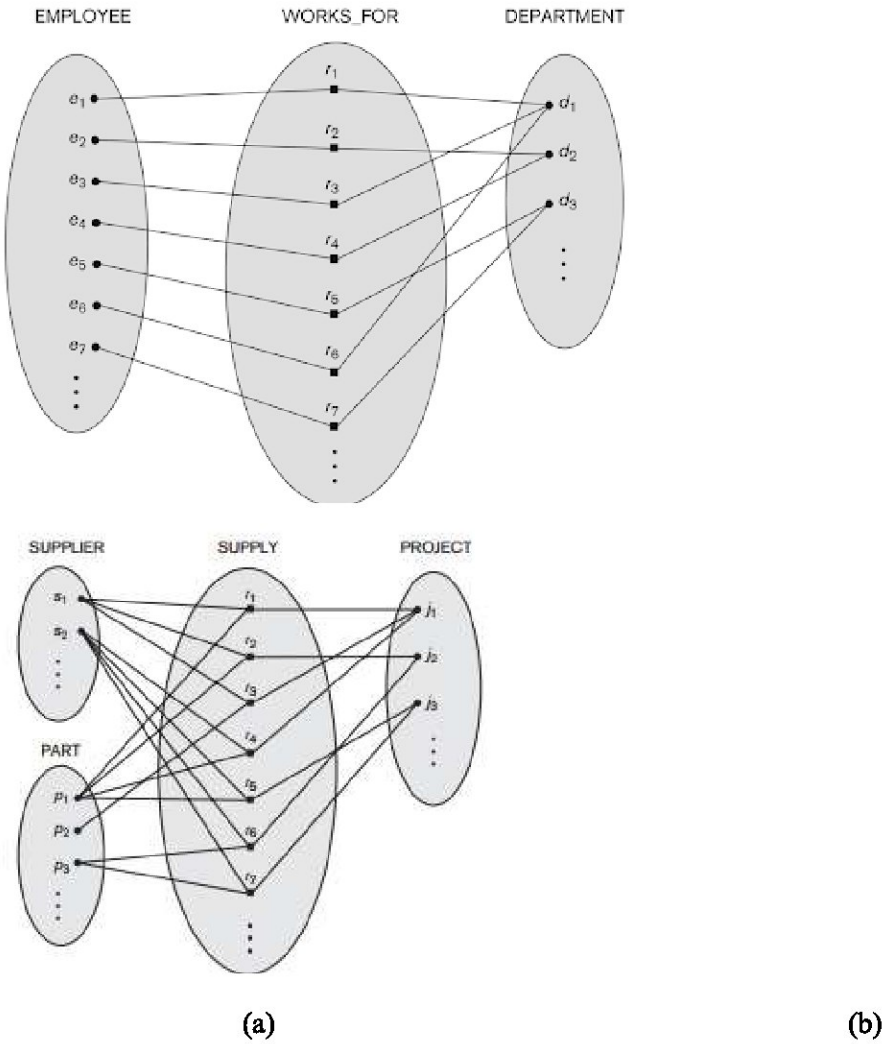


Figure 2.8- (a) A relationship set WORK_FOR between two entity types EMPLOYEE and DEPARTMENT. (b) A relationship set SUPPLY among three entity types SUPPLIER, PROJECT and PART[1].

For example, the Figure 2.8 shows a relationship set WORK_FOR between two entity types Employee and Department. In this relationship set, employee entities e1,e3 and e6 works for the department d1; employee e2 and e4 work for department d2; employee e5 and e7 work for department d3. In ER diagrams, the relationship types are represented with diamond shaped boxes which are connected to participating entity types by straight lines. A relationship name is shown inside the diamond shaped box.

Structure Constraints on Binary Relationship Types

Degree of a Relationship Type: The degree of a relationship type R is the number of entity types participating in the relationship type R. A relationship type of degree 2 is called binary relationship and a relationship type of degree 3 is called ternary relationship.
For example, the WORK_FOR relationship type shown in Figure 2.8(a) is a binary rel ationship because it involves two entity types EMPLOYEE and DEPARTMENT. The relationship type SUPPLY as shown in Figure 2.8(b) involves three entity types SUPPLIER, PROJECT and PART is a ternary relationship. The relationships can be of any degree but, binary relationships are most common.

Cardinality Ratio for Binary Relationship: The cardinality ratio for a binary relationship is the number of entities of an entity type to which another entity of another entity type can be associated via a relationship type. For example, the binary relationship type WORK_FOR involving entity types DEPARTMENT: EMPLOYEE is one to many (1:N) because each department employs many employees, but an employee works for only one department. Similarly, the relationship type MANAGES (as shown in Figure 3.9) between entity types DEPARTMENT and EMPLOYEE is one to one (1:1) because a department is managed by only one employee called manager and an employee as a manager manages only one department. Likewise, the relationship type WORK_ON (as shown in Figure 3.9) between two entity types EMPLOYEE and PROJECT has a cardinality ratio many to many (M:N) because an employee works for several projects and a project has several employees. This way, the possible cardinality ratios for binary relationship types are one-to-one,one to many, many to one and many to many. In the ER diagram, the cardinality ratio between two entity types related by a relationship is written on their respective lines connecting to each other by the relationship.

Participation Constraints: A participation of an entity set E in a relationship set R can be either total or partial participation. The participation is said to be total if each entity in E participates in at least one relationship in R. Otherwise, the participation is said to be partial, if only some entity in E participates in relationship set R. For example, the participation of an EMPLOYEE entity set in a relationship set WORK_FOR (as shown in figure 3.9) is total because every employee works for a department. There is no employee who does not work for any department. But, the participation of the entity the set Employee in relationship set MANAGES (as shown in Figure 3.9) is partial because every employee is not a manager who manages a department. In the ER diagram, a partial and a total participation is represented by a single solid line and double solid lines respectively. The cardinality ratio and participation constraints together are called structural constraints of a relationship set.

3.6 Weak Entity Types

An entity type that does not contain sufficient attributes to form a primary key is known as a weak entity type. Onthe other hand, an entity type that has a primary key is called as strong entity set. The entities of a weak entity type cannot be uniquely identified by their own. So, it must be associated with a strong entity type. The strong entity type associated with the weak entity type is called an identifying or owner entity set. The relationship type that associates the weak entity type with the identifying entity type is called an identifying relationship. For example, consider Figure 3.9 where a weak entity set DEPENDANT is associated with a strong entity set or identifying entity set EMPLOYEE with an identifying relationship DEPENDANT_OF.

A weak entity set always has a total participation with its identifying relationship because the weak entity set cannot be identified without its identifying entity set. So, the participation of the weak entity set DEPENDENT with its identifying relationship DEPENDENT_OF is total (indicated by double lines) as shown in figure 3.9. This means that every dependent entity must be related to some employee entity.
Although, a weak entity type does not have a primary key, it has a partial key or discriminator which can differentiate among all those entities that are dependent on a strong entity. It is a set of one or more attributes which can uniquely identify each weak entity for a given owner entity. In our example, the dependent name is unique for every employee, however it is not unique for dependent entities. In ER diagram, weak entity type and identifying relationship are represented by double rectangle and double diamond respectively. The partial key of a weak entity type is denoted by a dashed line.

3.7 Refining ER Design for the company Database

We can now refine the ER diagram shown in Figure 3.1 by removing some attributes from entity types that are refined into relationships. The following relationship types along with its cardinality ratio and participation constraint are identified from the requirements specified in section 3.3.

1. MANAGES: It is a one to one relationship between EMPLOYEE and DEPARTMENT. The EMPLOYEE participation in the relationship MANAGES is partial because every employee cannot be manager for some department. However, DEPARTMENT participation is total because every department must have a manager. WORKS_FOR: is a one to many relationship between DEPARTMENT and EMPLOYEE entity types. The participation of both department and employee entities are total because every department must have some employees and each employee must be associated to a department.

2. CONTROLS: It is a one to many relationship between DEPARTMENT and PROJECT entity types. The participation of DEPARTMENT is partial because some departments may not control any projects. While, the participation of PROJECT in CONTROLS relationship type is total because each project must be controlled by a department.

3. SUPERVISION: It is a one to many relationship between EMPLOYEE (as supervisor role) and EMPLOYEE (as supervisee role). The participation of both side entity types in the SUPERVISION relationship is partial because every employee cannot have a supervisor or can be a supervisor.

4. WORKS_ON: It is a many to many relationship between EMPLOYEE and PROJECT because a project can have several employees working on it and vice versa. There is total participation for both EMPLOYEE and PROJECT because every employee must work on some project and every project must have some employees.

5. DEPENDANTS_OF: It is a one to many relationship between EMPLOYEE and DEPENDANT. The participation of EMPLOYEE is partial, while that of DEPENDANT is total in the DEPENDANT_OF relationship type.

Figure 3.9 shows a refined ER Diagram after including the above refinement process.

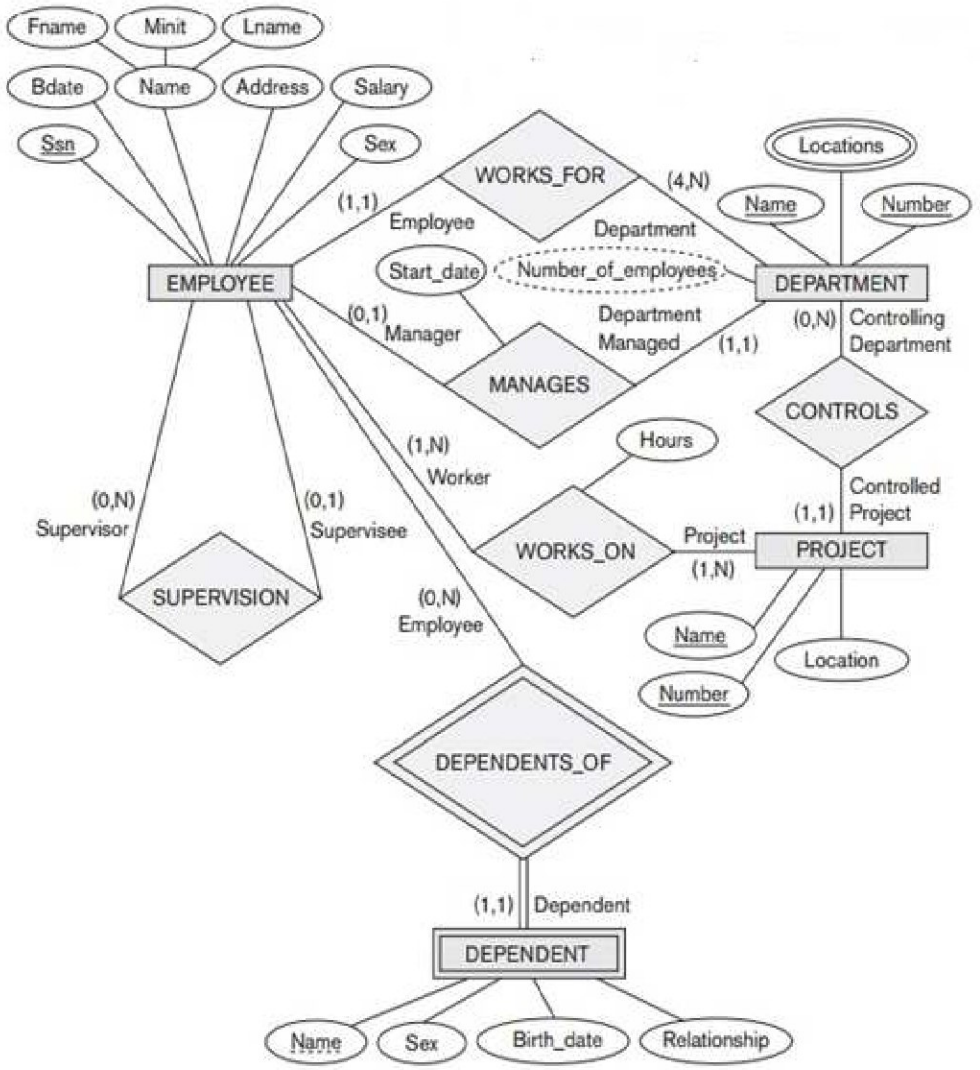


Figure 3.9- Refined ER Diagram of the company database[1].

Check your progress

1. In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set E1 to entity set E2. Assume that E1 and E2 participate totally in R and that the cardinality of E1 is greater that the cardinality of E2. Which one of the following is true about R?
   a) Every entity in E1 is associated with exactly one entity in E2.
   b) Some entity in E1 is associated with more than one entity in E2.
   c) Every entity in E2 is associated with exactly one entity in E1.
   d) Every entity in E2 is associated with at most one entity in E1.

3.8 Summary of ER Diagram

The set of various notations of ER diagram that we have used so far are summarized in Figure 2.10.

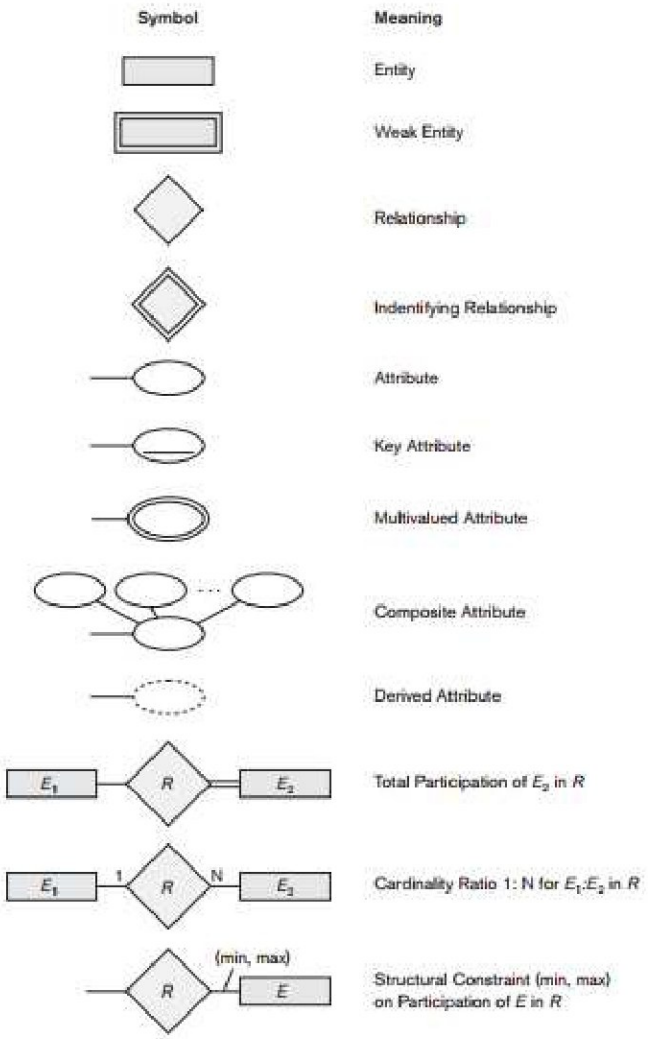


Figure 2.10- Summary of notation of ER diagram[1].

Naming Conventions: In the ER Diagram, we choose names of entity types and relationship types that convey meaning of them. Singular names are given to each entity type because this name represents to each individual entity of the entity type. The name of each entity type and relationship type is written with capital letters and its attributes name is written with capitalized letters. The role names in the relationships are written with lowercase letters. Commonly, entity types attributes names are nouns, while relationship types names are verbs. The relationship types names are chosen in such a way that the ER diagram is readable from left to right and top to bottom.

Design Choices for ER Diagram:

The schema design in the ER Diagram is an iterative refinement process. Initially, an initial design of schema is proposed which is then refined iteratively until the desired design is achieved. This refinement process is described through following considerations:

1. A concept is first modelled as an attribute and then it is refined into a relationship. Often, two complementary attributes are refined to a relationship type and these attributes are then removed from respective entity types.

2. An attribute which exists in several entity types is refined as a new entity type. For example, if an attribute Dependent occurs in multiple entity types STUDENT, INSTRUCTOR and COURSE, then we will make a new entity type DEPARTMENT.

3. An entity type which relates to only one entity type is refined as an attribute of the other entity type. For example, if an entity type DEPARTMENT relates to another entity type STUDENT, then it is refined as a new attribute in STUDENT entity type.

3.9 Conversion of ER Diagram to tables (or Relational Schema)

In this section, you will learn how an ER Diagram can be converted to relational schemas. The relational schemas also incorporate various constraints as represented in ER Diagram.

Representation of strong entity set:

1. **Strong Entity Set with simple attributes-**

A strong entity set E containing only simple descriptive attributes a1,a2……...an can be converted into a relational schema E with attributes a1,a2…...an. The primary key of the strong entity set serves as the primary key of the relational schema. For example, consider the entity set PROJECT of the ER diagram in Figure 3.9, which contains three attributes: Name, Number and Location. This entity set can be mapped to a relational schema PROJECT with the attributes:

PROJECT(Name, Number, Location)

Any one of the attributes: Name or Number (as chosen by database administrator) can serve as a primary key for the relational schema. Here, Name is the primary key for the relational schema PROJECT.

2. **Representation of Strong Entity Set with complex attributes-**

An entity Set E containing composite attributes is transformed to a relational schema by separating each composite attribute to its component attributes. For example, the entity set EMPLOYEE in Figure 3.9contains a composite attribute Name and its component attributes are FName, Mname and Lname. The relational schema derived from the entity set EMPLOYEE includes attributes Fname, Mname and Lname along with other simple attributes.

EMPLOYEE (SSn, Bdate, Fname,Mname,Lname, Address, Salary, Sex)

Representation of weak entity set: Consider a weak entity set A with attributes a1,a2,....am and its identifying entity set B consists of attributes b1,b2…...bm. The weak entity set A is transformed into a relational schema A with its primary key as a combination of its discriminator attribute and the primary key of B along with following attributes:

{a1,a2……..am} Union {b1,b2…bn}

While creating the primary key of the weak entity set, we also create a foreign key constraint on A. The attributes b1,b2…..bn reference the primary key of another relation B.

For example, consider the weak entity set DEPENDANT and its identifying entity set EMPLOYEE in Figure 7.15. The weak set DEPENDENT is transformed to a relational schema DEPENDANT with the following attributes:

DEPENDANT (Name, Ssn, Sex, Birth_date, Relationship)

Here, the primary key is Name, Ssn and the foreign key Ssn references the primary key of the EMPLOYEE relation.

Representation of Relationship set: Consider a relationship set R with attributes a1,a2,....am formed from union of primary keys of participating entity sets in R and b1,b2….bn are its deceptive attributes (if any). We can represent the schema of relational set R with following attributes:

{a1,a2……..am} Union {b1,b2…bn}

A. **Primary Key:** The primary key of the relation schema R can be chosen as follows:

   1. In a binary one to many or many to one relationship set, its primary key is the same as the primary key of the participating entity set from many side of the relationship set.
   2. The primary key of a binary one to one relationship set is taken from the primary key of either of the participating entity sets.
   3. The primary key of a many to many relationship set is the union of the primary keys of each participating entity set.

B. **Foreign Key:**We can also create a foreign key constraint on the relationship set R. The attributes of R which are the primary key of its participating entity set act as a foreign key of the relational schema R. For example consider the relationship set MANAGES (shown if Figure 3.9) which involves two participating entities EMPLOYEE and DEPARTMENT. The primary keys of the EMPLOYEE and DEPARTMENT entity sets are Ssn and Name respectively. Start_data is a descriptive attribute of the relationship set R. The primary key of the relation schema R will be the union of the primary keys {Ssn, Name}. We can create Ssn attribute as the foreign key of relation schema R which referencing the primary key of EMPLOYEE entity set.

Redundancy of schemas: A relational schema of a relationship set R will be redundant and does not present in the database, if the relational set R links a weak entity set to a strong entity set (i.e. one of the participating entity set is weak).For example, consider the relationship set DEPENDENTS_OF which involves EMPLOYEE and DEPENDENT as the participating entity sets. Since, the DEPENDENT is a weak entity set, its relational schema contains attributes: Name, Ssn, Sex, Birth_date, Relationship. The relational schema of DEPENDENTS_OF contains attributes: Name, Ssn. So every tuple which present is DEPENDENTS_OF relation would also be present in DEPENDENT relation. Hence, DEPENDENTS_OF schema is redundant and should not be presented in the database.

Combination of schemas:

Consider a relationship set AB involving entity set A and B. As discussed earlier, we require three schemas A, B and AB to represent these entity and relationship sets. In order to reduce redundancy further, we can combine the schema of relationship set as follows:

1. **Many to one relationship set:** If the relationship set AB is many to one from entity set A to entity set B and participation of entity set A in the relationship set AB is total (i.e. every entity in the entity set A participates in the relationship set AB). Then we can combine the schema of relational set AB with the schema of entity set A and its attributes are the union of attributes of both schemas A and AB.

2. **One to one relationship set:** If the relationship set AB is one to one from an entity set A to entity set B, then we can combine the schema of relationship set AB with either entity set A or B. The primary key of the combined schema is the primary key of the participating entity set schema into which the relationship set schema is merged.

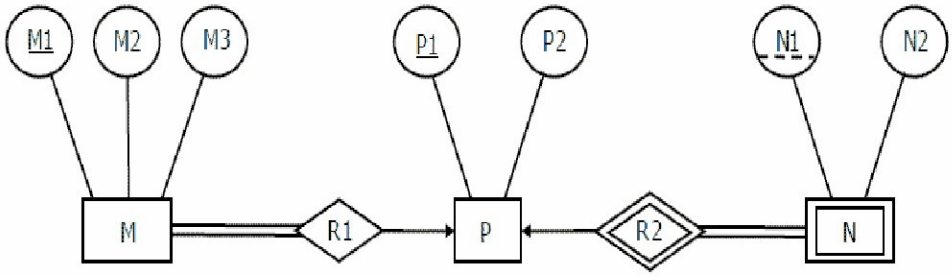
We remove the foreign key constraint of the relationship schema, if it references to the primary key of the entity set schema into which the relationship schema is merged. We add other foreign key constraint to the combined schema which referencing to other remaining entity set schema.

Representation of Composite and Multi valued Attributes: If an entity set consists of a composite attribute, then each component attribute is included asa separate attribute in the entity set schema. We discards the composite attribute itself in the schema. For example, the EMPLOYEE entity set containing a composite attribute "Name" will contain attributes in its schema: Fname, Minit and Lname.
If an entity set E contains a multivalued attribute M, then a separate schema is created for M, which contains an attribute M along with the attributes of the primary key of the entity set E. All the attributes of the new relation M serve as its primary key. The attributes of schema M that acts as the primary key of entity set E serve as foreign key referencing to primary key of the entity set E. For example, consider the multivalued attribute Locations of DEPARTMENT entity set. We create a relation Locations with following attributes:

Locations(Locations, Name)

The primary key of the relation Locations is Locations, Name. The Name attribute of Location schema is the foreign key referencing the primary key Name of the DEPARTMENT schema.

Illustrative Question: Consider the following ER diagram.



b) What is minimum number of tables needed to represent M, N, P, R1, R2.
c) What are the attribute set for these tables?

Solution:

Entity set: In the above ER diagram M and P are strong entities set, while N is a weak entity set. We require 3 tables to represent each entity set. The weak entity set when mapped to relational table, its discriminator attribute N1 along with the primary key P1 of entity set P act as the primary of table N. The relation schemas of these three tables are:

M(M1,M2,M3)

P(P1,P2)

N(N1,P1,N2)

Relationship set: The above ER diagram contains two relationship sets R1 and R2. Also, the participation of the entity set M and N are from many side of relationship set R1 and R2 respectively. The participation of entity set M and N are total which are indicated by double lines without arrow. So the relational schema of R1 and R2 can be merged with relation schema of entity sets M and N respectively. The primary key of the combined schema is the primary key of the entity set schema into which the relationship set schema is merged. Each combined schema M and N also requires an additional attribute P1 as a foreign key. Therefore relationship set R1 and R2 do not require any table separately but the table M and N will now contain additional attribute P1 which is the foreign key.

M(M1,M2,M3,P1)

N(N1,P1,N2,P1)

P(P1,P2)

Therefore the above ER diagram requires minimum 3 tables.

Check your progress

1. What is the min and max number of tables required to convert an ER diagram with 2 entities and 1 relationship between them with partial participation constraints of both entities?
2. Let E1 and E2 be two entities in an E/R diagram with simple single-valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one-to-many and R2 is many-to-many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model?

3.10 Summary

In summary:

- Entity type, entity set, weak entity set and strong entity set in ER diagram.
- We explained different types of attributes in ER diagram which includes: Single valued attribute, multivalued attribute, simple attribute, composite attribute, derived attribute.
- We discussed Relationship Types, Relationship Sets, Roles, and Structural Constraints of ER diagram along with total and partial participation in a relationship set.
- We illustrated how the cardinality ratio and participation constraints of a relationship set are represented in the ER diagram.
- We learned various notations and naming convention in ER diagram.
- We explained how an ER diagram can be reduced to relational schemas and how their attributes, primary key and foreign key are determined.

3.11 Terminal Questions

1. Differentiate between the following:
   a. Single valued attribute vs multivalued attribute
   b. Simple attribute vs composite attribute
2. What is the derived attribute?
3. Explain briefly the following terms.
   a. Entity
   b. Attribute
   c. Key attribute
   d. Composite key
   e. Degree of a relationship type
4. Describe the purpose of ER modelling.
5. How do the cardinality ratio and participation constraints of a relationship set are represented in the ER diagram.
6. How does the total participation differ from partial participation in a relationship set?
7. What is a weak entity set? How a weak entity set is represented in the ER diagram?
8. What do you mean by partial key or discriminator of a weak entity set? How is it represented in the ER diagram?
9. How any entity set in an ER diagram can be transformed into a relational schema.
10. Illustrate with an example how the primary key of a relational schema derived from a weak entity set is chosen.
11. Explain with a simple example how an entity set containing a composite attribute is transformed to a relational schema.
12. What are various naming conventions in ER Diagram?
13. What are Design Choices for an ER Diagram?
14. How is a primary key and foreign key of a binary relation schema determined?
15. Briefly explain redundant schema during reduction to relational schema from ER diagram.

16. Explains the scenario when a relational schema can be combined with one of its participating entity sets.
17. How do we represent composite and multivalued attributes in the schema of the entity set?

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

Uttar Pradesh Rajarshi Tondon
open University

Master of Computer Science

MCS - 109

Database Management System

Block 2

Query Language and Database Design Concepts

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Prayagraj

Prof. Suneeta Agarwal Member
Department of CSE
MNNIT, Prayagraj

Dr. Upendra Nath Tripathi Member
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare Member
Associate Professor, Department of Computer Science
University of , Prayagraj

Ms. Marisha — **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Course Preparation Committee

Mr. Manoj Kumar Balwant — **Author(Block 2 : Unit 4, 5, 6, 7)**
Assistant Professor (computer science)
School of Sciences, UPRTOU Prayagraj

Dr. Abhay Sexena — **Editor**
Professor and Head, Department of Computer Science
Dev Sanskriti Vishwavidyalya, Haridwar, Uttrakhand

Prof. Ashutosh Gupta
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU, Prayagraj

Mr. Manoj Kumar Balwant — **Course Coordinator**
Assistant Professor (computer science)
School of Sciences, UPRTOU, Prayagraj

UPRTOU, Prayagraj-2021
ISBN –

BLOCK INTRODUCTION

The second block is organized into four units. The fourth unit gives a basic idea of relational algebra upon which the database language SQL is designed. This unit explains use of unary relational operations (Select, Project and Rename), binary relational operations (CROSS PRODUCT, JOIN, DIVISION) and relational algebra operation from Set Theory (union, intersection, set difference). The fifth unit starts with similarities between SQL and relational algebra. It illustrates the order in which any SQL query evaluates. After that it explains nested query and correlated nested query and demonstrates how these queries are evaluated. Finally, it introduces various SQL commands for defining schema, changing schema and specifying basic constraints. The sixth unit presents the concept of functional dependencies. It illustrates how to determine these functional dependencies of any relational instance. It introduces attribute closure which helps in finding candidate keys of a relation from its functional dependencies. It presents functional dependency set closure, minimization of a functional dependency set and determining when two functional Dependency Sets are said to be equal. The last unit discusses various anomalies which arise due to the redundancy of data and the concept of normalization to get rid of anomalies and the data redundancies. It explains decomposition of a relational schema to different normal forms to ensure lossless and dependency preserving decomposition. Finally, it discusses multivalued functional dependency and join dependency upon which the fourth and fifth normal form are based.

UNIT-4 Relational Algebra

Structure

4.1 Introduction

SQL is considered as major reason for success of the relational database. It is a database language for data definitions, manipulations and updates. SQL is based on relational algebra and tuple relational calculus because it borrowed concepts from them. The relational algebra provides a basis for implementing and optimizing query languages such as SQL. In this unit, we will discuss various relational algebra operations such as σ(Segma), π (pi) and ρ(rho). The result of any relational algebra expression always gives a Relation. The operations σ(Segma), π (pi) and ρ(rho) are unary operations because they work on the single relation while others are binary operations.

4.2 Objectives

After study of this unit you will able to

- Understand basic ideas of relational algebra.
- Use unary relational operations: Select, Project and Rename.
- Analyze binary relational operations: CROSS PRODUCT, JOIN, DIVISION.
- Explain relational algebra operation from Set Theory: union, intersection, set difference.

4.3 Unary Relational Operations: Select, Project and Rename

Selection: The SELECT operation chooses only those tuples from a relation R that satisfy a selection condition. In other words, it picks some of the rows of a table and discard so there rows based on a specified condition. A general syntax of the selection operation is:

$\sigma_{<\text{selection condition}>}(R)$

Here, the symbol σ called as segma represents selection operation. The selection condition is specified with an expression that puts a condition on the attributes of relation R. The R can also be any relation algebra expression whose result is a relation. For example, consider the two statements given below.

$\Sigma_{Roll\_No=2}$(STUDENT)

$\sigma_{MARKS\_PERCENTAGE>80}$(STUDENT)

The first statement retrieves the tuple of STUDENT relation where student roll number is 2. The second statement selects those tuples from STUDENT relation whose percentage of marks is greater than 80. The result of these statements are shown in Figure 4.1.

Roll_no	**Name**	**Stream**	**Percentage_Marks**
1	Sharad	Mathematics	77
2	Anil	Mathematics	76
3	Dinesh	Biology	82
4	Pankaj	Commerce	81

(a) STUDENT Table

Roll_no	**Name**	**Stream**	**Percentage_ Marks**
2	Anil	Mathematics	76

(b) $\sigma_{Roll\_No=2}$(STUDENT)

Roll_no	**Name**	**Stream**	**Percentage_Marks**
3	Dinesh	Biology	82
4	Pankaj	Commerce	81

(c) $\sigma_{MARKS\_PERCENTAGE>80}$(STUDENT)

Figure 4.1 – Result of selection operator on STUDENT Table.

The general structure of <selection condition> of selection operation consists of following clauses:

<attribute name><comparison op><constant value>
or
<attribute name><comparison op><attribute name>

Here <attribute name> can be any attribute of relation R, <comparison op> can be any one of the operators {=, <, ≤, >, ≥, ≠}, and <constant value> is a constant value from

the attribute domain. The multiple selection conditions can also be combined by the standard Boolean operators {and, or, and not} or {∧, ∨, ¬}. For example, to retrieve students from STUDENT relation who opted for mathematics and their percentage of marks less than 40 with the following SELECT operation:

$\sigma_{(Subject='mathematics' \text{ AND } Percentage\_Marks<40)}$ (STUDENT)

A sequence of SELECT operations can also be applied and this can be done in any order. The sequence of SELECT operations can be specified with AND conjunctive as shown below:
σ<cond1>(σ<cond2>(...(σ<condn>(R)) ...)) = σ<cond1> AND<cond2> AND...AND <condn> (R)

In SQL query, the SELECT condition is specified with WHERE clause. For example, the SQL query for the relational algebra expression "$\sigma_{(Subject='mathematics' \text{ AND } Percentage\_Marks<40)}$ (STUDENT)" would be:

```
SELECT *
FROM STUDENT
WHERE Subject='mathematics' AND Percentage_Marks<40;
```

Projection: The PROJECT operation retrieves tuples with certain attributes as specified in the operation and discards others. In other words, if we want to select tuples containing only certain column so fatablethen we use the PROJECT operation. A general syntax of the PROJECT operation is:

$\pi_{<attribute\ list>}(R)$

Here, π (pi) symbol is the PROJECT operation, and the <attribute list> is a set of desired attributes of the relation R that we want to include in the final relation. The R can be any relational algebra expression whose result is a relation. The result of the PROJECT operation is a new relation which contains all tuples of the specified relation but corresponding to attributes specified in the <attribute list> and they will appear in the same order. For example, the following statement as shown in Figure 4.2 will give tuples with student's Name and Percentage_Marks from STUDENT relation.

Name	Percentage_Marks
Sharad	77
Anil	76
Dinesh	82
Pankaj	81

Figure 4.2 –$\pi_{Name, Percentage\_Marks}$(STUDENT)

If <attribute list> contains only nonkey attributes of R then duplicate tuples are possible. The PROJECT operation removes duplicate tuples so that it results in a set of distinct tuples to form a valid relation.

In SQL query, the attribute list is specified with SELECT clause. For example, the SQL query for relational algebra expression "$\pi_{\text{Name and Percentage\_Marks}}$(STUDENT)" would be:

SELECT DISTINCT Name, Percentage_ Marks FROM STUDENT;

Rename(ρ): Rename operator ρ(rho) is used to give a new name to a relation. A general syntax for rename operation is:

ρ(Relation2, Relation1)

For example, to rename a STUDENT relation to ALUMNI, we can use rename operator as follows:

ρ(ALUMNI, STUDENT)

We can also use rename operator to create a relation STUDENT_NAMES with Roll_no and Name from STUDENT relation and this can be done as follows:

ρ(MATHSTUDENT,$\pi_{(\text{ROLL\_NO,Name})}(\sigma_{(\text{Subject='mathematics'})}(\text{STUDENT}))$)

In SQL query, the rename operation is specified with an AS clause. For example, the SQL query for above relational algebra expression would be:

SELECT DISTINCT Roll_No, Name FROM STUDENT AS MATHSTUDENT WHERE Subject='mathematics'

Check your progress

1. Suppose R_1(A, B) and R_2(C, D) are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. Consider the relational alzebra expression given below:
   $\pi_B(r1) - \pi_C(r2) = \emptyset$
   If data in r_1 and r_2 satisfy referential integrity constraints, Explain is the above expression ALWAYS TRUE?

2. What is the optimized version of the relation algebra expression $\pi_{A1}(\pi_{A2}(\sigma_{F1}(\sigma_{F2}(r))))$ where $A1$, $A2$ are sets of attributes in r with $A1 \subset A2$ and $F1$, $F2$ are Boolean expressions based on the attributes in r?
   a) $\pi_{A1}(\sigma_{(F1 \wedge F2)}(r))$
   b) $\pi_{A1}(\sigma_{(F1 \vee F2)}(r))$
   c) $\pi_{A2}(\sigma_{(F1 \wedge F2)}(r))$
   d) $\pi_{A2}(\sigma_{(F1 \vee F2)}(r))$

4.4 Binary Relational Operations: CROSS PRODUCT, JOIN and DIVISION

CARTESIAN PRODUCT (CROSS PRODUCT) Operation: The cross product operation is denoted by ×. For two relation R(A1, A2, ..., An) and S(B1, B2,

..., Bm), the cross product operation R(A1, A2, ..., An) × S(B1, B2, ..., Bm) joins these two relations to give a new relation Q such that tuples in the relation Q is combination of each tuple from relation R with every other tuples from relation S. The new relation Q(A1, A2, ..., An, B1, B2, ..., Bm) contains n+m attributes and in the same order as they appear in the product operation. If the relation R has nR tuples and the relation S has nStuples then R × S give nR * nS tuples. The general syntax of product operation is:

Relation1 X Relation2

For example, consider the relations STUDENT and SPORT shown in Figure 4.3. The result of cross product operation STUDENT X SPORT is shown in Figure 5.4.

Roll_no	Name	Stream	Percentage_Marks
1	Sharad	Mathematics	77
2	Anil	Mathematics	76
3	Dinesh	Biology	82
4	Pankaj	Commerce	81

a) STUDENT relation

Roll_no	Sports
1	Badminton
3	Chess
5	Chess
7	Cricket

b) SPORT relation

Figure 4.3- a) STUDENT relation b) SPORT relation

Roll_no	Name	Stream	Percentage_Marks	Roll_no	Sports
1	Sharad	Mathematics	77	1	Badminton
1	Sharad	Mathematics	77	3	Chess
1	Sharad	Mathematics	77	5	Chess
1	Sharad	Mathematics	77	7	Cricket
2	Anil	Mathematics	76	1	Badminton

2	Anil	Mathematics	76	3	Chess
2	Anil	Mathematics	76	5	Chess
2	Anil	Mathematics	76	7	Cricket
3	Dinesh	Biology	82	1	Badminton
3	Dinesh	Biology	82	3	Chess
3	Dinesh	Biology	82	5	Chess
3	Dinesh	Biology	82	7	Cricket
4	Pankaj	Commerce	81	1	Badminton
4	Pankaj	Commerce	81	3	Chess
4	Pankaj	Commerce	81	5	Chess
4	Pankaj	Commerce	81	7	Cricket

Figure 4.4- Result of cross product operation STUDENT X SPORT

In SQL, the query for cross product operation of two or more relations is specified with relation names after FROM clause. For example, the SQL query for relational algebra expression "STUDENT X SPORT" would be:

SQL> SELECT STUDENT.Roll_no, Name, Stream, Percentage_Marks, SPORT.Roll_no, Sports FROM STUDENT, SPORT;

Join Operation: The join operation or Conditional Join ⋈ is used to join two or more relations based on some condition. The general structure of a JOIN operation on two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bm) is given as:

R⋈<join condition>S

The result of the JOIN operation gives a new relation Q(A1, A2, ..., An, B1, B2,... , Bm) with n + m attributes and in the order of attributes of R followed by S. The relation Q has all possible combinations of tuples from R and S, which satisfy the join condition. The tuples of Q contains each tuple from relation R combined with every other tuples from relation S but the tuples combinations for which the join condition evaluates to true are included in relation Q. While, the CARTESIAN PRODUCT give all combinations of tuples included in the result. A general form of join condition can be expressed as:

<condition> AND <condition> AND...AND <condition>

Here, each <condition> is of the form Ai θ Bj where Ai and Bj are attributes of R and S having the same domain, and θ (theta) is one of the comparison operators {=, <, ≤, >, ≥, ≠}.

For example, consider the relations STUDENT and SPORT shown in Figure 4.3. The result of conditional join operation STUDENT$\bowtie_{\text{STUDENT.ROLL_no} > \text{SPORT.Roll_no}}$ SPORT is shown in Figure 4.5.

Roll_no	Name	Stream	Percentage_Marks	Roll_no	Sport
2	Anil	Mathematics	76	1	Badminton
3	Dinesh	Biology	82	1	Badminton
4	Pankaj	Commerce	81	1	Badminton
4	Pankaj	Commerce	81	3	Chess

Figure 4.5- Result of conditional join operation STUDENT$\bowtie_{\text{STUDENT.ROLL_no} > \text{SPORT.Roll_no}}$SPORT.

The conditional join operation is similar to cross product followed by selection and projection operation. For example, the above join operation can also be specified as given below:

$\sigma_{(\text{STUDENT. Roll\_no} = \text{SPORT.Roll\_No})}(\text{STUDENT} \times \text{SPORT})$

Eqijoin: An equijoin is a special type of the conditional join operation where the only join condition allowed is the equality condition between a pair of attributes. The result of eqiijoin contain same values for the two attributes, but still both attributes appears in the final result.

Natural Join: A natural join is a special type of equijoin where the join condition equality is imposed on only those attributes that appear in both relations andthe common attributes appear only one time in the final result.Since, the result of natural join contains the same values for the two attributes, only one of the two attributes appears in the final result. For example, consider the relation STUDENT and SPORT shown in Figure 4.3. The result of natural join operation STUDENT $\bowtie$ SPORT is shown in Figure 4.6.

Roll_no	Name	Stream	Percentage_Marks	Sports
1	Sharad	Mathematics	77	Badminton
3	Dinesh	Biology	82	Chess

Figure 4.6- Result of the natural join operation STUDENT $\bowtie$ SPORT.

The join condition for above relational expression is equality condition on common attribute i.e. Roll_no. If the join attributes have the same names then there is no need to specify equality of attributes explicitly as the join condition.

Inner and Outer Join: The join operations that we have seen so far ignore the tuples that do not match the join condition. These join operations are generally called as inner joins. If we want a join operation to also include all tuples of R or all tuples of S or all tuples of both R and S that do not match the join condition is called as outer join. There are three outer joins: Left outer Join, Right Outer Join and Full outer Join.

Left outer Join: The left outer join on two relation R and S contains all tuples of inner join R⋈S along with the tuples of left side relation R that fail the join condition. The tuples of R that fail the join condition will have null values for corresponding to values of attributes of relation S. For example, consider the relation STUDENT and SPORT shown in Figure 4.3. The result of left outer join operation STUDENT $⋈_{STUDENT.ROLL_no>SPORT.Roll_no}$SPORT is shown in Figure 4.7.

Roll_no	Name	Stream	Percentage_Marks	Roll_no	Sports
2	Anil	Mathematics	76	1	Badminton
3	Dinesh	Biology	82	1	Badminton
4	Pankaj	Commerce	81	1	Badminton
4	Pankaj	Commerce	81	3	Chess
1	Sharad	Mathematics	77	Null	Null

Figure 4.7- Result of the left outer join operation STUDENT $⋈_{STUDENT.ROLL_no>SPORT.Roll_no}$SPORT.

Right Outer Join: The right outer join on two relation R and S contains all tuples of inner join R⋈S along with the tuples of right side relation R which fail the join condition. The tuples of S that fail the join condition will have null values for attributes values of relation R. For example, consider the relation STUDENT and SPORT shown in Figure 4.3. The result of right outer join operation STUDENT $⋈_{STUDENT.ROLL_no>SPORT.Roll_no}$SPORT is shown in Figure 4.8.

Roll_no	Name	Stream	Percentage_Marks	Roll_no	Sports
2	Anil	Mathematics	76	1	Badminton
3	Dinesh	Biology	82	1	Badminton
4	Pankaj	Commerce	81	1	Badminton
4	Pankaj	Commerce	81	3	Chess
Null	Null	Null	Null	5	Chess
Null	Null	Null	Null	7	Cricket

Figure 4.8- Result of the right outer join operation STUDENT $\bowtie_{STUDENT.ROLL\_no>SPORT.Roll\_no}$ SPORT.

Full outer Join: The full outer join on two relation R and S contains all tuples of inner join R⋈S along with the tuples of left and right side relation which fail the join condition. The tuples of R which fail the join condition will have null values for attributes of relation S and vice-versa. For example, consider the relation STUDENT and SPORT shown in Figure 4.3. The result of full outer join operation STUDENT $\bowtie_{STUDENT.ROLL_no>SPORT.Roll_no}$SPORT is shown in Figure4.9.

Roll_no	Name	Stream	Percentage_Marks	Roll_no	Sports
2	Anil	Mathematics	76	1	Badminton
3	Dinesh	Biology	82	1	Badminton
4	Pankaj	Commerce	81	1	Badminton
4	Pankaj	Commerce	81	3	Chess
1	Sharad	Mathematics	77	Null	Null
Null	Null	Null	Null	5	Chess
Null	Null	Null	Null	7	Cricket

Figure 4.9- Result of the full outer join operation STUDENT $\bowtie_{STUDENT.ROLL_no>SPORT.Roll_no}$SPORT.

Check your progress

1. Consider two relations R1(A,B) with the tuples(1, 5), (3, 7) and R2(A, C) = (1, 7), (4, 9). Assume that R(A,B,C) is the full natural outer join of R1 and R2. What are the tuples present in R(A,B,C)?

2. Consider the relations r(A, B) and s(B, C), where relation s contains B as a primary key and B of relation r is a foreign key referencing B of relation s. Let ⋈ denote the natural left outer-join operation. Assume that r and s contain no null values. Consider the query given below:
   I. $r⋈(\sigma_{B<5}(S))$
   II. $\sigma_{B<5}(r⋈S)$
   III. $\sigma_{B<5}(r⋈s)$
   IV. $\sigma_{B<5}(r) ⋈ s$
   Explain why do the above four queries are equivalent?

Division Operator: The division operation is denoted by /. Consider the relations SUBJECT and COURSE shown in Figure 4.10. Suppose we want to retrieve names of subjects which are taught in all courses. We can get this result by using division

operation / between names of subjects taught in some courses and names of all courses running. This can be expressed with relational algebra shown below:

$\pi_{Subject\_Name}(SUBJECT) / \pi_{Course\_Name}(COURSE)$

The result of full outer join operation $\pi_{Subject\_Name}(SUBJECT) / \pi_{Course\_Name}(COURSE)$ is shown in Figure4.11.

Subject_Name	**Course_Name**
Mathematics	BCA
Database	BCA
C language	BCA
Database	MCA

a) SUBJECT relation

Course_Name
MCA
BCA

b) COURSE relation

Figure 4.10- a) SUBJECT and b) COURSE relation

Subject_Name
Database

Figure 4.11- Result of division operation $\pi_{Subject\_Name}$(SUBJECT) / π_{Course_Name}(COURSE).

In general, the division operation on two relation A(X,Y) / B(Y) retrieves values of X for which there exist <X,Y> tuples for all Y values of relation B. The operation will be only valid if attributes of relation B are subset of relation A. The division operation can also be expressed by a sequence of π, $\times$, and – operations as follows:
$A(X,Y) / B(Y) = \pi_X(A) - \pi_X(\pi_X(A) \times B - A)$

Illustrative Example: Consider the following relations:
Supplier (Sid, Sname, rating)
Parts (Pid, Pname, colour)
Catalog(Sid, Pid, cost)

What are the relational alzebra expressions for following queries?

a. Retrive Sid of all suppliers whose rating greater than 7.
   Solution: $\pi_{Sid}(\sigma_{rating>7}(Supplier))$

b. Retrive Sid of all suppliers who supplies some red part.
   Solution:
   $\pi_{Sid}(\sigma_{color='red'}(Parts \bowtie Catalog))$
   or
   $\pi_{Sid}(\sigma_{color='red'}(Parts) \bowtie Catalog)$
   or
   $\pi_{Sid}(\sigma_{color='red'}(Parts) \bowtie \pi_{Sid,Pid}(Catalog)$

c. RetriveSname of all suppliers who supplies some red parts
   Solution:$\pi_{Sname}(\sigma_{color='red'}(Parts \bowtie Catalog) \bowtie Supplier)$

d. Retrive Sid of suppliers who supply some red parts or some green parts.
   Solution:$\pi_{Sid}(\sigma_{color='red' \vee color='green'}(Parts) \bowtie \pi_{Sid,Pid}(Catalog)$

e. Retrive Sid of suppliers who supply some red parts and some green parts.
   Solution:$\pi_{Sid}(\sigma_{color='red' \wedge color='green'}(Parts) \bowtie \pi_{Sid,Pid}(Catalog)$

f. Retrive Sid of suppliers who supply most expensive part.
   Solution:
   ρ (T1, Catalog)
   ρ (T2, Catalog)
   $\pi_{Sid}(\pi_{Sid,Pid}(Catalog) - \pi_{T1.Sid,\ T1.Pid}(\sigma_{T1.cost<T2.cost}(T1 \times T2))$

Check your progress

1. Consider the relational schema given below, where eId of the relation dependent is a foreign key referring to empId of the relation employee.
   Employee (empId, empName, empAge)
   dependent (depId, eId, depName, depAge)

   Assume that every employee has at least one associated dependent in the dependent relation. Consider the following relational algebra query:

   $\pi_{empId}(employee) - \pi_{empId}(employee \bowtie_{(empId=eID) \wedge (empAge \leq depAge)} Dependent$

   The above query evaluates to the set of empIds of employees whose age is greater than that ofall of his/her dependents. Is this statement true?

2. Consider a database that has the relation schema CR (Student Name, Course Name).An instance of the schema CR is as given below.

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB

SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

$T1 \leftarrow \pi_{CourseName}(\sigma_{StudentName='SA'}(CR))$
$T2 \leftarrow CR \div T1$
What are the rows in the result of T2?

4.5 Relational Algebra Operation from Set Theory

Union(∪), Interaction (∩) and set difference (-) are relational algebra operations from set theory. These operations can be applied on binary relations which are union compatible. Two relations R(A1, A2, …, An) and S(B1, B2, …, Bn) are said to be union compatible if:

1. They have the same number of attributes.
2. Each corresponding pair of similar attributes has the same domain or datatype.

For example, relations S1(Roll_no, Name) and S2(Roll_no, Name , Percentage_Mark) are not union compatible because S1 has 2 attributes while S2 has 3 attributes. Also, S2 (Roll_no, Name) and S2(Roll_no, Percentage_Marks) are not union compatible because datatype of Name is string while that of Percentage_Marks is integer numbers. But, relations S5(Roll_no, Name) and S6(S_ID, S_Name) are UNION COMPATIBLE because both have 2 attributes and their domains are also same which are integer and string respectively.

- **Union(∪):** The union operation of two relations R∪S is a relation which includes all tuples of relation R, all tuples of relationS and removes duplicate tuples. For example, consider the relations SUBJECT1 and SUBJECT2 shown in Figure 4.12.The result of SUBJECT1 ∪ SUBJECT2 is shown in Figure 4.13.

- **Intersection (∩):** The intersection of two relation R and S is represented by R∩S and it is a relation which includes all tuples that are present in both relations R and S. For example, the result of SUBJECT1 ∩SUBJECT2 is shown in Figure 4.14.

- **Set Difference (-):** The set difference of two relations R and S is represented by R-S gives a new relation which includes all tuples that are present in R but not in S. For example, the result of SUBJECT1 - SUBJECT2 is shown in Figure 4.15.

Subject_Name	Course_Name
Mathematics	BCA
Database	BCA
C language	BCA
Database	MCA

a) SUBJECT1

Subject_Name	Course_Name
Mathematics	BCA
Operating System	MCA
Computer Network	MCA

b) SUBJECT2

Figure 4.12- Relations SUBJECT1 and SUBJECT2 stores subjects and courses information.

Subject_Name	Course_Name
Mathematics	BCA
Database	BCA
C language	BCA
Database	MCA
Operating System	MCA
Computer Network	MCA

Figure 4.13- Result of SUBJECT1∪SUBJECT2 operation.

Subject_Name	Course_Name
Mathematics	BCA

Figure 4.14- Result of SUBJECT1∩SUBJECT2 operation.

Subject_Name	Course_Name
Database	BCA
C language	BCA
Database	MCA

Figure 4.15- Result of SUBJECT1-SUBJECT2 operation.

Check your progress

1. Given two union compatible relations $R_1(A,B)$ and $R_2(C,D)$. The result of the operation $R_1 \bowtie_{A=C \wedge B=D} R_2$ is equivalent to which one of the following.
   a) $R_1 \cup R_2$
   b) $R_1 \times R_2$
   c) $R_1 - R_2$
   d) $R_1 \cap R_2$.

2. Let R(P,Q,R1,R2,R3) and S(P,Q,S1,S2) be two relations schema, where {P,Q} is the key for both schemas. Consider the following two relational algebra expressions:
   I. $\pi_P (R \bowtie S)$
   II. $\pi_P (\pi_{P,Q} (R) \cap \pi_{P,Q}(S))$

   Explain whether the above two expressions are equivalent or not?

4.6 Summary

- We acquired basic ideas of relational algebra.
- We illustrated how to use unary relational operations: Select, Project and Rename work.
- We explained Binary relational operations: CROSS PRODUCT, JOIN, DIVISION and saw their applications.
- We discussed relational algebra operators from Set Theory: union, intersection, set difference.

4.7 Terminal Questions

1. Differentiate between entity Integrity constraints and key constraints.
2. What do you mean by union compatible relations?
3. Explain the differences between inner join and outer join.
4. Explain differences between left outer join, right outer join and full join with a suitable example.
5. Discuss division operation with a suitable example.
6. Write short notes on following relational algebra operations:
   i. Selection
   ii. Projection
   iii. Rename
7. What is the condition that the set operations must satisfy before it can be applied on any binary relations?

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

UNIT-5 Structure Query Language

Structure

5.1 Introduction

SQL is a standard language designed for accessing and manipulating data in relational databases. Generally, we refer to SQL as a query language. But, it can perform more than just querying a database. It can be used for defining structure of database, modifying data in database and specifying security constraints. SQL is based on relational algebra and tuple relational calculus because it borrowed concepts from them for implementing and optimizing query. All relational databases such as Oracle, MySql and MS access use SQL as their standard language. This unit gives you basic concepts for understanding SQL and its working.

5.2 Objectives

After study of this unit, you will able to:

- Understand similarity between SQL and relational algebra and the order in which any SQL query evaluates.
- Apply the set operators and check union compatible condition that must be satisfied by set operators.
- Use nested query and correlated nested query and understand how these queries are evaluated.
- Apply various SQL commands for different operations: Schema Change Statements, Specifying Basic Constraints in SQL, Data Definition and Data Types, INSERT, DELETE, and UPDATE.

5.3 Similarity between SQL and Relational Algebra

SQL uses the terms table, row, and column for relation, tuple, and attribute, respectively. There are lots of similarities between SQL and relational algebra expression because SQL borrows many concepts from relational algebra. This similarity can be illustrated with the following basic structure of SQL and its equivalent relational algebra expression.

SELECT DISTINCT $A_1,A_2 \ldots A_n$ FROM $R_1,R_2 \ldots R_n$ WHERE P

$$\Big\Vert$$

$$\pi_{A1,A2 \ldots An}(\sigma_P (R_1,R_2 \ldots R_n))$$

The SQL performs relational algebra product operation (×) with FROM clause, performs selection operation (σ) with WHERE clause and projection operation (π) with select clause.

Basic Structure: The result of any SQL query is not a relation because duplicate tuples are possible in the result. A general structure of an SQL expression contains basic clauses select, from and where along with optional clauses. The optional clauses can be used with the SQL to get desired results. The general structure of an SQL query has following form:

SELECT [DISTINCT] $A_1,A_2 \ldots A_n$ FROM $R_1,R_2 \ldots R_n$ [WHERE P] [GROUP BY attribute] [HAVING condition] [ORDER BY attributes [DESC]]

Here $A_1,A_2 \ldots A_n$ are attributes from relations $R_1,R_2 \ldots R_n$ and P is predicate condition on attributes of relations. The order of execution of each clause is as follows:

1. **FROM Clause:** In the SQL query, FROM clause is evaluated first. The FROM clause gives relational algebra product operation × of relations in the SQL clause. For example, consider the relations Book and Book_ Price shown in Figure 5.1.

ID	Title	Author	Publisher
001	Operating System	Galvin	Wiley India
002	Algorithm	Sahani	Universities Press
003	Computer Network	Forouzan	Tata McGraw-Hill
004	DBMS	Navathe	Pearson

c) Book relation

ID	Price	Quantity
001	669	50
002	595	70
004	885	60

d) Book_Price relation

Figure 5.1- The details of books are stored in relations Book and Book_Price.

The SQL query to retrieve the publisher of all books which are present in Library:

SELECT Publisher FROM Book, Book_Price WHERE Book.ID= Book_Price.ID

Publisher
Wiley India
Universities Press
Pearson

Figure 5.2- Result of SQL query when applied on Book and Book_Price relations.

Result of the above SQL query is shown in Figure 5.2. Similar to Relational Algebra, SQL also uses relation_name.attribute_name to avoid ambiguity if a same attribute name appears in more than one participating relations. For example, Book.ID= Book_Price.ID are used to differentiate between ID attribute of Book and Book_Price relation.

2. **WHERE clause:** The WHERE clause is executed next after FROM clause in SQL query. WHERE clause performs selection operation σ (segma) on attributes of participating relations. For example, the SQL query to retrieve title and author of all books whose price is greater than Rs 600 is given by:

SELECT Title, Author FROM Book, Book_Price WHERE Book.ID= Book_Price.ID AND Price>600.

The result of the above SQL query is shown in Figure 5.3. SQL uses the logical connectives AND, OR, NOT in WHERE clause to combine more than one predicate conditions. SQL allows us to compare strings and arithmetic expressions in predicate conditions with comparison operators <, <=, >, >=, =,and<>.

Title	Author
Operating System	Galvin
DBMS	Navathe

Figure 5.3- Result of SQL query when applied on Book and Book_Price relations.

3. GROUP BY:

Aggregation Operations: SQL offers several aggregate functions which take a set of values as input and produce a single value as output. SQL supports five aggregate functions which can be applied on any attribute of a relation. These functions are following:

- SUM([DISTINCT] attribute-name): It finds the sum of values in the specified attribute of a relation.
- COUNT([DISTINCT] attribute-name):It finds the number of values in a specified attribute.
- AVG([DISTINCT] attribute-name): It finds the average of all values in the specified attribute of a relation.
- MAX(attribute-name): It finds the maximum value in the specified attribute of a relation.
- MIN(attribute-name): It finds the minimum value in the specified attribute of a relation.

The DISTINCT clause is optional in all these functions. For example, consider the STUDENT relation shown in Figure 5.4.

SID	Branch	Marks
S01	IT	68
S02	EC	66
S03	EC	67
S04	CS	73
S05	IT	74
S06	CS	76

Figure 5.4- STUDENT relation containing branch and mark details of each student.

The SQL query to find average marks of CS branch of the relation is given by:

SELECT AVG(Marks) from STUDENT where branch = 'CS'

Result of the above SQL query is shown below.

Marks
74.5

Sometimes, we want to apply aggregate operation to multiple groups instead of a single group. We can specify this by using GROUP BY clause. The SQL GROUP BY clause is used to form groups which place the attribute entries with the same values

into one group. For example, the SQL query to find average marks of each branch is given below:

SELECT Branch, AVG (Marks) from STUDENT GROUP BY Branch

The result of above SQL query is given in Figure5.5.

Branch	Marks
CS	74.5
IT	71
EC	66.5

Figure 5.5- Result showing average marks of each branch of the STUDENT table.

If we wish to apply some condition on groups rather than tuples, then HAVING clause can be used after the groups are formed. For example, the SQL query to find only those branches whose average marks are greater than 70 is given as:

SELECT Branch, AVG (Marks) FROM STUDENT GROUP BY Branch HAVING AVG (Marks) > 70

The result of above SQL query is given below:

Branch	Marks
CS	74.5
IT	71

If a SQL query contains both WHERE and HAVING clause then it applies WHERE clause first. The tuples which satisfy WHERE predicate condition are then placed into groups by GROUP BY clause. SQL then applies HAVING clause which removes groups that do not satisfy HAVING condition. In the end, the SELECT clause retrieves tuples from remaining groups. For example, the SQL query to display only those branches which belong to CS or EC and their average marks are greater than 70 is given as:

SELECT Branch, AVG (Marks) FROM STUDENT WHERE Branch='CS' OR branch='EC' GROUP BY Branch HAVING AVG (Marks) > 70

The result of above SQL query is given below:

Branch	Marks
CS	74.5

4. **HAVING clause:** As discussed above, the HAVING clause evaluates after the GROUP BY clause. The HAVING clause applies a predicate condition on each group and selects only those groups which satisfy the predicate condition.

5. **SELECT:** The asterisk symbol " * " after SELECT clause gives result from all attributes. The SELECT may also contain arithmetic operations on attributes of a relation. For example consider the STUDENT relation shown in Figure 5.4. The SQL query to find detail of all students with Mark attribute is added 10 for each tuple in the result is given by:

   SELECT SID, branch, Marks+10 FROM STUDENT

   We can also specify desired attributes to be displayed as tuples in the result. This is specified with attribute names after the SELECT clause as given in above SQL query.

6. **DISTINCT:** This clause is optional and it is used to remove duplicate tuples in the result of SQL query. The DISTINCT clause comes after the SELECT clause. For example consider the STUDENT relation shown in Figure 5.4. The SQL query to list all Branches in the college is given as:

   SELECT DISTINCT Branch FROM STUDENT

7. **ORDER BY:** SQL offers ORDER BY clause which is executed in the end and it is used to control the order in which the tuples are displayed in the result. For example consider the STUDENT relation shown in Figure 5.4. The SQL query to display the details of student in ascending order of branch is as follows:

   SELECT * FROM STUDENT ORDER BY Branch

   By default, the ORDER BY clause lists the result in ascending order. We may specify sort order ASC for ascending order and DESC for descending order. For example, SQL query to list the tuples of student details in descending order is:

   SELECT * FROM STUDENT ORDER BY Branch DESC

Check your progress

1. Which of the following is aggregate function in SQL?
   a) Avg
   b) Select
   c) Ordered by
   d) Distinct

2. The employee information in a company is stored in the relation
   Employee (name, sex, salary, deptName)
   Consider the following SQL query:
   select deptNamefrom Employee
   where sex = 'M'
   group by deptName
   having avg (salary) > (select avg (salary) from Employee)

The above SQL query returns the names of the department in which.....

5.4 Set Operations

SQL offers UNION, INTERSECTION and EXCEPT operations to perform relational algebra ∪, ∩,and−. Similar to relational algebra set operation, the participating relations in SQL query must also be union compatible for applying these operations. The SQL UNION operation between two relations gives a set of all tuples from both relations and removes duplicate tuples. The SQL Intersection operation between two relations gives all tuples that are present in both relations. The Except operation between two relations R-S gives all tuples that are present in R but not in S. For example, consider Book and Book_Price relations shown in Figure 5.1. The SQL query to display all book IDs of both Book and Book_Price relations is given as:

(SELECT ID FROM Book) UNION (SELECT ID FROM Book_Price)

UNION operation automatically removes duplicate tuples from the result. But if we want to retain duplicates in the result, then the UNION ALL clause can be used as given below.

(SELECT ID FROM Book) UNION ALL (SELECT ID FROM Book_Price)

The SQL query to display all book IDs that are present in both Book and Book_Price relations is given as:

(SELECT ID FROM Book) INTERSECTION (SELECT ID FROM Book_Price)

The first part of the above SQL query gives SID of books 001,002,003 and 004. While, the second part gives 001,002 and 004. So the final result of the above INTERSECTION query gives SID 001,002 and 004. As discussed earlier, if we want to retain duplicates in the final result then INTERSECTION ALL and EXCEPT ALL can be used in place of INTERSECTION and EXCEPT. One important thing to note that UNION, INTERSECTION and EXCEPT can be used on any two tables which are union compatible. This means the number of columns in both tables should be equal and when columns of both tables considered in an order both should have the same data types.

5.5 Nested Query and Correlated Nested Query

Nested Query: A nested query is a query which has another query inside it. The inner query fetches existing values from a table and then it is used as a comparison condition for execution of outer query. The Nested query executes from innermost query to outermost query. The execution of each inner query is independent of its subsequent outer query. But, the result of each inner query is used as a comparison condition for execution of its subsequent outer query. The "IN" and "NOT IN" keywords are used to combine each outer query with its subsequent nested query. For example consider the Book and Book_Price table shown in Figure. The SQL query to retrieve IDs of books whose quantity is either 50 or 60 is given as.

SELECT ID from Book where ID

```
IN
(SELECT ID from Book_Price where Quantity = 50 OR Quantity=60)
```

In the above SQL query, the inner query will give a set with IDs 001 and 004 because their quantities are 50 and 60 respectively. The outer query will return IDs of books which are equal to any member of the set with IDs 001 and 004 (as returned by inner query). So, the final result will contain the ID of books 001 and 004.

If we wish to find Title of books whose quantity is neither 50 nor 60, its SQL query is given by:

```
SELECT Title from Book where ID
NOT IN
(SELECT ID from Book_Price where Quantity = 50 OR Quantity=60)
```

In the above SQL query, the inner query will give a set with IDs 001 and 004 because their quantities are 50 and 60 respectively. The outer query will return Titles of books whose IDs are not the member of the set with IDs 001 and 004 (as returned by inner query). So, the final result will contain Titles of books "Operations System" and "DBMS".

- **"IN" Equivalent to " =ANY":** The IN" operator is equivalent to " =ANY". So, both keywords have the same effect and give the same result when used in SQL query.

- **"NOT IN" Equivalent to "<>All":** The "NOT IN" operator is equivalent to "<>ANY". Both keywords have the same effect and give the same result when used in a SQL query.

The keywords "ANY" and "ALL" can also be combined with >, >=, <, <=, and <> with nested query to get desired results. But, this will give different results as compared to "IN"and "NOT NOT" operators.

Correlated Nested Query: In nested query, we have seen that an inner query is completely independent of its subsequent outer query. In Correlated Nested Query, the inner query is dependent on its subsequent outer query. More precisely, the inner query executes every time for each tuple of the table in the outer query. The "EXIST" operator is used to compare for each tuple of the outer query table whether the result of the inner query is non empty. In the correlated query, the where clause of the nested query contains some attribute of the table of the outer query. For example, again consider the tables shown in Figure. The correlated nested query to find Titles of books whose price less than Rs 600 is given by:

```
SELECT Title from Book where
EXISTS
(SELECT * from Book_Price where Book.ID== Book_Price.ID AND Price<600)
```

In the above SQL query, for each tuple of Book table, it finds a set of tuples from Book_Price table where Book.ID== Book_Price.ID AND Price<600 by executing the inner query. If the set contains at least one tuple, then the inner query returns true and the book ID contains in the final result. This happens for each book ID of the Book table. The final result of the above query contains book titles "Algorithm".

We can also use "NOT EXISTS" operator in the correlated nested query. For example, if we want to find Book Titles whose price is not less than Rs 600, the correlated nested query can be given as:

SELECT Title from Book where
NOT EXISTS
(SELECT * from Book_Price where Book.ID== Book_Price.ID AND Price<600)

The result of the above SQL query contains book titles "Operations System" and "DBMS".

Check your progress

1. Consider the table employee(empId, name, department, salary). Assuming that department 5 has more than one employee. What is the output of following SQL query:
   Select e.empId From employee e
   Where not exists
   (Select * From employee s where s.department = "5" and s.salary >=e.salary)

2. Consider the following relation
   Cinema (theater, address, capacity)
   What is the output of the following SQL query?
   SELECT P1. AddressFROM Cinema P1
   WHERE P1. Capacity> = All (select P2. Capacity from Cinema P2)

5.6 Comparison with NULL

SQL supports a special value called NULL. There are three different interpretations of NULL in SQL.

1. **Unknown value (a value exists but not known):** If a person does not know his date of birth then the Date_of_Birth attribute would be NULL for this person.

2. **Unavailable value (a value exists but intentionally not shown):**An attribute Personal_Mobile would be NULL for a person if he does not want to show his personal mobile number in the database.

3. **Not applicable (a value is undefined for some tuples under an attribute):**An attribute Spouse_Name would be NULL for a person who is unmarried.

So each NULL value is different from other NULL values when used in a table. It is impossible for SQL to determine which of the meanings of NULL is intended when NULL is used in the database table. When SQL encounters any NULL value in comparison, the result is considered as Unknown. The SQL uses three logic values: TRUE, FALSE, and UNKNOWN. The result of any comparison in SQL can be TRUE or FALSE or UNKNOWN when logical connectives AND, OR, and NOT are used.

SQL offers "IS" and "IS NOT" operators to check whether an attribute value is NULL. For example, the SQL query to find Book ID whose quantity is not NULL is:

SELECT ID from Book_Price where Quantity IS NOT NULL

5.7 SQL Commands

SQL offers various commands to interact with relational databases. These commands are broadly classified into three groups based on their nature: DDL - Data Definition Language, DML - Data Manipulation Language, DCL - Data Control Language

	Command	Description
DDL - Data Definition Language	CREATE	Creates a new table, a view of a table, or other object in the database.
	ALTER	Modifies an existing database object, such as a table.
	DROP	Deletes an entire table, a view of a table or other objects in the database.
DML - Data Manipulation Language	SELECT	Retrieves certain records from one or more tables.
	INSERT	Inserts a new tuple in a table.
	UPDATE	Modifies tuples of a table.
	DELETE	Deletes tuples.
DCL - Data Control Language	GRANT	Gives a privilege to users.
	REVOKE	Takes back privileges granted to users.

5.8 INSERT, DELETE, and UPDATE Statements in SQL

SQL offers three commands to modify a database: INSERT, DELETE, and UPDATE.

INSERT: This command is used to add single tuples in an existing table. The Insert command takes the table name and a list of values for all attributes in the same order in which attributes appear in the table. For example, the SQL query to insert a new book in the Book table is:

INSERT INTO Book
VALUES (005, 'Let Us C', 'yashwantkanetkar', 'BPB Publication')

DELETE: The delete command is used to delete tuples from a relation. The where clause in delete command is used to specify tuples to be deleted. Based on the condition specified with where clause, zero or more tuples are deleted by a single delete command. For example, the below statement removes all those tuples from the Book table in which author is "Galvin".

DELETE FROM Book
WHERE Author='Galvin'

If the WHERE clause is not specified in the delete command, it will delete all tuples from the table. For example, the below statement deletes all records from the Book table.

DELETE FROM Book

However, the table still remains in the database with empty tuples. If we wish to delete table definition, DROP command can be used. For example, the below statement completely removes the Book table from the database.

DROP TABLE Book

UPDATE: The update command is used to modify attributes values of a table. A WHERE clause is used in an update command to select tuples to be modified in a table. For example, the following command modified values of a book tuple with ID= 005.

UPDATE Book
SET Title = 'C Programming Language', Author = 'Dennis Ritchie'
WHERE ID=005;

The update command can also be used to modify several tuples with a single update command. For example, the below SQL query modifies several tuples from the Book table that contain Publisher as ' PHI Learning'.

UPDATE Book_Price
SET Quantity = Quantity + 50
WHERE Publisher = ' PHI Learning'

5.9 SQL Data Definition and Data Types

In SQL, a new table is created with CREATE TABLE command by specifying table name, its attributes and initial constraints. Each attribute is specified with an attribute name, its data type and an attribute constraint (such as NOT NULL) which is optional. The data type specifies the domain of values for an attribute. The key, entity integrity, and referential integrity constraints are specified within the CREATE TABLE statement after all the attributes of the table are declared. The key and various constraints can also be declared later using ALTER TABLE command. For example, the following SQL statements create two tables, Book and Book_Price.

CREATE TABLE Book (ID INT, Title VARCHAR(15), Author VARCHAR(15), Publisher VARCHAR(15),

PRIMARY KEY ID)

CREATE TABLE Book_Price (ID INT, Price INT, Quantity INT,
PRIMARY KEY ID,
FOREIGN KEY(ID) REFERENCES Book(ID))

5.10 Specifying Basic Constraints in SQL

1. **Specifying Attribute Constraints and Attribute Defaults:** We can specify an attribute constraint on an attribute such as NOT NULL, if we do not want NULL values for this attribute. For example, the primary key of any relation can never be NULL value. An attribute ID with INT data type can be set NOT NULL with the following statement in CREATE TABLE command.

   ID INT NOT NULL

   Another constraint can be to restrict the domain of attribute values. For example, if we want to restrict integer values for an attribute Quantity between 50 to 100, then we can declare the attribute Quantity of Book_Price table as follows:

   Quantity INT NOT NULL CHECK (Quantity > 50 AND Quantity<100)

   We can also define a default value for an attribute with DEFAULT<value> clause, if an explicit value is not provided for the attribute. For example, consider we wish to specify a default value of 1 to Price attribute of the Book_Price table when price is not provided during tuple insertion. This can be specify with following statement:

   Price INT NOT NULL DEFAULT 1

2. **Specifying Key and Referential Integrity Constraints:**
   primary key: SQL offers a primary key clause to define one or more attributes as a primary key of a table. For example, consider the student table with the following attributes and SID as a primary key.

   Student (SID INT NOT NULL, S_Name VARCHAR(15) NOT NULL, Gender CHAR, Class INT,
   PRIMARY KEY (SID))

   Referential integrity constraints: It is a property of a data which states that if a value of an attribute references another value of other attribute of either same or different table, then the referenced value must exist and valid. A referential integrity constraint is also specified with FOREIGN KEY clause. For example, we can define SID as foreign key of Marks table that references primary key SID of Student table with following statement:

   Marks (SID INT NOT NULL, Subject VARCHAR(15), Marks FLOAT,
   PRIMARY KEY (SID, Subject),
   FOREIGN KEY(SID) REFERENCES Student(SID)
   ON DELETE SET NULL ON UPDATE CASCADE)

When a tuple is inserted or deleted or a primary key or foreign key value is modified in a table, then the referential integrity constraint is violated. Under this scenario an action must be specified which includes SET NULL, or CASCADE or SET DEFAULT for both ON DELETE and ON UPDATE operations. For example, consider the previous SQL statement. The SET NULL clause sets those values of foreign key to NULL that references the deleted value of the primary key of Student table. We also use the SET DEFAULT clause to set the default value of foreign key if primary key value of referenced tuple in the Student table is updated. ON UPDATE operation. The ON DELETE CASCADE operation deletes all those tuples of the Marks table that references the deleted value of primary key of Student table. The ON UPDATE CASCADE operation changes the values of the foreign key in Book relation with new value of the primary key of Student table.

5.11 Schema Change Statements in SQL

Even after a table is created, we can change the schema of the table i.e. adding and dropping table, attributes, constraints and other schema elements. SQL offers DROP and ALERT commands to perform these operations.

1. **DROP command:** DROP TABLE command is used to remove a table including all its data and constraints. The table will no longer be available in the database and it cannot be accessed. For example, the following SQL statement permanently removes the Student table from the database.

   DROP TABLE Student

2. **ALTER command:** The alter table is used to add, delete and modify columns as well as constraints on an existing table. It can be used to perform following operations:
   a. **Add column:** A general syntax of SQL statement to add a new column in an existing table is:
      ALTER TABLE table_nameADD column_name data type

   b. **Drop column:** A general syntax of a SQL statement to drop an existing column in an existing table is:
      ALTER TABLE table_nameDROP COLUMN column_name

   c. **Modify column:** A general syntax of a SQL statement to alter a column in an existing table is:
      ALTER TABLE table_name ALTER COLUMN column_name data type

   d. **Add and Drop constraints:**
      The general syntax of a SQL statement to add a check constraint to an existing table is:
      ALTER TABLE table_name ADD CONSTRAINT Constraint_name CHECK (CONDITION)

      The general syntax of a SQL statement to drop or remove a constraint from an existing table is:

ALTER TABLE table_name DROP CONSTRAINT Constraint_name;

e. **Add and Drop primary key:** The general syntax of a SQL statement to add a new primary key constraint to an existing table is:
ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...)

The general syntax of a SQL statement to drop a primary key constraint from an existing table is.
ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey;

Check your progress

1. Which SQL command is used to delete a table in SQL?
2. DBMS provides the facility of accessing data from a database through
   a) DDL
   b) DML
   c) DBA
   d) Schema
3. Select operation in SQL is equivalent to which one of the following:
   a) the selection operation in relational algebra
   b) the selection operation in relational algebra, except that select in SQL retains duplicates
   c) the projection operation in relational algebra
   d) the projection operation in relational algebra, except that select in SQL retains duplicates

5.12 View and Trigger in SQL

A view in SQL is a virtual table, whose tuples are computed when needed from a base table. The view does not physically exist while the base table physically exists and its tuples are actually stored in a database. A view is an alternate way of specifying a table, which is accessed frequently during frequent issues of query on the view. For example, consider a SQL query which takes a table that is a natural join of two tables: Student and Marks. Every time when the query executes, it retrieves two base tables Student and Marks from the database and performs natural join of these two tables. Instead, we can use a view that holds the result of join operation of these two tables and act as a single virtual table. A view can be used to store the result of a SQL query and it can be used as a table in another SQL query.

Trigger: A trigger in SQL is a stored procedure which automatically invokes an action when an event such as database update operation occurs. For example, consider a Student table which has attributes age and date_of_birth along with other attributes. We can create a trigger which automatically computes and inserts age whenever a new tuple is inserted.

5.13 Summary

- We understand similarity between SQL and relational algebra and the order in which any SQL query evaluates.

- We learned the set operators and union compatible condition that must be satisfied by set operators before applying on any two relations.

- We explained nested query and correlated nested query and how these queries are evaluated.

- We see that how SQL treats NULL values.

- We illustrated how to use various SQL commands which includes: Schema Change Statements, Specifying Basic Constraints in SQL, Data Definition and Data Types, INSERT, DELETE, and UPDATE.

5.14 Terminal Questions

1. Briefly describe the order in which any SQL query evaluates.
2. What are the conditions that must be satisfied by set operators before applying on any two relations?
3. What do you mean by union compatible?
4. Explain with example how SQL evaluates nested query and correlated nested query.
5. How does SQL treat NULL values?
6. Which of the following is/are true with reference to 'view' in DBMS?
    a) A 'view' is a special stored procedure executed when certain event occurs.
    b) A 'view' is a virtual table, which occurs after executing a pre-compiled query.
7. How SQL checks NULL values of any particular attribute?
8. What do you mean by view and trigger in SQL?
9. Explain with suitable examples how we change schemas of a relation.
10. Describe the various constraints offered by SLQ which we can apply on attributes of a relation?
11. Illustrate how we can apply different constraints offered by SLQon attributes of a relation.

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

UNIT-6 Functional Dependency Theory

Structure

6.1 Introduction

So far, we have seen that a relation scheme contains a number of attributes and a relational database schema consists of many such relational schemes. We discussed how to derive a database schema using the ER data model. The model helps us to identify entity types, relationship types and their attributes in any real world database. It provides us a tool for logical grouping of attributes into their respective relations. However, it still does not provide us any way to measure the quality of a relation schema. We still need a formal way to measure goodness of a relation schema. In this unit, we will discuss theories that have been developed to evaluate qualities of a relational schema. This includes information preservation and minimum redundancy that forms the ultimate goal of a good relation schema. The information preservation means maintaining all ER model concepts including entity types, relationship types and attribute type. Minimum redundancy means minimizing multiple copies of the same data. We will start this unit with illustration of some good and bad relation schemas. After that, we will introduce the concept of functional dependency which forms criteria for decomposition of relation schemes. In the next unit, we will see various types of decomposition in normal formsto reduce the redundancy of data.

6.2 Objectives

After study of this unit you will able to:

- Understand functional dependencies and how to determine these functional dependencies from a given relational instance.
- Explain attribute closure and how we can use the attribute closure to find candidate keys of a relation from its functional dependencies.
- Describe functional dependency set closure of any relation with given functional dependency set and how to minimize any functional dependency set.
- Find when two functional Dependency Sets are said to be equal.

6.3 Functional Dependency

In the previous section, we informally discussed various issues and their solutions in database design. Now, we introduce a tool called Functional Dependency (FD) to formally discuss all the above database design issues. This forms a basis for Normal Forms of relation schemas.

Let us consider a relation schema R = {A1, A2, ... , An}. Assume X and Y are two subsets of attributes of relation schema R. A functional dependency from X to Y represented by X--->Y exists, if and only if for any two tuples t1 and t2:
If t1.X=t2.X, then t1.Y=t2.Y

This means the values of attributes set X of a tuple t1 uniquely determine the values of attributes set Y. Or, the values of attributes set Y of a tuple is uniquely determined by the values of attributes set X. The functional dependency is usually abbreviated as FD. The set of attributes X is called the left side of the FD and the set of attributes Y is called the right side of the FD.

X	Y
a1	b1
a1	b1

Figure 6.1 – XY1 table.

In the above relation XY1 shown in figure 6.1, a functional dependency X---->Y exists because when values of attribute X for tuple t1 and t2 are equal, this implies values of attribute Y for tuples t1 and t2 are also equal.

X	Y
a1	b1
a1	b2

Figure 6.2 – XY2 table.

In the above relation XY2 shown in figure 6.2, a functional dependency X---->Y does not exist because the values of attribute X for tuples t1 and t2 are equal but values of attribute Y for tuples t1 and t2 are not equal.

W	X	Y	Z
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Figure 6.3 – WXYZ table.

Now consider the relation shown in Figure 6.3. Here, a functional dependency {W, X} → Y exists because the attributes values corresponding to these attributes do not violate the FD constraint. The FD Y → X also exists because if t2.Y=t3.Y, then t2.X=t3.X and the attributes values for other tuples also do not violate the FD constraint. But the FD W→ X does not exist because t1.W=t2.W but t1.X ≠ t2.X. Similarly, the FD Z → Y also does not exist because here t3.Z=t4.Z but t3.Y ≠t4.Y.

For convenience, an attribute set is simply represented with all attributes concatenated together with removed parentheses and commas. Now, consider the STUDENT table shown in Figure 6.4.

Student_ID	Student_Name	Course_ID	Course_Name	Faculty_ID	F_Salary
S01	Sharad	CS01	DBMS	F03	80,000
S02	Denesh	CS04	Algorithm	F02	60,000
S03	Vikash	CS01	DBMS	F03	80,000
S04	Saukat	CS04	Algorithm	F02	60,000
S05	Surender	CS04	Algorithm	F02	60,000

Figure 6.4 - A STUDENT table.

We can verify that the following functional dependencies:

1. Student_ID----> Student_Name (This FD holds because the values of Student_ID attribute are unique)
2. Course_ID, Course_Name---->Faculty_ID(This FD also holds because if Course_ID and Course_Name are same for any two tuples in the table, the values of corresponding Faculty_ID are also same)
3. Faculty_ID ---->F_Salary(This FD also holds because any two tuples with same Faculty_ID will also have same values of F_Salary)

6.4 Trivial Functional Dependencyand Properties of Functional Dependency

Trivial Functional Dependency:Consider two attribute sets P and Q over a relation R. A functional dependency P--->Q is said to be trivial FD if Q is a subset of P (P⊇ Q). For example consider a relation R(ABC) with following FDs are trivial:

A---->A (A is already subset of A)
AB--->A (A is a subset of AB)
AB--->B (B is a subset of AB)

But, the following FDs are not trivial:
A--->B (B is not subset of A)
AB--->C (C is not subset of AB)

A--->BC (BC is not subset of A)

Inference Rules for Functional Dependencies: Given a relation schema R with attribute sets x,y and z. The following properties of functional dependency are called Armstrong's axioms which hold true for any relation schema.

1. **Reflexive:** If x ⊇ y then x--->y
2. **Transitivity:** If x--->y and y--->z then x--->z
3. **Augmentation:** If x--->y then xz--->yz
4. **Union:** If x--->y and x--->z then x--->yz
5. **Decomposition:** If x--->yz then x--->y and x--->z

6.5 Attribute Closure (X+)

Attribute closure of a set X of attributes of a relation R with FD set F is a set of all attributes that are functionally determined from X using the functional dependencies present in F. The attribute closure can be determined by following procedure:

1. $X^+ = X$
2. repeat
   oldX$^+$:= X$^+$;
   For each functional dependency Y→Z in F do
   if Y ⊆ X$^+$ then X$^+$= X$^+$ ∪ Z
   until (X$^+$ = = oldX$^+$);

The above procedure starts with first assigning all attributes of X to X^+. Next, we use the inference rules of FD and add attribute to X^+ recursively using each functional dependency in F. We repeat this process unless no new attribute can be added to X^+.For example, consider a relation schema R(ABCD) with functional dependency set {A--->B, B--->C, C--->D}. The attribute closure of A and BC can be obtained as follows:

$(A)^+$={A}
={A,B}(using functional dependencyA--->B)
={A,B,C} (using functional dependencyB--->C)
={A,B,C,D}(using functional dependencyC--->D)

$(BC)^+$={B,C}
={B,C,D} (using functional dependencyC--->D)

Super keys: Let us consider a relational schema R with an attribute set X. If the attribute closure of X (i.e. X^+) determines all attributes of the relation R, then X is super key for the relation R.

For example, if R(ABCD) is a relation schema with functional dependencies {A--->B, B--->CD}, then AB is super key of relation R(ABCD) because (AB)+={A,B,C,D}.

Candidate Key: A candidate Key is a minimum number of attributes whose closure gives all the attributes of a relation. In other words, if a closure of any proper subset of a super key does not give all attributes of the relation then the super key becomes the candidate key for the relation. So, a candidate key is the minimal super key and all candidate keys are super keys but not all super keys are candidate keys.

For example, consider a relational schema R(ABCD)with functional dependencies {AB--->C, C--->D, B--->EA}. Let us find the candidature key of R. Since, attribute B is not present on the right side of any functional dependencies. Start with B and find its attribute closure.

(B)+={B,E,A,C,D}.

Because, attribute closure of B gives all attributes of the relation, it is a candidate key.Since the attribute B cannot be obtained from any functional dependency, it is the only candidate key of the relation.

Illustrative question: Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A–>B, BC–>D, E–>C, D–>A}. What are the candidate keys of R?

Solution: Since attributes E and H are not present in the right hand side of any functional dependency, every candidate key must contain E and H. Start with EH and find their attribute closure.

$(EH)^+$= {EHC}

So it is not a candidate key. Add left side attributes of any functional dependency and then find their attributes closure.

$(AEH)^+$= {B,C,D,A,E,H}

$(A)^+$= {A,B}

So AEH is the candidate key.

Attribute A can be obtained from D–>A.

$(DEH)^+$= {D,A,B,E,H,C}

So DEH is also a candidate key.Attribute D can be obtained from BC–>D

$(BCEH)^+$={BCDAEH}

But attribute C can be obtained from E–>C. So, BEH is also a candidate key.

Check your progress

1. Consider the following table.

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the above table?
(a) XY -> Z and Z -> Y
(b) YZ -> X and Y -> Z
(c) YZ -> X and X -> Z
(d) XZ -> Y and Y -> X

2. Consider the relation X(P, Q, R, S, T, U) with functional dependenciesF = {{P, R} → {S,T}, {P, S, U} → {Q, R}}. Which of the following is the trivial functional dependency in F+ is closure of F?
   (A) {P,R}→{S,T}
   (B) {P,R}→{R,T}
   (C) {P,S}→{S}
   (D) {P,S,U}→{Q}

3. Consider a relation R(ABCDEF) with the following functional dependency set {C->F, E->A, EC->D, A->B}. Which of the following are false?
   a) (CD) + = {C,D,F}
   b) (EC) +={E,C,F,A,D,B}
   c) (AE) += {A,B,E}
   d) (AC) += {A,E}

6.6 Functional Dependency Set Closure

A closure of any functional dependency set is a set of all functional dependencies logically implied from the original functional dependency set. Given a relation R with functional dependency set F, a closure of F represented by F^+ can be determined by following procedure:

1. Initialize the functional dependency set closure with empty $F^+=\{\}$.
2. For each subset $X \subseteq R$.
   - Find attribute closure X^+
   - For each $Y \subseteq X^+$
     - Add a functional dependency X –> Y to F^+

To illustrate the above procedure, consider a FD set {A–>B, C–>B} for a relation R(ABC). Closure of the given FD set can be found by finding closure of all possible subsets of attributes of the relations.

1. Initialize an empty functional dependency closure set F^+.

2. $\Phi^+ = \Phi$
   It is also counted as a functional dependency. Add these functional dependencies to F^+.

3. $A^+ = AB$
   This attribute closure gives 2 attributes, so it gives total $2^{(\text{number of attributes})} = 4$ functional dependencies which are as follows:
   A–>Φ

A–>A
A–>B
A–>AB
Add these functional dependencies to F^+.

4. B+ = B
   Now, this closure also contains 1 attributes which gives total $2^{(\text{number of attributes})}$=2 functional dependencies:
   B–>Φ
   B–>B

   Add these functional dependencies to F^+.

5. C+= CB
   The atribute closure C^+ contains 2 attributes, it gives total $2^{(\text{number of attributes})}$ =4 functional dependencies which are as follows:
   C–>Φ
   C–>C
   C–>B
   C–>BC
   Add these functional dependencies to F^+.

5. AB+= AB
   This attribute closure gives 2 attributes. It gives total $2^{(\text{number of attributes})}$ =4 functional dependencies which are as follows:
   AB–>Φ
   AB–>A
   AB–>B
   AB–>AB
   Add these functional dependencies to F^+.

6. BC+= BC
   This attribute closure gives 2 attributes. It gives total $2^{(\text{number of attributes})}$=4 functional dependencies which are as follows:
   BC–>Φ
   BC–>B
   BC–>C
   BC–>BC
   Add these functional dependencies to F^+.

7. AC+= ACB
   This attribute closure gives 3 attributes. It gives total $2^{(\text{number of attributes})}$=8 functional dependencies which are as follows:
   AC–>Φ
   AC–>A
   AC–>B
   AC–>C
   AC–>AC
   AC–>BC
   AC–>AB
   AC–>ABC
   Add these functional dependencies to F^+.

8. ABC+= ABC
This attribute closure also gives 3 attributes. It gives total $2^{(\text{number of attributes})}=8$ functional dependencies which are as follows:
ABC–>Φ
ABC–>A
ABC–>B
ABC–>C
ABC–>AC
ABC–>BC
ABC–>AB
ABC–>ABC

So, the total number of functional dependencies logically implied from the given FD set is 1+4+2+4+4+4+8+8=35.

Membership Test:

6.7 Equality Between two Functional Dependency Sets

Any two functional dependency sets F and G are said to be equal if closure of FD set F (i.e. F+) is equal to the closure of FD set G (i.e. G^+). Alternately, we can also say the two FD set F and G are equal if

1. F covers G: This means all FDs of G are logically implied from the FD set F.
2. G covers F: This means all FDs of F are logically implied from the FD set G.

For example, consider two FD set F and G as follows:
F={A–>B, AB–>C, D–>ACE}
G={A–>BC,D–>AE}
We can check whether the two FD sets are equal or not by applying following steps:

1. Check whether F covers G: Take left side attributes of each FD of G and find its attribute closure using FDs of F.
$(A)^+$={A,B,C}
Since, the attribute closure A^+ contains BC attributes, this implies that the functional dependency A–>BC can be derived from existing FD set of F.

(D)+={D,A,C,E}
Since, the attribute closure D^+ contains attributes AE, this means the functional dependency D–>AE can also be derived from existing FD set of F.
So, all FDs of G are logically implied from FD set F.

2. Check whether G covers F: Take left side attributes of each FD of F and find its attribute closure using FDs of G.
(A)+={A,B,C}
So, the functional dependency A–>B, can be obtained from existing FDs of G.

(AB)+={A,B,C}
So, the functional dependency AB–>C, can be obtained from existing FDs of G.

(D)+={D,A,E,B,C}
So, the functional dependency D–>ACE, can be obtained from existing FDs of G.

Also, all FDs of F are logically implied from FD set G.

Therefore two FDs F and G are equal.

Check your progress

4. Consider following functional dependencies are given:
   AB->CD, AF->D, DE->F, C->G , F->E, G->A
   Which one of the following options is false?

(a)CF+ = {ACDEFG} (b)BG+ = {ABCDG}
(c)AF+ = {ACDEFG} (d)AB+ = {ABCDFG}

5. Consider the relation scheme R=(E,F,G,H,I,J,K,L,M,N) and the set of functional dependencies{EF→G, F→IJ, EH→KL, K→M, L→N}on R. What is the key for R?
   (a) {EF}
   (b) {EFH}
   (c) {EHKL}
   (d) {E}

6. Find closure of a functional dependency set F={A–>B, B–>A} of relation R(AB).

6.8 Minimization of a Functional Dependency Set

Minimal Functional Dependency Set:A set of functional dependencies F is said to be minimal if it satisfies following conditions:

1. Each FD in F contains a single attribute on its right hand side.
2. Any proper subset of X in FD X--->Y does not determine Y.
3. We cannot remove any FD from F which is equivalent to the original FD set F.

A minimal set of functional dependencies set F can be determined by following steps:

1. Replace each FD X → {A1, A2, ..., An} in F by the n functional dependencies X →A1, X →A2, ..., X → An.
2. Remove extraneous or redundant attributes from the left hand side of each FD. This can be found by finding attribute closure of each attribute of LHS.
3. Remove the redundant functional dependency X→Y if any from the functional dependency set obtained from step 2.

For example, assume that after applying step 1, we get the following FD: {A → C, AB → C, C → D, C → E, CD → E}. Find attribute closure of each attribute on the LHS.
(i) A+ = ACDE
(ii) B+ = B
(iii) C+ = CDE
(iv) D+ = D
(v) E+ = E

From (i), the attribute closure of A gives the attribute C. So, the FD AB → C contains redundant attribute B and it can be removed.

Similarly, from (iii), the attribute closure of C contains attribute E. This means the FD CD → E contains D as a redundant attribute and so D can be removed from it.

Now, no more redundant attributes can be found in any FD. So the final result after applying step 2 is as follows:
{A → C, A → C, C → D, C → E}

Finally, remove redundant functional dependency if any present. For example, after applying step 2, the FD set contains A → C as redundant FD which can be removed safely.

Explanatory Question: Find the minimal FD set of F:
F= {A ---> BC, B--->C, A --->B, AB --->C}

Solution: Minimal set of FD can be found by following steps:

1. Rewrite the FD set F such that the Right Hand Side (RHS) of each FD should contain a single attribute.
   {A ---> B, A ---> C, B--->C, A --->B, AB --->C}

2. Remove redundant attributes: Find attribute closure of each attribute of the LHS of FD set F.
   (i) A+ = ABC
   (ii) B+ = BC
   (iii) C+ = C
   From (i), the attribute closure of A gives attribute C. This means B is a redundant attribute in AB→ C, and so B can be removed. Now, no more redundant attributes can be found. So the final result after applying step 2 is as follows:
   {A ---> B, A ---> C, B--->C, A --->B, A--->C}

3. Remove redundant functional dependency if any. The FD set we get after applying step 2 contains A ---> B two times and A--->C also two times. So these FDs can be removed safely without loss of any FD in the given FD set. Now the FD set becomes {A ---> B, A ---> C, B--->C}. By transitive property of FD, A ---> C is logically implied from A ---> B and B--->C. So the final minimal FD set contains :{A ---> B, B--->C}.

Check your progress

1. Find the minimal cover of following functional dependency set:
   {A → B, ABCD → E, EF → GH, ACDF → EG}

2. What is the minimal cover of following functional dependency set F?
   F = {A →BC, B →CE, A →E}

6.9 Properties of Decomposition

We decompose a relation R into a set of sub relation {R1,R2,R3...Rn} if the relation R is not in appropriate normal form. When a relation R is decomposed into a set of sub

relations {R1,R2,R3...Rn}, then the decomposition should be lossless as well as functional dependency preserving decomposition.

1. Lossless Join Decomposition: Let a relational schema R is decomposed into R1,R2,R3...Rn. In general the natural join of among the decomposed relations R1 ⋈ R2 ⋈ R3 … Rn is superset of R i.e. R1 ⋈ R2 ⋈ R3...Rn ⊇ R. The decomposition is said to be lossless if the natural join among decomposed relations is exactly equal to original relation R i.e R1 ⋈ R2 ⋈ R3…..Rn = R. This means a lossless decomposition does not contain any false tuples after performing the natural join operation among decomposed relations. The false tuples represent erroneous information. If a decomposition is not lossless decomposition, then it is called lossy decomposition.

- Decomposition is lossy if R1 ⋈ R2...Rn ⊃ R
- Decomposition is lossless if R1 ⋈ R2...Rn = R

For example, consider a relation R(ABC) is decomposed into relations R1(AB) and R2(BC) as given below in figure 6.5 :

A	B	C
2	5	4
1	5	3
4	3	2

a) R(ABC)

A	B
2	5
1	5
4	3

b) R1(AB)

B	C
5	4
5	3
3	2

c) R2(BC)

A	B	C
2	5	4
2	5	3
4	3	2
1	5	4
1	5	3
4	3	2

d) Result of R1 ⋈ R2

Figure 6.5-The decomposition of R(ABC) into R1(AB),R2(BC) is lossy because R1 ⋈ R2 ⊃ R.

We can clearly see that the Result of R1 ⋈ R2 ⊃ R, which mean the result, contains false tuples. So it is a lossy decomposition.
Consider the same relation R, but now it is decomposed into two relation R1(AB) and R2(AC) as shown below in Figure 6.6.

A	B
2	5
1	5
4	3

a) R1(AB)

A	C
2	4
1	3
4	2

b) R2(AC)

A	B	C
2	5	4
1	5	3

4	3	2

c) Result of R1 ⋈ R2

Figure 6.6-The decomposition of R(ABC) into R1(AB),R2(BC) is lossless because R1 ⋈ R2 = R.

Ifwe are given functional dependencies of a relation. We can use the functional dependencies of the relation to show when a decomposition is lossless. Let a relation R is decomposed into two relations R1 and R2. The decomposition is said to be lossless if it satisfy following conditions:

1. Attributes of R1 U Attributes of R2 = Attributes of R
2. Attributes of R1 ∩ Attributes of R2 ≠ Φ
3. Common attributes of R1 and R2 must be a super key of at least one relation (R1 or R2).

For example, consider the decomposition of R(ABC) into R1(AB) and R2(AC) as shown in figure 6.6. We can clearly see that the common attribute A is super key of both relations R1 and R2. So it is a lossless decomposition.

Illustrative question: Identify whether the following decomposition with given functional dependencies are lossy or lossless.

a. R(ABCD) with FD={AB--->C, C--->A, C--->D}
   Decomposition={R1(AB), R2(ACD)}

b. R(ABCD) with FD={A--->B, B--->C, C--->D}
   Decomposition={R1(AB),R2(BCD)}

Solution:

a. The decomposition will be lossless if
   1. AB U ACD = ABCD, which is true.
   2. AB ∩ ACD ≠ Φ, which is true.
   3. The common attribute between R1(AB), R2(ACD) is A. A is not the super key of either R1 or R2 because

      The attribute closure of A does not gives all attributes of either R1 and R2 :
      A+ = A
      So, the decomposition is lossy.

b. The decomposition will be lossless if
   1. AB U BCD = ABCD, which is true.
   2. AB ∩ BCD ≠ Φ, which is true.
   3. The common attribute between R1(AB), R2(BCD) is B which is the super key of R2. This is because the attribute closure of B gives all attributes of R2 table :
      B+ = BCD

      So, the decomposition is lossless.

2. Dependency Preserving Decomposition: Let a relational schema R with functional dependency set F is decomposed into R1,R2,R3...Rn with functional dependency set F1,F2,F3...Fn respectively. In general the union of all functional dependency set F1,F2,F3...Fn is subset of F i.e F1 ∪ F2 ∪ F3...∪ Fn⊆ F. If a union of all functional dependency set F1,F2,F3...Fn is proper subset of F i.e F1 ∪ F2 ∪ F3...∪ Fn⊂ F, then the decomposition is dependency not preserving decomposition. But if, the union of all functional dependencies set F1,F2,F3...Fn is equal to F i.e F1 ∪ F2 ∪ F3...∪ Fn= F, then the decomposition is dependency preserving decomposition.

- Dependency preserving decomposition if F1 ∪ F2 ∪ F3...∪ Fn= F
- Dependency not preserving decomposition if F1 ∪ F2 ∪ F3...∪ Fn⊂ F

Illustrative question: Let R (A, B, C, D) be a relational schema with the following functional dependencies: {A → B, B → C, C → D and D → B}. The relation R is decomposed into
R1(AB), R2(BC), R3(BD). Determine whether it is dependency preserving or dependency not preserving decomposition.

Solution: Let F1, F2 and F3 after functional dependency set of R1(AB), R2(BC), R3(BD).
Find functional dependency set F1 by finding attribute closure of each combination of attributes of R1(AB).
A+=ABCD
B+= BCD
AB+= ABCD
F1={A→B} (only consider the attributes which are part of R1)

Now, find functional dependency set F2 from the attribute closure of each combination of attributes of R2(BC).
B+=BCD
C+=CDB
BC+= BCD
So F2= {B→C,C→B} (only consider the attributes which are part of R1).

Finally find functional dependency set F3 from the attribute closure of each combination of attributes of R3(BD).
B+=BCD
D+=DBC
BD+=BCD
SO, F3= {B→D, D→B} which are the part of R3(BD).

In the functional dependency set of R(ABCD), A → B is present in F1, B → C presents in F2, D → B presents in F3, and C → D can be obtained indirectly from C→B of F2 and B→D of F3. So this decomposition is functional dependency preserving decomposition.

Check your progress

1. Let R(X,Y,Z,W) be a relational schema with the following functional dependencies:
   {X→Y,Y→Z,Z→WX→Y,Y→Z,Z→W and W→YW→Y}

The decomposition of R into (X,Y),(Y,Z),(Y,W) is

a) Lossy join, but is dependency preserving
b) Lossless join, but is not dependency preserving
c) Lossless join and dependency preserving
d) Lossy join and not dependency preserving

2. Given R(A,B,C,D,E)with the FD Set F(A->B, A->C, DE->C, DE->B, C->D). Consider this decomposition : R1(A,B,C), R2(B,C,D,E) and R3(A,E). Then, which of the following is true for this decompositions:
   a) Lossy join, but is dependency preserving
   b) Lossless join, but is not dependency preserving
   c) Lossless join and dependency preserving
   d) Lossy join and not dependency preserving

6.10 Summary

In summary:

- We learned Functional Dependencies and how to determine these functional dependencies from the given relational instance.
- We understand attribute closure and how we can use the attribute closure to find candidate keys of a relation from its functional dependencies.
- We illustrated to find functional dependency closure of any relational instance.
- We described equivalence of two functional Dependency Sets and learned to find when two functional Dependency Sets are said to be equal.
- We learned to find minimization of a functional dependency set of a relation.

6.11 Terminal Questions

1. Explain the two properties that the decomposed relations should satisfy.
2. How do you determine whether the decomposed relations satisfy lossless and dependency preserving decomposition or not?
3. What are the differences between attribute closure and functional dependency closure?
4. When do any two functional Dependency Sets are said to be equal?
5. Find the minimal functional dependency set of {PQ-->R, PR-->Q, Q--->S,QR--->P, PQ--->T}.
6. Explain with suitable example how you find candidate keys of a relation from its functional dependencies.
7. What do you mean by Functional Dependencies of a relation?

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

UNIT-7 Normalization

Structure

7.1 Introduction

In this unit, we will discuss other database design concepts that are used extensively in commercial database design. The very first step in a database design involves designing of an ER model which is later mapped to relational model. In the subsequent step, functional dependencies are identified and then primary keys are determined from these functional dependencies. Later, the undesirable functional dependencies are removed through the normalization process. In the previous unit, we discussed concepts of functional dependency, lossless join and dependency preserving decomposition. Now based on these concepts, we will discuss normal forms such as 2NF, 3NF, and BCNF and use them to achieve desired decompositions. A good database design should satisfy two properties: lossless join and dependency preserving decomposition. Further, we will discuss the concept of multivalued dependency (MVD) and explain fourth and fifth normal form.

7.2 Objectives

After studying this unit, you will able to:

- Explain various types of anomalies, which area rising due to the redundancy of data.
- Understand normalization process and apply various normal forms on any relational schema.
- Decompose a relational schema which is not in third normal form to a set of decomposed relations in third normal form such that each decomposed relation satisfies both lossless and dependency preserving decomposition.
- Perform decomposition of a relational schema which is not in BCNF into a set of decomposed relations where each one is in forth normal forms.
- Explain multi valued functional dependency and forth normal form along with join dependency and fifth normal form.

7.3 Problems Caused by Redundancy of Data

Redundancy is multiple copies of the same data present in a database table. For example, consider the STUDENT table shown in Figure 7.1. The attributes Course_ID, Course_Name, Faculty_ID and Faculty_Salary contain multiple entries of cs-01, Java, F-01 and 70,000. This redundancy of data causes basically three problems:

Student_ID	Student_Name	Course_ID	Course_Name	Faculty_ID	Faculty_Salary
001	Sharad	cs-01	Java	F-01	70,000
002	Dinesh	cs-01	Java	F-01	70,000
003	Anil	cs-02	Python	F-02	70,000
004	Nikhil	cs-01	Java	F-01	70,000

Figure7.1- A sample STUDENT table of a student database.

1. **Update Anomaly:** When a database table contains multiple copies of the same data, an update of the data requires change at multiple places or all copies of the data. This causes an update operation costly and results in update anomaly. For example, consider the STUDENT table shown in Figure7.1.

Student_ID	Student_Name	Course_ID	Course_Name	Faculty_ID	Faculty_Salary
001	Sharad	cs-01	Java	F-01	80,000
002	Dinesh	cs-01	Java	F-01	80,000
003	Anil	cs-02	Python	F-02	70,000
004	Nikhil	cs-01	Java	F-01	80,000

Figure 7.2- Result of updation in salary of F-01 from Rs 70,000 to 80,000 in STUDENT table.

If we want to change the value of one of the attributes, say salary of F-01 from Rs 70,000 to 80,000, then we must update all records of students who are taught by F-01. In other words, we must update the salary of F-01 in all places (as shown in above Figure7.2). Otherwise, if we make changes in some places, then it causes different salaries of the same faculty. Updation of redundant copies of the same data makes the update operation too costly.

2. **Insertion Anomaly:** Redundancy of data causes insertion of some unexisted dummy data when a new data is inserted. This causes insertion anomaly. For example, consider again the STUDENT table shown in Figure7.1.

Student_ID	Student_Name	Course_ID	Course_Name	Faculty_ID	Faculty_Salary
001	Sharad	cs-01	Java	F-01	70,000
002	Dinesh	cs-01	Java	F-01	70,000
003	Anil	cs-02	Python	F-02	70,000
004	Nikhil	cs-01	Java	F-01	70,000
NULL	NULL	cs-03	C	F-03	55,000

Figure 7.3- Result of adding a new faculty F-03 in STUDENT table.

If we want to add a new faculty F-03 who is not yet teaching, we need to insert some dummy value like NULL for other attributes as shown in Figure 7.3. This forces the NULL values for Student_ID attribute also, but it cannot be NULL because it is a primary key.

3. **Deletion Anomaly:**In delete anomaly, deletion of some data causes deletion of other useful data also. For example, inSTUDENT table shown in Figure7.1.

Student_ID	Student_Name	Course_ID	Course_Name	Faculty_ID	Faculty_Salary
001	Sharad	cs-01	Java	F-01	70,000
002	Dinesh	cs-01	Java	F-01	70,000
003	Anil	cs-02	Python	F-02	70,000
004	Nikhil	cs-01	Java	F-01	70,000

Figure 7.4- Result of deleting a student with Student_Name Anil in STUDENT table.

If we delete the record of Anil, this also forces the deletion of faculty information of F-02. So, the deletion of one record results in the deletion of other useful information also.

Decomposition of a relation schema: We saw that the three anomalies cause difficulties in maintaining consistency of the data during updation, insertion and deletion of some data. During updation and insertion of new data, some redundant work needs to be done. Sometimes, the deletion of a data causes accidental loss of other data. There is also wastage of storage space due to dummy data like NULLs. The various problems arise due to these three anomalies can be overcome by splitting the original table into two or more tables. For example, consider the STUDENT table shown in Figure7.1 is decomposed into two separate tables as given in Figure7.5. The first table StudentDetail consists of attributes: Student_ID, Student_Name and Course_ID. The Other table Course_Detail consists of attributes: Course_ID, Course_Name, Faculty_ID and Faculty_Salary.

Student_ID	Student_Name	Course_ID
001	Sharad	cs-01
002	Dinesh	cs-01
003	Anil	cs-02
004	Nikhil	cs-01

a) StudentDetail table

Course_ID	Course_Name	Faculty_ID	Faculty_Salary
cs-01	Java	F-01	70,000
cs-02	Python	F-02	70,000

b) Course Detail table

Figure7.5 - Decomposition of a student table into two tables.

We can easily examine that:

1. An update will take place at only one place i.e. in a single tuple. For example if we consider the decomposed tables, a change in salary of a faculty requires updating only one record.But, in case of STUDENT table, this requires changes in several records.

2. Any insertion of a data does not insert an unexisted dummy data. For example, in the decomposed tables, insertion of a new faculty does not require the addition of dummy data for attributes Student_ID and Student_Name.But, in case of STUDENT table, this requires addition of NULL values corresponding to other attributes.

3. Deletion of any data does not cause removal of any other useful information. For example in decomposed tables, if we delete the record of Anil, it does not remove faculty information of F-02.

7.4 Normalization

Normalization is a process of structuring a relation based on normal forms to achieve following properties:

1. Minimizing redundancy of data.

2. Minimizing the insertion, deletion and update anomalies.

A normal form is a condition which indicates the degree to which a relation is normalized. The relation that does not satisfy the condition of normal form is decomposed into smaller relations. The main goals of the normalization process are:

1. 0% redundancy
2. Lossless Join Decomposition
3. Dependency Preserving Decomposition

There are six normal forms: 1NF, 2NF, 3NF, BCNF, 4NF and 5NF.

First Normal Form(1NF): A relation is in first Normal form if it contains only single valued attributes. If a relation contains any multi valued attribute, then the relation is not in first normal form. For example, consider the STUDENT1 table with a primary key S_ID shown in Figure7.6. It contains a multivalued attribute C_Name which contains multiple values for C_Name in each tuple. So this table is not in the first normal form.

S_ID	**S_Name**	**C_Name**
S01	Sharad	Algorithm, DBMS
S02	Denesh	Algorithm, Java
S03	Vikash	DBMS, Java

Figure7.6 - STUDENT1 table.

We can convert **STUDENT1** table to first normal form in any one of the three ways:

1. If we remove the multivalued attribute C_Name from STUDENT1 table and place it in a new table along with the primary key of the STUDENT1 table. Now, both STUDENT2 and STUDENT3 tables are in first normal form which is shown in Figure7.7.

S_ID	**S_Name**
S01	Sharad
S02	Denesh
S03	Vikash

a. STUDENT2 table.

S_ID	**C_Name**
S01	Algorithm
S01	DBMS
S02	Algorithm

S02	Java
S03	DBMS
S03	Java

b. STUDENT3 table.

Figure7.7- STUDENT2 and STUDENT3 tables.

2. If we expand each value of the attribute C_Name as a separate tuple, the table becomes first normal form. The restructured table STUDENT4 is shown in Figure 7.8 where {S_ID, C_Name} serves as its primary key. Since it contains all single valued attributes, it is in first formal form.

S_ID	S_Name	C_Name
S01	Sharad	Algorithm
S01	Sharad	DBMS
S02	Denesh	Algorithm
S02	Denesh	Java
S03	Vikash	DBMS
S03	Vikash	DBMS, Java

Figure7.8 - STUDENT4 table.

3. If we know the maximum number of courses any student can opt, let us say 2, then we can replace the **C_Name attribute** with two attributes: C_Name1 and C_Name2. The resulting table STUDENT5 shown in Figure 7.9 is in first formal form.

S_ID	S_Name	C_Name1	C_Name2
S01	Sharad	Algorithm, DBMS	DBMS
S02	Denesh	Algorithm	Java
S03	Vikash	DBMS	Java

Figure7.9 - STUDENT5 table.

In the above three solutions, the first is the best solution because it does not suffer from redundancy. The second solution suffers from redundancy of data.

7.5 Second Normal Form(2NF)

A relation R is in second normal form if it satisfies two conditions:

1. It is in first normal form

2. It does not contain any partial functional dependency. The partial functional dependency means if any proper subset of a candidate key functional determines any non prime attribute (attribute which are not a part of the candidate key) it is called partial functional dependency.

For example, consider a relation R(ABCD) with a candidate key {BC}. The attributes B and C are prime attributes while A and D are non-prime attributes. The functional dependencies B--->D and C--->AD are partial functional dependencies.

A relational schema that is not in second normal form can be decomposed into a set of second normal form relations as follows:

1. Create a new relation for each partial functional dependency which contains all the participating attributes of the partial functional dependency.

2. There should be a relation with the original primary key and attributes which are fully functionally dependent on the original primary key.

In a functional dependency BC---> D, if neither B--> D nor C-->D holds true, then the attribute D is fully functionally dependant on the candidate key BC.The decomposition by the above algorithm is both lossless-join and dependency-preserving.

Illustrative questions: Consider a relation R(ABCDE) with functional dependencies {AB--->C, C--->D, B--->E}. Is this relation is in second normal form. If not then decompose it into second normal form.

Solution: First we need to find candidate keys of relation R(ABCDE). Since attributes AB do not contain on the right side of any functional dependency, every candidate key must contain AB. So, start with AB and find attribute closure of it.
AB+= {ABCDE}
A^+= {A}
B^+= {BE}
Closure of attributes AB gives all attributes of relation R while closure of its subset A and B does not gives all attributes of the relation. So, AB is candidate key of the relation. Since, attributes A and B cannot be obtained from any functional dependencies, AB is the only candidate key of the relation.

The functional dependency B--->E is a partial functional dependency because B is a proper subset of candidate key and E is non-prime attribute (E is not a part of candidate key). So, the relation is not in second normal form.

The relation R can be decomposed into a set of relations such that each of them is in second normal form as follows:

1. Since, B--->E is partial functional dependency, create a new relation R1(BE) with attributes B and E.
2. The other relation R2(ABCD) should contain original candidate key AB and other attributes CD that are fully functionally dependent on it. The attribute E

is not fully functionally dependant on AB because the functional dependency B-->E holds true, so E is not included in R2.

The resulting R1(BE) and R2(ABCD) are in second normal form.

Check your progress

1. Consider a relation R(ABCD) with functional dependency F:{ AD → B, AB → C }. Is the functional dependency AB → C partial or total dependency?

2. Given the following relational schemes for a library database:
   Book (Title, Author, Catalog_no, Publisher, Year, Price)
   The following functional dependencies hold in above relation:
   a) Title Author -> Catalog_no
   b) Catalog_no -> Title, Author, Publisher,Year
   c) Publisher, Title,Year -> Price
   If {Author, Title} is the candidate key for the above relational scheme. Is this relational schema in second normal form?

3. Consider arelational schema R(A,B,C,D,E,P,G) with following FDs: {AB->CD, DE->P, C->E, P->C, B->G}. Whether relation schema R is in second normal form or not?

7.6 Third Normal Form(3NF)

A relational schema R with any non trivial functional dependency X--->Y is in third normal form if either of the following conditions holds:

a) X is a super key of R
   OR
b) Y is a prime attribute of R.

The redundancy of data on third normal form is less than second normal form. A relation which is not in third normal form can be converted into third normal form by decomposition of the original relation and set up a new relation for each functional dependency that violates third normal form condition. The other relation contains candidate keys of original relation along with candidate keys of each decomposed relations. The resulting decomposed relations are in 3NF that satisfy bothlossless-join and dependency-preserving decomposition.

Illustrative Question: Consider a relation R(ABCDE) with functional dependencies A--->BCDE, BC--->ADE and D--->E. Check whether it is in third normal form or not. If not, decompose it into third normal form.

Solution: In order to check if the relation is in third normal form, first we need to find its candidate keys. Since, all attributes of the relation are covered by the right side of all functional dependencies. Take left side attributes of any functional dependency and find its attribute closure.

(A)+= {BCDEA}

Since, closure of attribute A gives all attributes of the relation, so it is a candidate key. The attribute A can also be obtained from functional dependency BC--->ADE. Check whether BC is also the candidate key.

(BC)+= {BCADE}
(B)+= {B}
(C)+= {C}

So, BC is also a candidate key of the relation.

The candidate keys of the relation are {A} and {BC}.The functional dependency D → E violates the third normal form because neither D is a super key nor E is a prime attribute.

We can convert it to third normal form by decomposing the relation R(ABCDE) and set up a new relation R(DE) containing participating attributes of functional dependency D--->E. The other relation contains candidate key of original relation R and the candidate key of decomposed relation R(DE). Now decomposed relations R(ABCD) and R(DE) are in third normal form.

7.7 BOYCE-CODD NORMAL FORM (BCNF)

A relational schema R is in BCNF if for each functional dependency X-->Y that holds in R, X should be the super key of the relation.

BCNF normal form is stricter than third normal form. Every BCNF relation is also in third normal form but every third normal form is not in BCNF. For example, consider a relation TEACHER(Student, Course, Faculty) with functional dependencies {Student, Course}--->Faculty, Faculty--->Course and a candidate key {Student, Course}. The functional dependency Faculty--->Course violates BCNF condition. However, it satisfies the third normal form condition because Course is a prime attribute.

A relational schema R with a set of functional dependencies F which is not in BCNF can be decomposed into a set of BCNF relations as follows:

1. Set D = {R}
2. For each relational schema Q in D that is not in BCNF do
    - find a functional dependency X → Y in Q which violates BCNF;
    - replace Q in D by two relation schemas R1(Q - Y) and R2(X,Y);

The resulting decomposition from the above algorithm islossless-join but may or may not be dependency-preserving decomposition. If a lossless-join and dependency preserving decomposition are not possible with BCNF, we may consider decomposition with 3NF.

Illustrative Question: Consider a relation R = (ABCDE) with functional dependencies A--->B, BC--->D}. Is this relation in the BCNF?

Solution: First we need to find the candidate keys of the relation. Attributes ACE are not present on the right side of any functional dependency. Start with the attribute closure of ACE.

$(ACE)^+$= {ACEBD}

The attribute closure $(ACE)^+$gives all attributes of the relation, so ACE is a candidate key of the relation. Since, we do not get any attribute of the candidate key from the functional dependencies of R. So ACE is the only candidate key of the relation.
We know that all candidate keys are super keys. The functional dependencies A--->B and BC--->D violate BCNF condition because Left side attributes of both FDs are not the super keys.

We can decompose R = (ABCDE) into a set of BCNF relations as follows:

1. R(ABCDE)
2. Take the functional dependencyA--->B which violates the BCNF condition and decompose R into R1 and R2 by removing attribute B from R.

   R1(ACDE) and R2(AB)

3. Now consider the functional dependencyBC--->D that violates the BCNF condition. Itcan also be written as AC--->D because B can be obtained from A--->B. Now, decompose R1 into R11 and R12by removing attribute B from R as follows:

   R11(ACE), R12(ACD) and R2(AB)

4. Stop the further decomposition because all relations R11(ACE), R12(ACD) and R2(AB) are in BCNF.

A binary relation which contains only two attributes is always in BCNF. This is because there are only four scenarios possible:

1. R(AB) with functional dependency A--->B and candidate key A. So it is in BCNF.
2. R(AB) with functional dependency B--->A and candidate key B. So it is in BCNF.
3. R(AB) with functional dependency A--->B,R(AB) with functional dependencies A--->B and B--->A with candidate key A and B. So it is in BCNF.
4. R(AB) with no functional dependency with candidate key AB. So it is in BCNF.

Check your progress

1. Consider the relation schema Student_Performance (name, courseNo, rollNo, grade) has the following FDs:
   - name,courseNo->grade
   - rollNo,courseNo->grade
   - name->rollNo
   - rollNo->name

   What is the highest normal form of this relation scheme?

2. Let the set of functional dependencies F = {QR → S, R → P, S → Q} hold on a relation schema X = (PQRS). X is not in BCNF. Suppose X is decomposed

into two schemas and Z where Y = (PR) and Z = (QRS). Consider the two statements given below:
S1. Both Y and Z are in BCNF
S2. Decomposition of X into Y and Z is dependency preserving and a lossless.
Explain whether the above two statements are true or false?

3. Given the following two statements:
S1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.
S2: The FD set {AB->C, D->E, E->C} is a minimal cover of the set of functional dependencies {AB->C, D->E, AB->E, E->C}.
Explain whether the above two statements are true or false?

7.8 Multivalued Dependencies and Fourth Normal Form

In the previous section, we learned that in a functional dependency A -> B, each value of attribute A corresponds to only a single value of B. In other words, in a functional dependency A->>B, each value of A determines exactly one value of B. But, if for each value of A there exists multiple values of B then B is multivalued facts about A and it is represented as A ->> B. A multivalued dependency exists if there exists at least two multivalued facts A ->> B and A ->> C on the same attribute A within the same table R and attributes B and C are independent of each other. So, a multivalued dependency requires at least three attributes such that two attributes are dependent on a third attribute. For example consider a relation STUDENT (S_Name, Phone, C_name) as shown in Figure 7.10:

S_Name	Hobbies	C_name
Sandeep	Chess	C++
Sandeep	Badminton	Python
Nikhil	Table tennis	Java
Krisna	Badminton	C++
Krishna	Table tennis	Python

Figure7.10- A STUDENT (S_Name, Phone, C_name) table.

The above table contains two multivalued facts: S_Name->> Hobbies and S_Name->>C_name. They are read as S_Name multi determines Hobbies and S_Name multidetermines C_name. So, there exists multivalued dependencies: S_Name->> Hobbies and S_Name->>C_name in STUDENT table.

Fourth normal form(4NF): A relational schema R with a set of functional dependencies and multivalued functional dependencies is in fourth normal form, if each non trivial multivalued dependency X->>Y contains X as super key of the relation R. A multivalued dependency $X \rightarrow\rightarrow Y$ in R is called a trivial multivalued dependency if $Y \subseteq X$, or $X \cup Y = R$. Otherwise it is non trivial functional dependency.
A relation which is not in fourth normal form can be converted to forth normal form by creating a new relation for each non trivial multivalued dependency that violates

4NF condition. For example, the STUDENT table shown in Figure 4.14 is not in fourth normal form because it contains non trivial functional dependencies: S_Name->> Hobbies and S_Name->>C_name and S_Name is not a super key. It can be converted to fourth normal form by decomposing it into two tables STUDENT_HOBBIES(S_Name, Hobbies) and STUDENT_COURSE(S_Name,C_name). Now, the non trivial functional dependencies S_Name->> Hobbies and S_Name->>C_name become trivial in their respective tables.

7.9 Join Dependencies and Fifth Normal Form

We have seen the process of repeated decomposition during the normalization process to achieve 1NF, 2NF, 3NF and BCNF relations. Every 4NF relation also satisfies all the properties of BCNF. The relationship among these normal forms is shown in Figure 7.11. These relations always follow the property of lossless join decomposition. We can also achieve 4NF relation by removing each non trivial multivalued dependency which violates 4NF condition by repeated binary decomposition. However, sometimes it is not possible to obtain a lossless join decomposition of R by splitting into two relational schemas, but we can achieve by decomposition of R into more than two relational schemas.

Join Dependency: A join dependency represented by JD (R1,R2,R3...Rn) of a relational schema R exists if there exists lossless join decomposition of R into R1,R2,R3...Rn such that natural join among these relations equals to original relation R that is (R1⋈R2⋈R3...Rn)=R.

Fifth Normal Form(5NF): A relational schema R is in Fifth Normal with respect to a set of functional dependencies including multivalued dependencies and join dependency JD(R1, R2, ... Rn), if

1. JD(R1, R2, ... Rn) is non trivial join dependency. A join dependency JD(R1, R2, ... Rn) is said to be trivial if any relational schema Ri is equal to R.

2. Each Ri of JD(R1, R2, ... Rn) is a super key of R.

Most commercial applications use normal forms up to BCNF. The fifth normal is rarely used in practice because it is difficult to identify join dependency. If we consider all the groups of relations in first, second, third, BCNF, fourth and fifth normal form are represented by sets. Then, the relationship among various normal forms can be analyzed by following diagram shown in Figure 7.11.

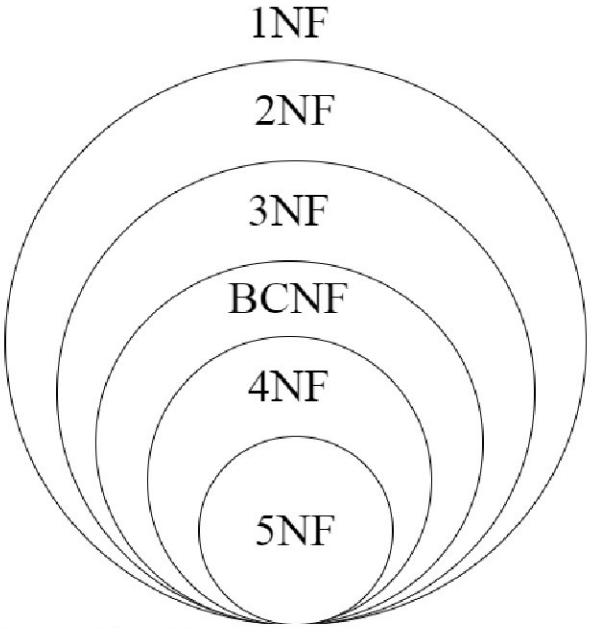


Figure7.11 - Relationship among various normal forms.

7.10 Summary

- We had seen various types of anomalies arise due to the redundancy of data.
- We explained normalization process and how it helps to get rid of redundancy of data through various normal forms.
- We learned first and second normal form and how to determine a relational schema in first and second normal forms or not.
- We understand about third normal form and learned to decompose a relational schema which is not in third normal form to a set of decomposed relations in third normal form which are both lossless and dependency preserving decomposition.
- We discussed BCNF normal form which is stricter than 3NF and performed decomposition of a relational schema which is not in BCNF into a set of decomposed relations all in forth normal forms. We seen that the decomposed relations are lossless but may or may not be dependency preserving decomposition.
- We explained multivalued functional dependency and forth normal form along with join dependency and fifth normal form.

7.11 Terminal Questions

1. What do you mean by redundancy of data? Explain various types of anomalies arise due to the redundancy of data.
2. What do you mean by transitive functional dependency?
3. When schema is said to be in second normal form?
4. Explain the third normal form with suitable example.
5. Is it possible to decompose a relational schema which is not in third normal form to a set of decomposed relations in third normal form? If possible, then is this decomposition is both lossless and dependency preserving decomposition?
6. What is the condition for a relational schema to be in BCNF?

7. Describe whether the decomposition of a relational schema which is not in BCNF in to a set of decomposed relations all in forth normal forms is possible which are both lossless and dependency preserving decomposition.
8. Explain multivalued functional dependency with suitable examples.
9. What do you mean by trivial multivalued functional dependency?
10. When a relation is said to be in fourth normal form?
11. Describe join dependency and fifth normal form.

BIBLIOGRAPHY

1. **R Elmasri, S Navathe, Fundamentals of Database Systems, 6th edition, Addison-Wesley, 2010.**
2. **R Ramakrishnan, J Gehrke, Database Management Systems, 3rd Ed., McGraw-Hill, 2002.**
3. **A Silberschatz, H Korth and S Sudarshan, Database System Concepts, 6th Ed., McGraw-Hill, 2010.**

Uttar Pradesh Rajarshi Tondon
open University

Master of Computer Science

MCS - 109

Database Management System

Block

3

Transaction Management & Emerging Databases

Course Design Committee

Prof. Ashutosh Gupta — **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Prayagraj

Prof. Suneeta Agarwal Member
Department of CSE
MNNIT, Prayagraj

Dr. Upendra Nath Tripathi Member
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

Dr. Ashish Khare Member
Associate Professor, Department of Computer Science
University of , Prayagraj

Ms. Marisha — **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Mr. Manoj Kumar Balwant — **Member**
Assistant Professor (Computer Science)
School of Science, UPRTOU Prayagraj

Course Preparation Committee

Dr. Parth Gautam — **Author(Block 3 : Unit 8, 9)**
Assistant Professor, Department of Computer Science
Dev Sanskriti Vishwavidyalya, Haridwar, Uttrakhand

Dr. Abhay Sexena — **Editor**
Professor and Head, Department of Computer Science
Dev Sanskriti Vishwavidyalya, Haridwar, Uttrakhand

Prof. Ashutosh Gupta
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU Prayagraj

Mr. Manoj Kumar Balwant — **Course Coordinator**
Assistant Professor (computer science)
School of Sciences, UPRTOU, Prayagraj

UPRTOU, Prayagraj-2021
ISBN –

BLOCK INTRODUCTION

This block deals with Transaction Management and Emerging DBMS. Here firstly, we are going to study about various transaction management strategies and mechanism. In the second phase, we will learn about emerging databases that are required to meet the needs of the emerging software applications and to manage varied data.

In the next unit of Transaction management, the concept of transaction and its properties along with states are discussed. We will understand, how concurrent transactions execute in the database by using the concept of serializability and other concurrency control techniques. Along with this we are going to cover recovery management mechanism which helps in attaining atomicity and durability of the database.

Further, in the unit of Emerging DBMS, we are going to understand the basic concept of some emerging databases. Here, our emphasis will be on discussing the design rules, architecture and user needs of these databases. This unit will also cover the applications and advantages of each of the databases

Unit- 8 Transaction Processing Concepts

Structure

8.1 Introduction to Transaction Processing

A user might see several operations on a database as a single unit. For Example, a very basic example of transferring funds from one account to another account looks like a single operation but from the view point of database it consists of several operations.

So, the collections of operations that form a single logical unit of work are called as Transactions. It is required here that every transaction that initiates must ensure its proper execution, not leaving the system in an inconsistent state. It would be unacceptable if one account is debited and another one is not credited.

Here, we will discuss all the basic concepts of transaction processing, its properties, state, managing concurrent transaction processing and recovery strategies.

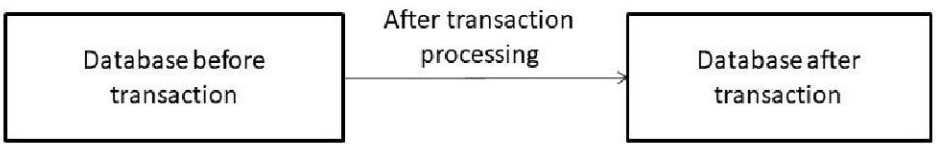
8.2 Objectives

After studying this unit, students can understand the concept of transaction its properties and states. We also discuss how concurrent transactions execute in the database by using the concept of serializability and other concurrency control techniques. This unit is also going to cover recovery management mechanism which helps in attaining atomicity and durability of the database.

8.3 Transaction and System Concept

A transaction is a sequence of read and writes operations on data items that logically functions as one unit of work. It provides and —"all-or-noting" proposition stating that each work –unitperformed in database must either complete in its entirety or have no effect whatsoever. There are some points to consider related with transaction processing:

- All actions of transactions will take place or none of all.
- If it succeeds, the effects of write operations persist (commit); if it fails, no effects of write operations persist (abort).
- It ensures consistent state despite of concurrent activity in the system, and despite failures that may occur.



A transaction is initiated by a user program wrapped in high level data manipulation language or programming language like SQL. Here, a transaction consists of all operations between the begin transaction and end transaction.

8.3.1 Transaction States

The transaction must be in one of the following states:-

1. **Active:-** This is a initial state, the transaction stays in this state while it is executing
2. **Partially committed:** The transaction is in this state when it has executed the final statement.
3. **Failed**: A transaction is in this state once the normal execution of the transaction cannot proceed.

4. **Aborted**: A transaction is said to be aborted when the transaction has rolled back and the database is being restored to the consistent state prior to the start of the transaction.
5. **Committed:** a transaction is in this committed state once it has been successfully executed and the database is transformed in to a new consistent

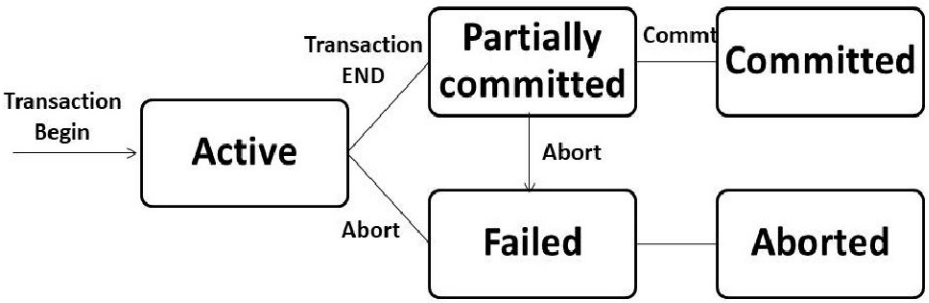


Figure 6.1 Transaction States

8.4 Desirable Properties of Transactions

To maintain integrity of data, the database system must follow four key properties of transactions.

8.4.1 Atomicity: It says "All or None ". This means either all operations of the transactions are reflected properly in the database, or none are.

8.4.2 Consistency: it ensures correctness of the database. This is achieved by execution of transaction in isolation i.e. No other transaction running concurrently. This preserves the consistency of the database.

8.4.3 Isolation: it indicates actions performed by a transaction are hidden or isolated from outside the transaction until it terminates. Suppose if two transactionsTi and Tj are executing, then in this case it appears to Ti that either Tj finished its execution before Ti started or Tj started execution after Tifinished.In this way, each transaction is unaware of the other transactions executing concurrently in the system

8.4.4 Durability: After successful completion of transaction, the changes that are made by all transaction should persist in the database even if the system goes down or crashes. All updates that done by transaction must become permanent on disk after commit action of transaction.

These properties are called as ACID properties. For better understanding of these properties, consider a simple banking system which have several accounts and allow set of transaction that access and make updates to these accounts. These transactions access the data using two operations:

- Read(X), it transfers data item X from the database to a local buffer of the transaction that call the read operation.
- Write(X), it transfers the data item X from the local buffer of the transaction that call the write operation to the database.

Let's assume Ti be a transaction that transfers 500 Rs from Account A to Account B. At begin of transaction balance in account A= 2000/- and B=3000/-. these transactions can be defined as:

```
Ti= read(A);
A:= A-500;
Write(A);
Read(B);
B:=B+500;
Write(B);
```

Now, we will understand ACID properties by taking an example of banking system.Suppose, during the execution of transaction Ti, any type of failure occurs that prevents the transaction Ti to complete successfully. It may also possible, that the write(A) operation executes but write(B) operation does not occur due to some system failure, in that case account A is debited but the account B is not credited.

A=1500;
B=3000;
Sum(A+B)=4500;

Here, the sum of A+B is not same as when the transaction Ti begins. This type of situations can lead the database in an inconsistent state.

So, these types of inconsistencies are deal by the property of **Atomicity** which ensures all operations of the transaction are reflected in the database or none are.

As **consistency** property of the database says, if the database is in consistent state before execution of transaction, then or termination, the database will also be in a consistent state.

Sum(A,B) == sum(A,B)
Before transaction begins after transaction terminates

So, to ensure consistency of the database, integrity constraints are applied which prevents database from an inconsistent state.

In the same example we discussed above of transferring funds from account A to B, we find the data base in an inconsistent state and in the same situation any other transaction running concurrently try to read the data of account A and B, it will find the incorrect data and may again leave the system in an inconsistent state.

So, avoid this type of problems, **Isolation** property of the transaction allows concurrently executing transactions is to execute serially- that is one after another.

As in the above example, a system failure takes place after the amount is deducted from the account A and not credited in account B. In this case loss of data takes place in main memory but data on disk is still present which guarantees durability of transaction.

Durability ensures the all the updates carried out on the database will persist even if there is a system failure. These updates are made before the execution of the transaction or after the successful completion of the database.

Check your Progress

1. Describe the Transaction States.
2. Explain the Properties of Transactions.

8.5 Scheduling and Recoverability

As transactions consists of set of operations and these operations make updates in database. These operations are executed in some order to make the transaction complete and to leave the system in a consistent state.

So, when multiple transactions are running concurrently then there is a need to maintain some sequence in which these transactions will execute. This process of sequencing transaction is termed as Scheduling of transaction.

8.5.1 Scheduling: when several transactions such as (T1, T2,……Tn) are running concurrently then the order of execution of each operation is known as Schedule. This can be understood by taking an example:

Suppose, there are two transactions T1 & T2 which are running concurrently and each transaction has own set of read and write operations on the database, then in this case, schedules determines the exact order of operations that are going to be performed on the database.

T1	T2
R(A)	
R(B)	

	R(A)
	R(B)

In the above table it is shown that operations of T1 are followed by T2.

However, this order of execution of operations of transactions may change. So, to deal with this, there are various types of serial and non-serial schedules.

8.5.1.1 Serial Schedule: Serial Schedule are those where for each pair of transaction T1 and T2, if T2 followed by T1, then T2 can only initiates its operation after the completion of T1.The given schedule is a serial schedule:

T1	T2
Read(A)	
A:A+500;	
Write(A)	
	Read(A)
	A:A+500;
	Write(A)

8.5.1.2 Non-Serial Schedule: In this type of schedules, interleaving of operations takes place. Here, the order of operations of the transaction can interleave the order of operations of other transaction and that result leaves the database in consistent state.

T1	T2
Read(A)	
A:A+500;	
	Read(A)
	A:A+500;
Write(A)	
	Write(A)

8.5.2 Recoverability

As in scheduling of transactions we have come across the order of execution of operations of transaction by maintaining the consistency of the database, assuming that there will be no transaction failure.

Now, if in any case, a transaction may not complete its execution due to any hardware failure, system crash or any software issue. So, in that case, system has to roll back the failed transaction. Here, it may also possible that any other transaction is dependent on failed transaction or used the data written by failed transaction type of schedules that are followed by the database system. So, there are schedules that are then it is also required to rollback those transactions too. To ensure this, we have to determine the applied from the view point of recovery from transaction failure. These schedules are discussed below:

8.5.2.1 Recoverable schedule is one where, for each pair of transaction Tin ad Tj, such that Tj reads a data item that was previously written by Ti, the the commit operation of Ti appears before the commit operation of Tj.

In other words, if a transaction Tj reads a value that is updated by Ti, then the commit of Tj must occur after the commit of Ti.

Ti	Tj
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Commit	
	Commit

Suppose, if due to some reason Ti fails before it commit, then in this situation Tj can be aborted and data can be recovered. Hence given schedule is recoverable.

8.5.2.2 Cascade less schedule are those where for each pair of transactions Ti and Tj ,such that Tj reads a data item previously written by Ti, then the commit operation of Ti appears before the read operation of Tj.

Ti	Tj

Read(A)	
Write(A)	
Commit	
	Read(A)
	Write(A)
	Commit

These types of schedules are recoverable and eliminate the drawback of cascading rollback.

In cascading rollback, due to failure of single transaction, a series of transaction that are dependent on failed transaction rolled back that leads to unnecessary work head.

Check your Progress

1. Explain Scheduling and its types.
2. What is Recoverability?

8.6 Serializability of Scheduling

In the previous topic of scheduling, we have seen there are serial and non-serial schedules. Serial schedules are those in which another transaction starts only after the completion of first transaction, it means it does not allow concurrent execution of transactions while the non-serial schedule, multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. So, serial schedules are always serializable but non serial schedules are needed to be checked for Serializability.

8.6.1 Serializability is a process of testing the schedules whether they are serializable or not. A Serializable schedule is a schedule which leaves the database system in a consistent state.

8.6.2 Types of Serializability

There are two types of Serializability of scheduling.

1. Conflict Serializability
2. View Serializability

8.6.2.1 Conflict Serializability: tells us whether a non-serial schedule is conflict serializable or not. Conflict serializable schedule is a schedule which can be converted to serial schedule after swapping its non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

Let's consider a schedule:-

T1	T2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

After swapping non-conflicting operations we get a serial schedule.

T1	T2
	R(A)
	R(B)
	W(B)
R(A)	
R(B)	
W(A)	

So, the above given schedule is conflict Serializable.

6.6.2.2 View Serializabilitytells us whether a non-serial schedule is view serializable or not.A schedule is said to be View serializable schedule if it is equivalent to its View Equivalent.Let's consider a non-serial schedule and its Serial schedule: -

T1	T2
R(X)	
W(X)	
	R(X)
	W(X)
R(Y)	
W(Y)	
	R(Y)
	W(Y)
T1	T2

R(X)	
W(X)	
R(Y)	
W(Y)	
	R(X)
	W(X)
	R(Y)
	W(Y)

Non Serial		Serial	
S1		S2	
T1	T2	T1	T2
R(X)		R(X)	
W(X)		W(X)	
	R(X)	R(Y)	
	W(X)	W(Y)	
R(Y)			R(X)
W(Y)			W(X)
	R(Y)		R(Y)
	W(Y)		W(Y)

S2 is the serial schedule of S1

The non-serial(S1) and its serial schedule(S2) are view equivalent, if they satisfy all the following conditions:

- Initial Read: Initial read of each data item in transactions must match in both schedules.
- Final Write: Final write operations on each data item must match in both the schedules.

- Update Read: If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item.

So, in the above given example two schedules S1 and S2 are view equivalent. Hence, the schedule S1 is view serializable schedule.

8.7 Transaction Support in SQL

As discussed earlier in the previous sections that transaction executes set of operations as single logical unit. Each of the transaction begins with a specific operation and ends when all the tasks in the given set of transaction complete successfully.

These set of operations performed against a database wrapped in Transaction SQL statements. If all the operations are executed successfully then the transaction is complete and then it will be committed and updates the database permanently. But, if any of the operation fails then the entire transaction will fail and complete transaction will be cancelled or rolled back.

So, to manage and control the transaction in the database following SQl commands are used.

- **BEGIN TRANSACTION**
- **SET TRANSACTION**
- **COMMIT**
- **ROLLBACK**
- **SAVEPOINT**

These transaction control commands are only used with the DML Commands such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

8.7.1 BEGIN TRANSACTION: It tells the starting point of a transaction.

Syntax:BEGIN TRANSACTION transaction_name;
Example: BEGIN TRANSACTION T1;

8.7.2 SET TRANSACTION Command: It is used to specify the read and write characteristics for the transaction.

Syntax:SET TRANSACTION [READ WRITE | READ ONLY];
Example:SETTRANSACTION T1rw;

8.7.3 COMMIT Command: It is used to save all the changes made by a transaction to the database. Let's understand it with the help of an example:

ID	NAME	AGE	CITY	SALARY
101	Akhilesh	25	Lucknow	15000.00
102	Roshan	22	Agra	10000.00
103	Ramesh	23	Varanasi	20000.00
104	Himanshu	21	Kanpur	25000.00
105	Ajay	25	Prayagraj	25000.00

Syntax:COMMIT;

Example:

```
SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;
SQL> COMMIT;
```

Output:

ID	NAME	AGE	CITY	SALARY
102	Roshan	22	Agra	10000.00
103	Ramesh	23	Varanasi	20000.00
104	Himanshu	21	Kanpur	25000.00

8.7.4 ROLLBACK Command: This command is used to undo the changes made by a transaction that are not committed or not permanently saved to the database.

Syntax:ROLLBACK;

Example:

```
SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;
SQL>ROLLBACK;
```

Output:

ID	NAME	AGE	CITY	SALARY
101	Akhilesh	25	Lucknow	15000.00
102	Roshan	22	Agra	10000.00
103	Ramesh	23	Varanasi	20000.00
104	Himanshu	21	Kanpur	25000.00
105	Ajay	25	Prayagraj	25000.00

8.7.5 SAVEPOINT Command: This command allows a transaction to rollback a certain operation in a transaction without rolling back the entire transaction. Here, it is required to create save points for all operation of the transaction. The ROLLBACK command is used to undo a mentioned save point

Syntax: SAVEPOINT *savepoint_name;*

ROLLBACK TO *savepoint_name;*

Example:

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=101;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL>DELETE FROM CUSTOMERS WHERE ID=102;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=103;
1 row deleted.
SQL> ROLLBACK TO SP2;
Rollback complete.
```

Output:

SQL> SELECT * FROM CUSTOMERS;

ID	NAME	AGE	CITY	SALARY
102	Roshan	22	Agra	10000.00
103	Ramesh	23	Varanasi	20000.00
104	Himanshu	21	Kanpur	25000.00
105	Ajay	25	Prayagraj	25000.00

4 rows selected.

Check your Progress

1. Explain Serializability and its type.
2. Describe SET TRANSACTION Command with example.
3. Explain Transaction Support in SQL.

8.8 Concurrency Control Techniques

As we have seen when database system allows concurrent execution of transaction then it is difficult to maintain Isolation property of transaction. However, to ensure that interleaving of transaction will not leave the system in inconsistent state, system follow some mechanism called as concurrency control techniques.

Concurrency control is a process of managing concurrent operations without conflicting each other and to ensure integrity of the database.

8.9 Concurrency Techniques for Concurrency Control

As in above discussed examples we have seen Read and Write operations on databases are applies in an interleaving manner. In this case, there are possibilities of some conflicts such as WW Conflicts, RW Conflicts and so on.

So, to overcome these types of issues, database system has some concurrency control techniques. These are:

- Lock-Based Protocols
- Timestamp-Based Protocols
- Validation-Based Protocols

8.9.1 Concurrency Control Based on Lock-Based Protocols

In this protocol, a transaction cannot read or write until it acquires an appropriate lock on the data item. In this way, while one transaction is accessing a data item, no other transaction can modify the same data item.

Here, lock is data variable which is related with data item that tells any operation is performing on the data item. A request for the lock is made to the concurrency control manager. Once the lock is granted, then only the transaction proceeds its execution. There are two types of lock:

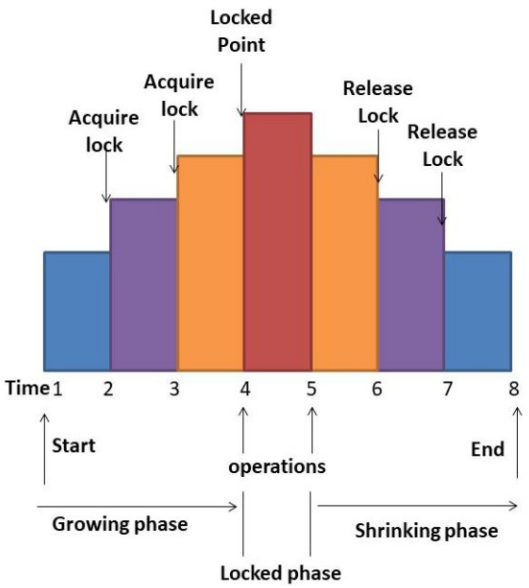
Shared lock: It is a read only lock. It can be shared among transaction as in this data can only read but cannot be modified by the transaction.

Exclusive lock: In this transaction can read as well as write a data item. Here, multiple transactions cannot modify the same data at a time. There are following types of lock-based protocols:

1. **Simplistic Lock Protocol:** In this lock is granted to a transaction before beginning of transaction and it is released after completing the transaction.
2. **Pre-claiming Lock Protocol:** This protocol evaluates all the data items on which they need locks. If all the required locks are granted then the

transaction begins otherwise it rolls back till all required locks are not granted. After completion of the transaction it releases all the issued locks.

3. **Two-phase locking (2PL):** This protocol works in three steps-
    - Firstly,when the transaction begins to execute, it requires permission for the locks it needs.
    - In second phase, all locks are granted to a transaction. The third phase is started as soon as the transaction releases its first lock.
    - In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



This protocol as termed as 2PL as it involves two phases:

Growing Phase: In this phase transaction may obtain locks but may not release any locks.

Shrinking Phase: In this phase, a transaction may release locks but cannot obtain any new lock

8.10 Concurrency Control Based on Timestamp Based Protocol

As Lock based protocol solved the problem of arising conflict among transaction of first lock at execution time. But sometimes this leads to incompatible database.

There is another method for determining the serializability order among transaction called as Timestamp Ordering Protocol.

Timestamp Ordering Protocol serializes the execution of concurrent transactions based on their Timestamps.

8.10.1 Timestamps

As with the creation of transaction Ti in the database system, a unique fixed timestamp TS(Ti) is attached to it. This timestamp is assigned in two ways:

1. Using the value of the system clock as a timestamp.
2. Using logical counter that is incremented after a new timestamp has been assigned.

Here, transactions are executed on the basis of the ascending order of the transaction creation. A transaction with older timestamp will execute first.

Let's assume, there are two transactions Ti and Tj, Where Ti has entered the system at time 0010 and Tj at 0020. Here, TS(Ti)< TS(Tj), so the transaction Ti will execute first.

Timestamp Values of data item Q can be of two types:

1. R_TS(Q) it denotes the largest timestamp of any transaction that executed Read (Q) successfully.
2. W_TS(Q) it denotes the largest timestamp of any transaction that executed Write (Q) successfully.

8.10.2 Timestamp Ordering Protocol ensures that any conflicting read/ write operations of a transaction are executed in timestamp order.

Condition1: Whenever a transaction Ti issues a Read (Q) operation and:

If W_TS(Q) >TS(Ti) then the operation is rejected.

If W_TS(Q) <= TS(Ti) then the operation is executed.

Timestamps of all the data items are updated.

Condition 2: when a transaction Ti issues a Write(Q) operation and :

If TS(Ti) < R_TS(X) then the operation is rejected.

If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

8.11 Validation Based Protocol

This protocol is known by optimistic concurrency control technique. This technique minimizes transaction conflicts. Here, transactions are allowed to execute without restrictions until it is committed. It involves three phases:

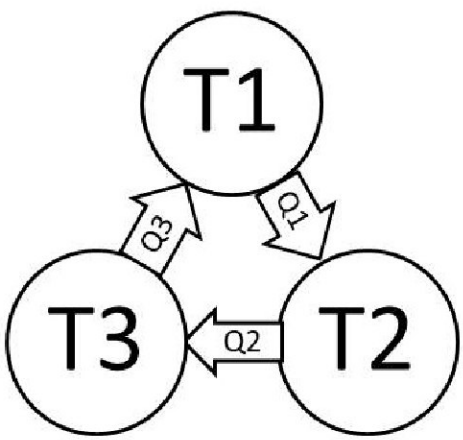
1. **Read Phase:** during this phase, a transaction can read the data item from the database but the update data item is stored in a local buffer, not in the actual database.
2. **Validation Phase:** here, transaction is validated to ensure that the changes made will not affect the integrity and consistency of database. If in the validation phase, database is consistent then a transaction goes to write phase otherwise transaction is rolled back.
3. **Write Phase**: as in this phase, all updates are reflected in the database permanently after successful validation.

Check your Progress

1. What is Concurrency Control?
2. Explain Timestamp Ordering Protocol.
3. Discuss Validation Based Protocol with its phases.

8.12 Deadlock Handling

Deadlock is a unwanted situation of a database, when n number of transactions are running and each transaction is waiting for another transaction for releasing a locked data item. Example Suppose, there are T1, T2,T3 transaction exists and these transactions are in a situation such that T1 is waiting for a data item (Q1) that is hold by T2, T2 is waiting for a data item(Q2) that is hold by T3 and T3 is waiting for a data item(Q3) that T1 holds.



In figure none of the transaction can complete its execution and leaving the system in a deadlock state.So, to recover from this situation system follows some deadlock handling mechanism:

- Deadlock Prevention
- Deadlock Detection

- Deadlock Recovery

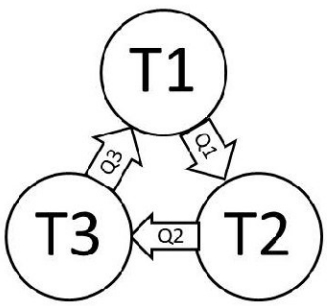
8.12.1 Deadlock Prevention

This helps in prevention of deadlock situation by avoiding condition of mutual exclusion, circular wait, no preemption for a data item. Prevention For deadlock prevention, system uses timestamp-based prevention mechanism is used. Here, the system uses timestamps to decide whether a transaction should wait or roll back. There are mainly two deadlock prevention schemes using timestamps are:

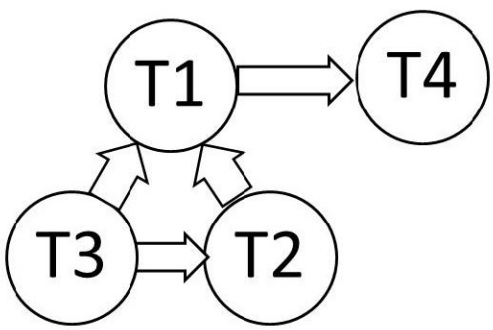
1. **Wait-Die scheme**: If T1 is older than T2, T1 is allowed to wait. Otherwise, if T1 is younger than T2, T1 is aborted and later restarted.
2. **Wound wait scheme:If T1 is older than T2, T2 is aborted and later restarted. Otherwise, if T1 is younger than T2, T1 is allowed to wait.**

8.12.2 Deadlock Detection

As a deadlock in database system occurs when a transaction waits for a indefinite time to obtain a lock. To detect a deadlock situation, a wait -for graph is used.If a wait-for graph contains a circular wait loop, then each transaction involved in the graph is said to be in deadlocked.Let's understand a situation by taking an example: There are three transactions T1, T2 and T3. T1 is waiting for T2; T2 is waiting for T3 to release the lock. Here a circular waiting loop exists in the wait for graph.



Hence, to resist the system from deadlock situation, system keeps on checking for the circular wait loop in the graph. If there is no cycle in the graph, the system is not in a deadlock state.



8.12.3 Deadlock Recovery

After detection of a deadlock in a system, recovery mechanism is applied to recover a database from a deadlock state. Most commonly transactions are rolled back to break a deadlock. There are three actions which are taken for deadlock recovery:

- **Selection of a Victim:** From a set of transaction which occurs deadlock, it is a process to select those transactions which are to be rolled back.
- **Rollback:** It is to determine how far a particular transaction is rolled back. A transaction can be either rolled back partial to a specific point or can be rolled back completely.
- **Starvation:** When a victim transaction never completes its execution, then it is termed as a starvation.

Check your Progress

1. What is Deadlock?
2. Describe the concept of **Deadlock Handling.**
3. Explain Deadlock Recovery.

8.13 Database Recovery Techniques Based On Immediate Update

A database system can face failure due to various reasons such as disk crash, power outage, software error, catastrophe and so on. But in all these situations, database systems should robust enough to recover the lost data to maintain the atomicity and durability of the transactions.

8.13.1 Database recovery on immediate update

Database recovery techniques are applied before the commit point or the transaction. In an immediate update scheme, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible.If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back. This recovery scheme follows two recovery procedures:

- undo(Ti): it restores the value of the data items updated by transaction Ti to its old values.
- redo(Ti): it sets the value of the data items updated by transaction Ti to its new values.

This technique is known as undo/redo algorithm.

8.14 Failure Classification

A failure of the database system can be of following types:

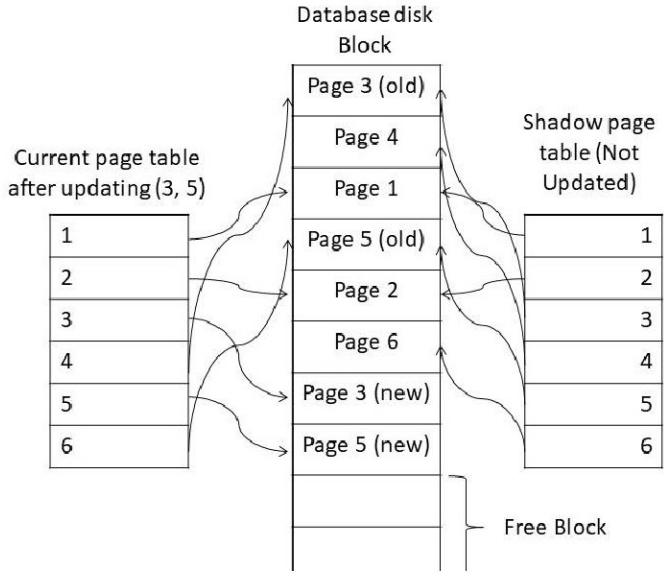
- **Transaction Failure:** A transaction can be failed or aborted due to two types of errors:
- **Logical errors:** errors in the logic of a program resist a transaction from execution.
- **System Errors:** undesirable state such as deadlock, resist a transaction to execute normally.
- **System Crash:** this failure can be raised due to hardware malfunction, failure of a database software or operating system corruption that results into loss of data of the volatile storage and lefts the transaction processing to a halt.
- **Disk Failure:** it involves formation of bad sectors, head crash, unreachability of the disk or data transfer loss.

8.15 Shadow Paging

Shadow paging is a database recovery technique. Here, database is considered as made up of fixed size of logical units of storage which are as *pages*. Pages are mapped into physical blocks of storage, with help of the page table which allow one entry for each logical page of database. This method uses two-page tables named **current page table** and **shadow page table**.

- **Current page table** - used to point to most recent database pages on disk. Entries present in current page table may be changed during execution.
- **Shadow page table** - Shadow page table is used when the transaction starts which is copying current page table. It resides on disk, and is never modifies during execution.

After successful execution of transactions, both tables become identical.The concept of shadow Paging is illustrated with the help of given example:



Here, two write operations are performed of Page 3 and 5. Before execution of write operation, current page table is pointing to old page on disk. When write operation of Page 3 starts, following steps takes place:

1. Firstly, it searches for available free block on disk.
2. After finding a free block, it copies page 3 to free block which is represented by Page 3 (New).
3. Now current page table points to Page 3 (New) on disk but shadow page table points to old page 3 because it is not modified.
4. The changes are now propagated to Page 3 (New) which is pointed by current page table.

To commit transaction following steps should be followed:

1. Flush all modified pages in main memory to physical database.
2. Output current page table to disk.
3. Make the current page table the new shadow page table. For this, keep a pointer to the shadow page table at a fixed (known) location on disk. And update the pointer to point to current page table on disk.

Once pointer to shadow page table has been written, transaction is committed.In case of failure during execution of transaction before committing, it is only needed to free modified database pages and discard current page table. The modified pages are available through the shadow page table.If the transaction is performed successfully, the entries of the shadow page table are discarded and the current page table is again copied to the shadow page table.

8.16 Log Based Recovery

As it is the most common recovery mechanism. Here, it maintains a structure for recording database updates termed as log. This log maintains a sequence of log

records, which is it a file that contains all read, write, and commit activities of transaction in the database.A log record holds following fields:

- **Transaction identifier (Ti):** Unique Identifier of the transaction that performed the write operation.
- **Data item(X):** Unique identifier of the data item written, a location of the data item on the disk.
- **Old value(V1):** Value of data item prior to write.
- **New value(V2):** Value of data item after write operation.

There are log records to record significant events during transaction processing such as start, commit or abort of a transaction. We denote these types of records as:

- **<Ti start>:** when a transaction Ti starts.
- **<Ti commit>:** when a transaction Ti commits.
- **<Ti abort>:** when a transaction Ti aborts.

These records allow undo and redo operations in case of failure of a transaction. For log-based recovery database system consults the log record to identify which transaction needs to be redo and which needs to be undo. We can see situations, when undo and redo operations are applied on a database:

- Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
- Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

For log records to be useful for recovery from system and disk failures, these log records must reside on stable storage.

8.17 Failure with Loss of Nonvolatile Storage

In the above recovery mechanisms, we had discussed about the recovery of data loss from volatile memory. But in some rare situations, failure of nonvolatile storage results in data loss.So, to recover data from failure of non-volatile storage *dump* scheme is used. It dumps the entire content of the database to stable memory periodically. No transaction can be active during the dump procedure. Procedure to dump the database follows steps:

1. Output all log records currently residing in main memory onto stable storage.
2. Output all buffer blocks onto the disk.
3. Copy the contents of the database to stable storage.
4. Output a record <dump> to log on stable storage.

To recover from disk or non-volatile storage, the system-

1. Restore database from most recent dump.
2. Consult the log and redo all transactions that committed after the dump

This is also known as an archival dump.

Check your Progress

1. Discuss on Failure Classification.

2. Explain Log Based Recovery.

8.18 Summary

Transaction refers to a unit of program execution that accesses and updates a data item. The transaction processing takes place in such a way that any failure cannot leave the database in an inconsistent state. Transaction is required to maintain the ACID (atomicity, consistency, isolation and durability) properties. As concurrent transactions have the ability to use the CPU cycle to its fullest but execution of concurrent transactions can lead to inconsistency.

It is therefore mandatory for the system to control the interleaving transactions. Schedules are maintained by system to achieve consistency. Serial schedules are consistent while non-serial schedules can cause inconsistency in database. So, the concept of Serializability is introduced to make the database consistent by testing the schedules whether they are serializable or not. Along with this concurrency control techniques based on locking, timestamp and validation are used. Some techniques result into deadlocks which is also deal by database systems using deadlock handling techniques.

As a computer system like any other electrical device are prone to failure. These failures cab be because of disk crash, power loss, software errors or any catastrophe. In case of any failure, state of database system may no longer be consistent. Then it is necessary to recovery the data loss. Recovery schemes such as logs, shadow paging, periodic dumps are followed by data bases to maintain the atomicity and durability property.

Terminal Questions

1. What isTransaction Processing?
2. Why we need of Concurrency Control in Transaction Processing.
3. Discuss the Database Recovery Techniques.
4. What does the ACID acronym for? Describe each property individually.
5. What is the difference between the following transaction states: Aborted vs Failed and Active vs Committed?
6. What do you understand by Deadlock? Explain its features with an example.
7. what is concurrency control ? Explain it with a suitable technique.
8. What is shadow paging? How it is differ from Log based recovery?

Unit 9 - Emerging Trends in DBMS

Structure:

9.1 Introduction

From the last four decades, businesses relied on relational database management systems (RDBMSs)—that used Structured Query Language (SQL) as the programming language. But the recent applications and emergence of network demand a completely different set of requirements in terms of the underlying database models. The conventional relational database model is no longer appropriate for these types of applications. Therefore, this unit will introduce emerging databases such as Object - oriented, distributed databases, client server database and knowledge database, their functionality and requirement.

9.2 Objectives

By the end of this unit, you should be able to understand about the various database models and their basic functioning. This will give you a view how these databases are different from conventional database systems and helps the application programs.

9.3 Introduction to object-oriented Database Management System

In early 1960s computerized databases were there with the availability of disks and drums that helps in maintaining data. Then in 1970s, Database technology improves and aims to make the data independent from the logic of application programs. This enables the different application programs to access the data concurrently. The first-generation databases were navigational; here data was accessed through record pointers moving from one record to another. With more advent of

technology, this was followed by relational model, which focused on data than pointers for data retrieval. These kinds of databases are more popular till date.

The Relational databases have many features and support in maintain data and information of commercial systems and business applications. RDBMS are capable to handle simple and fixed collection of data types, support high level queries, query optimization, transaction backup and crash recovery.

Despite all the above features, a number of limitations exist with relational model:

- Many other application domains need complex kinds of data such as CAD/CAM, multimedia repositories, and document management. To support such applications, DBMSs must support complex data types.The lack of support for new data types such as graphics, xml, 2D and 3D data.
- With the advent of Object-Oriented methodologies and languages,it was quite difficult to map data of application programs.

Hence, to cope up with all arising problems and to meet out the needs of complex data, object database systems were developed.

Object Oriented Database (OODB) implements OO concepts such as object identity, polymorphism, encapsulation and inheritance to provide access to persistent objects using any OO-programming language.

OODB also provides a unified environment when dealing with complex data such as 2D and 3D graphics by proper mapping of object orientation and databases. These databases are designed to work well object oriented programming languages such as Objective-C, Java, and Python.

9.3.1 Basic Object Oriented concepts - object, attribute, OID, class, method, encapsulation, class hierarchy, single/multiple inheritance, extensibility, complex object, overloading, overriding, polymorphism, user-defined typeAs OODB follow the concepts of Object data model. So, in this both data and its relationship are combined together in a single data structure. This data structure as a whole is termed as object.

9.3.1.1 Object: it is an abstraction of real-world entity. This entity conceptually exists and can be identified distinctly such as person, employee, student, book etc. these objects have structural properties which are defined by set of attributes and behavioural properties which are defined by methods. Each object is associated with a

logical non-reusable and unique object identifier (OID). The OID of an object is independent of the values of its attributes

9.3.1.2 Attributes: the properties of objects which help in identifying the objects are termed as attributes.

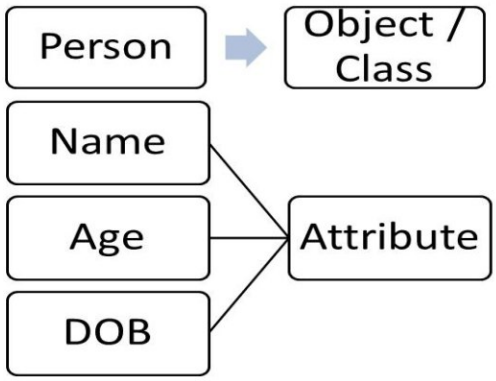


Figure 7.1 – Attribute & Object

9.3.1.3 Class: Objects which are similar in nature or have same attributes are grouped together in a class. So, we can say class is a collection of similar objects which have same attributes and behavior.

Classes are classified as lexical classes and non-lexical classes.

1. A lexical class contains objects that can be directly represented by their values.
2. A non-lexical class contains objects, each of which is represented by a set of attributes and methods. Instances of a non-lexical class are referred to by their OIDs. Example person and employee part are non-lexical classes.

9.3.1.4 Method: These are the procedures which modify the objects state and it allows one object to communicate with other.A message passing system is used to call a method of an object. A method's specification is represented by a method signature, which have the method name and information on the types of the method's input parameters and its functionality and output.

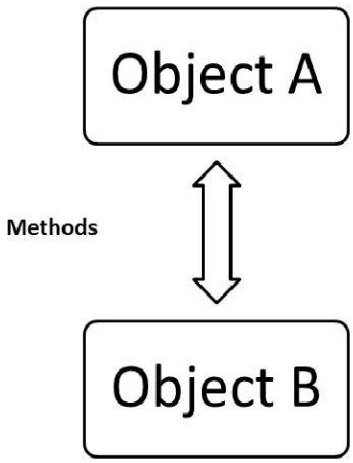


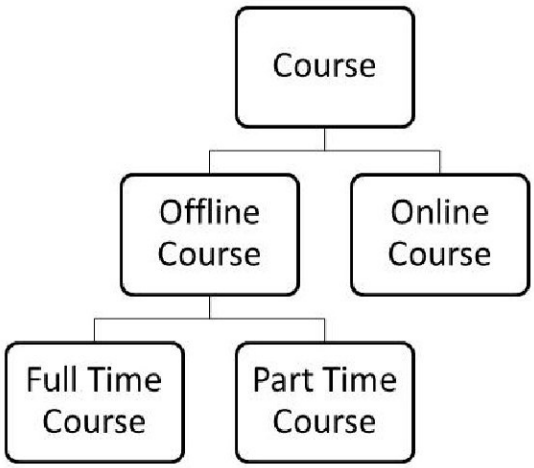
Figure 7.2 Methods

E.g. when an employee is fired, we need to delete the employee information from the employee file, delete the employee from the employee-project file, and insert

the employee information into a history file, etc. One method called "Fire-employee" can be defined that incorporates this sequence of actions.

9.3.1.5 Class Hierarchy: when classes are arranged that represents an upside down tree and have parent child relationship. By this it supports Inheritance.

9.3.1.6 Inheritance: As like parent child relationship, it allows a class to inherit properties (attributes and methods) from its super classes.



In this figure full time and part time course Inherit the properties (attributes and methods) of class Course.

Figure 7.3 Inheritance

9.3.1.7 Abstraction:in this some aspects of an entity are detailed which are needed and rest are ignored.

9.3.1.7 Encapsulation: it is a binding mechanism by which we can bind state(attributes) and behaviour(methods) of an object together.

9.3.1.8 Generalization: It is method to create a superclass is called generalization.

9.3.1.9 Specialization: It is process of forming a sub class is called specialization.

9.3.1.10 Polymorphism: It means —many forms. It is dynamic feature which executes at run time of program. It involves the concept of overriding and overloading.

9.3.2 Object Definition Language: There is a standardized language for defining the structure of object oriented databases. ODL defines three components of the object-oriented data model: Abstraction, Inheritance and Encapsulation

9.3.3 Object Query Language: As like SQL, it includes declarative statements. Besides this OQL includes more language constructs which permit for object-oriented design such as operation invocation and inheritance. Syntax or OQL query structure looks very similar to SQL but the results returned are different. As OQL query returns a set of objects.Example: OQL query to obtain Voter names who are from the state of Uttarakhand

Select distinct v.name From voters v Where v.state = "Uttarakhand"

Voter Id	Name	State
V1	Ramesh	Uttarakhand
V2	Harish	Uttarakhand
V3	John	Goa

Result from SQL table with rows
Name
Ramesh Harish

Result from OQL collection of Objects	
String	String
Ramesh	Harish

9.3.4 Applications for OO databases

The applications that use complex data types and needed high performance include:

- Computer-aided design and manufacturing (CAD/CAM)
- Computer-integrated manufacturing (CIM)
- Computer-aided software engineering (CASE)
- Geographic information systems (GIS)
- Many applications in science and medicine
- Document storage and retrieval

Check your Progress

1. What is OO database?
2. Describe the Basic Object Oriented concepts.
3. Explain Object Query Language with example.
4. Explain the application of OO Database.

9.4 Introduction to client/Server Database

As the name suggests these type of databases have two components that are client and the server.

9.4.1 Client: A client represents any end user which makes a request. It may be a application program, computer -mobile with a software application.

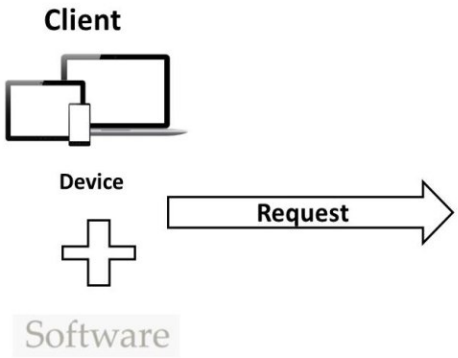


Figure 7.4 Client

9.4.2 Server: Server is a centralized computer that provides services to all attach clients. Itaccepts the request of clients and maintains a connection according to a defined protocol. For example file server, web server, mail server etc.

9.4.3 Working of Client-server Database Architecture in DBMS

In client / server architecture many clients connected with one server. The server is centralized; it provides services to all clients. All clients request to the server for different Service. The server responds according to the client's request.

Client/server architecture is a computing model in which the server hosts (computer), send and manages most of the resources and works to be required by the client. In this type of architecture has one or more client computers attached to a central server over a network. This system shares different resources.

Client/server architecture is also called as a networking computing model and client-server network because all the requests and demands are sent over a network.

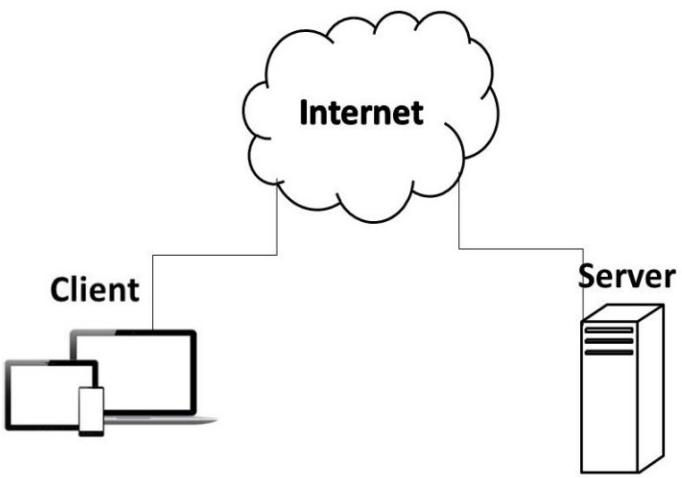


Figure 7.5 Client/server architecture

This architecture is basically working on three layers/ levels which shows how client access the data or response at each level. This is also termed as threetier architecture as it follows three basic layers that are:

- Presentation tier
- Application tier
- Database tier

9.4.3.1 Presentation tier: this is a top most layer which provides a user interface. Here, a user makes a request without knowing about the existence of the database beyond this layer. In this user sends a request by using an application program or any software.

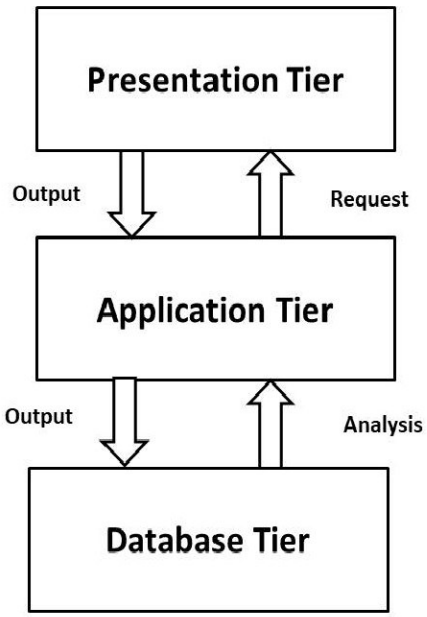
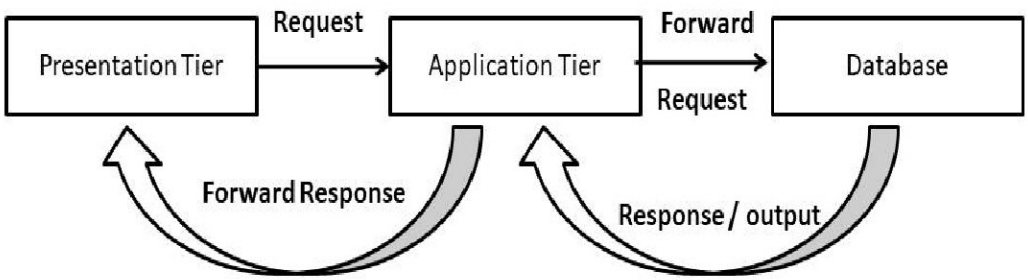


Figure 7.6 Client/server architecture

9.4.3.2 Application tier: This layer behaves as a middle man between presentation and database layer. It takes the request from above layer and validates the request then forwards it to database tier. Then in reverse manner, it takes the response from database layer and forwards it to presentation layer.



9.4.3.3 Database tier: This layer consistsofdatabase, which allows storage and retrieval of data. This layer process the request arrived from the application layer and take a response and forward that response to the application layer.

Check your Progress

1. What is Client / server database?
2. Describe the Basic Client-server Database concepts.
3. Explain Client-server Database Architecture.

9.5 Introduction to Distributed Database

Initially when database came into existence, it was single system architecture. But with the increasing demand of data base applications, voluminous amount of data and with the dispersed users around the worldcreates a heavy load on a single database server. Hence all these increasing factors influence the performance, reliability, concurrency and security of the database.

So, to overcome these issues, a new mechanism of allocating users and DB server is introduced. This new concept is known as Distributed database system (DDB).

9.5.1 Distributed Database: it is collection of multiple, logically interrelated databases that are distributed over a computer network.

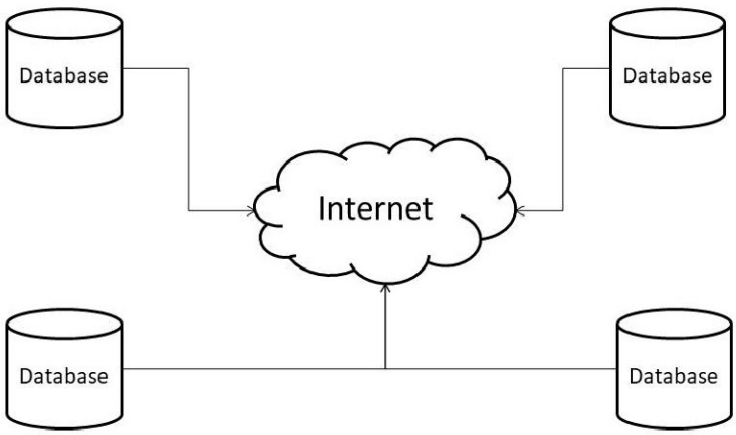


Figure 7.8 Distributed Database

Here, different database server is created and are placed at different locations rather than at single location. All these remote or distributes server kept in sync with each other in order to maintain consistency. In this user can access any of database without knowing its location.

9.5.2 Distributed Data Storage:

There are two data storage processes involved in order to ensure that the distributed database is reliable and are working efficiently. These are replication and Fragmentation.

9.5.2.1 Replication: In this multiple copy of data are stored at different locations for faster retrieval and fault tolerance.

9.5.2.2 Fragmentation: In this, a relation is portioned into several fragments and stored at different locations. It is of two types horizontal(tuples) fragmentation and vertical(attributes) fragmentation.

9.5.3 Features of Distributed database systems:

- Data is stored at multiple locations rather than on a single location.

- All database servers are interconnected to any computer network.
- Distributed database logically seems a single database.
- It provides full functionality of database management system.

9.5.4 Types of Distributed Database Systems

DDBs are classified in two types:

1. Homogeneous DDB
2. Heterogeneous DDB

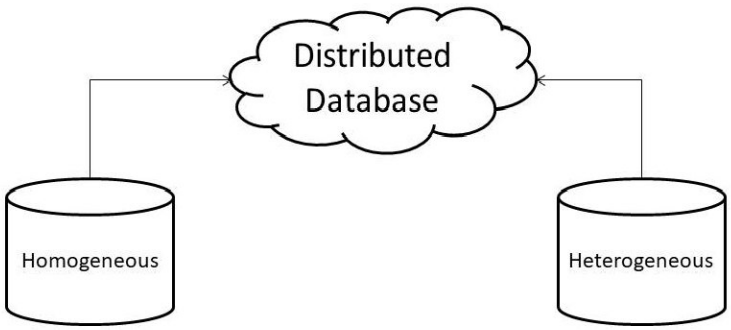


Figure 7.9 Types of Distributed Database

9.5.4.1 Homogeneous DDB: in this, all interconnected databases have identical database systems in terms of software, hardware, operating systems, database management software's and all components that are essential for having a database. These DDBs also have common global schema over DDBMS.

9.5.4.2 Heterogeneous DDB: this is contrast to homogeneous DDBs. Here, databases over the network may be different along with other components that are required for the managing database. In this, Database schema at one location can differ from another location.

9.5.4.4 Advantages of Distributed Databases

- Allows sharing of data by maintaining location transparency, network transparency, naming transparency.
- Improves availability and reliability of data even on failure of any of the system.
- Providing better performance by reducing network load and time.
- It also reduces the operating cost.
- It is easier to expand DDBs by adding new database server at different location.

9.5.4.5 Disadvantages of Distributed Databases

- Increases the complexity, asits difficult to maintain all of them to work together, to keep them in sync, coordinate and make them work efficiently.
- Difficult to maintain integrity.
- Security of data is also an issue as data is scattered over the network.
- Deadlock handling is difficult in DDBs.
- Fragmentation of data and its distribution is also challenge.

Check your progress

1. What is Distributed database?
2. Describe the Basic types of Distributed Databases.
3. Write down the advantages and disadvantages of Distributed Database.

9.6 Introduction to Knowledge Databases

Knowledge databases are the database management system that manages the knowledge in a specific domain and exhibits reasoning power as like human behaviour. It facilitates the collection, organization and retrieval of knowledge.

Most of the knowledge database is based on Artificial intelligence that helps databases in not only storing and retrieving data efficiently but also in making decisions smartly. Knowledge databases use the techniques ranging from traditional relational databases to data warehousing. It also uses the data from previous experiences as a part of knowledge base.

9.6.1 Knowledge

Knowledge can be defined as evidences and ideologies collected by human-kind or the act, fact or state of knowing. Knowledge is stored in computer systems in symbolic structures as like neurons of human being. Here these structures are in the form of collections of magnetic spots and voltage states. There are some examples that are representing facts:

1. Meera is clever then geeta.
2. Rajesh wedded to Rashmi.

These statements express some facts. To understand and make use of this knowledge, a person needs other world knowledge and the ability to reason with it. So, to make decisions on complex problems, knowledge-based databases come into picture.

9.6.2 Knowledge Databases

Knowledge based database (KBDB) are systems that depend on a rich base of knowledge to perform difficult tasks. KBDB use database concepts and models to store and retrieve knowledge. As these systems helps in managing the knowledge then termed as Knowledge management systems.

These database systems typically can help link and integrate all available knowledge sources, including explicit knowledge (various kinds of databases stored in existing information systems) and inexplicit knowledge (practical experience, skills, thought and thinking method in the brain of the experts / people) to form knowledge databases of various kinds. These databases facilitate the people:

- To find out the knowledge they need from disordered information.

- In providing most optimal knowledge to the most optimal people in the most optimal time to enable them to make the most appropriate decision-making.
- Solve complicated problems with relative ease.

9.6.3 Types of Knowledge Database

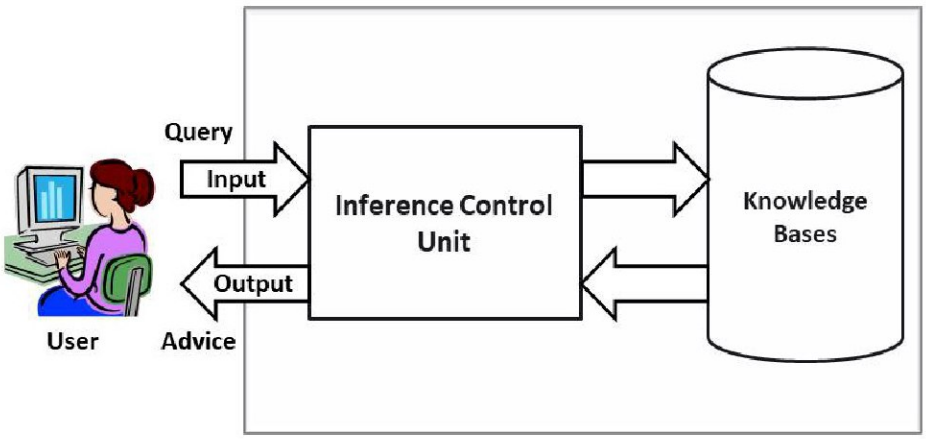
There are two types of knowledge database:

1. **Human Readable KBDB:** as they enable people to access and use the knowledge stored in documents, manuals, troubleshooting information and frequently answered questions.
2. **Machine ReadableKBDB:** the information stored is machine readable. The solutions provided by them are based onautomated deductive reasoning. Here, information shared is linear and is limited in interactivity, unlike the human interaction which is query based.

9.6.4 Components ofKnowledge Databases

Four main components of Knowledge Databases

1. Input / query
2. Inference Control Unit
3. Knowledge Bases
4. Output / Advice



9.6.4.1 **Input / query** - With the help of a user interface, a user makes input/output query to communicate with the knowledge base system to find a solution.

9.6.4.2 **Inference Control Unit** - The inference engine is known as the brain of the system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.

9.6.4.3 Knowledge bases - It is similar to a database that contains information and rules of a particular domain or subject. The more the knowledge makes more precise decision.

9.6.4.4 Output / Advice - After getting the response from the inference engine, it displays the output to the user.

9.6.5 Advantages of Knowledge Databases

- It helps in making precise and faster decision making.
- Resolves the problem quickly.
- Minimizes the possibility of errors.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- Enhances performance by better and expert knowledge.

9.6.6 Limitations of Knowledge Databases

- It is difficult to maintain accurate and precise knowledge.
- It cannot produce a creative output for different scenarios as a human being can.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- It cannot learn from itself and hence requires manual updates.
- It's also difficult to choosing and implementing knowledge management technology

9.6.7 Applications of Knowledge Databases

Knowledge based database systems are used in various fields such as:

- Medical Diagnosis Systems
- Engineering Systems
- Quality Management Systems
- Geographical Information Systems
- Expert Systems
- Client Service Software /Incident Management System

Check your progress

1. What is Knowledge Databases?
2. Describe the Basic types of Knowledge Databases.
3. Write down the advantages and disadvantages of Knowledge Databases.
4. Explain the main Components of Knowledge Databases.

9.7 Summary

In the past few years, however, there has been an increasing need for handling new data types in databases, such as temporal data, spatial data, multimedia data, and geographic data and so on. This has resulted into the development of new database technologies to handle new data types and applications.

Object Oriented Database (OODB) implements OO concepts such as object identity, polymorphism, encapsulation and inheritance to provide access to persistent objects using any OO-programming language.

Client/server database is a computing model in which the server hosts (computer), send and manages most of the resources and works to be required by the client.

Distributed Database is collection of multiple, logically interrelated databases that are distributed over a computer network.

Knowledge databases are the database management system that manages the knowledge in a specific domain and exhibits reasoning power as like human behaviour and helps in making intelligent solutions.

Terminal Question

1. Define DDBs and state is advantages and disadvantages.
2. How would you define object orientation? What are some of its benefits?
3. What is the difference between an object and a class in the object-oriented data model?
4. Define Knowledge Databases and its main Components.
5. Explain applications of Knowledge Databases.
6. Explain the data storage process in Distributed Database.
7. Discuss the working of Client-server Database.
8. Define Client in Client-server Database.
9. Differentiate between Object Definition Language and Object Query Language.
10. Define Object Query Language in OO database.

Notes

Notes

Notes