

DATABASE MANAGEMENT SYSTEM

WRITTEN BY
Mr.B.SANTHOSH KUMAR



NTDBPS-2017



Kongunadu Publications India Pvt Ltd

Database Management System

By

B.Santhosh Kumar,

*Associate Professor, Department of CSE (UG & PG),
SNS College of Technology, Sathy Main Road (NH-209),
Vazhiyampalayam, Saravanampatti Post,
Coimbatore – 641 035.*



Kongunadu Publications India Pvt Ltd

Printed in India

ISBN : 978-93-86770-44-8

Database Management System

B.Santhosh Kumar

English

2017

Pages: 142

Price Rs: 110/-

© Copyright Reserved by the Author

Publication

Printed, Published by

KONGUNADU PUBLICATIONS INDIA PVT LTD

118-Mettur Road, Opp:Kalyan Silks, Erode-638 011.

☎ 94422 51549, 97919 51549

✉ kongunadupublications@gmail.com

🌐 www.kongunadupublications.com

Distributed by

Kongunad Publishing House,

Erode-11.

89251 45386

ACKNOWLEDGEMENT

First of all I extend my heart felt gratitude to the management of SNS college of Technology, Dr.V.S.Velusamy, Founder Trestee, Dr. S.Rajalakshmi, Correspondent, Dr. S.N.Subramanian, chairman and Dr. V.P. Arunachalam, Director for providing us with all sorts of supports in completion of this book.

I record my indebtedness to my principal Dr. S.Chenthur Pandian for his guidance and sustained encouragement for the successful completion of this book.

I am highly grateful to Dr. S.Karthik, Professor & Dean, Department of Computer Science and Engineering, for his valuable suggestions and guidance throughout the course of this book.

I profoundly grateful to Dr.T.Kalaikumaran, Professor & Head, Department of Computer Science and Engineering, for his consistent encouragement and directions to prove my book and completing the book work in time.

I would like to extend my Special thanks to family members, Parents and my friends and who are all inspired us to start up this work.

Finally I would also like to thank my publishers for accept and gave valuable suggestion throughout the work.



Publisher Note

"If you would not be forgotten as soon as you are dead, either write something worth reading or do something worth writing" - words of Benjamin Franklin are highly inspirational and instrumental in our publishing career. Kongunadu Publication India Pvt Ltd is one of the leading book publishing concerns in Erode, Tamil Nadu, for the past 18 years.

Humans are the supreme creation of God. It is bliss to be born as a human being. Yet, the fragrance of this bliss can be enjoyed only when the human, like a flower, blooms to his or her fullest potential. Our motto is to make a better literary world. With that insight we formulated this 'National Teachers Day Book Publishing Scheme 2017'. Experience seems to underscore that knowledge alone, based on valuable insights and high ethical standards, is the leveraging factor for uplifting the whole mankind.


We intend to publish a series of books that will help people expand and utilize their fullest potential, in order to lead a richer and fuller life. We dream about the bright intellectual flowers sprouting all over the world. In fulfillment of this dream, ours is a concrete step.


This book is just a pinch of sugar of the NTDBP Scheme 2017. We assure you that it definitely tastes good. Full credit goes to our author, who has penned it with full of energy and enthusiasm.

No development happens as a matter of chance. Nor any single human can make this miracle happen. But it has to be a conscious and planned effort by all. We are sure that with your continued encouragement and patronage, a better world can be carved out. This belief is at the core of our efforts.

Nall Natarajan

Director

 94422 51549

 nallnatarajan@gmail.com

CONTENTS

Chapter 1: Introduction to Data Base Management Systems	1-29
1.1 Introduction	
1.2 Data Processing Vs. Data Management Systems	
1.3 File Oriented Approach	
1.4 Database Oriented Approach to Data Management	
1.5 Characteristics of Database	
1.6 Advantages and Disadvantages of a DBMS	
1.7 Disadvantages of a DBMS	
1.8 Instances and Schemas	
1.9 Data Models	
1.10 Database Languages	
1.11 Data Dictionary	
1.12 Database Administrators and Database Users	
1.13 DBMS Architecture and Data Independence	
Chapter 2: Data Modeling Using Entity-Relationship Approach	30-55
2.1 Introduction	
2.2 Data Modeling In the Context of Database Design	
2.3 The Entity-Relationship Model	
2.4 Data Modeling As Part of Database Design	
2.5 Steps in Building the Data Model	
2.6 Developing the Basic Schema	
Chapter 3: Structured Query Language	56-80
3.1 Basic Structure	
3.2 Data Definition Language	
3.3 DML Commands	
3.4 Different Types of Joins	
3.5 Integrity Constraints	
3.6 Stored Procedure	

- 3.7 Triggers
- 3.8 Security
- 3.9 Advanced SQL Features
- 3.10 Embedded SQL
- 3.11 View

Chapter 4: Introduction to Distributed Databases

81-89

- 4.1 Distributed Databases
- 4.2 Data Replication
- 4.3 Data Fragmentation
- 4.4 Transparency
- 4.5 Client/Server Database
- 4.6 Benefits of client/server computing

Chapter 5: Relational Database Design

90-104

- 5.1 Pitfalls in Database Design
- 5.2 Functional Dependencies
- 5.3 Canonical Cover
- 5.4 Normalization

Chapter 6: Query Processing

105-116

- 6.1 Introduction
- 6.2 Query Optimization

Chapter 7: Text and Data Mining

117-136

- 7.1 Introduction
- 7.2 Historical Development
- 7.3 Working Principles
- 7.4 Organisations involved in Text and Data Mining
- 7.5 The Challenges
- 7.6 Implications of Text and Data Mining

Chapter-1

Introduction to Data Base Management Systems

A database is an integrated collection of logically related records or files consolidated into a common pool that provides data for one or more multiple uses. You can think of a database as an electronic filing system.

1.1 Introduction

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL.

A datum – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information. Hence, information is interpreted data – data supplied with semantics. Knowledge refers to the practical use of information. While information can be transported, stored or shared without many difficulties the same cannot be said about knowledge. Knowledge necessarily involves a personal experience. Referring back to the scientific experiment, a third person reading the results will have information about it, while the person who conducted the experiment personally will have knowledge about it.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for

storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. These concepts and technique form the focus of this book. This chapter briefly introduces the principles of database systems.

1.2 Data Processing Vs. Data Management Systems

Although Data Processing and Data Management Systems both refer to functions that take raw data and transform it into usable information, the usage of the terms is very different. Data Processing is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed format reports written to paper.

The term Data Management Systems refers to an expansion of this concept, where the raw data, previously copied manually from paper to punched cards, and later into data entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquities of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

1.3 File Oriented Approach

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called file processing systems. In a typical file processing systems, each department has its own files, designed specifically for those applications. The department itself works with the data processing staff, sets policies or standards for the format and maintenance of its files.

Programs are dependent on the files and vice-versa; that is, when the physical format of the file is changed, the program has also to be changed. Although the traditional file oriented approach to information processing is still widely used, it does have some very important disadvantages.

1.4 Database Oriented Approach to Data Management

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including

- A program to debit or credit an account
- A program to add a new account
- A program to find the balance of an account
- A program to generate monthly statements

System programmers wrote these application programs to meet the needs of the bank.

New application programs are added to the system as the need arises. For example, suppose that the savings bank decides to offer

checking accounts. As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs.

This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) came along, organizations usually stored information in such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages:

➤ **Data redundancy and inconsistency.**

Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

➤ **Difficulty in accessing data.**

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no

application program on hand to meet it. There is, however, an application program to generate the list of all customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program.

➤ **Both alternatives are obviously unsatisfactory.**

Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of \$10,000 or more. As expected, a program to generate such a list does not exist.

Again, the officer has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

➤ **Data isolation.**

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

➤ **Integrity problems.**

The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, \$25). Developers enforce these constraints in the system by adding appropriate code in the various application programs.

However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

➤ **Atomicity problems.**

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer \$50 from account A to account B.

If a system failure occurs during the execution of the program, it is possible that the \$50 was removed from account A, but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

➤ **Concurrent-access anomalies.**

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account A, containing \$500. If two customers withdraw funds (say \$50 and \$100 respectively) from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state.

Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value \$500, and write back \$450 and \$400, respectively. Depending on which one writes the value last, the account may contain \$450 or \$400, rather than the correct value of \$350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

➤ **Security problems.**

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a bank enterprise as a running example of a typical data-processing application found in a corporation.

1.5 Characteristics of Database

The database approach has some very characteristic features which are discussed in detail below:

Concurrent Use

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system.

An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

Structured and Described Data

A fundamental feature of the database approach is that the database systems do not only contain the data but also the complete definition and

description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data").

Separation of Data and Applications

As described in the feature structured data the structure of a database is described through metadata which is also stored in the database. Application software does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system of a database (DBMS) via a standardized interface with the help of a standardized language like SQL. The access to the data and the metadata is entirely done by the DBMS. In this way all the applications can be totally separated from the data. Therefore database internal reorganizations or improvement of efficiency do not have any influence on the application software.

Data Integrity

Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorized access (confidentiality) and unauthorized changes. Data reflect facts of the real world database.

Transactions

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state. In between the data are inevitable inconsistent. A transaction is atomic what means that it cannot be divided up any further. Within a transaction all or none of the actions need to be carried out. Doing only a part of the actions would lead to an inconsistent database state. One example of a transaction is the transfer of an amount of money from one bank account to another. The debit of the money from one account and the credit of it to another account make together a consistent transaction. This transaction is also

atomic. The debit or credit alone would both lead to an inconsistent state. After finishing the transaction (debit and credit) the changes to both accounts become persistent and the one who gave the money has now less money on his account while the receiver has now a higher balance.

Data Persistence

Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly by the user and must not be dependent on system features. Additionally data once stored in a database must not be lost. Changes of a database which are done by a transaction are persistent. When a transaction is finished even a system crash cannot put the data in danger.

1.6 Advantages and Disadvantages of a DBMS

Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced

professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine tuning the storage of the data to make retrieval efficient.

Concurrent access and crash recovery: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

Reduced application development time: Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

A DBMS is a complex piece of software, optimized for certain kinds of workloads (e.g., answering complex queries or handling many concurrent requests), and its performance may not be adequate for certain specialized applications. Examples include applications with tight real-time constraints or applications with just a few well-designed critical operations for which efficient custom code must be written. Another reason for not using a DBMS is that an application may need to manipulate the data in ways not supported by the query language.

In such a situation, the abstract view of the data presented by the DBMS does not match the application's needs, and actually gets in the way. As an example, relational databases do not support flexible analysis of text data (although vendors are now extending their products in this direction). If specialized performance or data manipulation requirements are central to an application, the application may choose not to use a DBMS, especially if the added benefits of a DBMS (e.g., flexible querying, security, concurrent access, and crash recovery) are not required. In most situations calling for

large-scale data management, however, DBMSs have become an indispensable tool.

1.7 Disadvantages of a DBMS

Danger of Overkill: For small and simple applications for single users a database system is often not advisable.

Complexity: A database system creates additional complexity and requirements. The supply and operation of a database management system with several users and databases is quite costly and demanding.

Qualified Personnel: The professional operation of a database system requires appropriately trained staff. Without a qualified database administrator nothing will work for long.

Costs: Through the use of a database system new costs are generated for the system itself but also for additional hardware and the more complex handling of the system.

Lower Efficiency: A database system is a multi-use software which is often less efficient than specialized software which is produced and optimized exactly for one problem.

1.8 Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema. Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language.

A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema. Database systems have several schemas, partitioned according to the levels of

abstraction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas that describe different views of the database.

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit physical data independence if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

We study languages for describing schemas, after introducing the notion of data models in the next section.

1.9 Data Models

Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. To illustrate the concept of a data model, we outline two data models in this section: the entity-relationship model and the relational model. Both provide a way to describe the design of a database at the logical level.

The Entity-Relationship Model

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationships among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities. Entities are described in a database by a set of attributes. For example, the attributes account-number and balance may describe one particular account in a bank, and they form attributes of the account entity

set. Similarly, attributes customer-name, customer-street address and customer-city may describe a customer entity.

An extra attribute customer-id is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city). A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier. A relationship is an association among several entities. For example, a depositor relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively. The overall logical structure (schema) of a database can be expressed graphically by an E-R diagram.

Relational Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. The data is arranged in a relation which is visually represented in a two dimensional table. The data is inserted into the table in the form of tuples (which are nothing but rows). A tuple is formed by one or more than one attributes, which are used as basic building blocks in the formation of various expressions that are used to derive meaningful information.

There can be any number of tuples in the table, but all the tuple contain fixed and same attributes with varying values. The relational model is implemented in database where a relation is represented by a table, a tuple is represented by a row, an attribute is represented by a column of the table, attribute name is the name of the column such as 'identifier', 'name', 'city' etc., attribute value contains the value for column in the row. Constraints are applied to the table and form the logical schema.

In order to facilitate the selection of a particular row/tuple from the table, the attributes i.e. column names are used, and to expedite the selection of the rows some fields are defined uniquely to use them as indexes, this helps in searching the required data as fast as possible. All the relational algebra operations, such as Select, Intersection, Product, Union, Difference, Project, Join, Division, Merge etc. can also be performed on the Relational Database Model. Operations on the Relational Database Model are facilitated with the help of different conditional expressions, various key attributes, pre-defined n constraints etc.

Other Data Models

The object-oriented data model is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions object oriented data model. The object-relational data model combines features of the object-oriented data model and relational data model. Semi structured data models permit the specification of data where individual data items of the same type may have different sets of attributes.

This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The extensible markup language (XML) is widely used to represent semi structured data. Historically, two other data models, the network data model and the hierarchical data model, preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are little used now, except in old database code that is still in service in some places. They are outlined in Appendices A and B, for interested readers.

1.10 Database Languages

A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates. In practice, the data definition and data manipulation

languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). For instance, the following statement in the SQL language defines the account table:

```
create table account (account-number char(10), balance integer)
```

Execution of the above DDL statement creates the account table. In addition, it updates a special set of tables called the data dictionary or data directory. A data dictionary contains metadata—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints. For example, suppose the balance on an account should not fall below \$100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

Data-Manipulation Language

Data manipulation is

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. There are basically two types:

Procedural DMLs require a user to specify what data are needed and how to get those data. Declarative DMLs (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms query language and data manipulation language synonymously.

This query in the SQL language finds the name of the customer whose `cust_id` is 192-83-7465:

```
select cust.customer-name from customer where cust.customer_id =  
192-83-7465
```

The query specifies that those rows from the table `customer` where the `customer-id` is 192-83-7465 must be retrieved, and the `customer-name` attribute of these rows must be displayed.

Queries may involve information from more than one table. For instance, the following query finds the balance of all accounts owned by the customer with `cust_id` 192-83- 7465.

```
select account.balance from depositor, account where  
depositor.cust_id = 192-83-7465 and depositor.account-number =  
account.account-number
```


There are a number of database query languages in use, either commercially or experimentally.

The levels of abstraction apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we emphasize ease of use. The goal is to allow humans to interact efficiently with the system. The query processor component of the database system translates DML queries into sequences of actions at the physical level of the database system.

1.11 Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such “data about data” were labeled metadata. The DBMS data dictionary provides the DBMS with its self-describing characteristic. In effect, the data dictionary resembles an X-ray of the company’s entire data set, and is a crucial element in the data administration function.

The two main types of data dictionary exist, integrated and stand alone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs especially older types, do not have a built in data dictionary instead the DBA may use third party stand-alone data dictionary systems.

Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access information up-to-date. A passive data dictionary is not updated automatically and usually requires a batch process to be run. Data dictionary access information is normally used by the DBMS for query optimization purpose.

The data dictionary’s main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to

limit their metadata to the data managed by the DBMS. Stand-alone data dictionary systems are more usually more flexible and allow the DBA to describe and manage all the organization's data, whether or not they are computerized. Whatever the data dictionary's format, its existence provides database designers and end users with a much improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts. Although, there is no standard format for the information stored in the data dictionary several features are common.

For example, the data dictionary typically stores descriptions of all:

- Data elements that are define in all tables of all databases. Specifically the data dictionary stores the name, datatypes, display formats, internal storage formats, and validation rules. The data dictionary tells where an element is used, by whom it is used and so on.

- Tables define in all databases. For example, the data dictionary is likely to store the name of the table creator, the date of creation access authorizations, the number of columns, and so on.

- Indexes define for each database tables. For each index the DBMS stores at least the index name the attributes used, the location, specific index characteristics and the creation date.

- Define databases: who created each database, the date of creation where the database is located, who the DBA is and so on.

- End users and The Administrators of the data base

- Programs that access the database including screen formats, report formats application formats, SQL queries and so on.

- Access authorization for all users of all databases.

- Relationships among data elements which elements are involved: whether the relationships are mandatory or optional, the connectivity and cardinality and so on.

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an especially flexible to for more general corporate resource management. The management of such an extensive data dictionary, thus, makes it possible to manage the use and allocation of all of the organization information regardless whether it has its roots in the database data. This is why some managers consider the data dictionary to be the key element of the information resource management function. And this is also why the data dictionary might be described as the information resource dictionary.

The metadata stored in the data dictionary is often the bases for monitoring the database use and assignment of access rights to the database users. The information stored in the database is usually based on the relational table format, thus, enabling the DBA to query the database with SQL command. For example, SQL command can be used to extract information about the users of the specific table or about the access rights of particular users.

1.12 Database Administrators and Database Users

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators.

Database Users and User Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer \$50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from

which the money is to be transferred, and the account to which the money is to be transferred.

As another example, consider a user who wishes to find her account balance over the World Wide Web. Such a user may access a form, where she enters her account number. An application program at the Web server then retrieves the account balance, using the given account number, and passes this information back to the user. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called fourth-generation languages, often include special features to facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

Online analytical processing (OLAP) tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and

West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category).

Another class of tools for analysts is data mining tools, which help them, find certain kinds of patterns in data.

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include: Schema definition. The DBA creates the original database schema by executing a set of data definition statements in the DDL.

- Storage structure and access-method definition.
- Schema and physical-organization modification.

The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Granting of authorization for data access.

By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure

that the database system consults whenever someone attempts to access the data in the system.

Routine maintenance.

Examples of the database administrator's routine maintenance activities are: Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding. Ensuring that enough free disk space is available for normal operations and upgrading disk space as required. Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

1.13 DBMS Architecture and Data Independence

Three important characteristics of the database approach are (1) insulation of programs and data (program-data and program-operation independence); (2) support of multiple user views; and (3) use of a catalog to store the database description (schema). In this section we specify architecture for database systems, called the three-schema architecture, which was proposed to help achieve and visualize these characteristics. We then discuss the concept of data independence.

The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 1.1, is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures

and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.

3. The external or view level includes a number of external schemas or user views.

Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

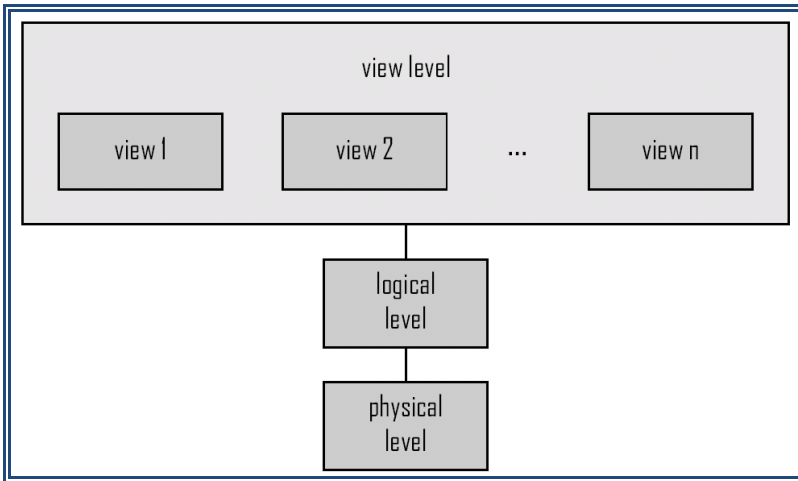


Figure 1.1 Three Schema Architecture

The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views, external schemas are specified in the same data model that describes the

conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.

Notice that the three schemas are only descriptions of data; the only data that actually exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings. These mappings may be time consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Data Independence

The three-schema architecture can be used to explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be

applied also to the conceptual schema without affecting the external schemas or application programs.

2. Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

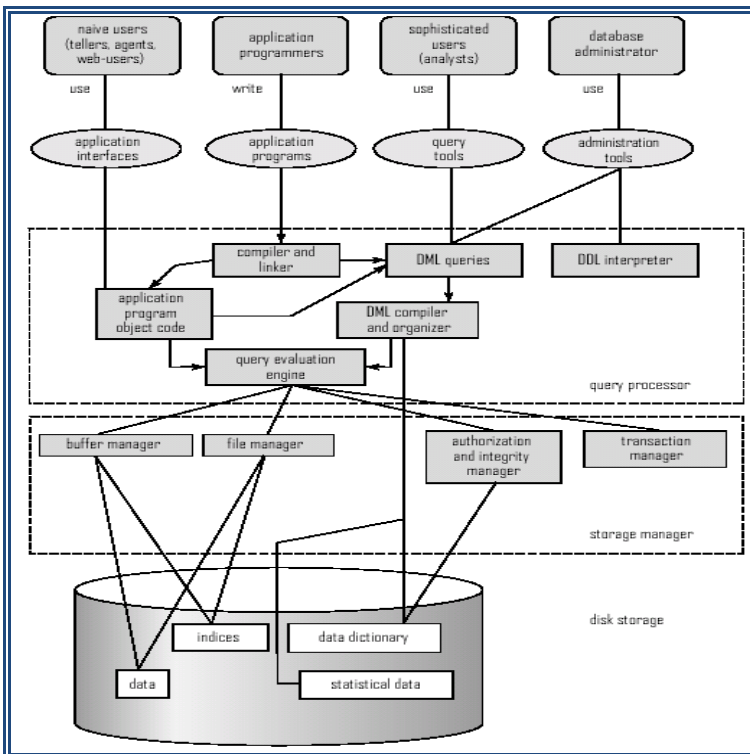


Fig 1.2 Overall System Structure

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence is accomplished because, when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three-schema architecture.

Types of Database System

Several criteria are normally used to classify DBMSs. The first is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model. The object data model was implemented in some commercial systems but has not had widespread use. Many legacy (older) applications still run on database systems based on the hierarchical and network data models.

The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can hence categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with personal computers. Multiuser systems, which include the majority of DBMSs, support multiple users concurrently.

A third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site.

A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites.

A recent trend is to develop software to access several autonomous preexisting databases stored under heterogeneous IBMSs. This leads to a federated DBMS (or multi database system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DBMSs use client-server architecture.

Summary

In this chapter we have discussed in a relatively informal manner the major components of a database system. We summarize the discussion below:

- ✓ A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database.
- ✓ A datum – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information.
- ✓ Knowledge refers to the practical use of information.
- ✓ The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema.

- ✓ The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas that describe different views of the database.
- ✓ Application programs are said to exhibit physical data independence if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.
- ✓ Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- ✓ A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates.
- ✓ One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA).

1.15 Key Words

DBMS, Data Integrity, Data Persistence, Instances, Schemas, Physical Schema, Logical Schema, Data Model, DDL, DML, Data Dictionary

1.16 Self-Assessment Questions

1. Why would you choose a database system instead of simply storing data in operating system files? When would it make sense not to use a database system?
2. What is logical data independence and why is it important?
3. Explain the difference between logical and physical data independence.

4. Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?
5. What are the responsibilities of a DBA?
6. Distinguish between logical and physical database design.
7. Describe and define the key properties of a database system. Give an organizational example of the benefits of each property.

Chapter 2:

Data Modeling Using Entity-Relationship Approach

2.1 Introduction

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects. As the name implies, the data model focuses on what data is required and how it should be organized rather than what operations will be performed on the data. To use a common analogy, the data model is equivalent to an architect's building plans.

A data model is independent of hardware or software constraints. Rather than try to represent the data as a database would see it, the data model focuses on representing the data as the user sees it in the "real world". It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

Methodology: There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. This document uses the Entity-Relationship approach.

2.2 Data Modeling In the Context of Database Design

Database design is defined as: "design the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications". The design process roughly follows five steps:

1. Planning and analysis
2. Conceptual design
3. Logical design
4. Physical design
5. Implementation

The data model is one part of the conceptual design process. The other, typically is the functional model. The data model focuses on what data should be stored in the database while the functional model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables.

The functional model is used to design the queries which will access and perform operations on those tables.

Components of a Data Model: The data model gets its inputs from the planning and analysis stage. Here the modeler, along with analysts, collects information about the requirements of the database by reviewing existing documentation and interviewing end-users.

The data model has two outputs. The first is an entity-relationship diagram which represents the data structures in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a data document. This document that describes in details the data objects, relationships, and rules required by the database. The dictionary provides the detail required by the database developer to construct the physical database.

Why is Data Modeling Important?

Data modeling is probably the most labor intensive and time consuming part of the development process. Why bother especially if you are pressed for time? A common response by practitioners who write on the subject is that you should no more build a database without a model than you should build a house without blueprints.

The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users.

The data model is also detailed enough to be used by the database developers to use as a "blueprint" for building the physical database. The information contained in the data model will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers. A poorly designed database will require more time in the long-term.

Without careful planning you may create a database that omits data required to create critical reports, produces results that are incorrect or inconsistent, and is unable to accommodate changes in the user's requirements.

2.3 The Entity-Relationship Model

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design. For the database designer, the utility of the ER model is:

It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.

It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.

In addition, the model can be used as a design plan by the database developer to implement a data model in specific database management software.

Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities.

Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, and INVOICES. An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one that does not rely on another for identification. A dependent entity is one that relies on another for identification.

An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

Special Entity Types Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

Associative entities and generalization hierarchies are discussed in more detail below.

Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

- Employees are assigned to projects
- Projects have subtasks
- Departments manage one or more projects

- Relationships are classified in terms of degree, connectivity, cardinality, and existence.

These concepts will be discussed below.

Attributes

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name.

The domain of an attribute is the collection of all possible values an attribute can have.

The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

Classifying Relationships

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

Degree of a Relationship

The degree of a relationship is the number of entities associated with the relationship.

The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

A binary relationship, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

Connectivity and Cardinality The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

A one-to-one (1:1) relationship is when at most one instance of a entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.

A one-to-many (1:N) relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

A department has many employees

Each employee is assigned to one department

A many-to-many (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is: employees can be assigned to no more than two projects at the same time; projects must have assigned at least three employees

A single employee can be assigned to many projects; conversely, a single project can have assigned to it many employee. Here the cardinality

for the relationship between employees and projects is two and the cardinality between project and employee is three.

Many-to-many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

Direction

The direction of a relationship indicates the originating entity of a binary relationship.

The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

Type

An identifying relationship is one in which one of the child entities is also a dependent entity. A non-identifying relationship is one in which both entities are independent.

Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either mandatory or optional. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of mandatory existence is the statement "every project must be managed by a single department". If the instance of the entity is not required, it is optional. An example of optional existence is the statement, "employees may be assigned to work on projects".

Generalization Hierarchies

A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a super type or generic entity. The lower-level of entities become the subtype, or categories, to the super type. Subtypes are dependent entities.

Generalization occurs when two or more entities represent categories of the same real-world object. For example, `Wages_Employees` and `Classified_Employees` represent categories of the same entity, `Employees`. In this example, `Employees` would be the supertype; `Wages_Employees` and `Classified_Employees` would be the subtypes.

Subtypes can be either mutually exclusive (disjoint) or overlapping (inclusive). A mutually exclusive category is when an entity instance can be in only one category. The above example is a mutually exclusive category. An employee can either be wages or classified but not both. An overlapping category is when an entity instance may be in two or more subtypes. An example would be a person who works for a university could also be a student at that same university. The completeness constraint requires that all instances of the subtype be represented in the supertype. Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of nesting is limited only by the constraint of simplicity. Subtype entities may be the parent entity in a relationship but not the child.

E-R Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin.

The symbols used for the basic ER constructs are:

- Entities are represented by labeled rectangles. The label is the name of the entity.
- Entity names should be singular nouns.
- Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.
- Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
- Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.
- Existence is represented by placing a circle or a perpendicular bar on the line.

Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

Examples of these symbols are shown in Figure 2.1 below:

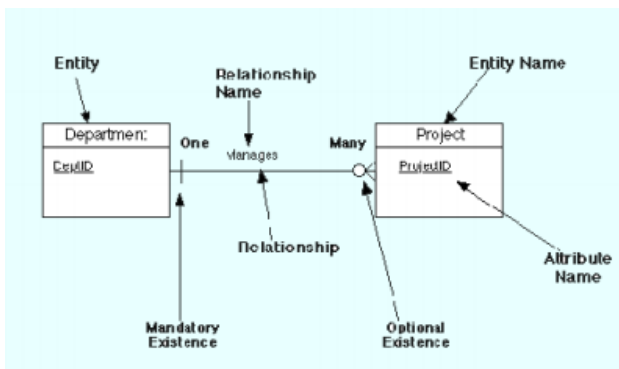


Figure 2.1 ER Notation

2.4 Data Modeling As Part of Database Design

The data model is one part of the conceptual design process. The other is the function model. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries that will access and perform operations on those tables.

Data modeling is preceded by planning and analysis. The effort devoted to this stage is proportional to the scope of the database. The planning and analysis of a database intended to serve the needs of an enterprise will require more effort than one intended to serve a small workgroup.

The information needed to build a data model is gathered during the requirements analysis. Although not formally considered part of the data modeling stage by some methodologies, in reality the requirements analysis and the ER diagramming part of the data model are done at the same time.

Requirements Analysis

The goals of the requirements analysis are:

- To determine the data requirements of the database in terms of primitive objects
- To classify and describe the information about these objects
- To identify and classify the relationships among the objects
- To determine the types of transactions that will be executed on the database and the interactions between the data and the transactions
- To identify rules governing the integrity of the data

The modeler, or modelers, works with the end users of an organization to determine the data requirements of the database. Information needed for the requirements analysis can be gathered in several ways:

Review of existing documents - such documents include existing forms and reports, written guidelines, job descriptions, personal narratives, and memoranda. Paper documentation is a good way to become familiar with the organization or activity you need to model.

Interviews with end users - these can be a combination of individual or group meetings. Try to keep group sessions to under five or six people. If possible, try to have everyone with the same function in one meeting. Use a blackboard, flip charts, or overhead transparencies to record information gathered from the interviews.

Review of existing automated systems - if the organization already has an automated system, review the system design specifications and documentation

The requirements analysis is usually done at the same time as the data modeling. As information is collected, data objects are identified and classified as entities, attributes, or relationship; assigned names; and, defined using terms familiar to the end-users.

The objects are then modeled and analysed using an ER diagram. The diagram can be reviewed by the modeler and the end-users to determine its completeness and accuracy. If the model is not correct, it is modified, which sometimes requires additional information to be collected. The review and edit cycle continues until the model is certified as correct.

Three points to keep in mind during the requirements analysis are:

1. Talk to the end users about their data in "real-world" terms. Users do not think in terms of entities, attributes, and relationships but about the actual people, things, and activities they deal with daily.

2. Take the time to learn the basics about the organization and its activities that you want to model. Having an understanding about the processes will make it easier to build the model.
3. End-users typically think about and view data in different ways according to their function within an organization. Therefore, it is important to interview the largest number of people that time permits.

2.5 Steps in Building the Data Model

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies, such as IDEFIX, specify a bottom-up development process where the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users. The sequences used for this document are:

1. Identification of data objects and relationships
2. Drafting the initial ER diagram with entities and relationships
3. Refining the ER diagram
4. Add key attributes to the diagram
5. Adding non-key attributes
6. Diagramming Generalization Hierarchies
7. Validating the model through normalization
8. Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. As noted above, the requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may

uncover problems or missing information which require more information gathering and analysis.

Identifying Data Objects and Relationships

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirements analysis for the purpose of:

- Classifying data objects as either entities or attributes
- Identifying and defining relationships between entities
- Naming and defining identified entities, attributes, and relationships
- Documenting this information in the data document

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

Although it is easy to define the basic constructs of the ER model, it is not an easy task to distinguish their roles in building the data model. What makes an object an entity or attribute? For example, given the statement "employees work on projects". Should employees be classified as an entity or attribute? Very often, the correct answer depends upon the requirements of the database. In some cases, employee would be an entity, in some it would be an attribute.

While the definitions of the constructs in the ER Model are simple, the model does not address the fundamental issue of how to identify them. Some commonly given guidelines are:

- Entities contain descriptive information
- Attributes either identify or describe entities
- Relationships are associations between entities

These guidelines are discussed in more detail below.

- Entities
- Attributes
 - Validating Attributes
 - Derived Attributes and Code Values
- Relationships
- Naming Data Objects
- Object Definition
- Recording Information in Design Document

Entities

There are various definitions of an entity:

"Any distinguishable person, place, thing, event, or concept, about which information is kept"

"A thing which can be distinctly identified"

"Any distinguishable object that is to be represented in a database"

"...anything about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.). For each entity type, certain attributes are stored".

These definitions contain common themes about entities:

- An entity is a "thing", "concept" or, "object". However, entities can sometimes represent the relationships between two or more objects. This type of entity is known as an associative entity.
- Entities are objects which contain descriptive information. If a data object you have identified is described by other objects, then it is an entity. If there is no descriptive information associated with

the item, it is not an entity. Whether or not a data object is an entity may depend upon the organization or activity being modeled.

- An entity represents many things which share properties. They are not single things. For example, King Lear and Hamlet are both plays which share common attributes such as name, author, and cast of characters. The entity describing these things would be PLAY, with King Lear and Hamlet being instances of the entity.
- Entities which share common properties are candidates for being converted to generalization hierarchies (See below)
- Entities should not be used to distinguish between time periods. For example, the entities 1st Quarter Profits, 2nd Quarter Profits, etc. should be collapsed into a single entity called Profits. An attribute specifying the time period would be used to categorize by time
- Not everything the users want to collect information about will be an entity. A complex concept may require more than one entity to represent it. Others "things" users think important may not be entities.

Attributes

Attributes are data objects that either identify or describe entities. Attributes that identify entities are called key attributes. Attributes that describe an entity are called non-key attributes. Key attributes will be discussed in detail in a latter section.

The process for identifying attributes is similar except now you want to look for and extract those names that appear to be descriptive noun phrases. Validating Attributes Attribute values should be atomic, that is, present a single fact. Having disaggregated data allows simpler programming, greater reusability of data, and easier implementation of changes. Normalization also depends upon the "single fact" rule being followed.

Common types of violations include:

- Simple aggregation - a common example is Person Name which concatenates first name, middle initial, and last name. Another is Address which concatenates, street address, city, and zip code. When dealing with such attributes, you need to find out if there are good reasons for decomposing them. For example, do the end-users want to use the person's first name in a form letter? Do they want to sort by zip code?
- Complex codes - these are attributes whose values are codes composed of concatenated pieces of information. An example is the code attached to automobiles and trucks. The code represents over 10 different pieces of information about the vehicle. Unless part of an industry standard, these codes have no meaning to the end user. They are very difficult to process and update.
- Text blocks - these are free-form text fields. While they have a legitimate use, an over reliance on them may indicate that some data requirements are not met by the model.
- Mixed domains - this is where a value of an attribute can have different meaning under different conditions

Derived Attributes and Code Values

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favour are:

- Derived data is often important to both managers and users and therefore should be included in the data model
- It is just as important, perhaps more so, to document derived attributes just as you would other attributes
- Including derived attributes in the data model does not imply how they will be implemented

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favour argue that many organizations have a long history of using coded attributes, that codes save space, and improve flexibility in that values can be easily added or modified by means of look-up tables.

Relationships

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities. For example: employees are assigned to projects.

As relationships are identified they should be classified in terms of cardinality, optionality, direction, and dependence. As a result of defining the relationships, some relationships may be dropped and new relationships added. Cardinality quantifies the relationships between entities by measuring how many instances of one entity are related to a single instance of another.

To determine the cardinality, assume the existence of an instance of one of the entities. Then determine how many specific instances of the second entity could be related to the first. Repeat this analysis reversing the entities. For example:

Employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example:

An employee may be assigned to a project

Mandatory relationships, on the other hand, are indicated by words such as must have. For example:

A student must register for at least three course each semester

In the case of the specific relationship form (1:1 and 1:M), there is always a parent entity and a child entity. In one-to-many relationships, the parent is always the entity with the cardinality of one. In one-to-one relationships, the choice of the parent entity must be made in the context of the business being modeled. If a decision cannot be made, the choice is arbitrary.

Naming Data Objects

The names should have the following properties:

- Unique
- Have meaning to the end-user

Contain the minimum number of words needed to uniquely and accurately describe the object For entities and attributes, names are singular nouns while relationship names are typically verbs.

Some authors advise against using abbreviations or acronyms because they might lead to confusion about what they mean. Other believe using abbreviations or acronyms are acceptable provided that they are universally used and understood within the organization.

You should also take care to identify and resolve synonyms for entities and attributes.

This can happen in large projects where different departments use different terms for the same thing.

Object Definition

Complete and accurate definitions are important to make sure that all parties involved in the modeling of the data know exactly what concepts the objects are representing.

Definitions should use terms familiar to the user and should precisely explain what the object represents and the role it plays in the enterprise. Some authors recommend having the end-users provide the definitions. If acronyms, or terms not universally understood are used in the definition, then these should be defined.

While defining objects, the modeler should be careful to resolve any instances where a single entity is actually representing two different concepts (homonyms) or where two different entities are actually representing the same "thing" (synonyms). This situation typically arises because individuals or organizations may think about an event or process in terms of their own function.

An example of a homonym would be a case where the Marketing Department defines the entity MARKET in terms of geographical regions while the Sales Departments thinks of this entity in terms of demographics. Unless resolved, the result would be an entity with two different meanings and properties.

Conversely, an example of a synonym would be the Service Department may have identified an entity called CUSTOMER while the Help Desk has identified the entity CONTACT. In reality, they may mean the same thing, a person who contacts or calls the organization for assistance with a problem. The resolution of synonyms is important in order to avoid redundancy and to avoid possible consistency or integrity problems.

Recording Information in Design Document

The design document records detailed information about each object used in the model. As you name, define, and describe objects, this information should be placed in this document. If you are not using an automated design tool, the document can be done on paper or with a word processor. There is no standard for the organization of this document but the document should include information about names, definitions, and, for attributes, domains. Two documents used in the IDEF1X method of modeling are useful for keeping track of objects. These are the ENTITY-ENTITY matrix and the ENTITY-ATTRIBUTE matrix.

The ENTITY-ENTITY matrix is a two-dimensional array for indicating relationships between entities. The names of all identified entities are listed along both axes. As relationships are first identified, an "X" is placed in the intersecting points where any of the two axes meet to indicate a possible relationship between the entities involved. As the relationship is further classified, the "X" is replaced with the notation indicating cardinality.

The ENTITY-ATTRIBUTE matrix is used to indicate the assignment of attributes to entities. It is similar in form to the ENTITY-ENTITY matrix except attribute names are listed on the rows.

Figure 2.2 shows examples of an ENTITY-ENTITY matrix and an ENTITYATTRIBUTE matrix.

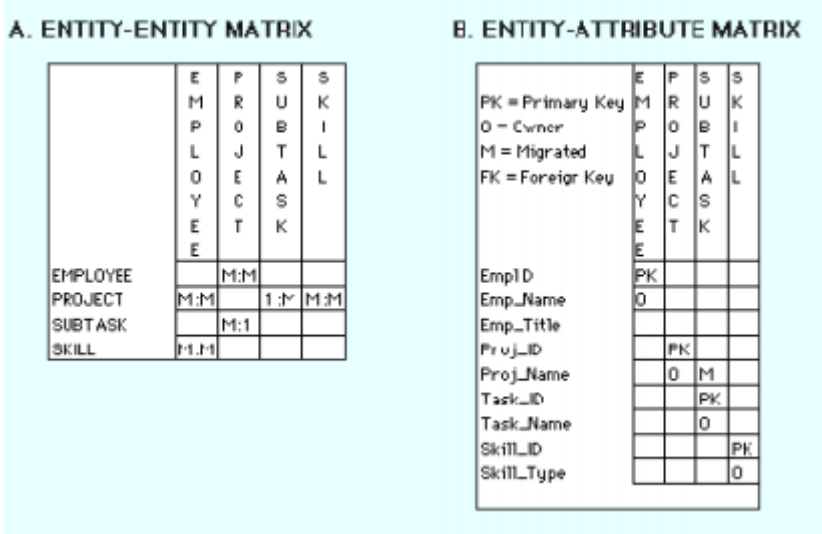


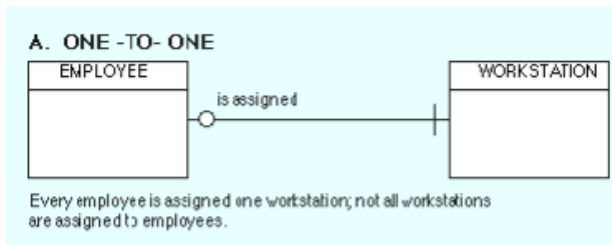
Figure 2.2

2.6 Developing the Basic Schema

Once entities and relationships have been identified and defined, the first draft of the entity relationship diagram can be created. This section introduces the ER diagram by demonstrating how to diagram binary relationships. Recursive relationships are also shown.

Binary Relationships

Figure 2.3 shows examples of how to diagram one-to-one, one-to-many, and many-to-many relationships.



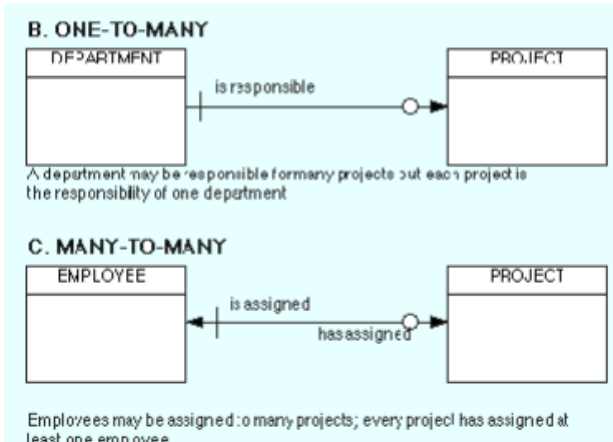


Figure 2.3 Example of Binary relationships

One-To-One

Figure 2A shows an example a one-to-one diagram. Reading the diagram from left to right represents the relationship every employee is assigned a workstation. Because every employee must have a workstation, the symbol for mandatory existence—in this case the crossbar—is placed next to the WORKSTATION entity. Reading from right to left, the diagram shows that not all workstation are assigned to employees. This condition may reflect that some workstations are kept for spares or for loans. Therefore, we use the symbol for optional existence, the circle, next to EMPLOYEE. The cardinality and existence of a relationship must be derived from the "business rules" of the organization.

For example, if all workstations owned by an organization were assigned to employees, then the circle would be replaced by a crossbar to indicate mandatory existence. One-to-one relationships are rarely seen in "real-world" data models. Some practioners advise that most one-to-one relationships should be collapsed into a single entity or converted to a generalization hierarchy.

One-To-Many

Figure 2B shows an example of a one-to-many relationship between DEPARTMENT and PROJECT. In this diagram, DEPARTMENT is considered the parent entity while PROJECT is the child. Reading from left to right, the diagram represents departments may be responsible for many projects.

The optionality of the relationship reflects the "business rule" that not all departments in the organization will be responsible for managing projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

Many-To-Many

Figure 2C shows a many-to-many relationship between EMPLOYEE and PROJECT. An employee may be assigned to many projects; each project must have many employee Note that the association between EMPLOYEE and PROJECT is optional because, at a given time, an employee may not be assigned to a project.

However, the relationship between PROJECT and EMPLOYEE is mandatory because a project must have at least two employees assigned. Many-To-Many relationships can be used in the initial drafting of the model but eventually must be transformed into two one-to-many relationships. The transformation is required because many-to-many relationships cannot be represented by the relational model. The process for resolving many-to-many relationships is discussed in the next section.

Recursive relationships

A recursive relationship is an entity is associated with itself. Figure 2.4 shows an example of the recursive relationship.

An employee may manage many employees and each employee is managed by one employee.

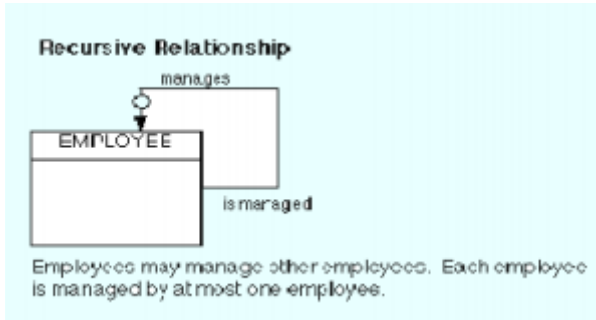


Figure 2.4 Example of Recursive relationships

Summary

- ✓ A data model is a plan for building a database. To be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure.
- ✓ The Entity-Relation Model (ER) is the most common method used to build data models for relational databases.
- ✓ The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram.
- ✓ The basic constructs of the ER model are entities, relationships, and attributes.
- ✓ Data modeling must be preceded by planning and analysis.
- ✓ Planning defines the goals of the database, explains why the goals are important, and sets out the path by which the goals will be reached.

- ✓ Analysis involves determining the requirements of the database. This is typically done by examining existing documentation and interviewing users.
- ✓ Data modeling is a bottom up process. A basic model, representing entities and relationships, is developed first. Then detail is added to the model by including information about attributes and business rules. The first step in creating the data model is to analyze the information gathered during the requirements analysis with the goal of identifying and classifying data objects and relationships
- ✓ The Entity-Relationship diagram provides a pictorial representation of the major data objects, the entities, and the relationships between them.

Key Words

ER Model, Database Design, Data Model, Schema, Entities, Relationship, Attributes, Cardinality

Self-Assessment Questions

1. A university registrar's office maintains data about the following entities:
 - Courses, including number, title, credits, syllabus, and prerequisites;
 - Course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;
 - Students, including student-id, name, and program;
 - Instructors, including identification number, name, department, and title.

Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct a E-R diagram for registrar's office. Document all assumptions that you make about the mapping constraints

2. Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match, and individual player statistics for each match. Summary statistics should be modeled as derived attributes.

3. Explain the significance of ER Model for Database design?

4. Enumerate the basic constructs of ER Model

Chapter 3:

STRUCTURED QUERY LANGUAGE

SQL-uses a combination of relational algebra and relational calculus constructs.

The SQL language has several parts:

Data Definition Language (DDL): The SQL DDL provides commands for defining relation schemas, deleting relations and modifying relation schemas.

Data manipulation Languages (DML): The SQL DML includes a query language based on both relational algebra and the tuple relation calculus. It includes commands to insert tuples into, delete tuples from and modify tuples in the database.

View Definition: The SQL DDL includes commands for defining views.

Transaction Control: SQL includes commands for specifying the beginning and ending of transactions.

Embedded SQL and Dynamic SQL: Embedded and dynamic SQL defines how SQL statements can be embedded within general-purpose programming languages such as C, C++, Java, PL/I, COBOL, Pascal and FORTRAN.

Integrity: The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are allowed.

Authentication: The SQL DDL includes commands for specifying access rights to relations and views.

3.1 Basic Structure

The basic structure of an SQL expression consists of three clauses: select, from and where.

The select clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.

The from clause corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

The where clause correspond to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the clause.

A SQL has the form

Select A₁, A₂, ..., A_n from r₁, r₂, ..., r_m where p

Where A_i – an attribute, r_i .relation and p-predicate.

Eg: *Select * from employee;*

Select emp_no, emp_name from employee where emp_city='ooty';

3.2 Data Definition Language

It is used to create a table, alter the structure of a table and also drop the table created.

Create command:

Syntax: create table <table name> (<columnname1> datatype(size), <columnname2> datatype(size),.....);

Eg: *create table employee (ename varchar2(10), eid number(5), address varchar2(10), salary number(5), designation varchar2(10));*

To view the table structure:

Syntax: desc <table name>

Eg: *desc employee*

Alter Command: It is used to add a new column or modify existing column definitions.

Syntax: alter table <table name> **add** (<new column name1> datatype(size), <new column name2> datatype(size).....);

alter table <table name> **modify** (column definition);

Eg: *alter table employee add (age number(2));*

alter table employee modify (eid number(8));

Truncate command: This is used to delete that records but retain the structure.

Syntax: truncate table <table name>

Eg: *truncate table employee;*

Drop Command: This is used to delete a table.

Syntax: drop table <table name>

Eg: *drop table employee;*

3.3 DML Commands

Insert command:

Syntax: insert into <table name> values (a list of data values)

// create table employee (ename varchar2(10), eid number(5), salary number(5));

Eg: *insert into employee values ('ABC', 50, 1000)*

Update Command: Changes be made by using update command.

Syntax: update <table name> set <field=value,...> where <condition>;

Eg: *update employee set eid=100 where ename='ABC';*

Delete Command: Rows can be deleted using delete command.

Syntax: delete from <table name> where <condition>

Eg: *delete from employee where eid=100;*

THE RENAME OPERATION:

SQL provides a mechanism for renaming both relations and attributes. It uses the as clause, taking the form

<Old name> as <new name>.

The “as” clause can appear in both the select and from clause.

Eg: *select ename as employee_name from employee;*
 alter table employee rename to emp;

STRING OPERATIONS:

The most commonly used operation on strings is pattern matching using the operator like.

- Percent (%) The % character matches any substring.
- Underscore (_): The _ Character matches any character.

Eg: Retrieve the names of the employees whose name is starting with the character ‘a’

Select ename from employee where name like ‘a%’;
Select ename from employee where name like ‘a_ _ _’;

ORDER BY CLAUSE:

The order by clause causes the tuples in the result of a query to appear in sorted order.

Eg: *select * from employee order by salary;*
 *select * from employee order by salary desc, eid asc;*

ARITHMETIC OPERATORS:

Standard arithmetic operations used are +, -, *, /

Eg: *select ename, eid, salary*10 from employee;*

SET OPERATIONS:

The SQL operations union, intersect and except operate on relations and correspond to the relational algebra operations $\cup, \cap, -$

UNION OPERATION:

It returns all distinct rows selected by both queries (It eliminates duplicate rows)

Eg: *select dno from employee **union** select dno from department;*

INTERSECT OPERATION:

It returns only rows that are common to both the queries.

Eg: *select dno from employee **intersect** select dno from department;*

EXCEPT OPERATION:

It returns all distinct rows selected only by the first query and not by the second.

Eg: *select dno from employee **except** select dno from department;*

AGGREGATE FUNCTIONS:

Aggregate functions are functions that take a collection of values as input and return a single value. SQL offers five built in aggregate function.

1. Average: avg
2. Minimum: min
3. Maximum: max
4. Total: sum
5. Count: count

Eg: *select ename, **avg**(salary), **max**(salary), **min**(salary) from employee;*

NULL values:

SQL allows the use of null values to indicate absence of information about the value of an attribute.

Eg: *select ename from employee where salary is null;*

NESTED SUBQUERIES:

A sub query is a select from where expression that is needed within another query.

Eg: *select ename, designation from employee where designation=(select designation from employee where ename='aaaa');*

Sub queries that return a set of values:

If a sub query can return more than one value you must specify how the returned values should be used in the where values.

Insert ANY or ALL between the comparison operators (=,!=,<,>,...) and the sub query.

Eg: *select dno, salary, designation from employee where salary ANY (select salary from employee where dno=10) order by salary desc;*

select dno, salary, designation from employee where salary ALL (select salary from employee where dno=10) order by salary desc;

Sub queries that return a list of values

IN and NOTIN will be used.

IN → = ANY; NOTIN → != ALL

Eg: *select ename, salary, designation from employee where dno=10 and job in (select designation from employee where dno=30);*

select ename, salary, designation from employee where dno=10 and job notin (select designation from employee where dno=30);

Sub queries that return more than one column

Eg: *select ename, designation from employee where (designation, salary)=(select designation, salary from employee where ename='aaa');*

Multiple sub queries

Eg: *select * from employee where designation in (select designation from employee where ename='aaa') or salary \rightarrow (select salary from employee where ename='bbb') order by designation, salary;*

Sub queries with multiple tables

Eg: *select * from employee where job=(select job from employee where dno=(delete dno from department where location='Chennai'));*

3.4 Different Types of Joins

- Simple join
- Self join
- Outer join

Simple join: It retrieves rows from two tables having a common column types.

- a) Equi join
- b) Non equi join

Equi Join: It combines rows that have equivalent values for the specified columns.

Eg: *select * from employee, department where employee.dno = department.dno;*

Non equi join: It specifies the relationship between columns belonging to different tables by making use of relational operators ($>$, $<$, $>=$, \leq , $<>$)

JOIN

Join is a query in which data is retrieved from two or more table. A join matches data from two or more tables, based on the values of one or more columns in each table.

Need for joins

In a database, where the tables are normalized one table may not give you all the information about a particular entity. For example, the employee table gives only the department ID, so if we want to know the department name and the manage name for each employee, and then you will have to get the information from the Employee and Department table. In other words, we will have to join the two tables.

So for comprehensive data analysis, we must assemble data from several tables. The relational model having made to partition the data and put in the different tables for reducing data redundancy and improving data independence relies on the join operation to enable to perform ad hoc queries that will combine the related data which resides in more than one table

Different types on joins are:

- Inner Join
- Outer Join
- Natural Join

Inner Join

It returns the matching rows from the tables that are being joined.

Consider following two relations:

- i) employee(emp_name, city)
- ii) Employee_salary(emp_name, dept, salary)

These two relations are shown in below figures:

Employee	
Emp_name	City
Hari	Pune
Raju	Mumbai
Sekar	Chennai
Jay	Delhi

Employee_salary		
Emp_name	Department	Salary
Hari	Computer	100000
Raju	Mechanical	80000
Ashok	Civil	65000
Jay	IT	90000

Eg: `select employee.emp_name, employee_salary.salary from employee inner join employee_salary on employee.emp_name=employee_salary.emp_name;`

Emp_name	Salary
Hari	100000
Raju	80000
Jay	90000

Eg: *select * from employee inner join employee_salary on employee.emp_name= employee_salary.emp_name;*

Emp_name	City	Emp_name	Department	Salary
Hari	Pune	Hari	Computer	100000
Raju	Mumbai	Raju	Mechanical	80000
Jay	Delhi	Jay	IT	90000

Outer Join:

When tables are joined using inner join, rows which contain matching in the join predicated are returned. Sometimes we need both matching and non-matching rows returned for the tables that are being joined. This kind of an operation is known as an outer join.

An outer is an extended form of the inner join. In this, the rows in one table having no matching rows in the outer table will also appear in the result table with nulls.

Types of outer join

The outer join can be any one of the following:

- Left outer join
- Right outer join
- Full outer join

Left outer join:

The left outer join returns matching rows from the tables being joined, and also non-matching rows from the left table in the result and places null values in the attributes that come from the right table.

Eg: *select employee.emp_name, employee_salary.salary from employee left outer join employee_salary on employee.emp_name= employee_salary.emp_name;*

Emp_name	Salary
Hari	100000
Raju	80000
Jay	90000
Sekar	null

Right outer join:

The right outer join returns matching rows from the tables being joined, and also non-matching rows from the right table in the result and places null values in the attributes that come from the left table.

Eg: `select employee.emp_name, employee_salary.salary, employee_salary.city from employee right outer join employee_salary on employee.emp_name= employee_salary.emp_name;`

Emp_name	City	Salary
Hari	Pune	100000
Raju	Mumbai	80000
Jay	Delhi	90000
Ashok	null	65000

3.5 Integrity Constraints

It is a mechanism used to prevent invalid data entry into the table.

Types

- Domain integrity constraints
- Entity integrity constraints
- Referential integrity constraints

Domain integrity constraints

Types

- a) Not null constraint
- b) Check constraint

Not null constraint: It is used to enforce that the particular column will not accept null values.

Eg: *create table employee (eid number(5) not null, ename varchar2(2));*

Check constraint: It is used to specify the conditions that each row must satisfy.

Eg: *create table employee (ename varchar2(10), eid number(5), salary number(5) constraint sal check (salary 500));*

sal- constraint name.

Entity integrity constraints

Types

- a) Unique Constraint
- b) Primary Key Constraint

Unique Constraint: It allows null values for the column and it is used to prevent duplication of values.

Eg: *create table employee (ename varchar2(10), eid number(5) unique)*

Primary Key Constraint: It will not allow null values for the column and it is used to prevent duplication of values.

Eg: *create table employee (ename varchar2(10), eid number(5) primary key, address varchar2(10));*

Referential integrity constraints

To establish a parent child relationship between two tables having a common column, we can use referential integrity constraints.

To implement this, we should define the column in the parent table as primary key and that same column in the child table as a foreign key referring to the corresponding parent entry.

Eg: *create table employee (ename varchar2(10), eid number(5) primary key, dno number(5) constraint fdno references department(dno), salary number(5));*

fdno- Constraint name

Before enabling these constraints, we ensure that dno column of the respective tables have been defined with either unique key or primary key constraint.

3.6 Stored Procedure

PL/SQL supports two types of sub programs. They are

- Procedures
- Function

Procedures are usually used to perform any specific task and functions are used to compute a value.

PROCEDURES

A Procedure is a sub program that performs a specific action. They syntax for creating a procedure is given below:

Syntax: Create or replace procedure <proc_name> [parameter_list] is

```
<local declaration>
Begin
(executable statements)
[exception] (exception handlers)
End;
```

A procedure has two parts:

- ❖ Specification
- ❖ Body

The procedure specification begins with the keyword **procedure** and ends with the procedure name or parameter list. The procedure body begins with the keyword **is** and ends with the keyword **end**.

Syntax to execute a procedure is given below:

```
SQL> exec <proc_name> (parameters);
```

```
Eg:   create or replace procedure interest (ir1 real,ir2 real)
      is old_bal number(9);
      c_id account.acc_no % type;
      cursor a is select balance,acc_no from account;
      begin
      open a;
      loop
      fetch a into old_bal,c_id;
      exit when a % notfound;
      if old_bal<5000
      then
      DBMS_OUTPUT.PUT_LINE('Balance less than min'||c_id);
      elsif old_bal between 5000 and 10000
      then
      update  account  set  balance=old_bal+(old_bal*ir1)  where
      acc_no=c_id;
      else
      update  account  set  balance=old_bal+(old_bal*ir2)  where
      acc_no=c_id;
      end if;
      end loop;
      close a;
      end;
```

Types of parameters pass to subprogram

There are three types of parameters: in, out and inout.

- i) In parameter: The in parameter mode is used to pass values to the subprogram when invoked.
- ii) Out parameter: The out parameter mode is used to return values to the caller of a subprogram.
- iii) Inout parameter: The inout parameter is used to pass initial values to subprogram, when invoked and it also returns updated values to the caller.

Eg: create or replace largest_value (a in number, b inout number, c out number) is

```
begin
c:=100;
if a>b and a>c then
b:=a;
c:=a;
else if b>a and b>c then
c:=b;
else
b:=c;
end if;
dbms_output.put_line('largest value='||b);
end;
```

FUNCTIONS:

A function is a subprogram that computes a value.

The syntax for creating a function is given below:

```
create or replace function <function_name>[argument list] return
datatype is (local declaration)
```

```
begin
(executable statements)
[exception]
(exception handler)
end;
```

Eg: create or replace largest_number (a in number, b in number, c in number) return number is

```
begin
if a>b and a>c then
return a;
else if b>a and b>c then
return b;
else
return c;
end if;
end;
```

3.7 Triggers

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. To design a trigger mechanism, we must need two requirements:

- 1) Specify when a trigger is to be executed. This broken up into an *event* that causes the trigger to be checked and as *condition* that must be satisfied for trigger execution to proceed.
- 2) Specify the *actions* to be taken when the trigger executes.

The above model of triggers is referred to as the *event-condition-action* model for trigger.

The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. Once we enter into trigger into the database, the DB system takes on the

responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

Need for triggers:

Triggers are useful mechanism for altering humans for starting certain tasks automatically, when certain conditions are met. For example, suppose that, instead of allowing negative account balances, the bank deals with overdrafts by setting the account balance to zero, and creating a loan in the amount of the overdraft. The bank gives this loan a loan_no identical to the account number of the overdrawn account.

For this example, the condition for executing the trigger is an update to the account relation that results in a negative balance value.

Suppose that Jones withdrawal of some money from an account made the account balance negative. Let 't' denote account tuple with a negative balance value. The actions to be taken are:

- Insert a new tuple 's' in the loan relation with

S[loan_no]=t[acc_no]

S[branch_name]=t[branch_name]

S[amount]=t[balance]

- Insert a new tuple 'u' in the borrower relation with

U[cus_name]=”jones”

U[loan_no]=t[acc_no]

- Set 't'[balance] to 0.

As another example of the use of the triggers, suppose a warehouse wishes to maintain a minimum inventory of each item; when the inventory level of an item falls below the minimum level, an order should be placed automatically. This is how the business rule can be implemented by triggers: on an update of the inventory level of an item, the trigger should

compare the level with the minimum inventory for the item, and if the level is at or below the minimum, a new order is added to an orders relation.

Triggers in SQL

SQL based database systems use triggers widely. Eg of SQL trigger is given below:

```
create trigger overdraft_trigger after update on account new row as
nrow
for each row
when nrow.balance<0
begin atomic
insert into borrower (select cus_name, acc_no) from depositor
where nrow.acc_no=depositor.acc_no);
insert into loan values(nrow.acc_no, nrow.branch_name,
nrow.balance);
update account set balance=0 where acc.acc_no=nrow.acc_no;
end
```

This trigger definition specifies that the trigger is initiated after any update of the relation current is executed. The referencing new row as clause creates a variable nrow, which stores the value of an updated row that the update. Then, for each row, when statement checks the value of balance whether it is less than zero or not.

If it is, then cus_name and acc_no of depositor for given acc_no inserted into borrower relation. A new tuple with values nrow.acc_no, nrow.branch_name and nrow.balance is inserted into loan relation and finally balance of that acc_no is set to zero in account relation.

3.8 Security

Security is a protection from malicious attempts to steal or modify data. The security should be provided at following levels:

1) Database system level:

Use authentication and authorization mechanisms to allow specific users access only to required data

2) Operating system level:

Operating system super-users can do anything they want to the database. Good OS level security is required.

3) Network Level:

Use encryption to prevent.

- Eavesdropping (unauthorized reading of messages).
- Masquerading (Pretending to be an authorized user or sending message supposedly from authorized users)

4) Physical Level:

- Here physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed.
- Computers must also be protected from floods, fire etc.,

5) Human level:

- Users must be screened to ensure that authorized users do not give access to intruders.
- Users should be trained on password selection and secrecy.

3.9 Advanced SQL Features

Some advanced SQL features are given below:

- Create a table with the same schema as an existing table:
Create table temp_account like account;
- SQL: 2003 allows subqueries to occur anywhere a value is required provided the subquery returns only one value. This applies to updates as well.

- SQL: 2003 allows subqueries in the from clause to access attributes of other relations in the from clause using the lateral construct:
Select c.customer_name, num_accounts from customer C,
lateral (select count(*) from account a where
a.customer_name=c.customer_name) as this_customer
(num_accounts);
- Merge construct allows batch processing of updates.
Merge into account as a using (select * from
funds_received ad f) on
(a.account_number=f.account_number) when matched then
update set balance=balance+f.amount;

3.10 Embedded SQL

Embedded SQLs are SQL statements included in the programming language. The programming language in which the SQL statements are included is called the host language. Some of the host languages are C, COBOL, Pascal, FORTRAN, PL/I etc., This embedded SQL source code is submitted to an SQL precompiler, which processes the SQL statements. Variables of the host language can be the program to be used by the SQL statements. The host language variables are used by the embedded SQL statements to receive of the SQL queries thus allowing the programming language to process the retrieved values.

Embedded SQL Features

Some of the basic features of the embedded SQL are given below:

- The embedded SQL statements appear in the host language. It usually does not matter whether the SQL statements are written in uppercase or lowercase. Usually the style of the host language is followed.

- Embedded SQL statements are prefixed by a delimiter- EXEC SQL- so that they can be distinguished from the host language statements.
- If an embedded SQL statement extends over multiple lines, the host language strategy for statement continuation is used.
- Every embedded SQL statement is terminated with a delimiter. In COBOL, it is END EXEC. In Adc, C, Pascal and PL/I, it is a semicolon.
- Host variables and SQL columns can have the same name.
- SQL statements can include reference to host variables. Such reference must be prefixed with a colon (:) to distinguish them from names of SQL objects like column names.

Advantages of Embedded SQL programs:

- The mixing of SQL statements with the programming language statements is an efficient way of merging the strengths of two programming environments. The programming language provides the flow of control, host variables, block structure, conditional branching, looping facilities and input/output functions etc. The SQL handles the database access and manipulation.
- The use of the precompiler shifts the CPU intensive parsing and optimization to the development phase. So the resulting executable program will be very efficient in the CPU usage.
- The program's run time interface to the private database routines is transparent to the application programmer.

Dynamic SQL

The dynamic SQL component of SQL allows programs to construct and submit SQL queries at run time. Using dynamic SQL, programs can

create SQL queries as strings at run time and can either have them executed immediately or have them prepared for subsequent use.

SQL defines standards for embedding dynamic SQL calls in a host language, such as C as in the following examples:

```
Char * sqlprog="update account set balance = balance*1.05 where  
acc_no=?"
```

```
EXEC SQL prepare dynprog from:sqlprog;
```

```
Char acc[10]="101";
```

```
EXEC SQL execute dynprog using :acc;
```

The dynamic SQL program contains `acc_no=?`, which is a placeholder for a value that is provided when the SQL program is executed.

Missing Information

- There are times when we want to add a tuple to a relation but don't have values for all attributes. Such values can be of type:
 - Non-existent value: We know that the attribute is inapplicable for that particular tuple
 - Unknown value: There is a value but we don't know it.
 - No-information: We don't know whether there is a value or not.
- DBMS's generally lump all these together.
- So what do we put in the tuple when information is missing?
- One possibility is to use a special value. Eg, if age is unknown, use 0, If SSN is unknown use 999999999999.
- This is not a proper solution to missing information.
- The proper solution to this problem is, use a value that is not in the domain. We call this a *null* value. This is not the same as a null/nil pointer,

- Primary keys mustn't have null values. Eg suppose regnum is the primary key of the relation and it consider that it can have null value.

RegNum	Surname	Firstname	DOB	Program
NULL	Smith	John	NULL	CSE
1234	John	Cena	12/02/82	CE
9645	Black	Burry	NULL	NULL

- As the primary key RegNum is having null value for some tuples, we can not uniquely identify tuple using primary key. Hence, primary key should not have null values.

3.11 View

A view is an imaginary table and it contains no data as such. Table contains data associated only with the table whereas views can contain data from multiple tables.

Creating a view:

To create views we are using create view statement.

Eg: create view v_1 as select eno, ename, dname from employee, department where employee.dno=department.dno;

Altering a view:

To alter a view that is to remove a column from a view, the replace command is used.

Eg: create or replace v_1 as select eno, ename, dname from employee, department where employee.dno=department.dno;

Destroying a view:

View can be dropped by using the DROP VIEW command.

Syntax: drop view <view name>

Eg: drop view v_1

Advantage:

- They provide table security by restricting access to a predetermined set of rows or columns of a table.
- They simplify commands for the user because they allow them to select information from multiple tables.
- They provide data in a different perspective than that of a base table by renaming columns without affecting the base table.

Summary

- ✓ SQL allows users to access data in relational database management systems
- ✓ There are three groups of SQL commands viz., DDL, DML and DCL.
- ✓ The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted.
- ✓ SQL language also includes syntax to update, insert, and delete records. These query and update commands together form the Data Manipulation Language (DML) part of SQL.
- ✓ The SQL Data Control Language (DCL) provides security for your database. The DCL consists of the GRANT, REVOKE, COMMIT, and ROLLBACK statements.
- ✓ Constraints are a way to limit the kind of data that can be stored in a table.
- ✓ Relational databases like SQL Server use indexes to find data quickly when a query is processed.

Key Words

SQL, DDL, DML, DCL, Constraints, Indexes

Self Assessment Questions

- 1) What does SQL stand for?
- 2) What SQL statement is used to delete table “Student”?
- 3) How can you insert a new record in table “Department”?
- 4) With SQL, how can you insert "GJU" as the "FName" in the "University" table?
- 5) How can you delete a record from table “student” where “RollNo”=GJU501?
- 6) Explain the use of Grant And Revoke Commands?
- 7) What are Transaction Control Language Commands?
- 8) Explain the ways to create a new user?

Chapter 4

Introduction to Distributed Databases

4.1 Distributed Databases

In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with one another through various communication media such as high speed networks or telephone lines. They do not share main memory or disks. The computer in a distributed system may vary in size and function, ranging from workstation up to mainframe systems.

The computers in distributed systems are referred to as sites or nodes.

- Distributed databases are geographically separated.
- They are separately administered.
- It has a slower interconnection.

Types of Distributed Database

Homogenous:

- i) All sites (nodes) have identical database management software, are aware of one another.
- ii) They are agreeing to cooperate in processing user's requests.
- iii) iLocal sites a portion of their autonomy in terms of their right to change schemas or DBMS software. That software must also cooperate with other sites in exchanging information about transactions to make transaction processing possible across multiple sites.

Heterogeneous:

- i) Different sites may use different schemas and different management system software. The sites may not be aware of one another.

- ii) They provide only limited facilities for cooperation in transaction processing.
- iii) The differences in schemas are often a major problem for query processing, while the difference in software becomes a difficulty for processing transactions that access multiple sites.

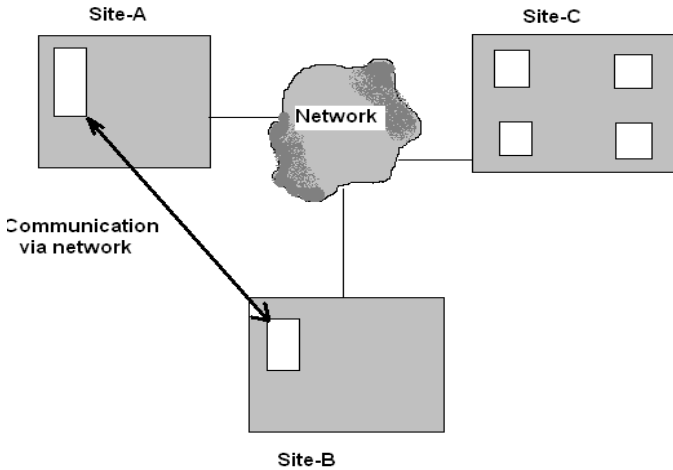


Fig 4.1. Distributed Architecture

Distributed Data Storage

Consider a relation 'r' that is to be stored in the database. There are two approaches storing this relation in the distributed database.

Replication

The system maintains several identical copies of the relation, and stores each replica at a different site.

Fragmentation

The system partitions the relation into several fragments, and stores each fragment at a different site.

4.2 Data Replication

There are a number of advantages and disadvantages of replication.

Availability

If one of the sites contain relation 'r' fails, then the relation 'r' can be found in another site, thus the system can continue to process queries involving 'r', despite the failure of one site.

Increased parallelism

In the case where the majority of accesses to the relation 'r' result in only the reading of the relation, then several sites can process queries involving 'r' in parallel. The more replicas of 'r' there are, the greater the chance that the needed data will be found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites

Increased overhead on update

The system must ensure that all replicas of a relation 'r' are consistent. Otherwise incorrect computation may result. Thus, whenever 'r' is updated, the update must be propagated to all sites contain replicas. The result is increased overhead.

4.3 Data Fragmentation

If relation 'r' is fragmented, 'r' is divided into a number of fragments r_1, r_2, \dots, r_n . These fragments contain sufficient information to allow reconstruction of the original relation r.

Two different schemas for fragmenting a relation:

- i) Horizontal fragmentation
- ii) Vertical fragmentation

Horizontal fragmentation: It splits the relation by assigning each tuple of 'r' to one or more fragments.

Vertical fragmentation: It splits the relation by decomposing the scheme R of relation 'r'.

4.4 Transparency

The user of a distributed system should not be required to know either where the data are physically located or how the data can be accessed at the specific local site. This characteristic is called as data transparency.

There are several terms of transparency.

- Fragmentation transparency: Users are not required to know how a relation has been fragmented.
- Replication transparency: Users view each data objects as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Users do not have to be concerned with what data objects have been replicated of where replicas have been placed.
- Location transparency: Users are not required to know the physical location of the data. The distributed database system should be able to find any data as long as data identifier is supplied by the user transaction.

4.5 Client/Server Database

As personal computers became faster, more powerful and cheaper there was a shift away from the centralized system architecture. Personal computers replace terminals connected to centralized systems. Correspondingly, personal computers assumed the user-interface functionality that used to be handled directly by the centralized systems. As a result, centralized systems today act as server systems that satisfy requests generated b client systems.

The figure shows the general structure of a client-server system.

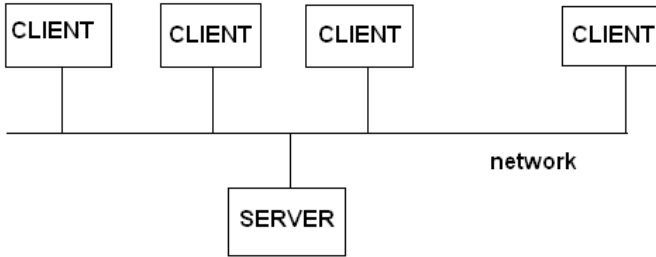


Fig.No:4.1 Client-Server System

Standards such as ODBC and JDBC were developed to interface clients with servers. Any client that uses the ODBC or JDBC interfaces can connect to any server that provides the interface.

The front-end (client) of a client/server system consists of tools such as forms, report writers, and graphical user interface facilities. The back end (server) manages access to structures, query evaluation and optimization, concurrency control, and recovery. The interface between the client and server is through SQL, or through an application program.

The figure shows front-end and back-end functionality.

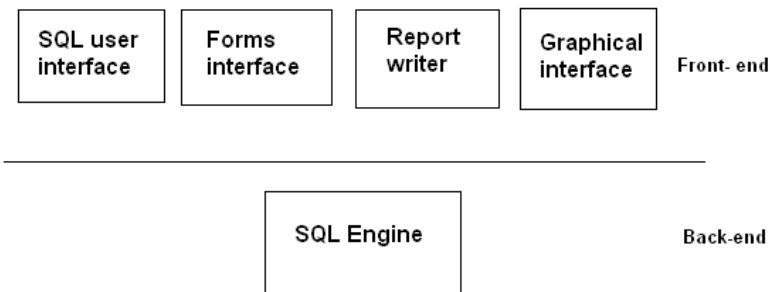


Fig.No:4.2 Front-end and Back-end Functionality

The most widely accepted form of client/server systems is the three-tier architecture. The components of this architecture are:

- The presentation (GUI) or user services
- Business rules or Business services
- Data server or data services.

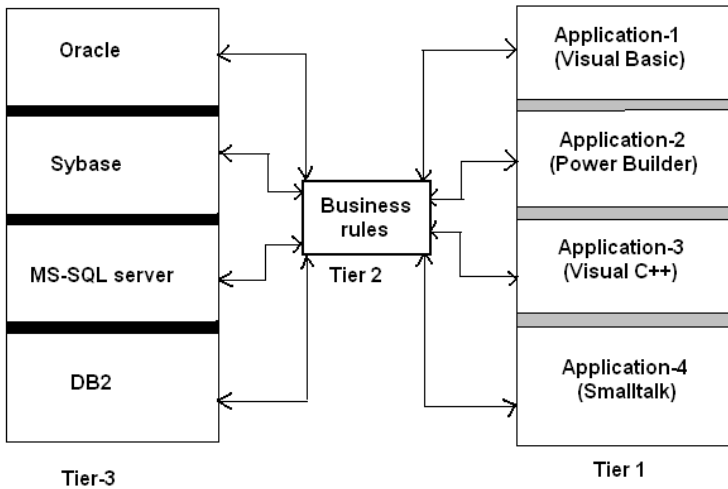


Fig.No:4.3 Three-tier Architecture

In three-tier architecture, all layers interact with each other. The client works as a query head and rules define how the query ought to be addressed and worked on. The database provides the answer and solutions to the client's query as determined by the rule base and the content of the database.

4.6 Benefits of client/server computing

1. Reduce operating costs: Client/server computing replaces expensive large systems with less expensive smaller ones networked together.

2. Platform Independence: It is not required to be locked into a single vendor's proprietary hardware or software environments.
3. Fast application Development: Advances in client/server tools have led to rapid application development.
4. Improved performance: With more processing power scattered throughout, the enterprise, the information is processed with faster response time.
5. Easier data access and processing: Online interactive client/server systems are a major improvement in usability over older batch-oriented systems. More people are able to access more data more quickly than ever before.

Summary

- ✓ Distributed databases bring the advantages of distributed computing to the database management domain.
- ✓ We can define a distributed database (DDB) as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user.
- ✓ The techniques that are used to break up the database into logical units, called fragments.
- ✓ Data replication, which permits certain data to be stored in more than one site
- ✓ Vertical fragmentation divides a relation "vertically" by columns.
- ✓ A fragmentation schema of a database is a definition of a set of fragments that includes all attributes and tuples in the database.
- ✓ An allocation schema describes the allocation of fragments to sites of the DDBS.

- ✓ The main feature that all DDBMS systems have in common is the fact that data and software are distributed over multiple sites connected by some form of communication network.

Key Words

Distributed Databases, Data Fragmentation, Replication, Distribution, Client Server Architecture

Self-Assessment Questions

- 1) What are the main reasons for and potential advantages of distributed
- 2) databases?
- 3) What are the main software modules of a DDBMS? Discuss the main functions of each of these modules in the context of the client-server architecture.
- 4) What is a fragment of a relation? What are the main types of fragments? Why is fragmentation a useful concept in distributed database design?
- 5) Why is data replication useful in DDBMSs? What typical units of data are replicated?
- 6) What is meant by data allocation in distributed database design? What typical units of data are distributed over sites?
- 7) How is a horizontal partitioning of a relation specified? How can a relation be put back together from a complete horizontal partitioning?
- 8) How is a vertical partitioning of a relation specified? How can a relation be put back together from a complete vertical partitioning?
- 9) Discuss what is meant by the following terms: degree of homogeneity of a

- 10) DDBMS, degree of local autonomy of a DDBMS, federated DBMS, distribution transparency, fragmentation transparency, replication, transparency, multi database system.
- 11) Discuss the naming problem in distributed databases.
- 12) Discuss the different techniques for executing an equijoin of two files located at different sites. What main factors affect the cost of data transfer?

Chapter 5

Relational Database Design

5.1 Pitfalls in Database Design

- Redundant information in tuples
- Update anomalies

Redundant Information in Tuple

One goal of schema design is to minimize the storage space that the base relations occupy. Grouping attributes into relation schemas has a significant effect on the storage.

For (eg), compare the space used by the two base relations EMPLOYEE and DEPARTMENT with EMP_DEPT base relation which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT.

In EMP_DEPT, the attribute values pertaining to a particular department are repeated for every employee who works for that department. In contrast, each department's information appears once in DEPARTMENT relation.

i.e, EMP_DEPT-(ename, eid, DOB, Address, dno, dname, dMGRid)

Update Anomalies

Types

- insertion anomalies
- Deletion anomalies
- Modification anomalies

Insertion anomalies: These can be differentiated into two types

- i) To insert a new employee tuple into EMP_DEPT, we must include the attribute values for the department that the employee works

for, or nulls (if the employee does not work for a department as yet).

- It is a difficult to insert a new department that has no employee as yet in EMP_DEPT relation. The only way is to place null values in the attributes for employee. This causes a problem because eid is a primary of EMP_DEPT.

ii) Deletion anomalies: If we delete for EMP_DEPT an employee tuple that happens to represents the last employee working for a particular department, the information concerning that department is lost from the database.

iii) Modification anomalies: In EMP_DEPT, if we change the value of one of the attributes of a particular department, say the manager of department 10 we must update the tuples of all employees work in that department. Otherwise the database will become inconsistent.

5.2 Functional Dependencies

Functional dependencies play a key role in differentiating good database design from bad database designs. A functional dependency is a type of constraint that is a generalization of the notation of key.

Definition:

A functional dependency is denoted is denoted by $X \rightarrow Y$, between two sets of attribute X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R.

The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$ also $t_1[Y] = t_2[Y]$.

This means that the values of the Y component of a tuple in r depends on, or are determined by, the values of the X component.

The values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. Y is functionally dependent on X.

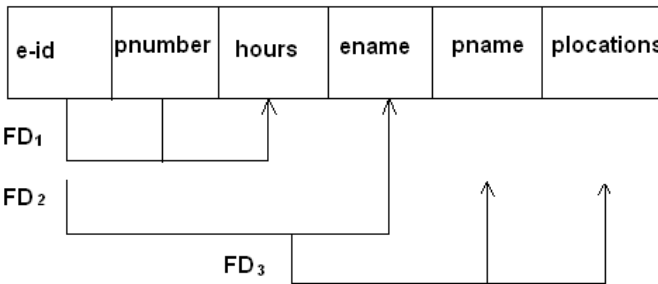
EMP-PROJ= {eid, pnumber, hours, ename, pname, plocation }

Eg., Consider the relation schema EMP-PROJ from the semantics of the attributes.

The following functional dependencies should hold.

- a) $Eid \rightarrow ename$
- b) $Pnumber \rightarrow \{pname, plocation\}$
- c) $\{eid, pnumber\} \rightarrow hours$

Diagrammatic notation for displays FD's



Each FD is displayed as a horizontal line. The LHS attributes of the FDs are connected by vertical lines to the lines representing FD, while the RHS attributes are connected by arrows pointing toward the attributes.

Inference Rules for Functional Dependencies

F is the set of functional dependencies that are specified on relation schema R. Numerous other FDs hold in all legal relation instance that satisfy the dependencies in F.

The set of all such dependencies are called closure of F and is denoted by F.

$$F = \{Eid \rightarrow \{ename, dob, address, dno\}\}$$

$$dno \rightarrow \{dname, dMGRid\}$$

We can infer the following additional FDs from F:

$$eid \rightarrow \{dname, dMGRid\}$$

$$dno \rightarrow dname, eid \rightarrow eid$$

An FD $X \rightarrow Y$ is inferred from a set of dependencies F specifies on R if $X \rightarrow Y$ holds in every relation state r that is legal extension of R. (ie) whenever r satisfies all the dependencies in F, $X \rightarrow Y$ also holds in r.

The closure F^+ of F is the set of all FDs that can be inferred from F. Inference rules that can be used to infer new dependencies from a given set of dependencies.

IR1 (reflexive rule): If $X \supseteq Y$ then $X \rightarrow Y$.

IR2 (Augmentation rule): $\{X \rightarrow Y\} \neq XZ \rightarrow YZ$

IR3 (Transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \neq X \rightarrow Z$

IR4 (Decomposition (or) Projective Rule): $\{X \rightarrow YZ\} \neq X \rightarrow Y$

IR5 (Union or Additive rule): $\{X \rightarrow Y, X \rightarrow Z\} \neq X \rightarrow YZ$

IR6 (Pseudo Transitive Rule): $\{X \rightarrow Y, WY \rightarrow Z\} \neq WX \rightarrow Z$

- IR1 states that a set of attributes always determines itself or any of its subsets. Because IR1 generates dependencies that are always true. Such dependencies are called trivial.
- IR2 says that adding same set of attributes to both LHS and RHS of a dependency results another valid dependency.
- IR4 we can remove attributes from RHS.

The inference rules IR1 through IR3 are known Armstrong's Inference Rules (or) Armstrong's Axioms. Thus for each set of attributes X,

we determine the set X^+ of attributes that are functionally determined by X based on F . X^+ is called the closure of X under F .

Algorithm: Calculate X^+ (Closure of Attributes Sets)

$X^+ := X$;

Repeat

For each functional dependency x in X^+ apply reflexivity and augmentation rules on x add the resulting functional dependencies to ' X '.

For each pair of functional dependencies x_1 and x_2 can be combined using transitivity add the resulting functional dependency X^+ .

Until X^+ does not change further.

5.3 Canonical Cover

An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

Definition: Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F .

- Attribute A is extraneous in α if $A \in \alpha$, and F logically implies $\{F - \{\alpha \rightarrow \beta\}\} \cup \{\{\alpha - A\} \rightarrow \beta\}$.
- Attribute A is extraneous in β if $A \in \beta$, and the set of functional dependencies

$\{F - \{\alpha - \beta\}\} \cup \{\alpha \{\beta - A\}\}$ logically implies F

A canonical cover F_c for F is a set of dependencies such that F logically implies all dependencies in F_c , and F_c logically implies all dependencies in F . F_c must have the following properties.

- No functional dependency in F_c contains an extraneous attribute.

- Each left side of functional dependency in F_c is unique. (ie) there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in such that $\alpha_1 = \alpha_2$.

(eg) Consider the following set F of functional dependencies on schema (A, B, C).

$$A \rightarrow BC$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

Compute the canonical cover for F.

- There are two functional dependencies with the same set of attributes on the left side of the arrow.

$$A \rightarrow BC$$

$$A \rightarrow B$$

These FD;s will be combine into $A \rightarrow BC$

- A is extraneous in $AB \rightarrow C$ because F logically implies $\{F == \{A \rightarrow BC\} \cup \{B \rightarrow C\}$. This assertion is true because $B \rightarrow C$ is already is our set of FDs.
- C is extraneous in $A \rightarrow BC$, since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$

Canonical cover is $A \rightarrow B \quad B \rightarrow C$.

Dependency Preservation:

This property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations.

5.4 Normalization

Normalization of data is a process of analyzing the given relation schema based on their FDs and primary keys to archive the desirable properties of

- i) Minimizing redundancy
- ii) Minimizing the insertion, deletion and update anomalies.

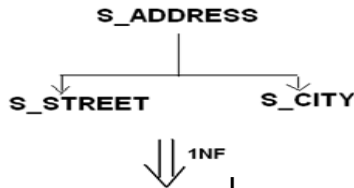
First Normal Form (1NF)

It states that the domain of an attribute must include only *atomic (Simple, indivisible) values* and that the value of any attribute in a tuple must be *a single value* from the domain of that attribute.

TABLE-A

S#	S_NAME	S_ADDRESS	P#	P_NAME	P_CITY	P_STATUS	QTY
S001	HCL	North Street, Chennai	P001	Mouse	Delhi	100	150
S002	IBM	West Street, Madurai	P001	Mouse	Mumbai	120	200
S002	IBM	West Street, Madurai	P002	Key Board	Pune	75	150
S003	DELL	South Street, Coimbatore	P003	Hard Disk	Delhi	100	180
S004	HCL	East Street, Trichy	P004	Dvd Drive	Pune	75	200

In the table-A the supplier address contains street and city. 1NF states that the domain of an attribute must include only atomic (Simple, indivisible) values. So we have to split S_ADDRESS into S_STREET and S_CITY

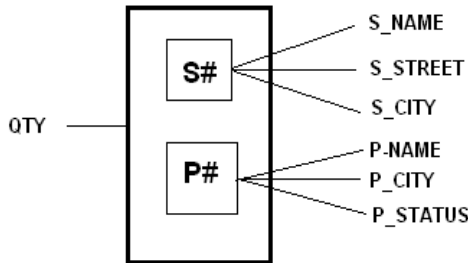


S#	S_NAME	S_STREET	S_CITY	P#	P_NAME	P_CITY	P_STATUS	QTY
S001	HCL	North	Chennai	P001	Mouse	Delhi	100	150
S002	IBM	West	Madurai	P001	Mouse	Mumbai	120	200
S002	IBM	West	Madurai	P002	Key Board	Pune	75	150
S003	DELL	South	Coimbatore	P003	Hard Disk	Delhi	100	180
S004	HCL	East	Trichy	P004	Dvd Drive	Pune	75	200

Second Normal Form (2NF)

A relation is said to be in the second normal form, if it is already in the first normal form and it has *no partial dependency (or) full functional dependency*.

In the above table B the key attributes are S# and P#. The non-key attributes must depend on key attribute. But the Table-B, S-NAME, S_STREET and S-CITY depend on S# and P_NAME, P_CITY and P_STATUS depends on P#. Only QTY depends on both key attributes S# and P#. So we have to split the table.



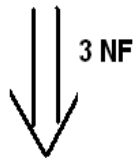
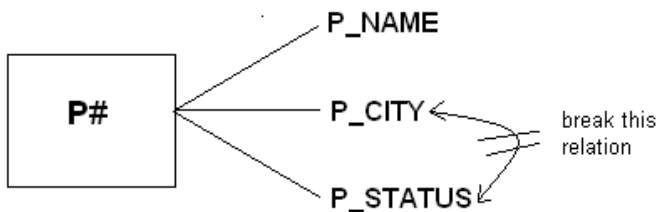
S#	S_NAME	S_STREET	S_CITY
S001	HCL	North	Chennai
S002	IBM	West	Madurai
S003	DELL	South	Coimbatore
S004	HCL	East	Trichy

P#	P_NAME	P_CITY	P_STATUS
P001	Mouse	Delhi	100
P001	Mouse	Mumbai	120
P002	Key Board	Pune	75
P003	Hard Disk	Delhi	100
P004	Dvd Drive	Pune	75

Third Normal Form (3NF)

A relation is said to be in the third normal form, if it is already in the second normal form and it has *no transitive dependency*.

In the above table C and E doesn't have transitive dependency. But in the Table-D the non-key attributes P_NAME, P_CITY and P_STATUS depend on P#. But P_STATUS transitively depends on P_CITY. So we have to break the table.



<u>P#</u>	P_NAME	P_CITY
P001	Mouse	Delhi
P001	Mouse	Mumbai
P002	Key Board	Pune
P003	Hard Disk	Delhi
P004	Dvd Drive	Pune

<u>P_CITY</u>	P_STATUS
DELHI	100
MUMBAI	120
PUNE	75

BOYCE-CODD NORMAL FORM (BCNF)

A relation is said to be in Boyce-Codd normal form if it is already in the third normal form and *every determinant is a candidate key*. It is a stronger version of 3NF.

Determinant: It is any field (Simple field or composite field) on which some other field is fully functionally determinant.

Comparison of BCNF and 3NF

1. 3NF design is always dependency preserving and lossless dependency preserving is difficult to achieve in BCNF sometimes.
2. BCNF strictly removes transitive dependency.
3. BCNF relation is in 3NF, but reverse is not possible

FOURTH NORMAL FORM (4NF)

A relation is said to be in the fourth normal form if it is already in BCNF and it has *no multi valued dependency*.

Consider an unnormalized relation that contains information about supplier, product and quantity. Each tuple contains supplier number, product number repeated for each supplier name and product name repeated for each product name.

<u>S#</u>	<u>P#</u>	<u>QTY</u>
S001	P001	100
	P002	150
S002	P001	100
	P002	150
S003	P001	200

Fourth normal form separates independent multivalued facts stored in one table into separate tables. So we rewrite the unnormalized data of the above table into normalized form.

<u>S#</u>	<u>P#</u>	<u>QTY</u>
S001	P001	100
S001	P002	150
S002	P001	100
S002	P002	150
S003	P001	200

FIFTH NORMAL FORM (5NF)

A relation is said to be in 5NF if it is already in 4NF and it has **no join dependency**.

Consider the Supplier_Product_Project relation.

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001

Supplier_Product_Project is obtainable if all the three projections are joined. Two projections if joined do not give back the original Supplier_Product_Project relations. We'll get extra tuple while we are joining two projections. If all the three relations are joined, then only we'll get the exact relations. This is called join dependency.

First we split the Supplier_Product_Project relation into three projections.

S#	P#
S001	P001
S001	P002
S002	P001

a) Supplier_Product

P#	J#
P001	J001
P001	J002
P002	J001

b) Product_Project

J#	S#
J001	S001
J002	S001
J001	S002

c) Project_Supplier



Join over parts

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001
S002	P001	J002

Extra Tuple

Supplier_Product & Product_Project

Join over
Project_Supplier

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001

Summary

- ✓ While designing a relational schema semantics of attributes, reducing the redundant values in a tuple, reducing null values in tuples and avoiding generation of spurious tuples are some of the issues that need to be taken care of.
- ✓ Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.
- ✓ Redundancy arises when a relational schema forces an association between attributes that is not natural. Functional dependencies can be used to identify such situations and suggest refinements to the schema.
- ✓ A functional dependency is a property of the semantics of the attributes in a relation. The semantics indicate how attributes relate to one another, and specify the functional dependencies between attributes.
- ✓ A table is in first normal form (1NF) if and only if all columns contain only atomic values, that is, each column can have only one value for each row in the table.
- ✓ A superkey is a set of one or more attributes, which, when taken collectively, allows us to identify uniquely an entity or table. Any subset of the attributes of a superkey that is also a superkey, and not reducible to another superkey, is called a candidate key.
- ✓ A primary key is selected arbitrarily from the set of candidate keys to be used in an index for that table.
- ✓ A table R is in Boyce-Codd normal form (BCNF) if for every nontrivial FD $X \rightarrow A$, X is a superkey.

Key Words

Database Design, Modification Anomalies, Decomposition, Functional Dependency, Normalisation, First Normal Form, Second Normal Form, Third Normal Form, BCNF, Lossless Join, Dependency Preservation, Super key, Candidate Key, Primary Key

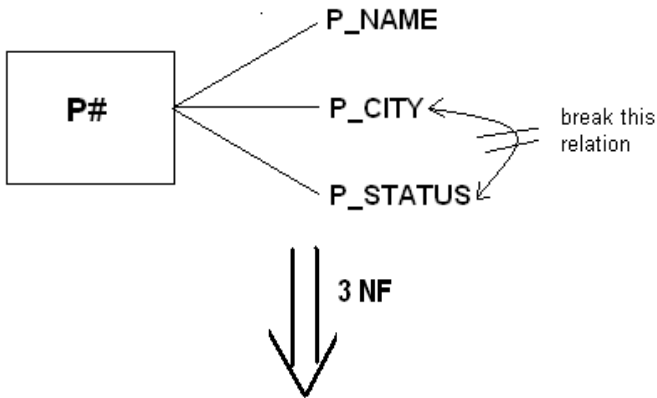
Self-Assessment Questions

- 1) What are the various guidelines that need to be taken care of while designing a relational schema?
- 2) Describe update, insert and delete anomalies with the help of examples.
- 3) Define functional dependencies?
- 4) Explain the inference rules?
- 5) Explain the concept of multi valued dependency?
- 6) What does the term unnormalized relation refer to? How did the normal forms develop historically?
- 7) Write down all the rules for normalization and explain with example.
- 8) Define first, second, and third normal forms when only primary keys are considered.

Third Normal Form (3NF)

A relation is said to be in the third normal form, if it is already in the second normal form and it has *no transitive dependency*.

In the above table C and E doesn't having transitive dependency. But in the Table-D the non-key attributes P_NAME, P_CITY and P_STATUS depends on P#. But the P-STATUS transitively depends on P_CITY. So we have to break the table.



<u>P#</u>	P_NAME	P_CITY
P001	Mouse	Delhi
P001	Mouse	Mumbai
P002	Key Board	Pune
P003	Hard Disk	Delhi
P004	Dvd Drive	Pune

<u>P_CITY</u>	P_STATUS
DELHI	100
MUMBAI	120
PUNE	75

BOYCE-CODD NORMAL FORM (BCNF)

A relation is said to be in Boyce-Codd normal form if it is already in the third normal form and *every determinant is a candidate key*. It is a stronger version of 3NF.

Determinant: It is any field (Simple field or composite field) on which some other field is fully functionally determinant.

Comparison of BCNF and 3NF

1. 3NF design is always dependency preserving and lossless dependency preserving is difficult to achieve in BCNF sometimes.
2. BCNF strictly removes transitive dependency.
3. BCNF relation is in 3NF, but reverse is not possible

FOURTH NORMAL FORM (4NF)

A relation is said to be in the fourth normal form if it is already in BCNF and it has *no multi valued dependency*.

Consider an unnormalized relation that contains information about supplier, product and quantity. Each tuple contains supplier number, product number repeated for each supplier name and product name repeated for each product name.

<u>S#</u>	<u>P#</u>	<u>QTY</u>
S001	P001	100
	P002	150
S002	P001	100
	P002	150
S003	P001	200

Fourth normal form separates independent multivalued facts stored in one table into separate tables. So we rewrite the unnormalized data of the above table into normalized form.

<u>S#</u>	<u>P#</u>	<u>QTY</u>
S001	P001	100
S001	P002	150
S002	P001	100
S002	P002	150
S003	P001	200

FIFTH NORMAL FORM (5NF)

A relation is said to be in 5NF if it is already in 4NF and it has **no join dependency**.

Consider the Supplier_Product_Project relation.

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001

Supplier_Product_Project is obtainable if all the three projections are joined. Two projections if joined do not give back the original Supplier_Product_Project relations. We'll get extra tuple while we are joining two projections. If all the three relations are joined, then only we'll get the exact relations. This is called join dependency.

First we split the Supplier_Product_Project relation into three projections.

S#	P#
S001	P001
S001	P002
S002	P001

a) Supplier_Product

P#	J#
P001	J001
P001	J002
P002	J001

b) Product_Project

J#	S#
J001	S001
J002	S001
J001	S002

c) Project_Supplier



Join over parts

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001
S002	P001	J002

Extra Tuple

Supplier_Product & Product_Project

Join over
Project_Supplier

S#	P#	J#
S001	P001	J001
S001	P001	J002
S001	P002	J001
S002	P001	J001

Summary

- ✓ While designing a relational schema semantics of attributes, reducing the redundant values in a tuple, reducing null values in tuples and avoiding generation of spurious tuples are some of the issues that need to be taken care of.
- ✓ Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.
- ✓ Redundancy arises when a relational schema forces an association between attributes that is not natural. Functional dependencies can be used to identify such situations and suggest refinements to the schema.
- ✓ A functional dependency is a property of the semantics of the attributes in a relation. The semantics indicate how attributes relate to one another, and specify the functional dependencies between attributes.
- ✓ A table is in first normal form (1NF) if and only if all columns contain only atomic values, that is, each column can have only one value for each row in the table.
- ✓ A superkey is a set of one or more attributes, which, when taken collectively, allows us to identify uniquely an entity or table. Any subset of the attributes of a superkey that is also a superkey, and not reducible to another superkey, is called a candidate key.
- ✓ A primary key is selected arbitrarily from the set of candidate keys to be used in an index for that table.
- ✓ A table R is in Boyce-Codd normal form (BCNF) if for every nontrivial FD $X \rightarrow A$, X is a superkey.

Key Words

Database Design, Modification Anomalies, Decomposition, Functional Dependency, Normalisation, First Normal Form, Second Normal Form, Third Normal Form, BCNF, Lossless Join, Dependency Preservation, Super key, Candidate Key, Primary Key

Self-Assessment Questions

- 1) What are the various guidelines that need to be taken care of while designing a relational schema?
- 2) Describe update, insert and delete anomalies with the help of examples.
- 3) Define functional dependencies?
- 4) Explain the inference rules?
- 5) Explain the concept of multi valued dependency?
- 6) What does the term unnormalized relation refer to? How did the normal forms develop historically?
- 7) Write down all the rules for normalization and explain with example.
- 8) Define first, second, and third normal forms when only primary keys are considered.

Chapter 6

Query Processing

6.1 Introduction

In this chapter we would like to discuss with you in detail about the query processing in DBMS. In this lesson you will find many repetitions from the previous chapters. That is included in this lesson purposefully in order to maintain the continuity and meaningfulness of the topic which we are going to deal with. So let's start the lecture with a bang.

In most database systems, queries are posed in a non-procedural language like SQL and as we have noted earlier such queries do not involve any reference to access paths or the order of evaluation of operations. The query processing of such queries by a DBMS usually involves the following four phases:

1. Parsing
2. Optimization
3. Code Generation
4. Execution

The parser basically checks the query for correct syntax and translates it into a conventional parse-tree (often called a query-tree) or some other internal representation.

If the parser returns with no errors, and the query uses some user-defined views, it is necessary to expand the query by making appropriate substitutions for the views. It is then necessary to check the query for semantic correctness by consulting the system catalogues and check for semantic errors and type compatibility in both expressions and predicate comparisons.

The optimizer is then invoked with the internal representation of the query as input so that a query plan or execution plan may be devised for

retrieving the information that is required. The optimizer carries out a number of operations. It relates the symbolic names in the query to data base objects and checks their existence and checks if the user is authorized to perform the operations that the query specifies.

In formulating the plans, the query optimizer obtains relevant information from the metadata that the system maintains and attempts to model the estimated costs of performing many alternative query plans and then selects the best amongst them. The metadata or system catalog consists of descriptions of all the databases that a DBMS maintains. Often, the query optimizer would at least retrieve the following information:

1. Cardinality of each relation of interest.
2. The number of pages in each relation of interest.
3. The number of distinct keys in each index of interest.
4. The number of pages in each index of interest.

The above information and perhaps other information will be used by the optimizer in modeling the cost estimation for each alternative query plan.

Considerable other information is normally available in the system catalog:

1. Name of each relation and all its attributes and their domains.
2. Information about the primary key and foreign keys of each relation.
3. Descriptions of views.
4. Descriptions of storage structures.
5. Other information including information about ownership and security issues.

Often this information is updated only periodically and not at every update/insert/delete.

Also, the system catalog is often stored as a relational database itself making it easy to query the catalog if a user is authorized to do so.

Information in the catalog is very important of course since query processing makes use of this information extensively. Therefore more comprehensive and more accurate information a database maintains the better optimization it can carry out but maintaining more comprehensive and more accurate information also introduces additional overheads and a good balance therefore must be found. The catalog information is also used by the optimizer in access path selection. These statistics are often updated only periodically and are therefore not always accurate.

An important part of the optimizer is the component that consults the metadata stored in the database to obtain statistics about the referenced relations and the access paths available on them. These are used to determine the most efficient order of the relational operations and the most efficient access paths. The order of operations and the access paths are selected from a number of alternate possibilities that normally exist so that the cost of query processing is minimized. More details of query optimization are presented in the next section.

If the optimizer finds no errors and outputs an execution plan, the code generator is then invoked. The execution plan is used by the code generator to generate the machine language code and any associated data structures. This code may now be stored if the code is likely to be executed more than once. To execute the code, the machine transfers control to the code which is then executed.

6.2 Query Optimization

Before query optimization is carried out, one would of course need to decide what needs to be optimized. The goal of achieving efficiency itself may be different in different situations. For example, one may wish to minimize the processing time but in many situations one would wish to minimize the response time. In other situations, one may wish to minimize the I/O, network time, memory used or some sort of combination of these

e.g. total resources used. Generally, a query processing algorithm A will be considered more efficient than an algorithm B if the measure of cost being minimized for processing the same query given the same resources using A is generally less than that for B.

To illustrate the desirability of optimization, we now present an example of a simple query that may be processed in several different ways. The following query retrieves subject names and instructor names of all current subjects in Computer Science that John Smith is enrolled in.

```
SELECT subject.name, instructor FROM student, enrolment, subject
WHERE
```

```
student.student_id = enrolment.student_id AND subject.subject_id =
enrolment.subject_id
```

```
AND subject.department = 'Computer Science' AND student.name =
'John Smith'
```

To process the above query, two joins and two restrictions need to be performed. There are a number of different ways these may be performed including the following:

1. Join the relations student and enrolment, join the result with subject and then do the restrictions.
2. Join the relations student and enrolment, do the restrictions, join the result with subject
3. Do the restrictions, join the relations student and enrolment, join the result with subject
4. Join the relations enrolment and subject, join the result with student and then do the restrictions.

Here we are talking about the cost estimates. Before we attempt to compare the costs of the above four alternatives, it is necessary to understand that estimating the cost of a plan is often non-trivial. Since normally a database is disk-resident, often the cost of reading and writing to disk dominates the

cost of processing a query. We would therefore estimate the cost of processing a query in terms of disk accesses or block accesses.

Estimating the number of block accesses to process even a simple query is not necessarily straight forward since it would depend on how the data is stored and which, if any, indexes are available. In some database systems, relations are stored in packed form, that is, each block only has tuples of the same relation while other systems may store tuples from several relations in each block making it much more expensive to scan all of a relation.

Let us now compare the costs of the above four options. Since exact cost computations are difficult, we will use simple estimates of the cost. We consider a situation where the enrolment database consists of 10,000 tuples in the relation student, 50,000 in enrolment, and 1,000 in the relation subject. For simplicity, let us assume that the relations student and subject have tuples of similar size of around 100 bytes each and therefore we can accommodate 10 tuples per block if the block is assumed to be 1 Kbytes in size. For the relation enrolment, we assume a tuple size of 40 bytes and thus we use a figure of 25 tuples/block. In addition, let John Smith be enrolled in 10 subjects and let there be 20 subjects offered by Computer Science. We can now estimate the costs of the four plans listed above.

The cost of query plan (1) above may now be computed. Let the join be computed by reading a block of the first relation followed by a scan of the second relation to identify matching tuples (this method is called nested-scan and is not particularly efficient. We will discuss the issue of efficiency of algebraic operators in a later section). This is then followed by the reading of the second block of the first relation followed by a scan of the second relation and so on. The cost of $R \bowtie X \bowtie S$ may therefore be estimated as the number of blocks in R times the number of blocks in S. Since the number of blocks in student is 1000 and in enrolment 2,000, the total number of blocks read in computing the join of student and enrolment is $1000 \times 2000 = 2,000,000$ block accesses. The result of the join is 50,000

tuples since each tuple from enrolment matches with a tuple from student. The joined tuples will be of size approximately 140 bytes since each tuple in the join is a tuple from student joined with another from enrolment. Given the tuple size of 140 bytes, we can only fit 7 tuples in a block and therefore we need about 7,000 blocks to store all 50,000 joined tuples. The cost of computing the join of this result with subject is $7000 \times 100 = 700,000$ block accesses. Therefore the total cost of plan (1) is approximately 2,700,000 block accesses.

To estimate the cost of plan (2), we know the cost of computing the join of student and enrolment has been estimated above as 2,000,000 block accesses. The result is 7000 blocks in size. Now the result of applying the restrictions to the result of the join reduces this result to about 5-10 tuples i.e. about 1-2 blocks. The cost of this restriction is about 7000 disk accesses. Also the result of applying the restriction to the relation subject reduces that relation to 20 tuples (2 blocks). The cost of this restriction is about 100 block accesses. The join now only requires about 4 block accesses. The total cost therefore is approximately 2,004,604.

To estimate the cost of plan (3), we need to estimate the size of the results of restrictions and their cost. The cost of the restrictions is reading the relations student and subject and writing the results. The reading costs are 1,100 block accesses. The writing costs are very small since the size of the results is 1 tuple for student and 20 tuples for subject. The cost of computing the join of student and enrolment primarily involves the cost of reading enrolment. This is 2,000 block accesses. The result is quite small in size and therefore the cost of writing the result back is small. The total cost of plan (3) is therefore 3,100 block accesses.

Similar estimates may be obtained for processing plan (4). We will not estimate this cost, since the above estimates are sufficient to illustrate that brute force method of query processing is unlikely to be efficient. The cost can be significantly reduced if the query plan is optimized. The issue of optimization is of course much more complex than estimating the costs

like we have done above since in the above estimation we did not consider the various alternative access paths that might be available to the system to access each relation.

The above cost estimates assumed that the secondary storage access costs dominate the query processing costs. This is often a reasonable assumption although the cost of communication is often quite important if we are dealing with a distributed system. The cost of storage can be important in large databases since some queries may require large intermediate results.

The cost of CPU of course is always important and it is not uncommon for database applications to be CPU bound than I/O bound as is normally assumed. In the present chapter we assume a centralized system where the cost of secondary storage access is assumed to dominate other costs although we recognize that this is not always true. For example, system R uses $\text{cost} = \text{page fetches} + w \text{ cpu utilization}$

When a query is specified to a DBMS, it must choose the best way to process it given the information it has about the database. The optimization part of query processing generally involves the following operations.

1. A suitable internal representation
2. Logical transformation of the query
3. Access path selection of the alternatives
4. Estimate costs and select best

We will discuss the above steps in detail.

Internal Representation

As noted earlier, a query posed in a query language like SQL must first be translated to an internal representation suitable for machine representation. Any internal query representation must be sufficiently powerful to represent all queries in the query language (e.g. SQL). The internal representation could be relational algebra or relational calculus

since these languages are powerful enough (they have been shown to be relationally complete by E.F. Codd) although it will be necessary to modify them from what was discussed in an earlier chapter so that features like Group By and aggregations may be represented. A representation like relational algebra is procedural and therefore once the query is represented in that representation, a sequence of operations is clearly indicated.

Other representations are possible. These include object graph, operator graph (or parse tree) and tableau. Further information about other representations is available in Jarke and Koch (1984) although some sort of tree representation appears to be most commonly used (why?). Our discussions will assume that a query tree representation is being used.

In such a representation, the leaf nodes of the query tree are the base relations and the nodes correspond to relational operations.

Logical Transformations

At the beginning of this chapter we showed that the same query may be formulated in a number of different ways that are semantically equivalent. It is clearly desirable that all such queries be transformed into the same query representation. To do this, we need to translate each query to some canonical form and then simplify.

This involves transformations of the query and selection of an optimal sequence of operations. The transformations that we discuss in this section do not consider the physical representation of the database and are designed to improve the efficiency of query processing whatever access methods might be available. An example of such transformation has already been discussed in the examples given. If a query involves one or more joins and a restriction, it is always going to be more efficient to carry out the restriction first since that will reduce the size of one of the relations (assuming that the restriction applies to only one relation) and therefore the cost of the join, often quite significantly.

Heuristic Optimization

In the heuristic approach, the sequence of operations in a query is reorganized so that the query execution time improves. Deterministic Optimization -- In the deterministic approach, cost of all possible forms of a query are evaluated and the best one is selected.

Common Subexpression

In this technique, common subexpressions in the query, if any, are recognised so as to avoid executing the same sequence of operations more than once.

Simple Hash Join Method

This method involves building a hash table of the smaller relation R by hashing each tuple on its hash attribute. Since we have assumed that the relation R is too large to fit in the main memory, the hash table would in general not fit into the main memory. The hash table therefore must be built in stages. A number of addresses of the hash table are first selected such that the tuples hashed to those addresses can be stored in the main memory.

The tuples of R that do not hash to these addresses are written back to the disk. Let these tuples be relation R'. Now the algorithm works as follows:

(a) Scan relation R and hash each tuple on its join attribute. If the hashed value is equal to one of the addresses that are in the main memory, store the tuple in the hash table. Otherwise write the tuple back to disk in a new relation R'.

(b) Scan the relation S and hash each tuple of S on its join attribute. One of the following three conditions must hold:

1. The hashed value is equal to one of the selected values, and one or more tuple of R with same attribute value exists. We combine the tuples of R that match with the tuple of S and output as the next tuples in the join.

2. The hashed value is equal to one of the selected values, but there is no tuple in R with same join attribute value. These tuple of S are rejected.

3. The hashed value is not equal to one of the selected values. These tuples are written back to disk as a new relation S' .

The above step continues till S is finished.

(c) Repeat steps (a) and (b) until either relation R' or S' or both are exhausted.

Grace Hash-Join Method

This method is a modification of the Simple Hash Join method in that the partitioning of R is completed before S is scanned and partitioning of S is completed before the joining phase. The method consists of the following three phases:

1. Partition R - Since R is assumed to be too large to fit in the main memory, a hash algorithm involves partitioning the relation into n buckets, each bucket corresponding to a hash table entry. The number of buckets n is chosen to be large enough so that each bucket will comfortably fit in the main memory.

2. Partition S - The second phase of the algorithm involves partitioning the relation S into the same number (n) of buckets, each bucket corresponding to a hash table entry. The same hashing function as for R is used.

3. Compute the Join - A bucket of R is read in and the corresponding bucket of S is read in. Matching tuples from the two buckets are combined and output as part of the join.

Hybrid Hash Join Method

The hybrid hash join algorithm is a modification of the Grace hash join method.

Aggregation

Aggregation is often found in queries given the frequency of requirements of finding an average, the maximum or how many times something happens. The functions supported in SQL are average, minimum, maximum, count, and sum. Aggregation can itself be of different types including aggregation that only requires one relation, for example finding the maximum mark in a subject, or it may involve a relation but require something like finding the number of students in each class. The latter aggregation would obviously require some grouping of the tuples in the relation before aggregation can be applied.

Summary

- ✓ The query processing by a DBMS usually involves the four phases Parsing Optimization, Code Generation, and Execution.
- ✓ The parser basically checks the query for correct syntax and translates it into a conventional parse-tree (often called a query-tree) or some other internal representation.
- ✓ The optimizer invokes the internal representation of the query as input so that a query plan or execution plan may be devised for retrieving the information that is required.
- ✓ Heuristic optimization often includes making transformations to the query tree by moving operators up and down the tree so that the transformed tree is equivalent to the tree before the transformations.

Key Words

Query Processing, Heuristics, Query Optimisation, Self-Assessment

Sample Questions

- 1) Discuss the following rules governing the manipulation of relational algebraic expressions?
- 2) Explain the algorithms used for the processing of join operation?
- 3) Explain how you could estimate costs while performing q
References/Suggested Readings
- 4) Data Base Systems by C.J.Date
- 5) Data Base Management Systems by Alexis Leon, Mathews Leon
- 6) <http://databases.about.com/library>

Chapter 7

TEXT AND DATA MINING

7.1 Introduction

A potentially useful intellectual tool for researchers is the ability to make connections between seemingly unrelated facts, and as a consequence create inspired new ideas, approaches or hypotheses for their current work. This can be achieved through a process known as text mining (or data mining if it focuses on non-bibliographic datasets).

Text/data mining currently involves analysing a large collection of often unrelated digital items in a systematic way and to discover previously unknown facts, which might take the form of relationships or patterns that are buried deep in an extensive collection.

These relationships would be extremely difficult, if not impossible, to discover using traditional manual-based search and browse techniques. Both text and data mining build on the corpus of past publications and build not so much on the shoulders of giants as on the breadth of past published knowledge and accumulated mass wisdom.

The claim currently being made for text and data mining is that they will speed up the research process and capitalise on work which has been done in the past in a new and effective way. However, a number of features need to be in place before this can happen. These include:

- ❖ Access to a vast corpus of research information
- ❖ In a consistent and interoperable form
- ❖ Freely accessible, without prohibitive authentication controls
- ❖ Covering digitised text, data and other media sources
- ❖ Unprotected by copyright controls (over creation of derivative works)
- ❖ A single point of entry with a powerful and generic search engine

- ❖ A sophisticated mechanism for enabling the machine (computer) to analyse the collection for hidden relationships

Currently the full potential for text/data mining is not being fulfilled because several of the above requirements are not being met. There are too many ‘silos’ of heavily protected document servers (such as those maintained independently by the many STM journal publishers) to provide the necessary critical mass of accessible data. There is also little interoperability between the various protocols and access procedures.

Text and data mining is still at an early stage in its development, but given the unrelated push towards an ‘open access’ environment (which undermines the ‘silo’ effect) text/data mining may become significant as a research tool within the next two to five years.

7.2 Historical Development

Forms of text and data mining have been around for some fifty years. The intelligence gathering community was an early recogniser of the usefulness of this technique. Artificial intelligence and diagnostics have also employed text and data mining procedures.

In the 1980’s abstracts in the MEDLINE database were used as a platform against which to test text mining approaches. Life science text has been used at the front-end of studies employing text mining largely because the payoffs in terms of drugs and health care are so high.

All this was a prelude to a shift in the way users came to terms with the information explosion. There were two more recent elements.

- The first is that ‘collecting’ digital material became different from the way physical collections were built up and used. In the print world filing cabinets became full of printed articles from which the user absorbed the content through some unclear form of osmosis. Now people find and collect things online.
- They build up collections, or personal libraries, of the digital items on their computers and laptops. The difference is that these personal

libraries – which often still go unread – are interrogated using more efficient electronic search and retrieval software

- The second change is that there is a new approach to digital ‘computation’. The processes of ‘search’ and ‘collections’ became disentangled. Google came along with its multiple services which raised the searching/discovery stakes. It offered access to a world of digital information much more extensive than that which was typical of a print-centric world.

The research community often assumes Google can reveal all the hidden secrets in the documents. But this is not the case, and it is the application of full-text mining software and data mining procedures which expose more of the relationships which exist between individual documents. These relationships are often hidden deeply within different parts of the growing mountain of documentation. Text mining builds on Google’s existence – it does not replace or compete with it.

To be really effective text and data mining requires access to large amounts of literature. This is the real challenge facing the widespread adoption of text/data mining techniques.

7.3 Working Principles

Text mining involves the application of techniques from areas such as information retrieval, natural language processing, information extraction and data mining. These various stages can be combined together into a single workflow.

Information Retrieval (IR) systems identify the documents in a collection which match a user’s query. The most well-known IR systems are search engines such as Google, which allows identification of a set of documents that relate to a set of key words. As text mining involves applying very computationally-intensive algorithms to large document collections, IR can speed up the discovery cycle considerably by reducing the number of documents found for analysis. For example, if a researcher is

interested in mining information only about protein interactions, he/she might restrict their analysis to documents that contain the name of a protein, or some form of the verb 'to interact', or one of its synonyms. Already, through application of IR, the vast accumulation of scientific research information can be reduced to a smaller subset of relevant items.

Natural Language Processing (NLP) is the analysis of human language so that computers can understand research terms in the same way as humans do. Although this goal is still some way off, NLP can perform some types of analysis with a high degree of success. For example:

- ❖ Part-of-speech tagging classifies words into categories such as nouns, verbs or adjectives
- ❖ Word sense disambiguation identifies the meaning of a word, given its usage, from among the multiple meanings that the word may have
- ❖ Parsing performs a grammatical analysis of a sentence. Shallow parsers identify only the main grammatical elements in a sentence, such as noun phrases and verb phrases, whereas deep parsers generate a complete representation of the grammatical structure of a sentence

The role of NLP is to provide the systems in the information extraction phase (see below) with linguistic data that the computer needs to perform its 'mining' task.

Information Extraction (IE) is the process of automatically obtaining structured data from an unstructured natural language document. Often this involves defining the general form of the information that the researcher is interested in as one or more templates, which are then used to guide the extraction process. IE systems rely heavily on the data generated by NLP systems. Tasks that IE systems can perform include:

- ❖ Term analysis, which identifies the terms in a document, where a term may consist of one or more words. This is especially useful

for documents that contain many complex multi-word terms, such as scientific research papers

- ❖ Named-entity recognition, which identifies the names in a document, such as the names of people or organisations. Some systems are also able to recognise dates and expressions of time, quantities and associated units, percentages, and so on
- ❖ Fact extraction, which identifies and extracts complex facts from documents. Such facts could be relationships between entities or events
- ❖ A very simplified example of the form of a template and how it might be filled from a sentence is shown in Figure 1. Here, the IE system must be able to identify that ‘bind’ is a kind of interaction, and that ‘myosin’ and ‘actin’ are the names of proteins. This kind of information might be stored in a dictionary or an ontology, which defines the terms in a particular field and their relationship to each other. The data generated during IE are normally stored in a database ready for analysis by the final stage, that of data mining.

Data Mining (DM) (often known as knowledge discovery) is the process of identifying patterns in large sets of data. When used in text mining, DM is applied to the facts generated by the information extraction phase. Continuing with the protein interaction example, the researcher may have extracted a large number of protein interactions from a document collection and stored these interactions as facts in a separate database.

By applying DM to this separate database, the researcher may be able to identify patterns in the facts. This may lead to new discoveries about the types of interactions that can or cannot occur, or the relationship between types of interactions and particular diseases, and so on.

The results of the DM process are put into another database that can be queried by the end-user via a suitable graphical interface. The data

generated by such queries can also be represented visually, for example, as a network of protein interactions.

Text mining is not just confined to proteins, or even biomedicine though this is an area where there has been much experimentation using text/data mining techniques. Its concepts are being extended into many other research disciplines. Increasing interest is being paid to multilingual data mining: the ability to gain information across languages and cluster similar items from different linguistic sources according to their meaning.

Text and data mining is a burgeoning new interdisciplinary field in support of the scientific research effort. There are a number of examples of such services in existence though few have so far broken through to become mainstream processes within the scientific research effort.

Examples of Text Mining

Research and development departments of major companies, including IBM and Microsoft, are researching text mining techniques and developing programmes to further automate the mining and analysis processes. Text mining software is also being researched by different companies working in the area of search and indexing in general as a way to improve their results. There are also a large number of companies that provide commercial computer programmes.

- AeroText - provides a suite of text mining applications for content analysis. Content used can be in multiple languages
- AlchemyAPI - SaaS-based text mining platform that supports 6+ languages. Includes named entity extraction, keyword extraction, document categorization, etc.
- Autonomy - suite of text mining, clustering and categorization solutions for a variety of industries
- Endeca Technologies - provides software to analyze and cluster unstructured text.

- Expert System S.p.A. - suite of semantic technologies and products for developers and knowledge managers.
- Fair Isaac - leading provider of decision management solutions powered by advanced analytics (includes text analytics).
- Inxight - provider of text analytics, search, and unstructured visualisation technologies. (Inxight was bought by Business Objects that was bought by SAP AG in 2008)
- Nstein - text mining solution that creates rich metadata to allow publishers to increase page views, increase site stickiness, optimise SEO, automate tagging, improve search experience, increase editorial productivity, decrease operational publishing costs, increase online revenues
- Pervasive Data Integrator - includes Extract Schema Designer that allows the user to point and click identify structure patterns in reports, html, emails, etc. for extraction into any database
- RapidMiner/YALE - open-source data and text mining software for scientific and commercial use.
- SAS - solutions including SAS Text Miner and Teragram - commercial text analytics, natural language processing, and taxonomy software leveraged for Information Management.
- SPSS - provider of SPSS Text Analysis for Surveys, Text Mining for Clementine, LexiQuest Mine and LexiQuest Categorize, commercial text analytics software that can be used in conjunction with SPSS Predictive Analytics Solutions.
- Thomson Data Analyzer - enables complex analysis on patent information, scientific publications and news.

- LexisNexis - provider of business intelligence solutions based on an extensive news and company information content set. Through the recent acquisition of Datops LexisNexis is leveraging its search and retrieval expertise to become a player in the text and data mining field.
- LanguageWare - Text Analysis libraries and customization tooling from IBM
- There has been much effort to incorporate text and data mining within the bioinformatics area. The main developments have been related to the identification of biological entities (named entity recognition), such as protein and gene names in free text. Specific examples include:
 - XTractor - discovering new scientific relations across PubMed abstracts. A tool to obtain manually annotated relationships for proteins, diseases, drugs and biological processes as they get published in the PubMed bibliographic database.
 - Chilobot - tool for finding relationships between genes or gene products.
 - Information Hyperlinked Over Proteins (iHOP) "A network of concurring genes and proteins extends through the scientific literature touching on phenotypes, pathologies and gene function. By using genes and proteins as hyperlinks between sentences and abstracts, the information in PubMed can be converted into one navigable resource"
 - FABLE - gene-centric text-mining search engine for MEDLINE
 - GoPubMed - retrieves PubMed abstracts for search queries, then detects ontology terms from the Gene Ontology and Medical Subject Headings in the abstracts and allows the

user to browse the search results by exploring the ontologies and displaying only papers mentioning selected terms, their synonyms or descendants.

- LitInspector - gene and signal transduction pathway data mining in PubMed abstracts.
- PubGene - co-occurrence networks display of gene and protein symbols as well as MeSH, GO, PubChem and interaction terms (such as "binds" or "induces") as these appear in MEDLINE records (that is, PubMed titles and abstracts).
- PubAnatomy - interactive visual search engine that provides new ways to explore relationships among Medline literature, text mining results, anatomical structures, gene expression and other background information.
- NextBio - life sciences search engine with a text mining functionality that utilises PubMed abstracts and clinical trials to return concepts relevant to the query based on a number of heuristics including ontology relationships, journal impact, publication date, and authorship.

Text mining not only extracts information on protein interactions from documents, but it can also go one step further to discover patterns in the extracted interactions. Information may be discovered that would have been extremely difficult to find, even if it had been possible to read all the documents – which in itself is an increasing impossibility.

7.4 Organisations involved in Text and Data Mining

A number of centres have been set up to build on the text and data mining techniques. These include:

The National Centre for Text Mining (NaCTeM)

The National Centre for Text Mining (NaCTeM) is the first publicly-funded text mining centre in the world. It provides text mining services for the UK academic community. NaCTeM is operated by the University of Manchester with close collaboration with the University of Tokyo and Liverpool University. It provides customised tools, research facilities and offers advice and provides software tools and services.

Funding comes primarily from the Joint Information Systems Committee (JISC) and two of the UK Research Councils, the BBSRC (Biotechnology and Biological Sciences Research Council) and EPSRC (Engineering and Physical Sciences Research Council). The services of the Centre are available free of charge for members of higher and further education institutions in the UK.

With an initial focus on text mining in the biological and biomedical sciences, research has since expanded into other areas of science, including the social sciences, the arts and humanities. Additionally, the Centre also organises and host workshops and tutorials and provides access to document collections and text-mining resources.

School of Information at University of California, Berkeley

In the United States, the School of Information at University of California, Berkeley is developing a program called BioText to assist bioscience researchers in text mining and. Analysis. A grant of \$840,000 has been received from the National Science Foundation to develop the search mechanism. Currently BioText runs against a database of some 300 open access journals. The project leader of BioText is Professor Marti Hearst.

TEMIS

TEMIS is a software organisation established in 2000 which has centres in France, Germany and the USA. It focuses on pharmaceutical and publishing applications and has a client base which includes Elsevier,

Thomson and Springer. Thomson Scientific uses TEMIS to rescue data which had been captured in another format (for example, the BIOSIS format) and restructures the data according to the Thompson house style. It can process three documents per second. MDL, a former Elsevier company, uses TEMIS to automatically extract facts. A new database is created from analysing text documents. Springer uses TEMIS to enrich journals with hyperlinks into major reference works.

UK PubMed Central (UKPMC)

Text and data mining will come under the agreed phased extensions of UKPMC developments, as adopted by the management and advisory group for UKPMC. Most of the text and data mining work will be channeled via University of Manchester (notably NaCTeM) and European Bioinformatics Institute (EBI), joint collaborators with the British Library on UKPMC.

Initially it was felt that the text mining work being done by Manchester and EBI were competitive, but it appears that EBI is focusing on indexing and NaCTeM on natural language processing. The ‘best of breed’ from both organisations will be incorporated to create a prototype text mining tool. Some parts already exist – genome and protein listing for example. But it is felt that the work is still some two years away from creating a fully effective system and interface. These tools will eventually plug-ins into UKPMC. NaCTeM’s ‘myexperiment’ will also be made available within British Library’s RIC (see previous ICSTI Insight on ‘Workflows’) However the work for RIC is a different project with NaCTeM.

Google

Google’s Search Appliance 6.0 searches 30 million documents and provides search across a variety of other internal and external sources - including file shares, intranets, databases, applications, hosted services and content management systems.

Microsoft Research

Microsoft's Text Mining, Search, and Navigation group undertakes research in information retrieval, machine learning, data mining, computational linguistics, and human-computer interaction. It is deeply involved with the academic community and works closely with the various Microsoft product teams. The primary contact is Chris Burges.

Other Centres

There are other developments taking place in text and data mining – notably at Sheffield University in the UK with their work on the Cancer grid and the National Cancer Research centre. There are also individuals who are pushing the boundaries of text and data mining. Professor Carol Goble, from Manchester University, is one such expert.

Another more controversial figure in this area is Dr Peter Murray Rust from University of Cambridge who has done much to advance the cause of text and data mining in the field of chemistry whilst challenging the very basis of the current scholarly publishing system. In effect we are seeing more and more peripheral use of text mining for specific applications, but so far it has not reached mainstream publishing activities for reasons outlined in the next section.

7.5 The Challenges

Intellectual Property Rights

As it stands, each publisher maintain their own 'digital silos' of information, and cross searching among these separate silos is undertaken more in the breach than the observance. Yet it is only through the dismantlement of the legal protections around such silos that effective text and data mining can take place. The greater the common document source being mined the more effective the results achieved.

Such a cross-silo approach could be achieved in a number of ways. Either through agreements reached with the existing publishers to allow cross searching of text files among publisher silos on a licence basis or

through the adoption by the industry at large of open access as the standard business model. Most databases which include a sweat of the brow activity may only be accessible if the customer has paid a subscription or licence fee.

Even if this hurdle is overcome, the terms of the subscription and licence may be such that the owner of the database will still not allow reformulation of the material in any way. Several commercial journal publishers have raised concerns that the creation of ‘derivative works’ could undermine the commercial opportunity facing their primary journals.

Nevertheless, a number of STM publishers have recently reached an agreement with the Wellcome Trust to allow text mining to take place on works which Wellcome has funded (through payment of author fees) but only within the terms of the licences agreed with each publisher.

This still remains restrictive as far as text mining is concerned. Licences would need to be changed to open up the database to unrestricted mining activity, even if they lead to derivative works being created. This is what the user community wants, this is what Science needs, this is what the traditional publishing industry wants to avoid. But we are seeing further instances of the licences slowly being adapted to meet this user demand.

The UK Model NESLI2 Licence for Journals

In the UK there is a central negotiating service offered by JISC whereby the 180 or so higher education institutions can be reached through a centrally negotiated contracts. The Model NESLI2 Licence for Journals has been in place for a number of years and has been the basis for special terms being given by publishers to UK academe as a result of the wide audience reached through a single negotiated agreement.

In May 2009 a new clause was added to the NESLI License which will come into effect for 2010. The relevant clause is given below

Permitted Uses

The clause has in fact only been added to the NESLi2 website in the past few months and will be used in the negotiations with publishers for 2010. As such we do not yet know how publishers may react to it. It was brought into the licence as JISC (and their negotiating partner, Content Complete) were aware that the academic community is likely to want this facility.

There will only be negotiations with a limited number of publishers for 2010, because there are already multi-year agreements with several of the big publishers in particular (for example, with Elsevier there are 3-4 year licences in place). It means, according to Content Complete, that these larger publishers will only have to confront this clause when their existing licence term expires and the new one comes up for re-negotiation.

Nature Publishing Group

The Nature Publishing Group (NPG) is one publisher which anticipated such developments early on and has agreed to a wider application of text and data mining techniques against its content.

Researchers can now data mine and text mine author manuscripts from NPG journals archived in PubMed Central, UK PubMed Central (UKPMC) and other institutional and subject repositories. The terms were developed in consultation with the Wellcome Trust. Under NPG's terms of reuse, users may view, print, copy, download and text and data-mine the content for the purposes of academic research. Re-use should only be for academic purposes; commercial reuse is not permitted.

The articles will be accessible via the UK PubMed Central OAI service (UKPMC-OAI), an implementation of the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). NPG's re-use terms will be included in the metadata of these archived manuscripts. However, there is a limit to the number of words which can be reused without permission as a result of a successful text mining of the Nature publications. \

Also, articles published by Nature Publishing Group (NPG) which are made available through academic repositories remain subject to copyright. Any reuse is subject to permission from NPG. The relevant part of the NPG licence is:

Wholesale re-publishing is prohibited

Archived content may not be published verbatim in whole or in part, whether or not this is done for Commercial Purposes, either in print or online.

This restriction does not apply to reproducing normal quotations with an appropriate citation. In the case of text-mining, individual words, concepts and quotes up to 100 words per matching sentence may be reused, whereas longer paragraphs of text and images cannot (without specific permission from NPG).

Moral rights

- ❖ All re-use must be fully attributed. Attribution must take the form of a link - using the article DOI - to the published article on the journal's website.
- ❖ All re-use must ensure that the authors' moral right to the integrity of their work is not compromised.

The complete terms of reuse can be seen at: http://www.nature.com/authors/editorial_policies/license.html#terms

Derivative Works

Derivative works are those new publications which are the result of mashing-up material from a number of publishers' copyrighted material. There may be no original primary research in the derivative work but it may have importance nonetheless in giving insight into something new from the sum of the (published) parts.

As indicated, publishers are suspicious that creating derivative works through text mining might compromise the publishers' commercial

chances – that it will create a whole new publishing activity which is a parasite on mainstream publishing. As such the publishing industry feels that it should control it in some way and be the recipient for any new commercial returns which are derived from mining their earlier work in an automated way.

However, there is an interesting question about being able to isolate any one publisher's work included in any particular text mining output. Though the publisher's server may have been interrogated by the text mining software, connecting the results of the mining process back to any one of the original information sources may prove difficult. Multiple results may have been derived from a wide variety of text sources and how can one be given credit for any one item? Computers are logical, not creative. Computation using text mining to create a derivative work is essentially a mechanical activity. Derivative works can therefore be based on hundreds or thousands of separate copyrighted works. Isolating the original ownership of an idea or wording may be impossible.

Text and data mining also needs to encompass the creation of extracts, translations and summaries of developments in various fields. Some of these derivative works are mechanically produced but others, such as creating a translation, still need elements of human creativity. So much so that copyright may be vested in the derived translation. There is an outstanding legal question of who can determine what is included from whom (copyright owner) in a newly derived work?

Technical Issues

A key technical issue is whether text/data mining is undertaken from a single large accumulated database held centrally, or else whether a federated search system is adopted with knowbots being launched to pull in results from remote and privately held databases. A centralised database also raises issues of resources. Not only in terms of the infrastructure to support a large central file but also in the support services necessary to run it. Computation can take place in a more controlled environment on a

single aggregated database, though this may not always be possible for a variety of technical and IPR reasons.

A distributed model raises issues around data normalisation, of performance levels, of other standardisation issues. A distributed or federated system requires conformity by all involved to common metadata standards to allow effective cross reference and indexing. If the need is to rely on a federated approach the issue of trust arises – trust that the remote database of text and data will always be there, curated and consistent in its approach to metadata creation and full-text production.

In support of a federated approach to text and data mining one can see the emergence of ‘the cloud’ as a mechanism for processing large amounts of data using the existing powerful computer resources made available by organizations such as Amazon, Yahoo, Microsoft, HP, etc. A federated powerful processing infrastructure is in place ‘in the cloud’.

7.6 Implications of Text and Data Mining

Providing a text/data mining facility for Science requires a new means of collaboration between existing and future stakeholders to accept data and text mining as being effective and acceptable processes. In particular, that such mining does not eliminate any significant role currently being performed by stakeholders, that it does not raise challenges and barriers to text/data mining applications, that it does not threaten publishers and librarians and their existence.

There is the rub. The battle will be whether the advantages which text and data mining confer are sufficiently powerful and attractive to the research community to enable it to sweep objections aside. At present all we can hypothesise is that data and text mining will happen – is happening in select areas – and will be another driver for change in the march towards full electronic publishing over the next few years. But how soon depends on a number of factors. Intellectual property rights and their protection will be at the forefront of these.

Text and data mining creates a new way of using information. It opens the horizons of researchers. But to fully appreciate the scope of the technology it requires some training for the researcher and the inclusion within their research process of text/data mining techniques.

Besides that it needs access to a large document database. As has been mentioned, this creates problems with regard to licensing. But text miners need text, and they need it in a form which is useful for the text mining systems.

Open Access

A review of text and data mining is not complete if one ignores other underlying trends in scientific communication. One of these is the changing business models which have come about in the past 6-8 years (in effect since the Budapest Initiative in 2002, the Bethesda Statement and the Berlin Declaration in 2003).

Text mining is believed to have a considerable commercial value. This is particularly true in scientific disciplines; in which highly relevant (and therefore monetarisable) information is often contained within written text. In recent years publishers have been effecting improvements to their publication systems without opening the doors to text and data mining. Some of the general initiatives taken, such as Nature's proposal for an Open Text Mining Interface (OTMI) and NIH's common Journal Publishing Document Type Definition (DTD) which has been adopted by many of the larger publishers, do provide semantic cues to machines to answer specific queries contained within text, but without going as far as removing publisher barriers to public access.

However, an earlier ICSTI 'Insight' (January 2009) gave a detailed expose of the open access movement. As far as text and data mining is concerned the success of this process for information extraction relies heavily on the ability to interrogate as wide a source of digital collections as possible, unencumbered by access control and authentication procedures. Gradually we are seeing open access come about, both in the 'gold' (author

pays) and the ‘green’ (self-deposition of items in subject or institutional repositories). Though the tables on subscription and licensing systems (which have been favoured by publishers in the past) have not been overturned, there is a gradual erosion of open access into the subscription system.

Some pundits claim that the ‘green’ movement has achieved a 15% market share – though this is disputable – and the ‘gold’ route has a 20% share (based on 4,000 journals being OA, though these journals are often smaller in size and therefore exaggerates the true market share). Nevertheless, there has been recent growing acceptance from one important sector of the industry, notably the research funding agencies that support for open access in all its forms is growing.

This means that over the next few years the amount of text which will be available for text mining purposes will increase albeit gradually rather than explosively. With regard to the data sources which are available for mining, the picture is much better. Here there has been limited legacy to protect raw datasets with authentication and access rights protocols.

In fact the European Commission adopted the position of “Initiatives leading to wider dissemination of scientific information are necessary, especially with regard to journal articles and research data produced on the basis of public funding.” on 14 February 2007 at the time of a major international conference held in Brussels to discuss in effect the open access movement. At the time of this conference, commercial STM publishers agreed among themselves that ‘data’ should be a free resource – in effect washing their hands of the problems of managing and curating data as an information resource for the scientific community.

“Raw research data should be made freely available to all researchers. Publishers encourage the public posting of the raw data outputs of research. Sets or sub-sets of data that are submitted with a paper to a journal should wherever possible be made freely accessible to other scholars”.

Open access of data is therefore more ripe for mining activities, and this is reflected in some of the early text/data mining work involving genomes, proteins etc in the area of bioinformatics. Open access is a leading factor in bringing text and data mining to the Science community.

About Author



B.Santhosh Kumar
Associate Professor
Department of CSE (UG & PG)
SNS College of Technology
Sathy Main Road(NH-209)
Vazhiyampalayam
Saravanampatti Post
Coimbatore - 641 035

B.Santhosh Kumar was born at Ooty, India on March 6, 1980. He completed M.E in Computer Science and Engineering from Anna University Thiruchirappalli and he is pursuing Ph.D in Computer Science and Engineering from Anna University, Chennai. Presently he is working as Associate Professor in the Department of Computer Science & Engineering, SNS College of Technology, Affiliated to Anna University, Chennai. He started his career as Lecturer at Merit International Institute of Technology and Lecturer at CSI College of Engineering. He is a Life Member of International Association of Computer Science and Information Technology (IACSIT) and member in the Institute of Electrical and Electronics Engineering (IEEE), Computer Society of India (CSI) and International Association of Engineers (IAENG). His research interest includes Data Base Management Systems, Data Mining and Web Mining. For his credential he published more than 25 papers in refereed International journals and 25 papers in conferences. He has also acted as an editorial Board Member and reviewer of many International Journals and IEEE & ASDF Conferences.



**KONGUNADU PUBLICATIONS
INDIA PVT LTD**

ISBN-978-93-86770-44-8

