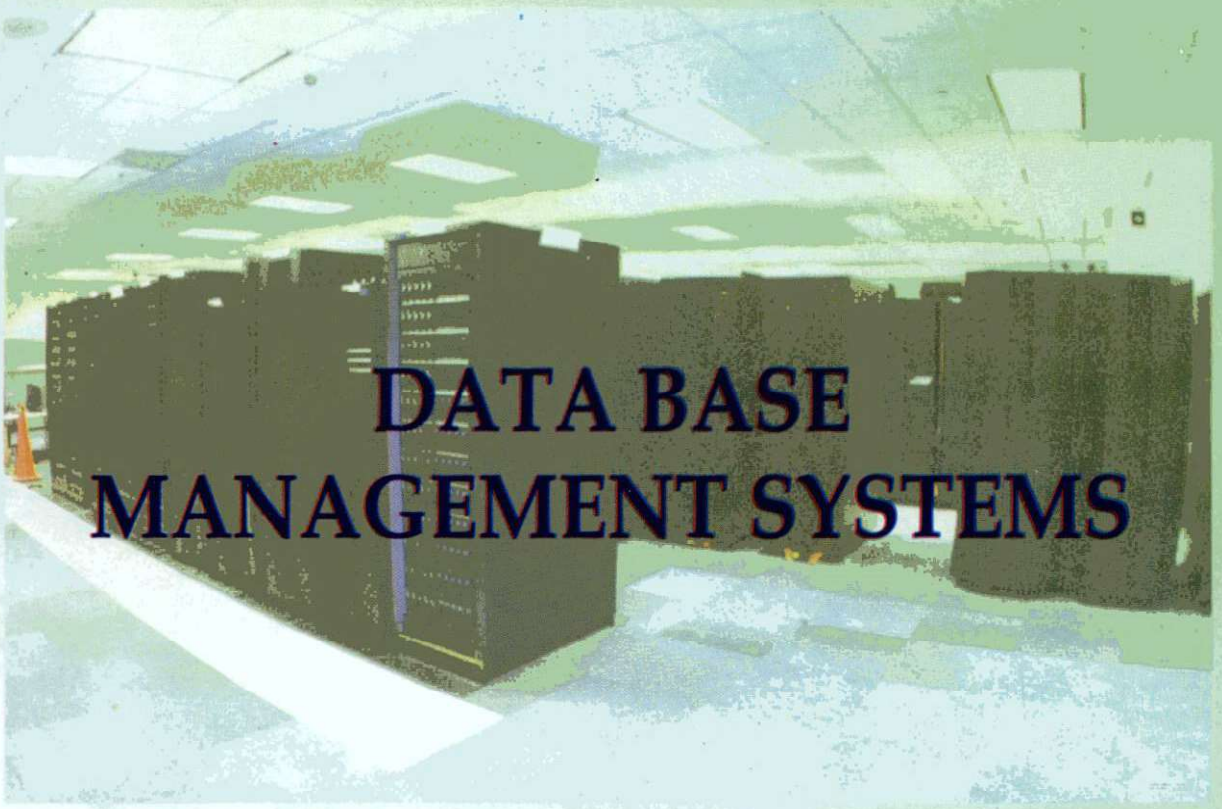
ಕರ್ನಾಟಕ ರಾಜ್ಯ ಮುಕ್ತ ವಿಶ್ವವಿದ್ಯಾನಿಲಯ
ಮಾನಸಗಂಗೋತ್ರಿ, ಮೈಸೂರು - 570 006

KARNATAKA STATE OPEN UNIVERSITY
Manasagangotri Mysore - 570 006

M.Sc. Computer Science

Second Semester



DATA BASE MANAGEMENT SYSTEMS

Course: 8

Module : 1 - 6

MSCS-508 DBMS

ಉನ್ನತ ಶಿಕ್ಷಣಕ್ಕಾಗಿ ಇರುವ ಅವಕಾಶಗಳನ್ನು ಹೆಚ್ಚಿಸುವುದಕ್ಕೆ ಮತ್ತು ಶಿಕ್ಷಣವನ್ನು ಪ್ರಜಾತಂತ್ರೀಕರಿಸುವುದಕ್ಕೆ ಮುಕ್ತ ವಿಶ್ವವಿದ್ಯಾನಿಲಯ ವ್ಯವಸ್ಥೆಯನ್ನು ಆರಂಭಿಸಲಾಗಿದೆ.

ರಾಷ್ಟ್ರೀಯ ಶಿಕ್ಷಣ ನೀತಿ 1986

The Open University System has been initiated in order to augment opportunities for higher education and as instrument of democrating education.

National Educational Policy 1986

ವಿಶ್ವ ಮಾನವ ಸಂದೇಶ

ಪ್ರತಿಯೊಂದು ಮಗುವು ಹುಟ್ಟುತ್ತಲೇ – ವಿಶ್ವಮಾನವ, ಬೆಳೆಯುತ್ತಾ ನಾವು ಅದನ್ನು 'ಅಲ್ಪ ಮಾನವ'ನನ್ನಾಗಿ ಮಾಡುತ್ತೇವೆ. ಮತ್ತೆ ಅದನ್ನು 'ವಿಶ್ವಮಾನವ'ನನ್ನಾಗಿ ಮಾಡುವುದೇ ವಿದ್ಯೆಯ ಕರ್ತವ್ಯವಾಗಬೇಕು.

ಮನುಜ ಮತ, ವಿಶ್ವ ಪಥ, ಸರ್ವೋದಯ, ಸಮನ್ವಯ, ಪೂರ್ಣದೃಷ್ಟಿ ಈ ಪಂಚಮಂತ್ರ ಇನ್ನು ಮುಂದಿನ ದೃಷ್ಟಿಯಾಗಬೇಕಾಗಿದೆ. ಅಂದರೆ, ನಮಗೆ ಇನ್ನು ಬೇಕಾದುದು ಆ ಮತ ಈ ಮತ ಅಲ್ಲ; ಮನುಜ ಮತ. ಆ ಪಥ ಈ ಪಥ ಅಲ್ಲ ; ವಿಶ್ವ ಪಥ. ಆ ಒಬ್ಬರ ಉದಯ ಮಾತ್ರವಲ್ಲ; ಸರ್ವರ ಸರ್ವಸ್ತರದ ಉದಯ. ಪರಸ್ಪರ ವಿಮುಖವಾಗಿ ಸಿಡಿದು ಹೋಗುವುದಲ್ಲ; ಸಮನ್ವಯಗೊಳ್ಳುವುದು. ಸಂಕುಚಿತ ಮತದ ಆಂಶಿಕ ದೃಷ್ಟಿ ಅಲ್ಲ; ಭೌತಿಕ ಪಾರಮಾರ್ಥಿಕ ಎಂಬ ಭಿನ್ನದೃಷ್ಟಿ ಅಲ್ಲ; ಎಲ್ಲವನ್ನು ಭಗವದ್ ದೃಷ್ಟಿಯಿಂದ ಕಾಣುವ ಪೂರ್ಣ ದೃಷ್ಟಿ.

ಕುವೆಂಪು

Gospel of Universal Man

Every Child, at birth, is the universal man. But, as it grows, we trun it into "a petty man". It should be the function of education to turn it again into the enlightened "universal man".

The Religion of Humanity, the Universal Path, the Welfare of All, Reconciliation, the Integral Vision - these ***five mantras*** should become view of the Future. In other words, what we want henceforth is not this religion or that religion, but the Religion of Humanity; not this path or that path, but the Universal Path; not the well-being of this individual or that individual, but the Welfare of All; not turning away and breaking off from one another, but reconciling and uniting in concord and harmony; and above all, not the partial view of a narrow creed, not the dual outlook of the material and the spiritual, but the Integral Vision of seeing all things with the eye of the Divine.

Kuvempu

Second Semester M. Sc. Computer Science

Module 3 Database Design

Module 4 Concept of Storage and Indexing

Module 5 Transaction Management and Recovery Techniques

Module 6 Case Studies

Preface

This material is prepared to give an overview of Database Management Systems (DBMS) for the Second Semester course in Computer Science curricula. It is suitable for both hardware and software-oriented students. To study the design details of database management system and the various concepts related to DBMS, this material has been prepared. The whole material is organized into six modules each with four units. Each unit lists out the objectives of study along with the relevant questions and suggested reading to better understand the concepts.

Module-1: Gives an introduction to Database Management Systems (DBMS). It starts with a brief history of DBMS. It compares DBMS verses file systems. It describes the overall structure of DBMS. It also describes the conceptual data models.

Module-2: Introduces the relational model concepts. It describes relational database schemes. It describes overview of relational algebra. An introduction to query language SQL is also given.

Module-3: Describes the database design theory. Functional dependencies and axioms on them discussed. Normal forms are introduced as a tool for better database design. Algorithms for relational database scheme design are given.

Module-4: Introduces the concept of storage and indexing. File organizations are explained. It covers primary, clustered and secondary indexes. Hash-based indexing is also discussed. It also describes tree-based indexing. At the end, comparisons of file organizations is given.

Module-5: Introduces the transaction management and recovery. The ACID properties are explained. Various types of schedules are covered. The motivation for concurrent execution and anomalies due to them covered. Concept of serializability is explained. Currency control schemes and ARIES recovery mechanisms are discussed.

Module-6: introduces Oracle and DB2. In both Oracle and DB2 database design and query tools are explained. SQL variations, storage and indexing, query processing and optimization are covered. Concurrency control and recovery, system architectures and database administrations tools are explained.

We thank everyone who helped us directly or indirectly in preparing this material. Without their support, this material would not have been a reality.

Dr. Suresha

Smt. L. Hamsaveni

UNIT 1: OVERVIEW OF DBMS

Structure:

1.0 OBJECTIVES

At the end of this unit you will be able to:

- Discuss history behind DBMS
- Explain the problems with conventional file systems
- State the advantages of DBMS
- Conceive the levels of abstraction

1.1 INTRODUCTION

This unit is about the need for a Database Management Systems (DBMS). It starts with definition of DBMS. It gives a brief history of DBMS. It compares DBMS with conventional file systems provided by operating systems. It describes problems with conventional file systems. It also gives the advantages of DBMS. At end you understand the levels of abstraction in a DBMS.

1.2. DEFINITION

A database is collection of interrelated data of an enterprise. Where as a **Database Management System (DBMS)** is a set of computer programs (a software) that controls the creation, maintenance, and the use of one or more database. DBMS is designed to assist in the collection and maintenance of utility data in large numbers. It is a system that makes information available when required. It allows different user application programs to easily access the same database

In large systems, a DBMS allows users and other software to store and retrieve data in a structured way. Instead of having to write computer programs to extract information, user can ask simple questions in a query language. Thus, many DBMS packages provide Fourth-generation programming language (4GLs) and other application development features. It helps to specify the logical organization for a database and access and use the information within a database. It provides facilities for enforcing data integrity, controlling data access, restoring the database from backups and managing concurrency. A DBMS also provides the ability to present database information to users in a more meaningful way.

1.3 HISTORY OF DBMS

First-generation DBMS designed by Charles Bachman in the company General Electric in the early 1960s, referred to as the Data Storage Integrated (Integrated Data Store) formed the basis for the network data model then by standardization Conference on Data System Languages (CODASYL). And at the end of 1960, IBM developed management information system (Information Management System) DBMS. IMS formed from the data on the representation framework, called the data model hierarchy.

Then in 1970, Edgar Codd (IBM San Jose Research Laboratory, USA), proposed the model called relational data model. By 1980, the relational data model became the most dominant. SQL query language was developed for relational database project as part of IBM's System R. SQL standardization was done at the end of 1980. SQL-92 was adopted

by the American National Standards Institute (ANSI) and International Standards Organization (ISO).

At the end of 1980 and beginning of 1990, many areas of the database systems developed. Some vendors extended the system with the ability to store new data types such as image and text, and ability of complex queries.

System-specific / special databases developed by many vendors to create a data warehouse, consolidate data from multiple base data. Furthermore, the DBMS enter the internet. At the time of the first generation of Web site store data exclusively in the file system operation, the current DBMS can be used to store data that can be accessed through a Web browser. Query degenerate can form via the Web, and format the response using the markup language such as HTML for easier viewing in a browser.

The importance of online data is increasing. Current areas such as this realized in the database multimedia, interactive video, digital libraries, the project scientists, such as project mapping, earth observation system project NASA property, etc.

1.4 FILE SYSTEM VERSUS A DBMS

A file is facility provided by the operating systems, to store, retrieve and update some data. Files are stored in specific locations on the hard disk (directories). The user can create new files to place data in, delete a file that contains data, rename the file, etc., all known as file management.

If the user wishes to perform some operation on the data he has placed in the file, such as viewing a list of his clients that celebrate their birthday in May, he has to scroll through all the data by himself in order to see the data he is interested in. Moreover, he has to know where he has put the files that contain the data, and if there are multiple files he has to remember to go through each one of them.

A Database Management System is intended to remove this burden of manually locating data, and having to scroll through it by allowing the user to create a logical structure for the data beforehand, and then allowing the user to place the data in the database that the DBMS is managing. In this way the DBMS abstracts away the physical concerns of organizing files, and provides the user with a logical view of the data. Note, that the DBMS will still (behind the scenes) place the data in files on the hard-disk.

DBMS may be multi-user and provides better utilization of resources compared to file management systems.

If we keep the organizational information in a conventional file processing system, then it has a number of disadvantages: Namely,

Difficulty in accessing the data:

File processing system provides a customized solution. In file processing system, we need to define the data-structures and write programs to access, manipulate, and maintain the data stored in files for each application.

DBMS provides a generic solution. We just need to logically define the structure of data to be stored. All the programs to access, manipulate, and maintain the data are provided by the DBMS.

Lack of data independence:

File processing system provides a specific solution to a particular problem for which it is implemented. For example, an employee information system cannot cater the library even if the same employees go to the library to borrow books.

In DBMS, there is no connection between the user/programmer with the data storage. Both the user and the programmer need not worry about how data is stored or accessed. They have to use the interfaces/libraries provided by DBMS to access the data. So, the data is portable and a single source of data can serve several users around a variety of applications.

Inconsistency in data:

In a file processing system, the data files are not protected fully, so they can be changed with out restrictions. Any one who is able to use the program to handle the data files can change anything on the data file so there is every possibility of data becoming inconsistent due to improper handling.

For example, a non-existing employee having borrowed books from library is classic example of inconsistency in data in a file processing system.

Data redundancy:

In a file processing system the process of getting inter-related is difficult, so mostly data is duplicated across data-files to ease the data access. Duplicated data is a cause of inconsistency because data updated at a single place is not available instantly to all the places where data is duplicated. DBMS tries to minimize or remove this data duplication by having references to the data, where it is stored.

Absence of constraints:

In a file processing systems all the programs accessing the data are independent, so having control over data access is a problem. DBMS have the ability to enforce constraints and also change them without affecting the programs in any way.

Flexibility:

The file processing system is a customized solution. It cannot be adopted for another problem or can be marketed. For each and every query, we have to write a program. It is a difficult task. There is very less re-use of code, since the code is tightly bound with data.

Integrity problems:

In a file processing system all the programs accessing the data are independent, so controlling the access of data is a very big problem. There are absolutely no constraints at the file level. Any constraints required need to written as a program and which is very complex. The DBMS have the ability not only to enforce constraints but also change them without affecting the program in any way.

Atomicity:

In a file processing system, there is no way to ensure the completion a transaction. Any partial transaction may lead to illegal data being inserted into data-files. For example, when we transfer an amount from one account to another account, if the transaction fails after with-drawl from one account and before deposit to another account results in illegal transaction. DBMS ensures multiple operations to be treated as one operation, so any transaction is carried out atomically, without having any partial execution effect on the database.

Concurrent access anomalies:

When many users want to update a single record at a time, this leads to concurrent access anomalies. Only the last successful update is stored. DBMS solves this problem by implementing concurrency control algorithms.

Security problems:

The security in a file processing system is limited. It provides security at course grain. Either the entire file can be made accessible or not to a user. A part of the data cannot be exposed while restrictions are provided for other parts of data.

1.5 ADVANTAGES OF DBMS

There are many advantages of DBMS. The important ones are:

- Greater flexibility
- Greater processing power
- Fits the needs of many medium to large-sized organizations
- Storage for all relevant data
- Provides user views relevant to tasks
- Ensures data integrity by managing transactions (ACID test = atomicity, consistency, isolation, durability)
- Supports simultaneous access
- Enforces design criteria in relation to data format and structure
- Provides backup and recovery
- Advanced security

The goals of a DBMS

There are many goals of DBMS. They are:

- Data storage, retrieval, and update (while hiding the internal physical implementation details)
- A user-accessible catalog
- Transaction support
- Concurrency control services (multi-user update functionality)
- Recovery services (damaged database must be returned to a consistent state)

- Authorization services (security)
- Support for data integrity services (i.e. constraints)
- Services to promote data independence
- Utility services (i.e. importing, monitoring, performance, record deletion, etc.)

The components to facilitate the goals of a DBMS may include the following:

- Query processor
- Data Manipulation Language preprocessor
- Database manager (software components to include authorization control, command processor, integrity checker, query optimizer, transaction manager, scheduler, recovery manager, and buffer manager)
- Data Definition Language compiler
- File manager
- Catalog manager

1.6. LEVELS OF ABSTRACTIONS IN A DBMS

The major purpose of a database system is to provide users with an abstract view of the system. The system hides certain details of how data is stored and created and maintained. The complexity is hidden from database users.

There are three levels of data abstraction: namely

1. Physical level: It deals with how the data is stored physically and where it is stored in database. This is the lowest level of abstraction.
2. Logical level: It describes what information or data is stored in the database (like what is the data type or what is the format of data? or the relationships among data).
3. View level: Describes *part* of the database for a particular group of users. End users work on view level. There can be different views of a database. This is the highest level of abstraction.

Figure 1.1 illustrates the three levels.

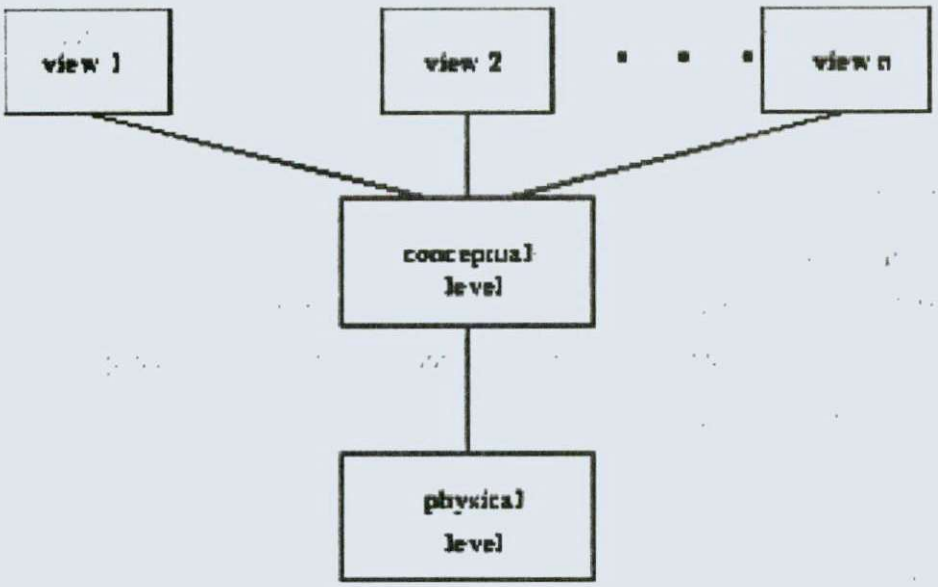


Figure 1.1: The three levels of data abstraction

1.7 SUMMARY

A database is collection of interrelated data of an enterprise. Where as a **Database Management System (DBMS)** is a set of computer programs (a software) that controls the creation, maintenance, and the use of one or more database. In this unit, we considered the historical perspective of DBMS. We also discussed the problems with conventional file processing compared to DBMS. We ended with levels of data abstraction.

1.8 KEYWORDS

Database: A database is collection of interrelated data of an enterprise

DBMS: DBMS stands for **Database Management System**, is a set of computer programs.

4GL: It stands for Fourth-generation programming language.

File: A file is facility provided by the operating systems, to store, retrieve and update some data.

ACID: A transaction has to satisfy the ACID property, which stands for atomicity, consistency, isolation, durability.

Abstraction: hiding certain details of how data is stored and created and maintained.

1.9 UNIT-END EXERCISES AND ANSWERS

1. Give an historical perspective of DBMS
2. Compare DBMS with conventional file processing
3. What are the advantages of DBMS?
4. What are the goals of DBMS?
5. Discuss levels of abstraction

Answers: SEE

1. 1.3
2. 1.4
3. 1.5
4. 1.5
5. 1.6

1.10 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta
- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan
- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke

UNIT 2: STRUCTURE OF DBMS

Structure:

2.0 OBJECTIVES

At the end of this unit you will be able to:

- Explain Data Independence
- State the Structure of DBMS
- Elucidate the people who deal with databases
- Discuss the DBMS Architecture

2. 1 INTRODUCTION

This unit is about structure of DBMS. It starts with Data Independence, where we study the importance of data independence and types of data independence. It gives a detailed structure of DBMS. It also describes various people who deal with databases.

2.2 DATA INDEPENDENCE

Separation of data from processing, either so that changes in the size or format of the data elements require no change in the computer programs processing them or so that these changes can be made automatically by the database management system.

These are the techniques that allow data to be changed without affecting the applications that process it. There are two kinds of data independence. The first type is data independence for data, which is accomplished in a database management system (DBMS). It allows the database to be structurally changed without affecting most existing programs. Programs access data in a DBMS by field and are concerned with only the data fields they use, not the format of the complete record. Thus, when the record layout is updated (fields added, deleted or changed in size), the only programs that must be changed are those that use those new fields.

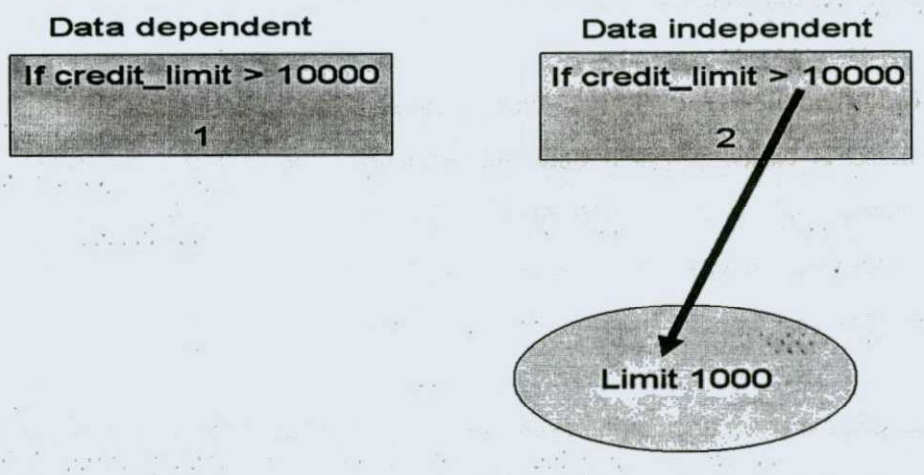


Figure: 2.1 Data Independence

Program number1 uses a hard-coded value to test credit limit. To change the limit, the program must be recompiled. Program number 2 retrieves the credit limit from a database. To change it, only the database must be updated, which is a simpler task. This is illustrated in Figure 2.1.

Data Independence Types

Data independence is the type of data transparency that matters for a centralized DBMS. It refers to the immunity of user applications to make changes in the definition and organization of data.

Data independence has two types:

1. Physical Independence and
2. Logical, Independence.

The term data independence can be explained as follows: Each higher level of the data architecture is immune to changes of the next lower level of the architecture.

Physical Independence: The ability to change the physical schema without changing the logical schema is called physical data independence. For example, a change to the internal schema, such as using different file organization or storage structures, storage devices, or indexing strategy, should be possible without having to change the conceptual or external schemas.

The logical scheme stays unchanged even though the storage space or type of some data is changed for reasons of optimization or reorganization. Physical data independence deals with hiding the details of the storage structure from user applications. The application should not be involved with these issues, since there is no difference in the operation carried out against the data.

Logical Independence: The ability to change the logical (conceptual) schema without changing the External schema (User View) is called logical data independence. For example, the addition or removal of new entities, attributes, or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.

The data independence and operation independence together gives the feature of data abstraction

2.3 STRUCTURE OF DBMS

The overall structure of DBMS looks as shown in the Figure 2.2.

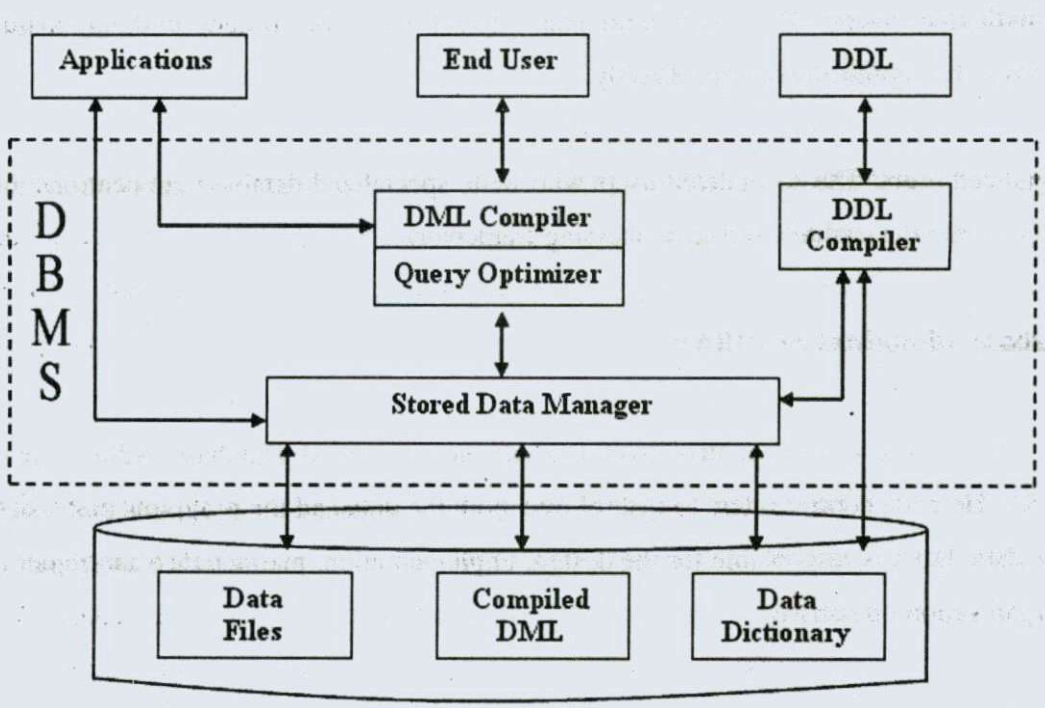


Figure 2.2: The overall structure of DBMS

2.4 PEOPLE WHO DEAL WITH DATABASES

People who deal with databases are called database users, differentiated by the way they interact with the system. There are four types of database users.

They are Naive users, Applications programmers, Sophisticated users, and Specialized users.

Naive users: The unsophisticated user who interact with the system by invoking the application programs that have been written previously.

Applications programmers: They are the computer professionals who write application programs.

Sophisticated users: The users who can interact with the system without writing programs, by submitting queries directly.

Specialized users: The specialized users who write specialized database applications that do not fit into the traditional data-processing framework.

Database Administrator (DBA):

A person having central control over the database is called Database Administrator (DBA). Here the control refers to control over both the data and the programs that access those data. DBA is responsible for the design, implementation, maintenance and repair of an organization's database.

The DBA has the following roles:

1. Granting of authorization for data access:
2. Maintaining database and ensuring its availability to users
3. Schema definition:
4. Storage structure and access-method definition:
5. Schema and physical-organization modifications
6. Controlling privileges and permissions to database users
7. Monitoring database performance
8. Transferring Data
9. Replicating Data
10. Database backup and recovery
11. Database security

DBA coordinates all the activities of the database system and has a good understanding of the enterprise's information resources and needs.

2.5 DBMS ARCHITECTURE

All DBMS do not conform to the same architecture.

- The three-level architecture forms the basis of modern database architectures.
- This is in agreement with the ANSI/SPARC study group on Database Management Systems.
- ANSI/SPARC is the American National Standards Institute/Standard Planning and Requirement Committee).
- The architecture for DBMSs is divided into three general levels:
- External
- Conceptual
- Internal

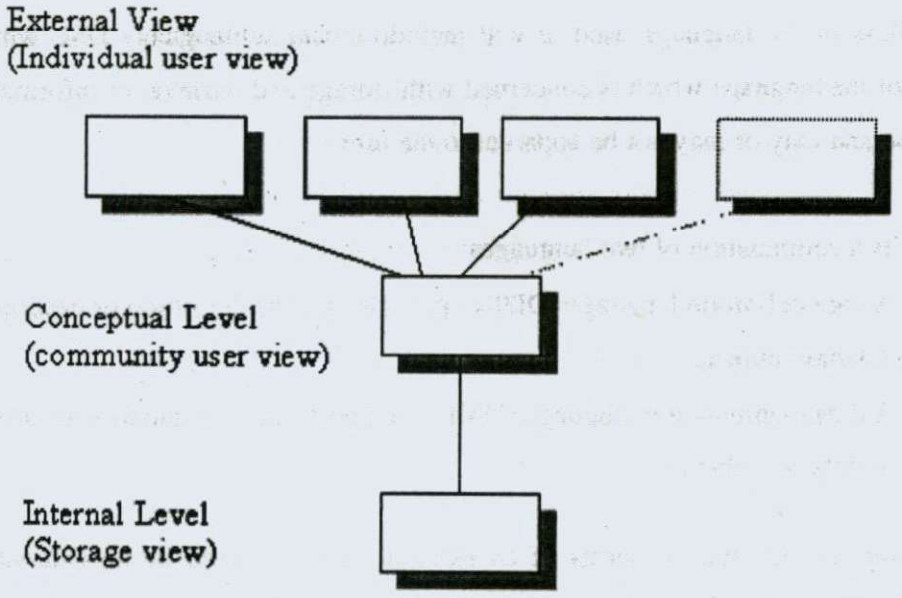


Figure 2.3: Three level architecture

1. The external level : concerned with the way individual users see the data
2. The conceptual level: can be regarded as a community user view a formal description of data of interest to the organization, independent of any storage considerations.
3. The internal level : concerned with the way in which the data is actually stored

External View

A user is anyone who needs to access some portion of the data. They may range from application programmers to casual users with ad-hoc queries. Each user has a language at his/her disposal.

The application programmer may use a high level language (e.g. COBOL) while the casual user will probably use a query language.

Regardless of the language used, it will include a data sublanguage DSL which is that subset of the language which is concerned with storage and retrieval of information in the database and may or may not be apparent to the user.

A DSL is a combination of two languages:

- A data definition language (DDL) - provides for the definition or description of database objects
- A data manipulation language (DML) - supports the manipulation or processing of database objects.

Each user sees the data in terms of an external view: Defined by an external schema, consisting basically of descriptions of each of the various types of external record in that external view, and also a definition of the mapping between the external schema and the underlying conceptual schema.

Conceptual View

- An abstract representation of the entire information content of the database.
- It is in general a view of the data as it actually is, that is, it is a `model' of the `realworld'.
- It consists of multiple occurrences of multiple types of conceptual record, defined in the conceptual schema.
- To achieve data independence, the definitions of conceptual records must involve information content only.
- Storage structure is ignored
- Access strategy is ignored
- In addition to definitions, the conceptual schema contains authorization and validation procedures.

Internal View

The internal view is a low-level representation of the entire database consisting of multiple occurrences of multiple types of internal (stored) records. It is however at one remove from the physical level since it does not deal in terms of physical records or blocks nor with any device specific constraints such as cylinder or track sizes. Details of mapping to physical storage are highly implementation specific and are not expressed in the three-level architecture.

The internal view described by the internal schema:

- Defines the various types of stored record
- What indices exist
- How stored fields are represented
- What physical sequence the stored records are in

In effect the internal schema is the storage structure definition.

2.6 SUMMARY

In this unit, we have discussed about data independence, structure of DBMS. We also described various people who deal with DBMS. At the end we have given architecture of DBMS.

2.7 KEYWORDS

Data independence: It refers to the immunity of user applications to make changes in the definition and organization of data.

Physical Independence: The ability to change the physical schema without changing the logical schema is called physical data independence

Logical Independence: The ability to change the logical (conceptual) schema without changing the External schema (User View) is called logical data independence.

Database Administrator (DBA): A person having central control over the database.

2.8 UNIT-END EXERCISES AND ANSWERS

1. What is data independence? Explain different types data independence.
2. Give overall structure of DBMS.
3. Who are people, who deal with DBMS?
4. What are the roles of DBA?
5. Explain the three-level architecture of DBMS.

Answers: SEE

1. 2.2
2. 2.3
3. 2.4
4. 2.4
5. 2.5

2.8 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta
- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan
- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke

UNIT 3: CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

Structure:

3.0 OBJECTIVES

At the end of this unit you will be able to

- Discuss the Data models for database design
- Elucidate An example database application
- Explain Some related terms
- Differentiate the Relationships, Roles and Structural constraints

3.1 INTRODUCTION

This unit is about data models and some related terminologies. Data modeling in software engineering is the process of creating a data model by applying formal data model descriptions using data modeling techniques. Data modeling is a method used to define and analyze data requirements needed to support the business processes of an organization. The data requirements are recorded as a conceptual data model with associated data definitions.

3.2 DATA MODELS

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. It is concept underlying the structure of a database. All the data models that have been proposed can be grouped into three groups, namely: object-based logical models, record-based logical data models and physical data models

Use data modeling:

- to manage data as a resource;
- for the integration of information systems;
- for designing databases/data warehouses (aka data repositories)

Object-Based Logical Models:

Object-based logical models are used in describing data at the conceptual and view levels. They are characterized by the facts that they provide fairly flexible structuring capabilities and allow data consistency to be specified explicitly. A more widely used object-based model is the entity-relationship model. The entity-relationship model is based on a perception of a real world.

Record-Based Logical Models:

Record-based logical models are also used in describing data at the conceptual and view levels. In the record-based logical models, the database is structured in fixed format records of several types. In the record-based logical models, the three most widely accepted models are the relational, network and hierarchical models.

Physical Data Models:

The physical data models are used to describe data at the lowest level. Among a very few physical data models, unifying and frame memory are widely known. Physical data models represent the design of data while also taking into account both the constraints

and facilities of a particular database management system. Generally, it is taken from a logical data model. Although it can also be engineered in reverse from a particular database implementation.

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

- Specification of all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS.

3.3 AN EXAMPLE DATABASE APPLICATION

A computer database application program is a computer program that interacts with the database by issuing an appropriate request to the DBMS. Users interact with the database through a number of application programs, written in some programming language.

Examples of Database Applications:

There are many database applications.

- Purchase from the supermarket
- Purchase using your credit card
- Booking a holiday at the travel agents
- Using the local library,
- Property management
- Banking transaction

We describe COMPANY as an example database application. The Company database keeps track of a company's employees, departments and projects.

1. The company is organised into departments. Each department has a unique name, a unique number and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
2. A department controls a number of projects, each of which has a unique name, a unique number and a single location.
3. We store each employee's name, Social Security number, address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date and relationship to the employee.

Company DB: initial conceptual design

We can identify four entity types:

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager and ManagerStartDate. Locations is a multi-valued attribute. Both Name and Number are (separate) key attributes.
2. An entity type PROJECT with attributes Name, Number, Location and ControllingDepartment. Both Name and Number are (separate) key attributes.
3. An entity type EMPLOYEE with attributes Name, SSN, Sex, Address, Salary, BirthDate, Department and Supervisor. It is not clear from the requirements if Name and Address should be composite or simple attributes.
4. An entity type DEPENDENT with attributes Employee, DependentName, Sex, BirthDate and Relationship. Omissions

We have not yet represented the fact that an employee can work on several projects, or the number of hours per week that an employee works on each project. This can be represented as either

- A multi-valued composite attribute of EMPLOYEE called WorksOn (with components Project, Hours) or
- A multi-valued composite attribute of PROJECT called Workers (with components Employee, Hours).

Preliminary design of entity types

DEPARTMENT
Name, Number, {Locations}, Manager, ManagerStartDate

PROJECT
Name, Number, Location, ControllingDepartment

EMPLOYEE
Name (Fname, MInit, Lname), SSN, Sex, Address, Salary,
BirthDate, Department, Supervisor, {WorksOn (Project,
Hours)}

DEPENDENT
Employee, DependentName, Sex, BirthDate, Relationship

Company DB: refined conceptual design
We can identify six relationship types:

1. MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. After questioning the users, we establish that DEPARTMENT participation is total. The relationship type has an attribute StartDate.

2. WORKS_FOR, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
3. CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total. After questioning the users, we establish that DEPARTMENT participation is partial.
4. SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). After questioning the users, both participations are determined to be partial.
5. WORKS_ON, defined, after consultation with the users, as an M:N relationship type between PROJECT and EMPLOYEE with attribute Hours. Both participations are determined to be total.

3.4 SOME RELATED TERMS

Now we discuss some related terms.

An attribute: An attribute is a property or characteristic of an object.

An entity: An entity is an object that exists and is distinguishable from other objects by a specific set of attributes. The existence need not be a material existence.

Entity types: An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Entity set: An entity set is a set of entities of the same type.

Key: A key is a set attributes that is used to identify records.

Let us consider the following employee-table and department-table as a sample tables for the purpose of explaining different key concepts.

Employee table with the attributes (columns):

1 Employee-id
2 Employee-name
3 Employee-address
4 Employee-department-id
5 Employee-salary
6 Employee-social-security-number

Department table with the attributes:

1. Department-id
2. Department-name
3. Department-location

Now we define various keys.

Super Key: A set of attributes (columns) that is used to identify the records (rows) in a table uniquely is known as Super Key. A table can have many Super Keys

From the above employee table, we can have many super keys. For example:

Employee-id
Employee-id, Employee-name
Employee-id, Employee-name, Employee-address
Employee-id, Employee-department-id
Employee-social-security-number
Employee-social-security-number, Employee-name

All the above are super keys and many more such combinations are possible.

Candidate Key: It can be defined as minimal Super Key or irreducible Super Key. In other words a set attributes that identifies the records uniquely but none of its proper subsets can identify the records uniquely.

From the above employee table, we can have the following candidate keys. For example:

Employee-id
Employee-social-security-number

Note that all the candidate keys are also super keys, but all super keys need not be necessarily candidate keys.

Primary Key: A Candidate Key that is used by the database designer for unique identification of each row in a table is known as Primary Key. A Primary Key can consist of one or more attributes of a table

From the above employee table, we can select any one candidate keys as primary key. For example: either Employee-id or Employee-social-security-number

Foreign Key: A foreign key is a set of attributes in one base table that points to the candidate key (generally it is the primary key) of another table. The purpose of the foreign key is to ensure referential integrity of the data

For example, in the above employee table, Employee-department-id is a foreign key.

Composite Key: If we use multiple attributes to create a Primary Key then that Primary Key is called Composite Key (also called a Compound Key or Concatenated Key).

For example, in the above employee table, if we can assume that Employee-name and Employee-address can uniquely identify the Employees in an organization, then (Employee-name, Employee-address) can be used as a primary key, which is a composite key.

Alternate Key: Alternate Key can be any of the Candidate Keys except for the Primary Key.

In the above Employee table, if we choose Employee-name as the primary key, then Employee-social-security-number is an alternate key.

Secondary Key: The attributes that are not even the Super Key but can be still used for identification of records (not unique) are known as Secondary Key. A secondary key is made on a field that you would like to be indexed for faster searches. We can have multiple Secondary Keys per table. The Attributes used for Secondary key are not the ones used for Super Key i.e. Secondary Key is not even be one of the Super Key.

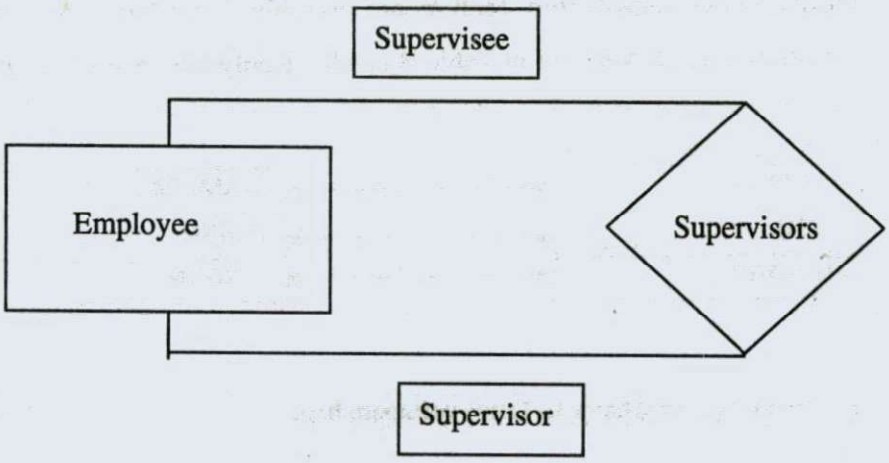
In the above Employee table, we can choose Employee-department-id as a secondary key. We can also choose Employee-address as another secondary key.

3.5 RELATIONSHIPS, ROLES AND STRUCTURAL CONSTRAINTS

A relationship: A relationship is an association (combination) among the instance of one or more entity type.

Role Names: Each entity type in a relationship plays a particular role. The role name specifies the role that a participating entity type plays in the relationship and explains what the relationship means. For example, in the relationship between Employee and Department, the Employee entity type plays the employee role, and the Department entity type plays the department role. In most cases the role names do not have to be specified, but in cases where the same entity participates more than once in a relationship type in different roles, then we have to specify the role.

For example, in the Company schema, each employee has a supervisor. We need to include the relationship "Supervises". However a supervisor is also an employee. Therefore the employee entity type participates twice in the relationship, once as an employee and once as a supervisor. Therefore we can specify two roles, employee and supervisor.



Constraints on Relationship Types:

Relationship types have certain constraints that limit the possible combination of entities that may participate in relationship. An example of a constraint is that if we have the entities Doctor and Patient, the organization may have a rule that a patient cannot be seen by more than one doctor. This constraint needs to be described in the schema.
There are two main types of relationship constraints, cardinality ratio, and participation.

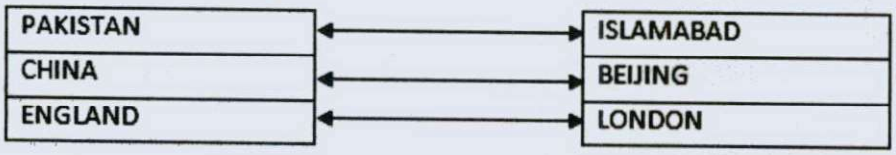
Relationship types:

Types of Relationships in RDBMS There are three types of relationships:

1. One to One relationship
2. One to Many (or Many to One) relationship
3. Many to Many relationship.

One to One relationship:

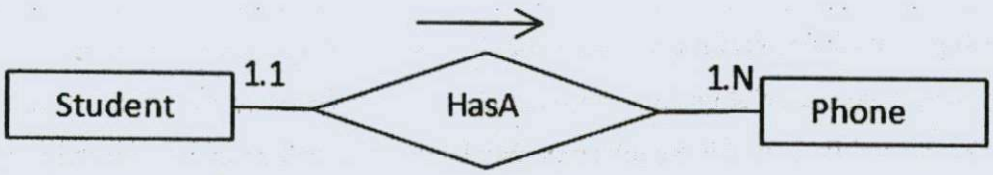
In a one-to-one relationship, each record in Table A can have only one matching record in Table B, and each record in Table B can have only one matching record in Table A



One to Many (or Many to One) relationship:

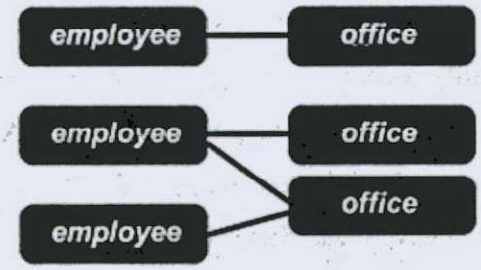
In a one-to-many relationship, a record in Table A can have many matching records in Table B, but a record in Table B has only one matching record in Table A

Consider the following relationship between *Student* and *Phone* entity. According to the relationship a student can have any number of phone numbers.



Many to Many relationship:

In a many-to-many relationship, a record in Table A can have many matching records in Table B, and a record in Table B can have many matching records in Table A



3.6 SUMMARY

Data modeling in software engineering is the process of creating a data model by applying formal data model descriptions using data modeling techniques. This unit introduced data models and some related terminologies. The data requirements are recorded as a conceptual data model with associated data definitions. We also defined relationships, roles and structural constraints in this unit.

3.7 KEYWORDS

Data model: A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

An attribute: An attribute is a property or characteristic of an object.

An entity: An entity is an object that exists and is distinguishable from other objects by a specific set of attributes. The existence need not be a material existence.

Entity types: An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Entity set: An entity set is a set of entities of the same type.

Key: A key is a set attributes that is used to identify records.

Super Key: A set of attributes (columns) that is used to identify the records (rows) in a table uniquely is known as Super Key. A table can have many Super Keys

Candidate Key: It can be defined as minimal Super Key or irreducible Super Key. In other words a set attributes that identifies the records uniquely but none of its proper subsets can identify the records uniquely

Primary Key: A Candidate Key that is used by the database designer for unique identification of each row in a table is known as Primary Key. A Primary Key can consist of one or more attributes of a table

Foreign Key: A foreign key is a set of attributes in one base table that points to the candidate key (generally it is the primary key) of another table. The purpose of the foreign key is to ensure referential integrity of the data

Relationship: A relationship is an association (combination) among the instance of one or more entity type.

3.8 UNIT-END EXERCISES AND ANSWERS

1. What is a data model? Explain different types of data models.
2. Describe an example database application.
3. Define Superkey, Candidate key, Primary key, and Foreign key.
4. With diagram, explain constraints on relationship types.

Answers: SEE

1. 3.2
2. 3.3
3. 3.4
4. 3.5

3.9 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta
- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan
- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke
- An Introduction to Database Systems
  C.J.Date

UNIT 4: ENTITY RELATIONSHIP MODEL

Structure:

4.0 OBJECTIVES

At the end of this unit you will be able to understand:

- ER Data model
- Design Issues with ER model
- Problems with ER data modeling

4.1 INTRODUCTION

This unit describes Entity-Relationship (ER) data model, which is used to describe data at the conceptual and view level. It is based on the perception of the real world. It also deals with design issues and problems with ER model.

4.2 ENTITY-RELATIONSHIP MODEL

The Entity-Relationship(ER) data model is used to describe data at the conceptual and view level. It is based on the perception of the real world. It perceives the real world as consisting of basic objects called entities, and relationship among these objects. It was developed to facilitate database design. The ER data model is very useful in mapping the meanings and interactions of real-world enterprises onto enterprises onto a conceptual schema. This represents the overall logical structure of the database.

The building blocks: entities, relationships, and attributes

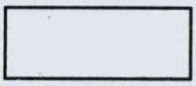
The ER data model employs the notion of attributes, entity sets and relationship sets.
An entity is an object that exists and is distinguishable from other objects. For instance, Rama with UID 890-12-3456 is an entity, as he can be uniquely identified as one particular person in India. An entity may be concrete (a person or a book, for example) or abstract (like a holiday or a concept).

An entity set is a set of entities of the same type (e.g., all persons having an account at a bank). Entity sets need not be disjoint. For example, the entity set employee (all employees of a bank) and the entity set customer (all customers of the bank) may have members in common.

An entity is represented by a set of attributes. e.g. name, S.I.N., street, city for ``customer" entity. The domain of the attribute is the set of permitted values (e.g. the telephone number must be seven positive integers). Formally, an attribute is a function which maps an entity set into a domain. Every entity is described by a set of (attribute, data value) pairs. There is one pair for each attribute of the entity set. e.g. a particular customer entity is described by the set {(name, Harris), (S.I.N., 890-123-456), (street, North), (city, Georgetown)}.

Symbol Used for representing the above concepts:

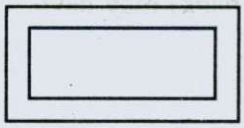
Attributes are represented by ellipses



Entity sets are represented by rectangles

Relationships among entities are represented by diamonds

Lines are used to link attributes to



Weak entity set

Primary key

Some example ER diagrams: The Figure 4.1 describes an ER diagram for a bank. The Figure 4.2 describes an ER diagram for a book shop. Where as Figure gives an ER diagram for a transport system.

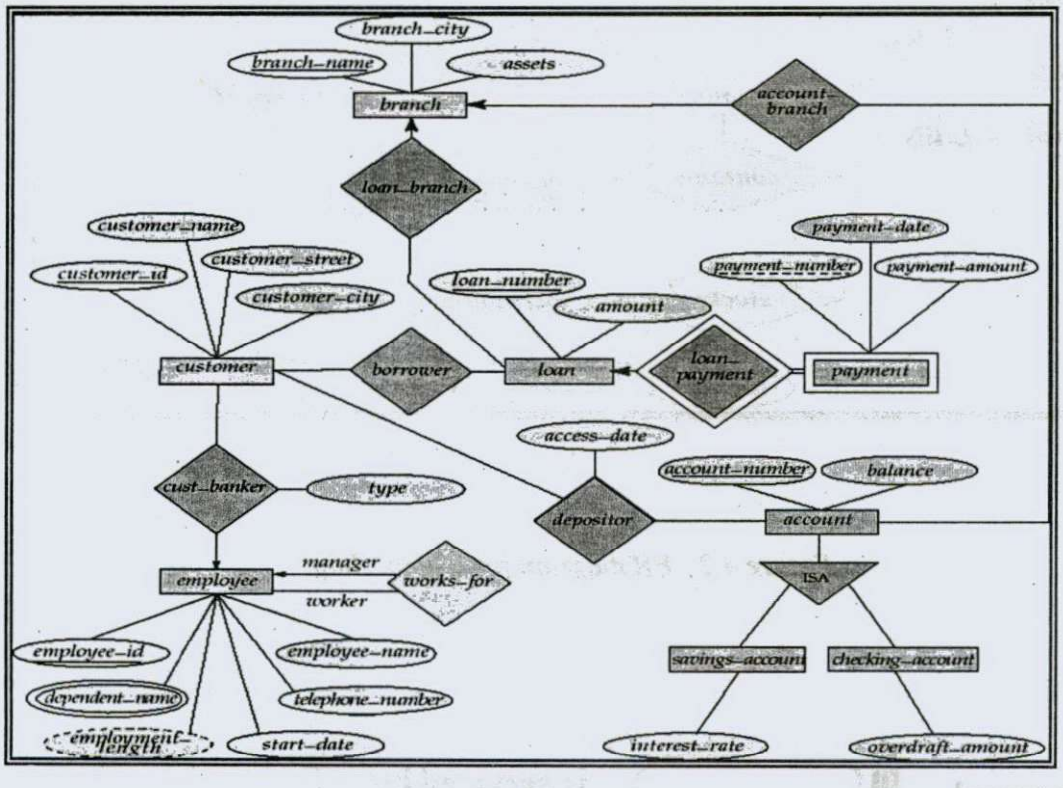


Figure 4.1: ER diagram for bank:

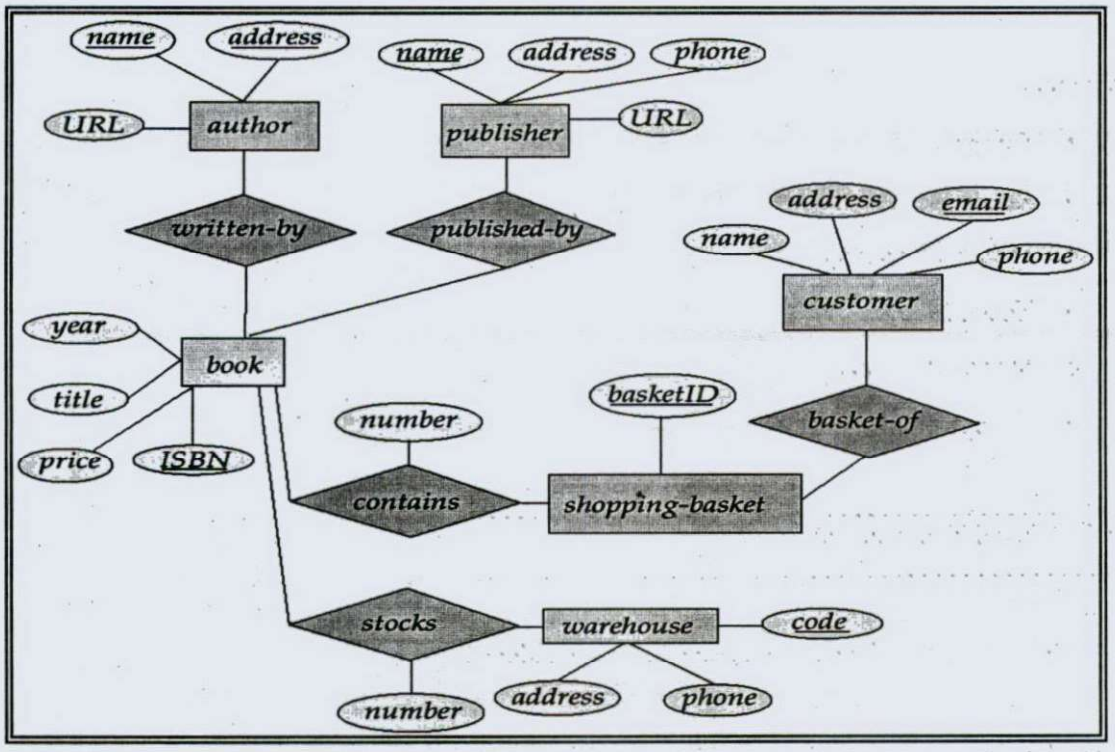


Figure 4.2: ER diagram for a book shop:

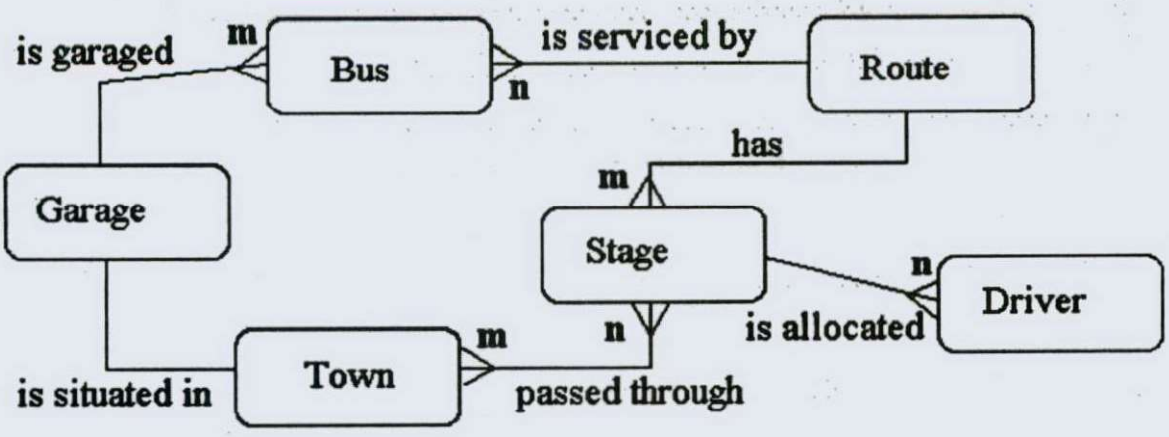


Figure 4.3: ER diagram for a transport system:

Attributes of the transport system:

- Bus (reg-no,make,size,deck,no-pass)
- Route (route-no,avg-pass)
- Driver (emp-no,name,address,tel-no)
- Town (name)
- Stage (stage-no)
- Garage (name,address)

4.3 DESIGN ISSUES

The followings are the design issues to be considered:

Use of entity sets vs. attributes: Choice mainly depends on the structure of the enterprise being modeled, and on the semantics associated with the attribute in question.

Use of entity sets vs. relationship sets: Possible guideline is to designate a relationship set to describe an action that occurs between entities.

Binary versus n -ary relationship sets: Although it is possible to replace any nonbinary (n -ary, for n > 2) relationship set by a number of distinct binary relationship sets, a n -ary relationship set shows more clearly that several entities participate in a single relationship.

Placement of relationship attributes

4.4 PROBLEMS ON ER MODELING

There are several problems that may arise when designing a conceptual data model. These are known as connection traps.

There are two main types of connection traps:

1. Fan traps
2. Chasm traps

Fan traps:

A fan trap occurs when a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous. It occurs when 1:m relationships fan out from a single entity.

Figure 4.4: Fan Trap

A single site contains many departments and employs many staff. However, which staff work in a particular department?

The fan trap is resolved by restructuring the original ER model to represent the correct association.

Figure 4.5: Resolved Fan Trap

Chasm traps:

A chasm trap occurs when a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences. It occurs where there is a relationship with partial participation, which forms part of the pathway between entities that are related.

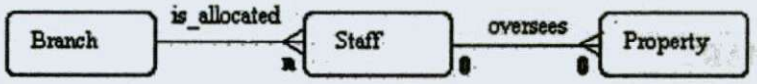


Figure 4.6: Chasm Trap

- A single branch is allocated many staff who oversee the management of properties for rent. Not all staff oversee property and not all property is managed by a member of staff.
- What properties are available at a branch?
- The partial participation of Staff and Property in the oversees relation means that some properties cannot be associated with a branch office through a member of staff.
- We need to add the missing relationship which is called `has' between the Branch and the Property entities.
- You need to therefore be careful when you remove relationships which you consider to be redundant.

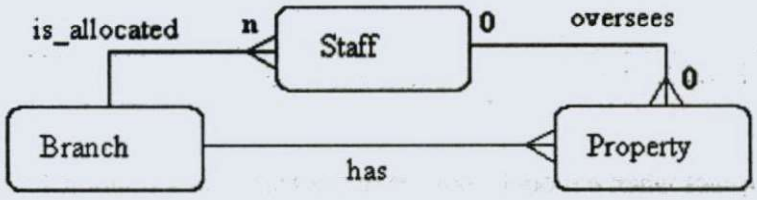


Figure 4.7: Resolved Chasm Trap

4.5 SUMMARY

In this unit, we have introduced Entity-Relationship (ER) data model, which is used to describe data at the conceptual and view level. We also discussed the design issues and problems with ER model. We studied some example ER diagrams to understand the conceptual database.

4.6 KEYWORDS

Entity-Relationship (ER) model: ER model is used to describe data at the conceptual, which is based on the perception of the real world.

Fan trap: A fan trap occurs when a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.

Chasm trap: A chasm trap occurs when a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences. It occurs where there is a relationship with partial participation, which forms part of the pathway between entities that are related.

4.7 UNIT-END EXERCISES AND ANSWERS

1. Explain various symbols used in ER diagrams.
2. What are design issues with ER diagram? Explain
3. What are problems on ER modeling? Explain
4. Construct an ER diagram for a bank.

Answers: SEE

1. 4.2
2. 4.3
3. 4.4
4. 4.2

4.8 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta
- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan
- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke
- An Introduction to Database Systems
  C.J.Date

UNIT 5: RELATIONAL MODEL CONCEPTS

Structure:

5.0 OBJECTIVES

At the end of this unit you will be able to :

- Explain Relational model
- Elucidate Relational constraints
- Discuss Relational Database schema
- State Update operations

5.1 INTRODUCTION

The Relational model was proposed by E.F.Codd in the year 1970. It is relatively new, compared to older models hierarchical model and network model. The relational model has become the most common commercial data model. Relations are actually the mathematical table. This unit introduces relational model concepts.

5.2 RELATIONAL MODEL

The relational database model is the most popular data model. It is very simple and easily understandable by information systems professionals and end users. Understanding a relational model is very simple since it is very similar to Entity Relationship Model. In ER model data is represented as entities similarly here data in represented in the form of relations that are depicted by use of two-dimensional tables. Also attributes are represented as columns of the table.

The basic concept in the relational model is that of a relation. In simple language, a relation is a two-dimensional table. Table can be used to represent some entity information or some relationship between them. Even the table for an entity information and table for relationship information are similar in form. Only from the type of information given in the table can tell if the table is for entity or relationship. The entities and relationships, which we studied in the ER model, are similar to relations in this model. In relational model, tables represent all the entities and relationships identified in ER model. Rows in the table represent records; and columns show the attributes of the entity. Figure 5.1 shows structure of a relation in a relational model.

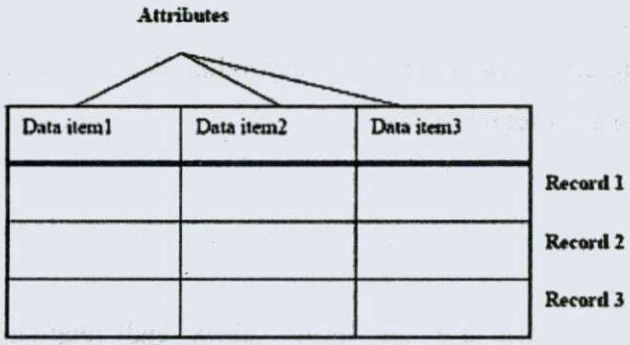


Figure 5.1 Structure of a relation in a relational model.

Structural Part of Relation

We start with some terminology first.

Relation: a table with columns and rows.

Attribute /field: named column

Tuple: row of relation (a record)

Degree: number of attributes in a relation. If the column/attribute is only one then relation is of degree one also known as unary. If a relation has two columns then it's known as degree 2 also known as binary.

Cardinality: number of rows (tuples) in a relation.

Domain: data type .Set of allowable values for one or more attribute.

Relational Database: collection of normalized relations.

Basic Structure:

A relation is basically a subset of all possible rows of their corresponding domains. In a general a relation of n attributes must be a subset of:

$D_1 \times D_2 \times \ldots. \times D_{n-1} \times D_n$

Tables are essentially relations. So, we can use the terms tables and relations interchangeably.

Properties of Relation:

- Each table should have a distinct name.
- Each cell should have one value (atomic value).
- Attribute names should be unique.
- Values of an attribute are from the same domain.
- Order of attribute has no significance.
- Each tuple should be distinct. It means no duplicate rows.
- Order of rows has no significance.

5.3 RELATIONAL CONSTRAINTS

In Relational model, there are various types of constraints. They are explained below.

Key constraints

Key constraint is implied by the existence of candidate keys. The intension includes a specification of the attribute(s) consisting of the primary key and specification of the attribute(s) consisting alternate keys, if any. Each of these specifications implies a uniqueness constraint (by definition of candidate key). In addition, primary key specification implies a no-nulls constraint (by integrity rule).

Referential constraints

Referential constraints are constraints implied by the existence of foreign keys. The intension includes a specification of all foreign keys in the relation. Each of these specifications implies a referential constraint (by integrity rule).

Other constraints: Many other constraints are possible in theory.

Examples:

1) salary >= 20000
2) age > 20

Participation constraints

If every entity of an entity set is related to some other entity set via a relationship type, then the participation of the first entity type is total. If only few member of an entity type is related to some entity type via a relationship type, the participation is partial.

5.4 RELATIONAL DATABASE SCHEMA

A database schema of a database system is its structure described in a formal language supported by the database management system (DBMS) and refers to the organization of data to create a blueprint of how a database will be constructed (divided into database tables). A relational database schema is the tables, columns and relationships that make up a relational database.

Schemata are generally stored in a data dictionary. Although a schema is defined in text database language, the term is often used to refer to a graphical depiction of the database

structure. In other words, schema is the structure of the database that defines the objects in the database.

The Purpose of a Schema: A relational database schema helps us to organize and understand the structure of a database. This is particularly useful when designing a new database, modifying an existing database to support more functionality, or building integration between databases.

The Figure 5.2 shows the relational schema for a teaching department with three entities, namely teacher, student, and course.

There are two intersection entities in this schema: Student and Course and Teacher and Course. These handle the two many-to-many relationships: 1) between Student and Course, and 2) between Teacher and Course. In the first case, a Student may take many Courses and a Course may be taken by many Students. Similarly, in the second case, a Teachers may teach many Courses and a Course may be taught by many Teachers.

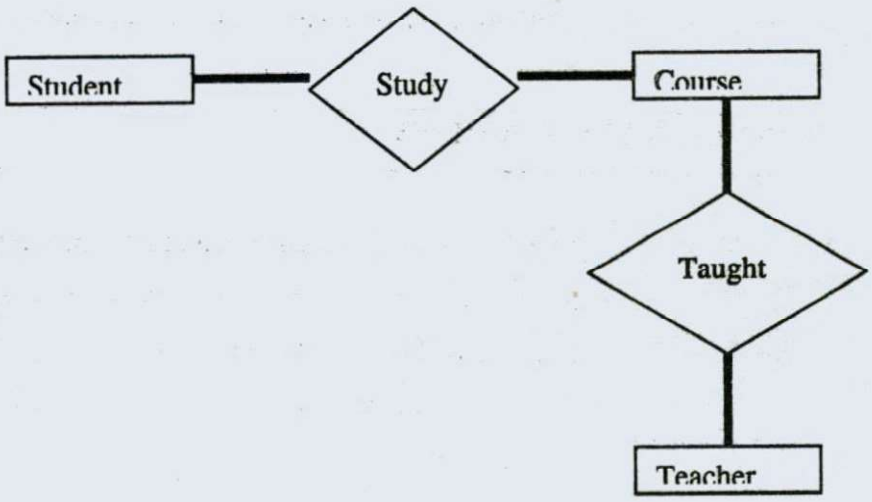


Figure 5.2: An example Relational database scheme

5.5 UPDATE OPERATIONS & DEALING WITH CONSTRAINTS VIOLATIONS

The basic update operations of the relational model are: Insert, Delete, and Update (or Modify). All integrity constraints specified on the database schema should not be violated by the update operations. Several update operations may have to be grouped together. Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints.

In case of integrity violation, several actions can be taken:

1. Cancel the operation that causes the violation (RESTRICT or REJECT option)
2. Perform the operation but inform the user of the violation
3. Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
4. Execute a user-specified error-correction routine

INSERT may violate any of the constraints:

1. Domain constraint: if one of the attribute values provided for the new tuple is not of the specified attribute domain.
2. Key constraint: if the value of a key attribute in the new tuple already exists in another tuple in the relation.
3. Referential integrity: if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation.
4. Entity integrity: if the primary key value is null in the new tuple.

DELETE may violate only referential integrity:

If the primary key value of the tuple being deleted is referenced from other tuples in the database: Can be remedied by several actions: RESTRICT, CASCADE, SET NULL.

1. RESTRICT option: reject the deletion.
2. CASCADE option: propagate deletion by deleting tuples that reference the tuple that is deleted.
3. SET NULL option: set the foreign keys of the referencing tuples to NULL.

One of the above options must be specified during database design for each foreign key constraint.

UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified.

Any of the other constraints may also be violated, depending on the attribute being updated:

1. Updating the primary key (PK): Similar to a DELETE followed by an INSERT. Need to specify similar options to DELETE.
2. Updating a foreign key (FK): May violate referential integrity.
3. Updating an ordinary attribute (neither PK nor FK): Can only violate domain constraints.

5.6 SUMMARY

The Relational model was proposed by E.F.Codd in the year 1970. It is relatively new, compared to older models hierarchical model and network model. The relational model has become the most common commercial data model. Relations are actually the mathematical table. This unit introduces has relational model concepts, relational constraints, relational database schema, and update operations.

5.7 KEYWORDS

A relation: A relations is actually a two-dimensional mathematical table.

The relational model: The relational model is a data model, which uses relations (tables) to represent both entity information and relationship between them.

Attribute: named column.

Tuple: row of relation (a record)

Degree: number of attributes in a relation.

Cardinality: number of rows (tuples) in a relation.

Domain: data type, set of allowable values for one or more attribute.

Relational Database: collection of normalized relations.

5.8 UNIT-END EXERCISES AND ANSWERS

1. Briefly explain the concepts of relational model
2. Discuss different types of relational constraints
3. Explain relational database schema
4. Discuss the constraints violations with update operations

Answers: SEE

1. 5.2
2. 5.3
3. 5.4
4. 5.5

5.9 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta

- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan

- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke

- An Introduction to Database Systems
  C.J.Date

UNIT 6: RELATIONAL ALGEBRA

Structure:

6.0 OBJECTIVES

At the end of this unit you will be able to:

- Explain Theory of Relational Algebra
- Elucidate Fundamental Relational Operations
- Discuss Additional Relational Operations
- State Expressing Queries in Relational Algebra

6.1 INTRODUCTION

The Relational Algebra: A relational algebra is an algebra defined over relations. The inputs are relations and the output is also a relation. Similar to normal algebra, except we use relations as values instead of numbers, and the operations and operators are different.

It is not used as a query language in actual DBMS. The relational algebra is a procedural query language.

6.2 FUNDAMENTAL RELATIONAL OPERATIONS

An operation is fundamental if it cannot be expressed with other operations. The relational algebra has six fundamental operations:

- Select (unary)
- Project (unary)
- Rename (unary)
- Cartesian product (binary)
- Union (binary)
- Set-difference (binary)

The Relational Operations produce a new relation as a result.

Let us consider the following relations (tables) as an example of a banking database in our further discussions:

customer relation over the Customer_Scheme

deposit relation over the Deposit_Scheme

borrow relation over the Borrow_Scheme

branch_scheme over the Branch_Scheme

client_realtion over the Client_Scheme

customer table:

Customer_name	Custermer_House_number	Custermer_Locality	Custermer_City	Custermer_State
Shiva	23	Indira Nagar	Bangalore	Karnataka
Kaveri	45	Saraswathi Puram	Mysore	Karnakata
Laloo	13	Gandhi Nagar	Patna	Bihar
Kumar	56	Balaji Nagar	Tirupati	Andra Pradesh
Khan	67	Treemurthi	Delhi	Dehli
Parvathi	23	Indira Nagar	Bangalore	Karnataka
Asha	47	Netaji Nagar	Mysore	Karnataka
Ganesh	79	Thana	Mumbai	Maharastra

deposit table:

Account_Number	Customer_Name	Branch_Name	Account_Balance
45	Kaveri	University	10000
56	Laloo	Redfort	1000000
78	Kumar	Majestic	20000
20	Shiva	Palace	30000
20	Parvathi	Palace	30000
21	Asha	Majestic	22000

borrow table:

Loan_Number	Customer_Name	Branch_Name	Loan_Amount
10	Laloo	Saltlake	20000
13	Kumar	University	30000
13	Asha	University	30000
75	Khan	Redfort	15000
55	Laloo	Majestic	25000
66	Khan	University	40000
18	Ganesh	University	65000
11	Kuamr	Majestic	13000
61	Laloo	Redfort	22000

branch table:

Branch_Name	Branch_City	Branch_Locality	Assets
Saltlake	Kolkata	Subhas Nagar	12000000
Redfort	Delhi	Puranadelhi	30000000
University	Mysore	Manasagangotri	1150000
Majestic	Bangalore	Gandhi Nagar	450000000
Palace	Mysore	Shivarampet	35000000

client table:

Customer_Name	Banker_Name
Kumar	Shiva
Asha	Shiva
Laloo	Kaveri
Khan	Shiva
Asha	Rama
Laloo	Rama

Select Operation (σ):

Select is denoted by a lowercase Greek sigma (σ), with the predicate appearing as a subscript. The argument relation (table) is given in parentheses following the σ. **Select** operation selects tuples (rows) that satisfy a given predicate from the input table.

For example, let us consider the query:

"select tuples (rows) of the *borrow* relation where the branch is **MAJESTIC**"

We can express it as:

$$\sigma_{\text{Branch\_name} = \text{"MAJESTIC"}} (\text{borrow})$$

The result of this operation consists of only one tuple as given:

Loan_Number	Customer_Name	Branch_Name	Loan_Amount
55	Laloo	Majestic	25000

We can allow comparisons using $=, \neq, <, \leq, >$ and $\geq$ in the selection predicate.

We can also allow the logical connectives $\vee$(or) and $\wedge$(and). For example:

$$\sigma_{\text{Branch\_name} = \text{"University"} \wedge \text{Amount} > 30000} (\text{borrow})$$

The result of this operation is:

Loan_Number	Customer_Name	Branch_Name	Loan_Amount
66	Khan	University	40000
18	Ganesh	University	65000

The Project Operation (Π):

Projection is denoted by the Greek capital letter pi (Π). Project outputs its argument relation for the specified attributes only. The attributes to be output appear as subscripts. Since a relation is a set, duplicate rows are eliminated.

For example, let us express the query:

"find the names of branches and names of customers who have taken loans"

We can express it as:

$$\Pi_{\text{Branch\_Name, Customer\_Name}} (\text{borrow})$$

The output of this operation is:

Branch_Name	Customer_Name
Saltlake	Laloo
University	Kumar
University	Asha
Redfort	Khan
Majestic	Laloo
University	Khan
University	Ganesh

We can also combine select and project operations as shown below:

$$\Pi_{\text{Customer\_Name}} (\sigma_{\text{Branch\_name} = \text{“MAJESTIC”}} (\text{borrow}))$$

This results in the displaying of customer names who have taken loans from MAJESTIC branch.

The output of this operation is:

Customer_Name
Laloo

We can think of select as taking rows of a relation, and project as taking columns of a relation.

The Cartesian Product Operation (×):

The cartesian product of two relations is denoted by a cross (×). It is a binary operation and requires two argument relations as input. For two input relations r_1 and r_2, it can be written as

$$r_1 \times r_2$$

The result of $r_1 \times r_2$ is a new relation with a tuple for each possible pairing of tuples from r_1 and r_2. In order to avoid ambiguity, the attribute names have attached to them the name of the relation from which they came. If no ambiguity will result, we drop the relation name. If r_1 has n_1 tuples, and r_2 has n_2 tuples, then $r = r_1 \times r_2$ will have $n_1 n_2$ tuples. The resulting scheme is the concatenation of the schemes of r_1 and r_2, with relation names added as mentioned. To find the clients of banker Rama and the city in

which they live, we need information in both client and customer relations. We can get this by writing

$$\sigma_{\text{Banker\_Name= "Rama"}} (\text{client} \times \text{customer})$$

We want rows where client.Customer_Name = customer.Customer_Name. So we can write to get just these tuples

$$\sigma_{\text{client.Customer\_Name= customer.Customer\_Name}} (\sigma_{\text{Banker\_Name= "Rama"}} (\text{client} \times \text{customer}))$$

Finally, to get just the customer's name and city, we can use a projection as shown:

$$\Pi_{\text{client.Customer\_Name, Customer\_City}} (\sigma_{\text{client.Customer\_Name= customer.Customer\_Name}} (\sigma_{\text{Banker\_Name= "Rama"}} (\text{client} \times \text{customer})))$$

The final output of the above query is:

Customer_name	Custermer_City
Laloo	Patna
Asha	Mysore

The Rename Operation (ρ)

The rename operation solves the problems that occur with naming when performing the Cartesian product of a relation with itself.

Suppose we want to find the names of all the customers who live with "Shiva" in the same locality, same city and same state.

We can get the information of Shiva by writing

$$\Pi_{\text{Customer\_Locality, Customer\_City, Customer\_State}} (\sigma_{\text{Customer\_name ="Shiva"}} (\text{customer}))$$

To find other customers with the same information, we need to reference the customer relation again:

$$\sigma_P(\text{customer} \times \Pi_{\text{Customer\_Locality, Customer\_City, Customer\_State}} (\sigma_{\text{Customer\_name} = \text{“Shiva”}} (\text{customer})))$$

Where ***P*** is a selection predicate requiring Customer_Locality, Customer_City, and Customer_State values to be equal.

The problem is how do we distinguish between the two Locality values, two City values and two State values appearing in the Cartesian product, as both come from a customer relation. This leads to an ambiguity. The solution is use the rename operator, denoted by the Greek letter rho(ρ).

To rename a relation, the general express is:

$$\rho_x (r)$$

to get the relation **r** under the name of **x**.

By using this to rename one of the two customer relations we can remove the ambiguity.

$$\Pi_{\text{customer.Customer.Name}} (\sigma_{\text{cust2.Customer\_Locality} = \text{customer.Customer\_Locality} \wedge \text{cust2. Customer\_City} = \text{customer.Customer.city} \wedge \text{cust2. Customer\_State} = \text{customer.Customer.State}} (\text{customer} \times (\Pi_{\text{Customer\_Locality, Customer\_City, Customer\_State}} (\sigma_{\text{Customer\_Name} = \text{“Shiva”}} (\rho_{\text{cust2}} (\text{customer}))))))$$

The output of the above query is:

Customer_name
Shiva
Parvathi

The Union Operation (∪):

The Union operation is a binary operation. This operation is denoted ∪ as in set theory. It returns the union (set union) of two compatible relations (tables). For a union operation r∪s to be legal, we require that r and s must have the same number of attributes and of the same type. To find all customers of the MAJESTIC branch, we must find everyone who has a loan or an account or both at the branch.

We need both borrow and deposit relations for this. We can express it using union operator as below:

$$\Pi_{Customer\_Name}\ (\sigma_{Branch\_Name = \text{“MAJESTIC”}}\ borrow)$$

$$\cup$$

$$\Pi_{Customer\_Name}\ (\sigma_{Branch\_Name = \text{“MAJESTIC”}}\ depsoit)$$

The output of the above query is:

Customer_name
Kumar
Laloo

As in all set operations, duplicates if any are eliminated.

The Set Difference Operation (_):

The Set difference operation is a binary operation. It is denoted by the minus sign (_). It returns tuples that are in one relation (first relation), but not in another(second relation).

Thus $r - s$ results in a relation containing tuples that are in r but not in s.

To find customers of the MAJESTIC branch who have an account there but no loan, we express it as:

$$\Pi_{Customer\_Name}\ (\sigma_{Branch\_Name = \text{“MAJESTIC”}}\ depsoit\ )$$

$$-$$

$$\Pi_{Customer\_Name}\ (\sigma_{Branch\_Name = \text{“MAJESTIC”}}\ borrow\ )$$

The output of the above query is:

Customer_name
Asha

6.3 ADDITIONAL RELATIONAL OPERATIONS

Additional relational operations are those which are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries. The list of additional operations consists of set-intersection, natural join, division, and assignment operations.

The Set Intersection Operation:

Set intersection is denoted by ∩ and returns a relation that contains tuples that are in both of its argument relations.

It does not add any power as such.

The general format of this operation is:

$$r \cap s = r - (r - s)$$

To find all customers having both a loan and an account at the REDFORT branch, we can write it as:

$$\Pi_{Customer\_Name} \ (\sigma_{Branch\_Name = \text{“REDFORT”}} \ depsoit \ )$$

$$\cap$$

$$\Pi_{Customer\_Name} \ (\sigma_{Branch\_Name = \text{“REDFORT”}} \ borrow \ )$$

The output of the above query is:

Customer_name
Laloo

The Natural Join Operation:

Very often we want to simplify queries on a Cartesian product. For example, "to find all customers having a loan at the bank and the cities in which they live", we need borrow and customer relations:

The query expression is:

$$\Pi_{\text{borrow.Customer\_Name, customer.Customer\_City}} \_$$

$$(\sigma_{\text{borrow.Customer\_Name= customer\_Customer\_Nanme}} (\text{borrow} \times \text{customer}))$$

Our selection predicate obtains only those tuples pertaining to Customer_name. This type of operation is very common, so we have the natural join, denoted by a ⋈ sign. Natural join combines a Cartesian product and a selection into one operation. It performs a selection forcing equality on those attributes that appear in both relation schemes. Duplicates are removed as in all relation operations. To illustrate this, we can rewrite the above query as

$$\Pi_{\text{Customer\_Name, Customer\_City}} (\text{borrow} \bowtie \text{customer})$$

The resulting relation is:

Customer_name	Custermer_City
Laloo	Patna
Kumar	Tirupati
Khan	Delhi
Parvathi	Bangalore
Asha	Mysore
Ganesh	Mumbai

Now let us make a more formal definition of natural join. Consider R and S to be sets of attributes. We can denote attributes appearing in both relations by $R \cap S$. We can denote attributes in either or both relations by $R \cup S$. Consider two relations $r(R)$ and $s(S)$. The natural join of r and s, denoted by $r \bowtie s$ is a relation on scheme $R \cup S$. It is a

projection onto $R \cup S$ of a selection on $r \times s$ where the predicate requires $r.A = s.A$ for each attribute A in $R \cap S$. Formally,

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}(r \times s))$$

Where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

To find the assets and names of all branches which have depositors living in Bangalore, we need customer, deposit, and branch relations and we can express the query as:

$$\Pi_{\text{Branch\_Name, assets}} \left(\sigma_{\text{Customer\_city} = \text{"Bangalore"}} (\text{customer} \bowtie \text{deposit} \bowtie \text{branch}) \right)$$

Note that $\bowtie$ is associative.

To find all customers who have both an account and a loan at the MAJESTIC branch, we can express it as:

$$\Pi_{\text{Customer\_Name}} \left(\sigma_{\text{Branch\_Name} = \text{"MAJESTIC"}} (\text{borrow} \bowtie \text{deposit}) \right)$$

This is equivalent to the set intersection version. We can see that there can be several ways to write a query in the relational algebra. If two relations $r(R)$ and $s(S)$ have no attributes in common, then $R \cap S = \emptyset$, and $r \bowtie s = r \times s$.

The Division Operation:

Division operation, denoted $\div$ is suited to queries that include the phrase ``for all". Suppose we want to "find all the customers who have an account at all branches located in Mysore".

Strategy: think of it as three steps. We can obtain the names of all branches located in Mysore by:

$$r_1 = \Pi_{\text{Branch\_Name}} (\sigma_{\text{Branch\_City} = \text{"Mysore"}} (\text{branch}))$$

We can also find all Customer_Name, Branch_Name pairs for which the customer has an account by

$$r_2 = \Pi_{\text{Customer\_Name, Branch\_Name}} ((\text{deposit}))$$

Now we need to find all customers who appear in r_2 with every branch name in r_1.

The divide operation provides exactly those customers:

$$\Pi_{\text{Customer\_Name, Branch\_Name}} ((\text{deposit}))$$

$$\div$$

$$\Pi_{\text{Branch\_Name}} (\sigma_{\text{Branch\_City} = \text{"Mysore"}} (\text{branch}))$$

Which is simply $r_2 \div r_1$.

Now, we can formally define division operation as: Let $r(R)$ and $s(S)$ be relations. Let $S \subseteq R$. The relation $r \div s$ is a relation on scheme $R - S$. A tuple t is in $r \div s$ if for every tuple t_s in s there is a tuple t_r in r satisfying both of the following:

$$t_r [S] = t_s [S]$$

$$t_r [R - S] = t[R - S]$$

These conditions say that the $R - S$ portion of a tuple t is in $r \div s$ if and only if there are tuples with the $r - s$ portion and the S portion in r for every value of the S portion in relation S. The division operation can be defined in terms of the fundamental operations.

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - r)$$

The Assignment Operation

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable. The assignment operation, denoted by ←, works like assignment in a programming language. We could rewrite our division definition as

$$temp \leftarrow \Pi_{R-S}(r)$$
$$temp - \Pi_{R-S}((temp \times s) - r)$$

Note that no extra relation is added to the database, but the relation variable created can be used in subsequent expressions. Assignment to a permanent relation would constitute a modification to the database.

6.4 EXAMPLES OF QUERIES IN RELATIONAL ALGEBRA

Now we consider some example queries in relational algebra. For these queries, let us consider an employee relational database with relations:

lives(Persons_Name, House_Number, Street, Locality_Name, Person_City)
works(Person_Name, Company_Name, Salary)
located_in(Company_Name, Company_City)
manages(Person_name, Manager_Name)

1) "Find the Person_Name, Locality_Name of all employees who stay in Bombay city".

$$\Pi_{\text{Branch\_Name, Locality\_Name}} (\sigma_{\text{Person\_City = "Bomaby"}} (\text{lives}))$$

2) "Find the name and city of all employees who work for WIPRO".

$$\Pi_{\text{Person\_Name, Person\_City}} (\sigma_{\text{lives.Person\_Name = works.Person\_Name } \Lambda \text{ Company\_Name = WIPRO}} (\text{lives} \times \text{works}))$$

3) "Find all employees who live on the same street, in the same locality, and in the same city as their manager".

$\Pi_{Customer\_Name}$ ($\sigma_{managerlives.Street = lives\_Street \wedge mangerlives.Locality = lives\_Locality \wedge mangerlives.City = lives\_City}$ ($\Pi_{lives.Person\_name = namanges.Person\_Name \wedge manages.Manager\_Nmae = managerlives.Persosn\_Name}$ (lives × manages × ($\rho_{managerlives}$ (lives)))))

4) "Find all employees who live in the same city as the company they work for".

$\Pi_{Customer\_Name}$ ($\sigma_{lives.Person\_Name = works.Person\_Name \wedge works\_Company\_Name = located\_in.Company\_Name \wedge lives.City = located\_in.City}$ (lives × works × located_in)

6.5 SUMMARY

A relational algebra is an algebra defined over relations. The inputs are relations and the output is also a relation. The relational algebra is a procedural query language. In this unit, we discussed theory of relational algebra, the fundamental operations, and the additional operation of relational algebra. We also studied how to express some queries in relational algebra.

6.6 KEYWORDS

Relational algebra: A relational algebra is an algebra defined over relations.

Fundamental operation: An operation is fundamental if it cannot be expressed with other operations.

Additional operation: An operation is additional if it cannot be expressed with the fundamental operations

6.7 UNIT-END EXERCISES AND ANSWERS

1. What are the fundamental operations in relational algebra?
2. .Explain select, project, Cartesian product, rename, union, and set difference operations with examples.
3. What are additional relational operations? Explain.
4. Express some example queries in relational algebra.

Answer: SEE

1. 6.2
2. 6.2
3. 6.3
4. 6.4

6.8 SUGGESTED READINGS

- Fundamentals of Database Systems
  By Ramez Elmasri, Shamkant B. Navathe, Durvasula V.L.N. Somayajulu, Shyam K.Gupta
- Database System Concepts
  By Avi Silberschatz, Henry F. Korth , S. Sudarshan
- Database Management Systems
  By Raghu Ramakrishnan and Johannes Gehrke
- An Introduction to Database Systems
  C.J.Date