LOVELY PROFESSIONAL UNIVERSITY

MANAGING DATABASE

Edited By

Dr. Anil Sharma

ISBN: 978-93-87034-71-6

Printed by
EXCEL BOOKS PRIVATE LIMITED
Regd. Office: E-77, South Ext. Part-I, Delhi-110049
Corporate Office: 1E/14, Jhandewalan Extension, New Delhi-110055
+91-8800697053, +91-011-47520129
info@excelbooks.com/projects@excelbooks.com
internationalalliance@excelbooks.com
www.excelbooks.com

for
Lovely Professional University
Phagwara

CONTENTS

Unit 1:	Database Fundamentals *Harjinder Kaur, Lovely Professional University*	1
Unit 2:	Database Relational Model *Pooja Gupta, Lovely Professional University*	12
Unit 3:	Structured Query Language *Sartaj Singh, Lovely Professional University*	37
Unit 4:	Advanced SQL *Sarabjit Kumar, Lovely Professional University*	59
Unit 5:	Integrity Constraints *Pawan Kumar, Lovely Professional University*	72
Unit 6:	Relational Language and Database Design *Pawan Kumar, Lovely Professional University*	84
Unit 7:	Relational Database Design *Mandeep Kaur, Lovely Professional University*	106
Unit 8:	Normalization *Sahil Rampal, Lovely Professional University*	127
Unit 9:	Transaction Management *Pooja Gupta, Lovely Professional University*	148
Unit 10:	Datalog and Recursion *Mithilesh Kumar Dubey, Lovely Professional University*	166
Unit 11:	Recovery System *Balraj Kumar, Lovely Professional University*	179
Unit 12:	Query Processing and Optimization *Kamlesh Lakhwani, Lovely Professional University*	196
Unit 13:	Parallel Databases *Manmohan Sharma, Lovely Professional University*	219
Unit 14:	Application Development and Administration *Manmohan Sharma, Lovely Professional University*	232

SYLLABUS
Managing Database

Objectives: The course aims at providing the students through insight on database management principles and practices. The major objectives of the course are to: Knowledge of DBMS and in terms of use and design

- Describe the main features and function of the DBMS;
- Describe the features of relational database and E-R models;
- Experience with SQL queries;
- Experience ER diagrams;
- Discuss the concept of Transaction, Recovery, Concurrency and Security of DBMS

Managing Database

Sr. No.	Description
1.	**Introduction to fundamentals of DBMS:** Database applications, Purpose of database systems, Views of data, Database languages, Relational Databases, Database Design, Transaction Management
2.	**Relational Databases:** Relational Model, Structure of Relational databases, fundamental, additional and extended relational algebra operations, Null Values, Modification of database
3.	**SQL:** Data Definition, datatypes, schema definition, Basic structure of SQL Queries, Creating tables, DML - SELECT, INSERT, DELETE and UPDATE operations, DDL commands
4.	**SQL:** Set Operations – UNION, INTERSECT, EXCEPT, Aggregate Functions, NULL values, Nested subqueries, Complex queries, Views, Joined relations Advanced SQL: Integrity constraints, Authorization: GRANT, REVOKE
5.	**Relational Languages:** Tuple Relational calculus, Domain relational calculus, Query by Example, Datalog
6.	**Database design and ER model:** Overview of Design process, Entity relationship model, constraints, ER Diagrams, ER Design issues, Weak entity sets, extended ER features
7.	**Relational Database Design:** Features, Atomic Domains and first normal form, Functional dependency theory decomposition using functional dependencies, decomposition using Multivalued dependencies, More normal forms, database design process
8.	**Transaction Management:** Concept of Transaction, Transaction State, Implementation of atomicity and durability, concurrent execution, Serializability, Recoverability, Implementation of Isolation, testing for Serializability
9.	**Concurrency Control:** Lock based protocols, Timestamp based protocols, Validation based protocols, Deadlock handling, Insert and Delete operations, Weak levels of consistency
10.	**Recovery system:** Failure classification, storage structure, recovery and atomicity, log-based recovery, recovery with concurrent transactions, buffer management, failure with loss of nonvolatile storage

Unit 1: Database Fundamentals

Notes

CONTENTS

Objectives

Introduction

1.1 Database Management Systems (DBMS)

1.2 Database System Applications

1.3 Characteristics of the Database Approach

1.4 Advantages of DBMS

1.5 Disadvantages of DBMS

1.6 Database Architecture

1.7 Summary

1.8 Keywords

1.9 Self Assessment

1.10 Review Questions

1.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Define database management system
- Explain database system applications
- State the characteristics and the database approach
- Discuss the advantages and disadvantages of database
- Discuss the database architecture

Introduction

The information storage and retrieval has become very important in our day-to-day life. The old era of manual system is no longer used in most of the places. For example, to book your airline tickets or to deposit your money in the bank the database systems may be used. The database system makes most of the operations automated. A very good example for this is the billing system used for the items purchased in a super market. Obviously this is done with the help of a database application package. Inventory systems used in a drug store or in a manufacturing industry are some more examples of database. We can add similar kind of examples to this list.

Apart from these traditional database systems, more sophisticated database systems are used in the Internet where a large amount of information is stored and retrieved with efficient search engines. For instance, http://www.google.com is a famous web site that enables users to search for their favorite information on the net. In a database we can store starting from text data to very complex data like audio, video, etc.

Notes

1.1 Database Management Systems (DBMS)

A database is a collection of related data stored in a standard format, designed to be shared by multiple users. A database is defined as "A collection of interrelated data items that can be processed by one or more application programs".

A database can also be defined as "A collection of persistent data that is used by the application systems of some given enterprise". An enterprise can be a single individual (with a small personal database), or a complete corporation or similar large body (with a large shared database), or anything in between.

Example: A Bank, a Hospital, a University, a Manufacturing company.

Data

Data is the raw material from which useful information is derived. The word data is the plural of Datum. Data is commonly used in both singular and plural forms. It is defined as raw facts or observations. It takes variety of forms, including numeric data, text and voice and images. Data is a collection of facts, which is unorganized but can be made organized into useful information. The term Data and Information come across in our daily life and are often interchanged.

Example: Weights, prices, costs, number of items sold etc.

Information

Data that have been processed in such a way as to increase the knowledge of the person who uses the data. The term data and information are closely related. Data are raw material resources that are processed into finished information products. The information as data that has been processed in such way that it can increase the knowledge of the person who uses it.

In practice, the database today may contain either data or information.

Data Processing

The process of converting the facts into meaningful information is known as data processing. Data processing is also known as information processing.

Metadata

Data that describe the properties or characteristics of other data.

Data is only become useful when placed in some context. The primary mechanism for providing context for data is Metadata. Metadata are data that describe the properties, or characteristics of other data. Some of these properties include data definition, data structures and rules or constraints. The Metadata describes the properties of data but do not include that data.

It allows the database designer and users to understand what data Exit, what the data mean, and what the fine distinctions are between seemingly similar data items. The management of Metadata is at least as a crucial as meaning the associated data since data without clear meaning can be confusing, misinterpreted or erroneous.

Notes

1.2 Database System Applications

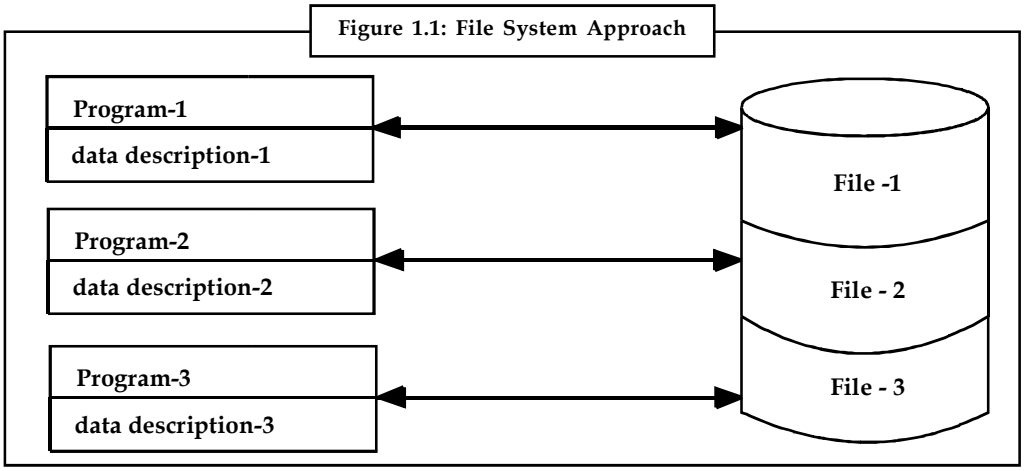
Databases are widely used. Here are some representative applications:

1. ***Banking:*** For customer information, accounts, and loans, and banking transactions.
2. ***Airlines:*** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner - terminals situated around the world accessed the central database system through phone lines and other data networks.
3. ***Universities:*** For student information, course registrations, and grades.
4. ***Credit card transactions:*** For purchases on credit cards and generation of monthly statements.
5. ***Telecommunication:*** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
6. ***Finance:*** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
7. ***Sales:*** For customer, product, and purchase information.
8. ***Manufacturing:*** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses / stores, and orders for items.
9. ***Human resources:*** For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

1.3 Characteristics of the Database Approach

A shared collection of logically related data along with the description of the data that suits to the needs of large enterprises.

Figure 1.1: File System Approach

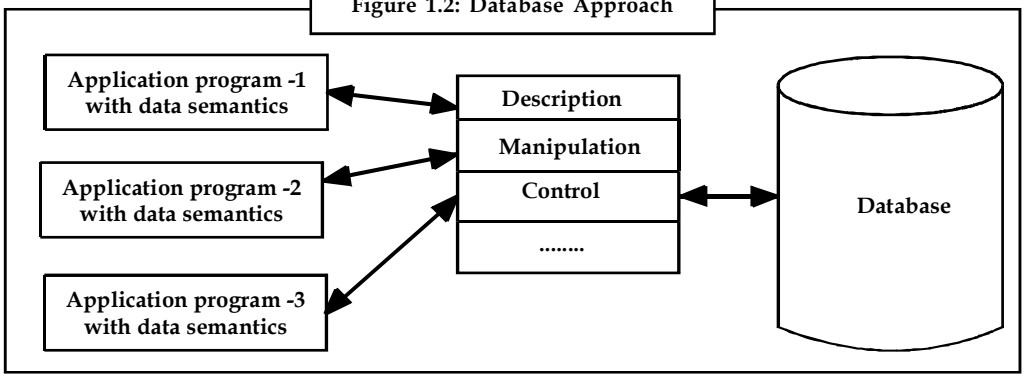


This unit describes the basic differences between the traditional way of processing, also called as file processing, and the database method of processing the data. Every operating system provides users to open, save, and close a file. The users can store appropriate information in these files. Take a look at the Figure 1.1 which shows the traditional file processing system that stores the program and data description in a file. The related information of a particular application is stored in various files named as File1, File2, etc., and these files are manipulated using Program1, Program2, etc. This is the method that was used in early days.

Notes

It means that without a DBMS, the data will simply be dumped into one or more files. For any updation, the files need to be opened and manually search for the line or record, update and then save the file. Now you can understand the difficulties involved in marinating this type of information storage.

Figure 1.2: Database Approach



With the advent of database systems, the file processing approach is no longer used. Now you can observe with the Figure 1.2 that the database is in the disk which in turn is controlled by the DBMS. In this approach the Application Program-1 along with its data semantics, Application Program-2 along with its data semantics, etc., interact with the database where the actual data and constraints are stored through the DBMS. The DBMS provides the necessary control and manipulation software modules for these application programs to access the data stored in the database.

This way the applications are free from the system dependent code and achieve program-data independence.

Task Find out the various sources of database management system.

Drawbacks of File Processing System

1. ***Catalog:*** In DBMS, the database structure is stored in a catalog and it also contains the storage details with constraints.

   The DMBS software must equally work with any number of database applications provided the catalog contains the structure and other details of that application. In file processing the data definition is part of the application program.

   Example: Record declaration in Pascal.

   Class or structure declaration in C++.

2. ***Program-data independence:*** In file processing, if changes are done in the structure of the file, then we may require changing the program design that accesses it. In DBMS the access programs are written independent of any specific files. This is called as program-data independence.

   The DBMS stores the data in such a way that the user need not be aware of these details. This concept is called as data abstraction and it may also be called as conceptual representation.

Notes

3. ***Views:*** A database may have many users and each one may be interested on a particular view of the application. A view is conceptually a table, but the records of this table are not stored in the database.

   Example: Consider the Student database in which we can think of two views:

   View 1: Students Grade in various courses. To obtain this information the tables Course and Grade_Report are to be joined and created as a view.

   View 2: If we want to know the Prerequisite Courses that a student needs to study, three tables are to be joined. These tables are nothing but Student, Section and Prerequisite.

4. ***Sharing and Transaction processing:*** A DBMS must provide control for various users trying to access the database.

   Example: Railway Reservation System with multiple counters.

   Whenever several users try to access the same application at the same time, we call this situation as concurrent transaction processing. Generally, the concurrent access is achieved with a simple Local Area Network (LAN). It is also possible to book railway tickets online i.e. through Internet.

1.4 Advantages of DBMS

One of the main advantages of using a database management system is that the organization can exert via the DBA, centralized management and control over the data. The database administrator is the focus of the centralized control. If any application requiring a change in the structure of a data record, the DBA makes the necessary modifications, which do not affect other applications or users of the record in question.

The following are the major advantages of using a Database Management System (DBMS):

1. ***Reduction of Redundancies:*** Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large mass of data. Another advantage of avoiding duplication is the elimination of the inconsistencies that tend to be present in redundant data files.

2. ***Data Independence and Efficient Access:*** Database application programs are independent of the details of auto representation and storage. In addition a DBMS provides efficient storage and retrieval mechanisms, including support for very large files, index structures and query optimization.

3. ***Data Integrity:*** Centralized control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity, which means that the data contained in the database is both accurate and consistent. Therefore, data values being entered for storage could be checked to ensure that they fall within a specified range and are of the correct format. For example, the value for the age of an employee may be in the range of 16 and 75. Also it should be ensured that if there is a reference to certain object, that object must exist. In the case of an automatic teller machine, for example a user is not allowed to transfer funds from a nonexistent savings account to a checking account.

4. ***Data Security:*** Confidential data must not be accessed by unauthorized persons. Different levels of security could be implemented for various types of data and operations.

Notes

5. ***Reduced Application Development Time:*** Since the DBMS provides several important functions required by applications, such as concurrency control and crash recovery, high level query facilities, etc., only application-specific code needs to be written.

6. ***Conflict Resolution:*** Since the database is under the control of the DBA, he should resolve the conflicting requirements of various users and applications.

   The DBA chooses the best file structure and access method to get optimal performance for the response-critical applications, while permitting less critical applications to continue to use the database, though with a relatively slower response.

7. ***Data Administration:*** By providing common base for a large collection of data that is shared by several users, a DBMS facilitates maintenance and data administration tasks. A good DBA can effectively ensure the fine-tuning, the data representation, periodic backups etc.

8. ***Concurrent Access and Crash Recovery:*** A DBMS supports the notion of a transaction and executes the actions of transactions in an interleaved fashion to obtain good performance, but schedules them in such a way as to ensure that conflicting operations are not permitted to proceed concurrently. Further, the DBMS maintains a continuous log of the changes to the data, and if there is a system crash, it can restore the database to a transaction- consistent state. That is, the actions of incomplete transactions are undone. Thus, if each complete transaction, executing alone, maintains the consistent of criteria, then the database state after recovery from a crash is consistent.

Task Discuss, what are the advantages of oracle instead of access.

1.5 Disadvantages of DBMS

The disadvantage of the DBMS system is overhead cost. The processing overhead introduced by the DBMS to implement security, integrity, and sharing of the data causes a degradation of the response and throughput times. An additional cost is that of migration from a traditionally separate application environment to an integrated one.

Even though centralization reduces duplication, the lack of duplication requires that the database be adequately backup so that in the case of failure the data can be recovered.

Backup and recovery operations are complex in a DBMS environment, and this is an increment in a concurrent multi-user database system. A database system requires a certain amount of controlled redundancies and duplication to enable access to related data items.

Centralization also means that the data is accessible from a single source, namely the database. This increases the potential severity of security breaches and disrupting of the operation of the organization because of down times and failures.

1.6 Database Architecture

The functional components of a database system can be broadly divided into query processor components and storage manager components. The query processor includes:

1. ***DML Compiler:*** It translates DML statements in a query language into low-level instructions that the query evaluation engine understands.

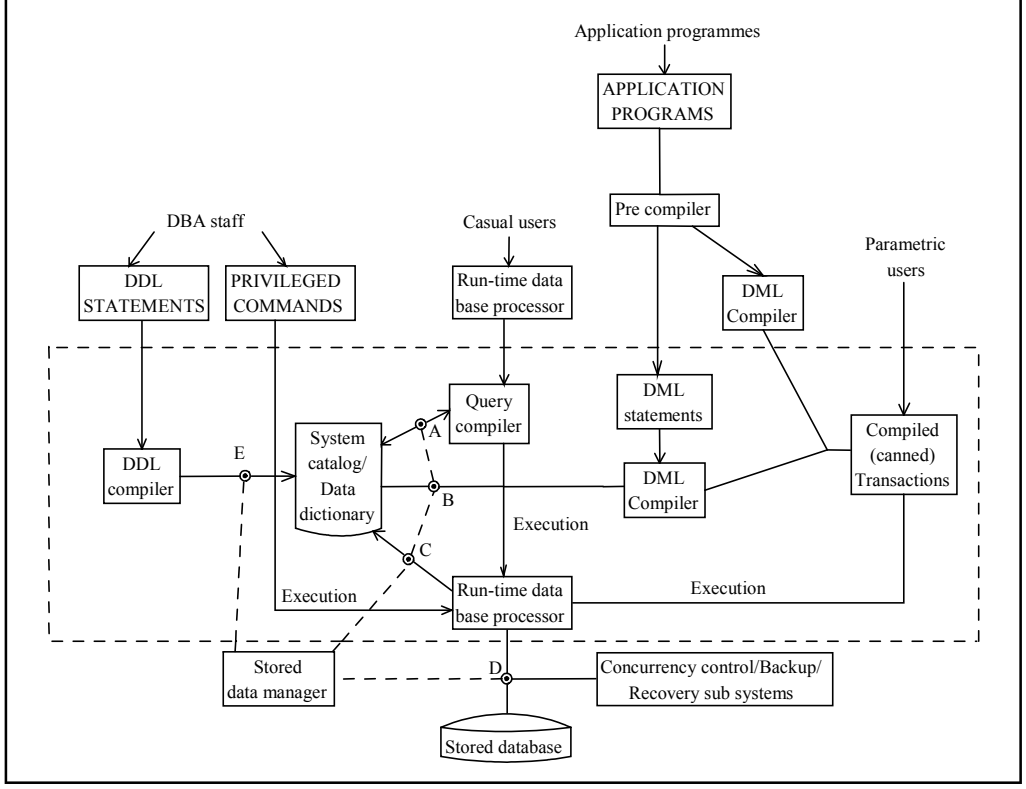
Notes

2. ***Embedded DML Pre-compiler:*** It converts DML statements embedded in an application program to normal procedure calls in the host language. The pre-compiler must interact with the DML compiler to generate the appropriate code.
3. ***DDL Interpreter:*** It interprets DDL Stateline its and records them in a set of tables containing metadata.
4. ***Transaction Manager:*** Ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
5. ***File Manager:*** Manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
6. ***Buffer Manager:*** Is responsible for fetching data from disk storage into main memory and deciding what data to cache in memory.

Also some data structures are required as part of the physical system implementation:

1. ***Data Files:*** The data files store the database by itself.
2. ***Data Dictionary:*** It stores metadata about the structure of the database, as it is used heavily.
3. ***Indices:*** It provides fast access to data items that hold particular values.
4. ***Statistical Data:*** It stores statistical information about the data in the database. This information used by the query processor to select efficient ways to execute a query.

Figure 1.3: Structure of DBMS



Notes

Case Study

Requirements Analysis

The owner of B&N has thought about what he wants and offers a concise summary:

"I would like my customers to be able to browse my catalog of books and to place orders over the Internet. Currently, I take orders over the phone. I have mostly corporate customers who call me and give me the ISBN number of a book and a quantity. I then prepare a shipment that contains the books they have ordered. If I don't have enough copies in stock, I order additional copies and delay the shipment until the new copies arrive; I want to ship a customer's entire order together. My catalog includes all the books that I sell. For each book, the catalog contains its ISBN number, title, author, purchase price, sales price, and the year the book was published. Most of my customers are regulars, and I have records with their name, address, and credit card number. New customers have to call me first and establish an account before they can use my Web site.

On my new Web site, customers should first identify themselves by their unique customer identification number. Then they should be able to browse my catalog and to place orders online."

DBDudes's consultants are a little surprised by how quickly the requirements phase was completed it usually takes them weeks of discussions (and many lunches and dinners) to get this done but return to their offices to analyze this information.

1.7 Summary

- A database is a collection of persistent data that is used by the application system of some enterprise. The enterprise may be a Bank, a Hospital, an Educational Institution, a Library, etc.
- The word persistence means once the data of the database is accepted by the DBMS, it can then be removed from the database only by some explicit request.
- It can not be deleted or modified because of some side effect just like the programming language variables.
- There are several advantages of storing the data in database rather than storing it in operating system files. An example, university database, to illustrate this concept was discussed.
- In the DBMS environment we speak of many people. For example, the main people involved are DBA, Database Designers, and various types of users.
- Though database approach has few disadvantages under some specific circumstances, the use of database is indispensable. The major implications of database approach are:
    - Potential for enforcing standards
    - Reduced application development time
    - Flexibility
    - Economically viable
    - Data integrity and security

Notes

- Database need not be used always. There are occasions where you manage your data with out a database.

1.8 Keywords

Data Abstraction: A database management system is a collection of interrelated files and a set of programs that allow users to access and modify these files. A major purpose of a database system is to provide users with an abstract view of the data. This is called data abstraction.

Data processing: The process of converting the facts into meaningful information is known as data processing. Data processing is also known as information processing.

Data: Data is the raw material from which useful information is derived.

Database: A shared collection of logically related data along with the description of the data that suits to the needs of large enterprises.

Metadata: Data that describe the properties or characteristics of other data.

1.9 Self Assessment

Choose the appropriate answer:

1. DBMS stands for:
   (a) Database Managerial System
   (b) Database Management System
   (c) Database Management Source
   (d) Development Management System
2. Data processing is also known as
   (a) Data programming
   (b) Data access
   (c) Information processing
   (d) Database sourcing
3. DDL stands for
   (a) Data Development Language
   (b) Data Document Language
   (c) Document Definition Language
   (d) Data Definition Language

Fill in the blanks:

4. ..... provides fast access to data items that hold particular values.
5. Data is commonly used in both singular and ..... forms.
6. The term data and ..... are closely related.
7. The primary mechanism for providing context for data is .....

Notes

8. A ..... is conceptually a table, but the records of this table are not stored in the database.
9. In DBMS the access programs are written independent of any specific .....
10. In file processing the data definition is part of the ..... program.

1.10 Review Questions

1. Define database. Explain the concepts in database environment.
2. List and explain various Database System Applications.
3. What are the differences between File processing systems and DBMS?
4. Write the advantages of DBMS.
5. Write short notes on Disadvantages of Database Management System.
6. What is Data independence? Explain the types of Data Independence.
7. What are the database languages? Explain the various languages.
8. What are the responsibilities of a DBA? List and explain them.
9. What is the role of Data user? Explain the types of users.
10. Explain the architecture of DBMS.
11. Explain the components of DBMS.
12. Write history of Database Systems.

Answers: Self Assessment

1.	(b)	2.	(c)
3.	(d)	4.	Indices
5.	plural	6.	information
7.	Metadata	8.	view
9.	files	10.	application

1.11 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

Notes

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 2: Database Relational Model

CONTENTS

Objectives

Introduction

2.1 Relational Model

2.1.1 Relational Model Concepts

2.1.2 Alternatives to the Relational Model

2.1.3 Implementation

2.1.4 Application to Databases

2.1.5 SQL and the Relational Model

2.1.6 Set-theoretic Formulation

2.2 Additional and Extended Relational Algebra Operations

2.2.1 Relational Algebra Expression

2.2.2 Set Operation of Relational Algebra

2.2.3 Joins

2.3 Summary

2.4 Keywords

2.5 Self Assessment

2.6 Review Questions

2.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe relational model
- Explain additional and extended relational algebra operations

Introduction

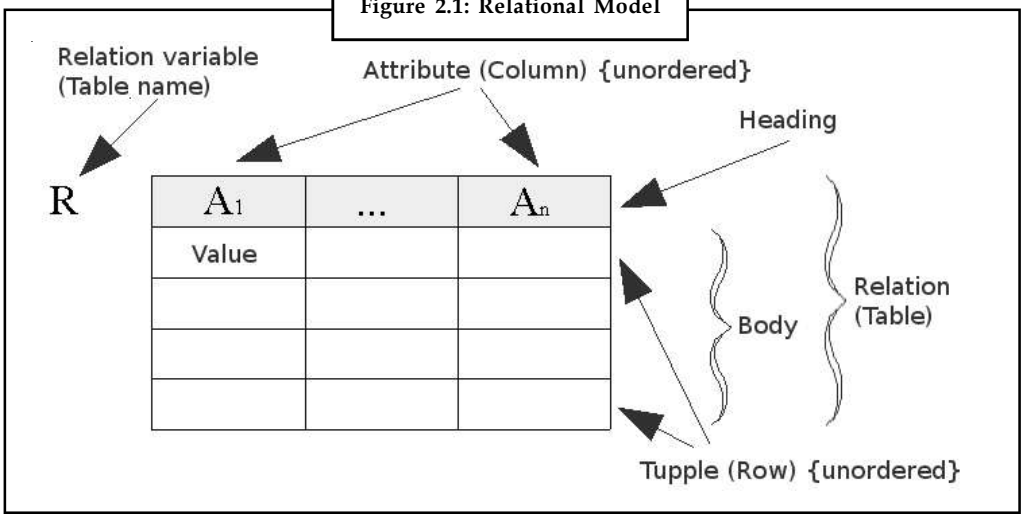
A relational database consists of a collection of tables that store particular sets of data. The invention of this database system has standardized the way that data is stored and processed. The concept of a relational database derives from the principles of relational algebra, realized as a whole by the father of relational databases, E. F. Codd. Most of the database systems in use today are based on the relational system.

2.1 Relational Model

The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar Codd.

Notes

Figure 2.1: Relational Model



Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite model (logic) of the database, i.e. a set of relations, one per predicate variable, such that all predicates are satisfied. A request for information from the database (a database query) is also a predicate.

2.1.1 Relational Model Concepts

The purpose of the relational model is to provide a declarative method for specifying data and queries: we directly state what information the database contains and what information we want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for getting queries answered.

IBM implemented Codd's ideas with the DB2 database management system; it introduced the SQL data definition and query language. Other relational database management systems followed, most of them using SQL as well. A table in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, it must be noted that SQL databases, including DB2, deviate from the relational model in many details; Codd fiercely argued against deviations that compromise the original principles.

2.1.2 Alternatives to the Relational Model

Other models are the hierarchical model and network model. Some systems using these older architectures are still in use today in data centers with high data volume needs or where existing systems are so complex and abstract it would be cost prohibitive to migrate to systems employing the relational model; also of note are newer object-oriented databases, even though many of them are DBMS-construction kits, rather than proper DBMSs. A recent development is the Object-Relation type-Object model, which is based on the assumption that any fact can be expressed in the form of one or more binary relationships. The model is used in Object Role Modeling (ORM), RDF/Notation 3 (N3) and in Gellish English.

The relational model was the first formal database model. After it was defined, informal models were made to describe hierarchical databases (the hierarchical model) and network databases (the network model). Hierarchical and network databases existed before relational databases,

Notes

but were only described as models after the relational model was defined, in order to establish a basis for comparison.

2.1.3 Implementation

There have been several attempts to produce a true implementation of the relational database model as originally defined by Codd and explained by Date, Darwen and others, but none have been popular successes so far. Rel is one of the more recent attempts to do this.

History

The relational model was invented by E.F. (Ted) Codd as a general model of data, and subsequently maintained and developed by Chris Date and Hugh Darwen among others. In The Third Manifesto (first published in 1995) Date and Darwen show how the relational model can accommodate certain desired object-oriented features.

Controversies

Codd himself, some years after publication of his 1970 model, proposed a three-valued logic (True, False, Missing or NULL) version of it in order to deal with missing information, and in his The Relational Model for Database Management Version 2 (1990) he went a step further with a four-valued logic (True, False, Missing but Applicable, Missing but Inapplicable) version. But these have never been implemented, presumably because of attending complexity. SQL's NULL construct was intended to be part of a three-valued logic system, but fell short of that due to logical errors in the standard and in its implementations.

The Model

The fundamental assumption of the relational model is that all data is represented as mathematical n-ary relations, an n-ary relation being a subset of the Cartesian product of n domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either true or false (and in particular no third value such as unknown, or not applicable, either of which are often associated with the concept of NULL). Some think two-valued logic is an important part of the relational model, where others think a system that uses a form of three-valued logic can still be considered relational.

Data are operated upon by means of a relational calculus or relational algebra, these being equivalent in expressive power.

The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the logical schema.

The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives. The access plans and other implementation and operation details are handled by the DBMS engine, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model.

The basic relational building block is the domain or data type, usually abbreviated nowadays to type. A tuple is an unordered set of attribute values. An attribute is an ordered pair of attribute name and type name. An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

Notes

A relation consists of a heading and a body. A heading is a set of attributes. A body (of an n-ary relation) is a set of n-tuples. The heading of the relation is also the heading of each of its tuples.

A relation is defined as a set of n-tuples. In both mathematics and the relational database model, a set is an unordered collection of items, although some DBMSs impose an order to their data. In mathematics, a tuple has an order, and allows for duplication. E.F. Codd originally defined tuples using this mathematical definition. Later, it was one of E.F. Codd's great insights that using attribute names instead of an ordering would be so much more convenient (in general) in a computer language based on relations. This insight is still being used today. Though the concept has changed, the name "tuple" has not. An immediate and important consequence of this distinguishing feature is that in the relational model the Cartesian product becomes commutative.

A table is an accepted visual representation of a relation; a tuple is similar to the concept of row, but note that in the database language SQL the columns and the rows of a table are ordered.

A relvar is a named variable of some specific relation type, to which at all times some relation of that type is assigned, though the relation may contain zero tuples.

The basic principle of the relational model is the Information Principle: all information is represented by data values in relations. In accordance with this Principle, a relational database is a set of relvars and the result of every query is presented as a relation.

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by constraints, declared as part of the logical schema and enforced by the DBMS for all applications. In general, constraints are expressed using relational comparison operators, of which just one, "is subset of", is theoretically sufficient. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints.

Interpretation

To fully appreciate the relational model of data, it is essential to understand the intended interpretation of a relation.

The body of a relation is sometimes called its extension. This is because it is to be interpreted as a representation of the extension of some predicate, this being the set of true propositions that can be formed by replacing each free variable in that predicate by a name (a term that designates something).

There is a one-to-one correspondence between the free variables of the predicate and the attribute names of the relation heading. Each tuple of the relation body provides attribute values to instantiate the predicate by substituting each of its free variables. The result is a proposition that is deemed, on account of the appearance of the tuple in the relation body, to be true. Contrariwise, every tuple whose heading conforms to that of the relation but which does not appear in the body is deemed to be false. This assumption is known as the closed world assumption.

For a formal exposition of these ideas, see the section Set Theory Formulation.

Task Write total number of rules given by E.F. Codd.

Notes

2.1.4 Application to Databases

A type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values true and false, and so on. The corresponding type names for these types might be the strings "int", "char", "date", "boolean", etc. It is important to understand, though, that relational theory does not dictate what types are to be supported; indeed, nowadays provisions are expected to be available for user-defined types in addition to the built-in ones provided by the system.

Attribute is the term used in the theory for what is commonly referred to as a column. Similarly, table is commonly used in place of the theoretical term relation (though in SQL the term is by no means synonymous with relation). A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column. An attribute value is the entry in a specific column and row, such as "John Doe" or "35".

A tuple is basically the same thing as a row, except in an SQL DBMS, where the column values in a row are ordered. (Tuples are not ordered; instead, each attribute value is identified solely by the attribute name and never by its ordinal position within the tuple.) An attribute name might be "name" or "age".

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table.

The heading of its assigned value at any time is as specified in the table declaration and its body is that most recently assigned to it by invoking some update operator (typically, Insert, Update, or Delete). The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query.

Notes In SQL the heading is not always a set of column definitions as described above, because it is possible for a column to have no name and also for two or more columns to have the same name. Also, the body is not always a set of rows because in SQL it is possible for the same row to appear more than once in the same body.

2.1.5 SQL and the Relational Model

SQL, initially pushed as the standard language for relational databases, deviates from the relational model in several places. The current ISO SQL standard doesn't mention the relational model or use relational terms or concepts. However, it is possible to create a database conforming to the relational model using SQL if one does not use certain SQL features.

The following deviations from the relational model have been noted in SQL. Note that few database servers implement the entire SQL standard and in particular do not allow some of these deviations. Whereas NULL is nearly ubiquitous, for example, allowing duplicate column names within a table or anonymous columns is uncommon.

Duplicate Rows

The same row can appear more than once in an SQL table. The same tuple cannot appear more than once in a relation.

Notes

Anonymous Columns

A column in an SQL table can be unnamed and thus unable to be referenced in expressions. The relational model requires every attribute to be named and referenceable.

Duplicate Column Names

Two or more columns of the same SQL table can have the same name and therefore cannot be referenced, on account of the obvious ambiguity. The relational model requires every attribute to be referenceable.

Column Order Significance

The order of columns in an SQL table is defined and significant, one consequence being that SQL's implementations of Cartesian product and union are both non-commutative. The relational model requires there to be no significance to any ordering of the attributes of a relation.

Views without CHECK OPTION

Updates to a view defined without CHECK OPTION can be accepted but the resulting update to the database does not necessarily have the expressed effect on its target. For example, an invocation of INSERT can be accepted but the inserted rows might not all appear in the view, or an invocation of UPDATE can result in rows disappearing from the view. The relational model requires updates to a view to have the same effect as if the view were a base relvar.

Columnless Tables Unrecognized

SQL requires every table to have at least one column, but there are two relations of degree zero (of cardinality one and zero) and they are needed to represent extensions of predicates that contain no free variables.

NULL

This special mark can appear instead of a value wherever a value can appear in SQL, in particular in place of a column value in some row. The deviation from the relational model arises from the fact that the implementation of this ad hoc concept in SQL involves the use of three-valued logic, under which the comparison of NULL with itself does not yield true but instead yields the third truth value, unknown; similarly the comparison NULL with something other than itself does not yield false but instead yields unknown. It is because of this behaviour in comparisons that NULL is described as a mark rather than a value. The relational model depends on the law of excluded middle under which anything that is not true is false and anything that is not false is true; it also requires every tuple in a relation body to have a value for every attribute of that relation. This particular deviation is disputed by some if only because E.F. Codd himself eventually advocated the use of special marks and a 4-valued logic, but this was based on his observation that there are two distinct reasons why one might want to use a special mark in place of a value, which led opponents of the use of such logics to discover more distinct reasons and at least as many as 19 have been noted, which would require a 21-valued logic. SQL itself uses NULL for several purposes other than to represent "value unknown".

Example: The sum of the empty set is NULL, meaning zero, the average of the empty set is NULL, meaning undefined, and NULL appearing in the result of a LEFT JOIN can mean "no value because there is no matching row in the right-hand operand".

Notes

Concepts

SQL uses concepts "table", "column", "row" instead of "relvar", "attribute", "tuple". These are not merely differences in terminology.

Example: A "table" may contain duplicate rows, whereas the same tuple cannot appear more than once in a relation.

Database

An idealized, very simple example of a description of some relvars and their attributes:

1. Customer (Customer ID, Tax ID, Name, Address, City, State, Zip, Phone)
2. Order (Order No, Customer ID, Invoice No, Date Placed, Date Promised, Terms, Status)
3. Order Line (Order No, Order Line No, Product Code, Qty)
4. Invoice (Invoice No, Customer ID, Order No, Date, Status)
5. Invoice Line (Invoice No, Line No, Product Code, Qty Shipped)
6. Product (Product Code, Product Description)

In this design we have six relvars: Customer, Order, Order Line, Invoice, Invoice Line and Product. The bold, underlined attributes are candidate keys. The non-bold, underlined attributes are foreign keys.

Usually one candidate key is arbitrarily chosen to be called the primary key and used in preference over the other candidate keys, which are then called alternate keys.

A candidate key is a unique identifier enforcing that no tuple will be duplicated; this would make the relation into something else, namely a bag, by violating the basic definition of a set. Both foreign keys and superkeys (which includes candidate keys) can be composite, that is, can be composed of several attributes. Below is a tabular depiction of a relation of our example Customer relvar; a relation can be thought of as a value that can be attributed to a relvar.

Customer Relation

Customer ID	Tax ID	Name	Address	[More fields....]
1234567890	555-5512222	Munmun	323 Broadway	...
2223344556	555-5523232	SS4 Vegeta	1200 Main Street	...
3334445563	555-5533323	Ekta	871 1st Street	...
4232342432	555-5325523	E. F. Codd	123 It Way	...

If we attempted to insert a new customer with the ID 1234567890, this would violate the design of the relvar since Customer ID is a primary key and we already have a customer 1234567890. The DBMS must reject a transaction such as this that would render the database inconsistent by a violation of an integrity constraint.

Foreign keys are integrity constraints enforcing that the value of the attribute set is drawn from a candidate key in another relation. For example in the Order relation the attribute Customer ID is a foreign key. A join is the operation that draws on information from several relations at once. By joining relvars from the example above we could query the database for all of the Customers, Orders, and Invoices. If we only wanted the tuples for a specific customer, we would specify this using a restriction condition.

Notes

If we wanted to retrieve all of the Orders for Customer 1234567890, we could query the database to return every row in the Order table with Customer ID 1234567890 and join the Order table to the Order Line table based on Order No.

There is a flaw in our database design above. The Invoice relvar contains an Order No attribute. So, each tuple in the Invoice relvar will have one Order No, which implies that there is precisely one Order for each Invoice. But in reality an invoice can be created against many orders, or indeed for no particular order. Additionally the Order relvar contains an Invoice No attribute, implying that each Order has a corresponding Invoice. But again this is not always true in the real world. An order is sometimes paid through several invoices, and sometimes paid without an invoice. In other words there can be many Invoices per Order and many Orders per Invoice. This is a many-to-many relationship between Order and Invoice (also called a non-specific relationship). To represent this relationship in the database a new relvar should be introduced whose role is to specify the correspondence between Orders and Invoices:

OrderInvoice (Order No, Invoice No)

Now, the Order relvar has a one-to-many relationship to the OrderInvoice table, as does the Invoice relvar. If we want to retrieve every Invoice for a particular Order, we can query for all orders where Order No in the Order relation equals the Order No in OrderInvoice, and where Invoice No in OrderInvoice equals the Invoice No in Invoice.

Task Advantages of NULL constraints in DBMS.

2.1.6 Set-theoretic Formulation

Basic notions in the relational model are relation names and attribute names. We will represent these as strings such as "Person" and "name" and we will usually use the variables r, s, t …and a, b, c to range over them. Another basic notion is the set of atomic values that contains values such as numbers and strings.

Our first definition concerns the notion of tuple, which formalizes the notion of row or record in a table:

Tuple: A tuple is a partial function from attribute names to atomic values.

Header: A header is a finite set of attribute names.

Projection: The projection of a tuple t on a finite set of attributes A is

$$t[A] = \{(a, v) : (a, v) \in t, a \in A\}$$

The next definition defines relation which formalizes the contents of a table as it is defined in the relational model.

Relation: A relation is a tuple (H,B) with H, the header, and B, the body, a set of tuples that all have the domain H.

Such a relation closely corresponds to what is usually called the extension of a predicate in first-order logic except that here we identify the places in the predicate with attribute names. Usually in the relational model a database schema is said to consist of a set of relation names, the headers that are associated with these names and the constraints that should hold for every instance of the database schema.

Notes

Relation Universe: A relation universe U over a header H is a non-empty set of relations with header H.

Relation Schema: A relation schema (H, C) consists of a header H and a predicate C(R) that is defined for all relations R with header H. A relation satisfies a relation schema (H, C) if it has header H and satisfies C.

Key Constraints and Functional Dependencies

One of the simplest and most important types of relation constraints is the key constraint. It tells us that in every instance of a certain relational schema the tuples can be identified by their values for certain attributes.

1. ***Superkey:*** A superkey is written as a finite set of attribute names.

   A superkey K holds in a relation (H, B) if:

   $K \subseteq H$ and

   there exist no two distinct tuples $t_1, t_2 \in B$ such that $t_1[K] = t_2[K]$.

   A superkey holds in a relation universe U if it holds in all relations in U.

   Theorem: A superkey K holds in a relation universe U over H if and only if $K \subseteq H$ and $K \to H$ holds in U.

2. ***Candidate Key:*** A superkey K holds as a candidate key for a relation universe U if it holds as a superkey for U and there is no proper subset of K that also holds as a superkey for U.

3. ***Functional Dependency:*** A functional dependency (FD for short) is written as $X \to Y$ for X, Y finite sets of attribute names.

   A functional dependency $X \to Y$ holds in a relation (H,B) if:

   $X, Y \subseteq H$ and

   $\forall$ tuples $t_1, t_2 \in B,\ t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

   A functional dependency $X \to Y$ holds in a relation universe U if it holds in all relations in U.

4. ***Trivial Functional Dependency:*** A functional dependency is trivial under a header H if it holds in all relation universes over H.

   Theorem: An FD $X \to Y$ is trivial under a header H if and only if $Y \subseteq X \subseteq H$.

5. ***Closure: Armstrong's axioms:*** The closure of a set of FDs S under a header H, written as S^+, is the smallest superset of S such that:

   (a) $Y \subseteq X \subseteq H \Rightarrow X \to Y \in S^+$ (reflexivity)

   (b) $X \to Y \in S^+ \wedge Y \to Z \in S^+ \Rightarrow X \to Z \in S^+$ (transitivity) and

   (c) $X \to Y \in S^+ \wedge Y \subseteq H \Rightarrow (X \cup Z) \to (Y \cup Z) \in S^+$ (augmentation)

   Theorem: Armstrong's axioms are sound and complete; given a header H and a set S of FDs that only contain subsets of H, $X \to Y \in S^+$ if and only if $X \to Y$ holds in all relation universes over H in which all FDs in S hold.

Notes

6. ***Completion:*** The completion of a finite set of attributes X under a finite set of FDs S, written as X^+, is the smallest superset of X such that: $Y \to Z \in S \wedge Y \subseteq X^+ \Rightarrow Z \subseteq X$.

   The completion of an attribute set can be used to compute if a certain dependency is in the closure of a set of FDs.

   Theorem: Given a set S of FDs $X \to Y \in S^+$, if and only if $Y \subseteq X^+$

7. ***Irreducible Cover:*** An irreducible cover of a set S of FDs is a set T of FDs such that:

   (a) $S^+ = T^+$

   (b) there exists $U \subset T$ no such that $S^+ = U^+$

   (c) $X \to Y \in T \Rightarrow Y$ is a singleton set and

   (d) $X \to Y \in T \wedge Z \subset X \Rightarrow Z \to Y \notin S^+$

Algorithm to Derive Candidate Keys from Functional Dependencies

INPUT: A set S of FDs that contain only subsets of a header H

OUTPUT: The set C of superkeys that hold as candidate keys in all relation universes over H in which all FDs in S hold begin

```
C := ϕ ; // found candidate keys
Q := { H }; // superkeys that contain candidate keys
while Q <> ϕ do
let K be some element from Q;
Q := Q - { K };
minimal := true;
for each X->Y in S do
K′ := (K - Y) ∪ X; // derive new superkey
if K′ ⊂ K then
minimal := false;
Q := Q ∪ { K′ };
end if
end for
if minimal and there is not a subset of K in C then remove all supersets of K from C;
C := C ∪ { K };
end if
end while
end
```

Notes

2.2 Additional and Extended Relational Algebra Operations

This language is used to manipulate the data in the data model. This languages uses a set of operators that are applied on two relations (input) and produces a new relation (output).

Using the different operators-defined by the relational algebra, queries are built and applied on the relations.

2.2.1 Relational Algebra Expression

Consists of operations when applied results into a new relation representing the different requests.

Unary Operations

Operations which operates on only one relation.

Binary Operations

Operations which operate on two relations.

To understand the concepts of relation algebra and relational calculus, the following database "COMPANY" will be used throughout the unit.

COMPANY database consist of five tables with attribute set as shown.

EMPLOYEE (Ename, Bid, Bdate, Address, Sex, Salary, DNo, Phonenumber, age)

DEPARTMENT **(DName, DNo**. Manager-id)

DEPT _LOCATION (DNo. Dlocation)

PROJECT (PName, PNumber. PLocation, DNo, Bid)

DEPENDENT (eid, Dep-Name. Sex, relationship)

The underlined attributes represent the primary key for the respective tables.

A compute view of "COMPANY" Database:

Employee

Ename	Eid	Bdate	Address	Sex	($) Salary	Dno	Phone number	Age (years)
John	12345261	10-8-1965	New Jersey	M	25000	7	773213218	41
Jack	12345262	12-5-1955	Chicago	M	55000	7	773421317	51
preen	12345263	20- 11- 1972	New York	M	20000	7	773421317	34
Brown	12345264	28-7-1968	Chicago	M	40000	8	773210192	28
Bill	12345265	25-4-1955	Detroit	F	35000	8	773271842	51
Jill	12345266	04-4-1965	New York	F	42000	8	773291828	41
Donald	12345267	02 8-1968	Detroit	M	20000	9	773423145	28
John	1 234526	10- 8- 1965	New Jersey	M	25000	8	773213218	41
John	12375261	10-8-1965	New Jersey	M	25000	9	773213218	41

Notes

Department

Dname	D.No	Manager-id	D.no	DLocation
Administration	7	12345261	7	New York
Research	8	12345265	8	Detroit
Head quarters	9	12345267	9	Chicago

Project

Eid	Pno	Pname	Plocation	Dno
12345261	21	Computerization	New York	7
12345265	31	Product A	Detroit	8
12345267	41	New benefits	Chicago	9
12345263	51	Soft drinks	New Jersey	7
12345264	61	Reorganization	Chicago	8
12345262	71	Product B	New Jersey	7

Dependent

Eid	Dep-Name	Sex	Relationship
12354261	Nivedita	F	Mother
12354262	Alan	M	Brother
12354263	Lizzzi	F	Sister
12354267	Roven	F	Daughter
12354265	Peter	M	Father
12354266	Kumar	M	Brother
12345267	Sunny	M	son

The different operations of relational algebra are:

Select (σ)

The symbol for the select operation is σ. This operation is used to select particular rows from the two relations, which satisfy the given condition.

The syntax for select operations (table name) <condition>

Example:

1. σ (DEPENDENT)

   Sex = F

   This can be read as "select from table DEPENDENT where Sex = F". The following table will be displayed.

Notes

Projection 1

Eid	Dep-Name	Sex	Relationship
12354261	Nivedita	F	Mother
12354263	Lizzi	F	Sister
12354267	Raven	F	Doughtier

2. σ (DNo = 7 AND Address = 'New Jersey') OR (DNo = 8 AND Address = 'Detroit') EMPLOYEE

   This condition selects the rows from employee which satisfy the conditions.

   DNo = 7 and address = 'New Jersey' OR the condition

   DNo = 8 and address 'Detroit'.

   If either of the conditions are satisfied then that row is selected.

   The boolean operators used here convey the following meaning.

   A_1 AND B_1 = TRUE if A, & B, are true i.e., the row must satisfy both conditions (A_1 & B_1) in order to get selected.

   $(A_1 \vee B_1) \Leftrightarrow$ A_1 OR B_1 = TRUE if either A_1 OR B_1 is true.

   $(A_1 \wedge B_1) \Leftrightarrow$ A_1 AND B_1 = False if either of A_1 & B_1 are false.

   $(A_1 \vee B_1) \Leftrightarrow$ A_1 OR B_1 = False if both of A, and B, are false.

   The resulting relation is,

Projection 2

Age	Ename	Eid	Bdata	Address	Sex	($) Salary	Dno	Phone No.
41	John	12345261	10-8-1965	New Jersey	M	25000	7	773213218
51	Bill	12345265	12-5-1955	Detroit	F	35000	8	773271142

   Some examples of the select operator:

   $\sigma_{\text{manager-id = 4341}}$ DEPARTMENT

   $\sigma_{\text{(Salary > 40000 AND age < 40) OR (Salary} \leq \text{20000 AND age} \geq \text{20)}}$ EMPLOYEE

   $\sigma_{\text{(Dep-Name = 'Raven') OR (relationship = 'Brother')}}$ DEPENDENT. Different comparison operators can be used with select operator.

   (a) $< \rightarrow$ Less than

   (b) $> \rightarrow$ Greater than

   (c) $= \rightarrow$ Equal to

   (d) $\neq \rightarrow$ Not equal to

   (e) $\leq \rightarrow$ Less than or equal to

   (f) $\geq \rightarrow$ Greater than or equal to.

   We can even use boolean operator "not" to negate a condition.

3. Display all the information of projects except those which are located in Chicago.

   $\sigma_{\text{Not Plocation = 'Chicago'}}$ PROJECT

Notes

Projection 3

Pno	Pname	Plocation	Dno	Eid
21	Computerization	New York	7	12345261
31	Product A	Detroit	8	12345265
51	Soft drinks	New Jersey	7	12345263
71	Product B	New Jersey	7	12345262

Project (π)

This operator is used to display the list of attributes/columns defined in the relation R.

The syntax for "Project" operator is

$\pi_{(attributes)}$(tablename)'R'

$\pi_{(columns)}$(table name)'

Example: To display project No, DNo from relation PROJECT,

we can write:

$\pi_{PNo, Dmo}$(PROJECT)

Now only these columns are selected and displayed while other columns are eliminated from the relation.

Projection 4

PNo	DNo
21	7
31	8
41	9
51	7
61	8
71	9

$\pi_{sex, salary}$ (EMPLOYEE)

This query will display the columns sex and salary from employee. However, it must be noted that the duplicate values are not repeated in the new relation. Only unique topples are listed. This elimination of duplicate values is called as duplicate elimination and hence reduces redundancy. The following relations will be displayed.

Projection 5

Sex	Salary
M	25000
M	55000
M	20000
M	40000
F	35000
F	42000

Notes

Notes The select operator is commutative in nature i.e.,

$\sigma_{\text{cand A}} \cdot \sigma_{\text{cand.B}}$ (R) $\Rightarrow \sigma_{\text{cand.B}}$ (σ_{TcanA} (R))

Whereas the project operator is not commutative in nature.

$\pi_{\text{attribute list}} \cdot$ A ($\pi_{\text{attributelist.B}}$ (Relation)) $\Rightarrow \pi_{\text{listA}}$ (Relation) if and only if attributes list

B is a subset of attribute list A.

The number of columns returned by the project operator is always equal to or less than the number of columns in the original relation.

To manipulate the data stored in a table we can combine the two operators discussed so far.

Example: If we want to display the emp-id, emp-name, Birthdate, salary of all the employees whose age is greater 30, we can write the expression as,

$\pi_{\text{eid, fname, Bdata, Salary, age}}$ ($\sigma_{\text{age} > 30}$ (EMPLOYEE))

This will return the following relation:

Projection 6

Eid	Ename	Bdate	($) Salary	Age (years)
12345261	John	10-8-1965	25000	41
12345262	Jack	12-5-1955	55000	51
12345263	Green	20-11-1972	20000	34
12345265	Bill	25-4-1955	35000	51
12345266	Jill	04-4-1965	42000	41

Other examples are

$\pi_{\text{Dep-Name, Sex,}}$ $\sigma_{\text{relationship} = \text{'Mother'}}$ OR $_{\text{relationship} = \text{'son'}}$ DEPENDENT

$\pi_{\text{PNo, PName,}}$ DNo ($\sigma_{\text{PLocation} = \text{'Detroit}}$PROJECT)

$\pi_{\text{Ename, Eid, DNo, age, address}}$ ($\sigma_{\text{(Salary} = 40000 \text{ AND Sex} = \text{'M')}}$ EMPLOYEE)

Selection and projection operators are unary operators, they can be applied to only single relation.

2.2.2 Set Operation of Relational Algebra

The different operators of relational algebra are as follows:

Union ($\cup$)

When applied on two relations, it returns all the rows which are either present in first relation or second relation or in both the relations. It does not return the rows which has the same tuple values - repeating rows. All the rows returned by this operator are unique. One of the constraints

Notes

of union operator is that both the relations must be union-compatible, i.e., both the relations must have the same number of columns and also these columns must have the same domain (type of data being stored in each column).

If A and B are two relations and are union compatible then union of these two relations is expressed as:

$A \cup B \rightarrow$ A B returns uniquely all the tuples of A and B.

Example: If we apply union operator on EMPLOYEE relation and projection 2 relation then we will get the original employee relation with-only unique tuples. To understand this, consider a simple student1 and student2 relation which represent juniors and seniors.

Student (S_1)

Roll No	Class	Name
2101	VIII	Ravi
2102	VII	Kumar
2103	VI	Sheena

Student (S_2)

Roll No	Class	Name
3101	IX	Sodir
3102	X	Dhani
3103	XI	Rahul

S_1S_2

Roll No	Class	Name
2101	VIII	Ravi
2102	VII	Kumar
2103	VI	Sheena
3101	IX	Sodir
3102	X	Dhani
3103	XI	Rahul

Intersection ($\cap$)

When applied on two relations, A and B, it ($A \cap B$) returns all the rows which are common to both the relations. Most importantly, this operator can be applied to only those relations that are union compatible.

Example: If we apply intersection operator ($\cap$) to S_1 and S_2 (above relations) then the result will be an empty relation as nothing is common between the two.

$S_1 \cap S_2$ = ~ empty relation.

Notes

If we apply intersection operator (n) on employee relation and projection 2 the resulting relation will be similar to projection 2 relation as it has only two tuples in common.

Set-difference (Minus '–')

This operator is also called "Minus" operator. If A and B are two relations and are union compatible then (A - B) will return all the rows which are in A bit not in B. Consider, this operator is applied to employee and projection 2 relations, then the resulting relation is,

EMPLOYEE Projection 2

Ename	Eid	Bdate	Address	Sex	($) Salary	Dno	Phone No.	Age (years)
Jack	12345262	12-5-1955	Chcago	M	55000	7	773313218	51
John	12345261	10-8-1965	New Jersey	M	25000	8	773213218	41
Green	12345263	20- 11-1 972	New York	M	20000	7	773421317	34
Brown	12345264	28-7-1968	Chicago	M	40000	8	773210192	28
Jill	12345266	04-4-1965	New York	F	42000	8	773291828	41
Donald	12345267	02-8-1968	Detroit	M	20000	9	773423175	28
John	12375261	10-8-1965	New Jersey	M	25000	9	773213218	41

Cross Product (*)

This operator returns all the tuples of relation A plus all the tuples of relation B. Cross product is denoted as, A * B - returns the Cartesian product of two relation. The resulting relation will be much bigger in size than the two relations.

A		B		
Column 1	**Column2**	**Column A**	**Column 1**	**Column B**
1	a	a_1	1	b_1
2	b	a_2	2	b_2
3	c	a_3	3	b_3

A * B

Column 1	Column 2	Column A	Column 1	Column B
1	a	a_1	1	b_1
1	a	a_2	2	b_2
1	a	a_3	3	b_3
2	b	a_2	1	b_1
2	b	a_2	2	b_2
2	b	a_2	3	b_3
3	c	a_3	1	B_1
3	c	a_3	2	b_2
3	c	a_3	3	b_3

Notes

Renaming (ρ)

This operator is used to rename the relation or attributes or both. The syntax for this operator is ρ (A (old name $\rightarrow$ ρ new name), B)

B is the old relation which being renamed as 'A'. The list which contains the new names for attributes is called as renaming list.

i.e., old name $\rightarrow$ rename. This list is used to rename the attributes.

Example: If we want to rename the relation DEPT-LOCATION to DLOCATION and also the attributes DNo to D-number and DLocation to D_L then we can write this as:

ρ (DLocation (DNo $\rightarrow$ D_number, DLocation $\rightarrow$ D_L) DEPT_LOCATION)

Its not compulsory to rename both relations and attributes. It depends on the user and this operator simplifies the presentation of different relations (with different names and different attributes). The exclusion of both the relation names and attribute names is meaningless.

Notes Both union and intersection operators are commutative and also associative.

$$\left.\begin{array}{l} A \cup B = B \cup A \\ A \cap B = B \cup A \end{array}\right\} \text{Commutative}$$

$$\left.\begin{array}{l} A \cap (B \cap C) = (A \cap B) \cap C \\ A \cup (B \cup C) = (A \cup B) \cup C \end{array}\right\} \text{Associative}$$

The 'Minus' operation is not commutative.

$A - B \neq B - A$.

2.2.3 Joins

Joins are used to combine the information of two relations which have at least one field in common. It is one of the most important operators of RDBMS. In simple words, a join operation can be defined as a cartesian product followed by selection or projection operators. The different forms of join operators are,

1. Conditional join
2. Equi-join
3. Natural join
4. Outer join.

Conditional Join

This join returns a relation that includes a set of rows from the cartesian product of two relations A and B such that each row satisfies a given condition C. This can be denoted as,

$$A \bowtie_c B$$

Notes

i.e., join A and B based on same condition. This join is equivalent to performing a cartesian product on two relations followed by a selection operator. Thus,

$$A \bowtie_c B = \sigma_c (A \times B)$$

The application of conditional join on relations employee and Dept_location results into a new relation whose degree is 11.

Degree

The degree of a relation is equal to the number of fields (columns).

Consider the equation

Projection 7

Employee $\bowtie_{D.\,No>7}$ Dept_location

This statement means:

The relation employee and Deptlocation are joined based on the condition DNo>7 and the resulting relation is projection 7. Projection 7 will include all the tuples from employee and dept-location where DNo>7.

The operators which can be used with condition in order to compare the two relations are "$>, <, \leq, \geq =, \neq$

Ename	Ed	Bdate	Address	Sex	Salary	Dno	Phone number	Age (years)	DNo	Address
Brown	12345264	28-7-1968	Chicago	M	40000	8	773210192	28	8	Detroit
John	12345261	1-08- 1965	New Jersey	M	25000	9	773213218	41	9	Chicago
Brown	12345264	28-7-1968	Chicago	M	40000	8	773271872	51	8	Detroit
Bill	12345265	25-7-1955	Detroit	F	35000	8	773271842	51	9	Chicago
Bill	12345265	25-7-1955	Detroit	F	35000	8	773271842	52	9	Chicago
Jill	12345266	04-4-1965	New York	F	42000	8	773291828	41	8	Detroit
Jill	12345266	04-4-1965	New York	F	42000	8	773291628	41	9	Chicago
Donald	12345267	02-8-1768	Detroit	M	20000	9	773423145	28	8	Detroit
Donald	1234526	62-8-1768	Detroit	M	20000	9	773423175	28	9	Chicago
John	12345261	10-8-1965	New Jersey	M	25000	8	773213218	41	8	Detroit

Project 7 = Employee $\bowtie_{DNo>7}$ Dept_location.

Equi-join

Equi-join is same as conditional join, the only difference being only equity '=' operator is used to join the two relations.

Example: We may join DEPARTMENT and DEPTJLOCATION relation with the condition that

DNo_1. Department = DNo_2.Dept_location.

Notes

Where DNo_1 and DNo_2 are two instances of respective relations. This condition indicates - join the tuples where $DNo_1 = DNo_2$. The degree of resulting relation will be the sum of degrees of two relation minus the number of fields they have in common. More precisely,

The degree of relation A is x

The degree of relation B is y and the number of common fields is z

Then degree of resulting relation = x + y – z.

Projection 8

DEPARTMENT $\bowtie_{No1.DEPARTMENT = DNo2.DEPT\text{-}LOCATION}$ DEPT_LOCATION projection 8 will contain all the fields of DEPARTMENT AND DEPT_LOCATION AND the common fields will be included only once.

Project 8: Department $\bowtie_c$ Dept-location

Where c = DNo_1.Department = DNo_2.Dept_location

DName	DNo	Manager-id	DLocation
Administration	7	2431	New York
Research	8	3731	Detroit
Head Quarters	9	4341	Chicago

Natural Join

This is the default join operation i.e., no condition is specified. Natural join is equivalent to cartesian product. If two relations have a common field then the application of natural join is equivalent to equi join and if no field is common then the natural join is cartesian product of the two relations.

We can denote the operation as,

A $\bowtie$ B where A and B are two relations.

If we apply natural join operation as Departments and dept-location then the result will be same as projection 8 as they have only DNo field in common.

Outer Joins

This is a special case "join" operator which considers the NULL values. Generally a 'join' operation performs the cross product of two tables and applies certain join conditions. Then it selects those rows from the cross product that satisfies the given condition. But with outer joins, DBMS allows us to select those rows which are common (satisfies the given) and even those rows that do not satisfy the given condition. To understand this, consider simple instances of project and department table as shown.

Department D_1			Project P_1	
Dept_Mid	DNo	PNo	PNo	Pname
101	2	11	44	D
97	5	22	11	A
120	4	33	22	B

If we perform join operation on these two tables.

Notes

SELECT *.D_1 * .P_1

FROM Department D_1, Project P_1

WHERE D_1 .PNo = P_1 .PNo

The result of this statement is as follows:

Dept_Mid	DNo	PNo	(PNo)	Pname
101	2	11	11	A
97	5	22	22	B

This table shows the simple join operation of two tables - only those rows are selected that satisfied the condition. However, if we want to include those rows that do not satisfy the condition, then we can use the concept of Outer joins.

There are three types of outer joins namely: (1) Left Outer Join, (2) Right Outer Join and (3) Full Outer Join.

1. ***Left Outer Join:*** Left outer join lists all those rows which are common to both the tables and also all those unmatched rows of the table which is specified at the left hand side.

   Example: The application of left outer join will result is the following table.

   SELECT * .D_1, * .P_1

   FROM Department D_1, LEFT OUTER JOIN Project P_1

   WHERE D_1.PNo = P_1.PNo.

   Result

Dept_Mid	DNo	(PNo)	(PNo)	Pname
101	2	11	11	A
97	5	22	22	B
120	4	33	NULL	NULL

   So, the left outer join resulted in a relation that have common rows from both the tables and also the row which does not have match in the other table. The values of the attributes corresponding to second table are NULL values.

2. ***Right Outer Join:*** Right outer join is same as the left outer join but the only difference is the unmatched rows of second table (specified on the right hand side) are listed along with the common rows of both the tables.

   SELECT *.D_1, *.P_1

   FROM Department D_1 RIGHT OUTER JOIN Project P_1

   WHERE D_1 .PNo = P_1 .PNo

Dept-mid	DNo	(PNo)	(PNo)	Pname
NULL	NULL	NULL	44	D
101	2	11	11	A
97	5	22	22	C

   The values of attributes for the first table are declared as NULL.

3. ***Full Outer Join:*** This is same as the right outer join and left outer join but only difference is unmatched rows of both tables are listed along with the common tows of the tables.

Notes

SELECT *.D_1 * .P_1

FROM Department D_1, FULLOUTER JOIN Project P_1

WHERE D_1.PNo = P_1.PNo

The following table shows the result

Dept_Mid	DNo	PNo	(PNo)	Pname
101	2	11	11	A
97	5	22	22	B
120	4	33	NULL	NULL
NULL	NULL	NULL	44	D

In this relation as you can see all the matched and unmatched columns of both the tables are displayed, the values for the unmatched attributes are entered as NULL.

Division

To understand the concept of division operator consider a simple example of two relations with attributes customer name, product id.

1. Customer (Customer name, product id)
2. Product (Product id)

The result of applying the division operator on relations customer and product is shown in the "Result" relation.

A division operator returns a relation (result). That is a collection of rows from first relation (customer) such that for every value in second relation (product). There exists a pair (customer name, product id) in the first relation (customer).

$$\Rightarrow \text{Customer/product} = \text{Result}$$

$$\frac{(\text{Customer, Product})}{(\text{Product id})} = (\text{Custmer name})$$

Customer		Product	Result = customer/product
Customer name	Product id	Product id	Customer name
Ravi	1021321	1021321	Ravi
Kumar	30991231	30991231	Sharma
Girish	2310219		Kumar
Sharma	1021321		Rahul
Rahul	30991231		

The division operation is reverse of cartesian product. We can note that

(Customer/product id) = (customer name) * (product id)

Customer = Result * Product

A binary relationship sets is a relationship between only two entities.

Notes

Case Study

Requirements Analysis

The owner of B&N has thought about what he wants and offers a concise summary:

"I would like my customers to be able to browse my catalog of books and to place orders over the Internet. Currently, I take orders over the phone. I have mostly corporate customers who call me and give me the ISBN number of a book and a quantity. I then prepare a shipment that contains the books they have ordered. If I don't have enough copies in stock, I order additional copies and delay the shipment until the new copies arrive; I want to ship a customer's entire order together. My catalog includes all the books that I sell. For each book, the catalog contains its ISBN number, title, author, purchase price, sales price, and the year the book was published. Most of my customers are regulars, and I have records with their name, address, and credit card number. New customers have to call me first and establish an account before they can use my Web site.

On my new Web site, customers should first identify themselves by their unique customer identification number. Then they should be able to browse my catalog and to place orders online."

DBDudes's consultants are a little surprised by how quickly the requirements phase was completed it usually takes them weeks of discussions (and many lunches and dinners) to get this done but return to their offices to analyze this information.

2.3 Summary

- The relation, which is a two-dimensional table, is the primary unit of storage in a relational database.
- A relational database can contain one or more of these tables, with each table consisting of a unique set of rows and columns.
- A single record is stored in a table as a row, also known as a tuple, while attributes of the data are defined in columns, or fields, in the table.
- The characteristics of the data, or the column, relates one record to another.
- Each column has a unique name and the content within it must be of the same type.

2.4 Keywords

Cross Product (*): This operator returns all the tuples of relation A plus all the tuples of relation B.

Equi-Joins: Equi-join is same as conditional join, the only difference being only equity '=' operator is used to join the two relations.

Joins: Joins are used to combine the information of two relations which have at least one field in common.

Outer Joins: This is a special case "join" operator which considers the NULL values. Generally a 'join' operation performs the cross product of two tables and applies certain join conditions.

Notes

2.5 Self Assessment

Fill in the blanks:

1. .....first formulated and proposed in 1969 by Edgar Codd.
2. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the .....
3. A ..... is a set of attributes.
4. A ..... is an accepted visual representation of a relation.
5. ..... is the term used in the theory for what is commonly referred to as a column.
6. ..... is a special case "join" operator which considers the NULL values.
7. ..... is a collection of facts, which is unorganized but can be made organized into useful information.
8. A recent development is the Object-Relation type-Object model, which is based on the assumption that any fact can be expressed in the form of one or more .....
9. A relation universe U over a header H is a non-empty set of relations with .....
10. ..... operator can be applied to only those relations that are union compat-ible.

2.6 Review Questions

1. Here we consider three tables for solving queries:

Instance S_1 of Sailors

sid	sname	rating	age
22	Dustbin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Instance S_2 of Sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Instance R_1 of Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

(a) Find the names of sailors who have reserved a red or a green boat.

Notes

(b) Find the names of sailors who have reserved at least two boats.

(c) Find the sids of sailors with age over 20 who have not reserved a red boat.

(d) Find the names of sailors who have reserved all boats. The use of the word all (or every) is a good indication that the division operation might be applicable.

(e) Find the names of sailors who have reserved all boats called Interlake.

2. Describe the relational model concept of DBMS.
3. What do you mean by alternatives to the relational model?
4. Describe NULL concept in database.
5 Describe various expressions of relational algebra.
6. Write short note on UNION and INTERSECTION.

Answers: Self Assessment

1.	Relational model	2.	logical schema
3.	heading	4.	table
5.	Attribute	6.	Outer joins
7.	Data	8.	binary relationships
9.	header H	10.	Intersection

2.7 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Unit 3: Structured Query Language

Notes

CONTENTS

Objectives

Introduction

3.1 Structured Query Language (SQL)

3.2 Data Definition

3.3 Data Types

3.4 Schema Definition

3.5 Basic Structure of SQL Queries

3.6 Creating Tables

3.7 DML Operations

3.7.1 SELECT Command

3.7.2 Insert Command

3.7.3 Update Command

3.7.4 Delete Command

3.8 DDL Commands for Creating and Altering

3.9 Set Operations

3.10 Aggregate Functions

3.11 Null Values

3.12 Summary

3.13 Keywords

3.14 Self Assessment

3.15 Review Questions

3.16 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe SQL and data types
- Explain the basic structures of SQL queries
- Know how to create tables
- Realise aggregate functions and null values

Notes

Introduction

SQL stands for Structured Query Language (SQL). It is the most widely used commercial relational database language. SQL has clearly established itself as the standard relational-database languages. There are numerous versions of SQL. The original SQL. Version was developed at IBM's San Jose Research Laboratory (now the Almaden Research Centre). This language, was originally called as Sequel, was used as part of System R Project in early 1970s. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language). SQL uses a combination of relational algebra and relational calculus. Although SQL language is referred as 'Query language', it contains many other capabilities besides querying a data-base. Within the DBMS, SQL will be used to create the tables, translate user requests, maintain the data dictionary, maintain the system catalog, update and maintain the tables, establish security, and carry out backup and recovery procedures.

3.1 Structured Query Language (SQL)

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

The SQL language has several parts:

1. ***Data-definition language (DDL):*** The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
2. ***Interactive data-manipulation language (DML):*** The SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.
3. ***View definition:*** The SQL DOL includes commands for defining views.
4. ***Transaction control:*** SQL includes commands for specifying the beginning and ending of transactions.
5. ***Embedded SQL and dynamic SQL:*** Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, Java, PUr, Cobol, Pascal, and Fortran.
6. ***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 disallowed.
7. ***Authorization:*** The SQL DDL includes commands for specifying access rights to relations and views.

3.2 Data Definition

Data definition in SQL is via the create statement. The statement can be used to create a table, index, or view (i.e., a virtual table based on existing tables). To create a table, the create statement

Notes

specifies the name of the table and the names and data types of each column of the table. Its format is:

create table (relation> (<attribute list>)

where the attribute list is specified as:

<attribute list> :: = <attribute name> (<data type>)[not null] <attribute list>

The data types supported by SQL depend on the particular implementation. However, the following data types are generally included: integer, decimal, real (i.e., floating point values), and character strings, both of fixed size and varying length. A number of ranges of values for the integer data type are generally supported, for example, integer and smallint. The decimal value declaration requires the specification of the total number of decimal digits for the value and (optionally), the number of digits to the right of the decimal point. The number of fractional decimal digits is assumed to be zero if only the total number of digits is specified.

<data type> :: = <integer> | <smallint> | <char(n)> | <float> | <decimal (p[q])>

In addition, some implementations can support additional data types such as bit strings, graphical strings, logical, data, and time. Some DBMSs support the concept of date. One possible implementation of date could be as eight unsigned decimal digits representing the data in the yyyymmdd format. Here yyyy represents the year, mm represents the month and dd represents the day. Two dates can be compared to find the one that is larger and hence occurring later. The system ensures that only legal date values are inserted (19860536 for the date would be illegal) and functions are provided to perform operations such as adding a number of days to a date to come up with another date or subtracting a date from the current date to find the number of days, months, or years. Date constants are provided in either the format given above or as a character string in one of the following formats: mm/dd/yy; mm/dd/yyyy; dd-mm-yy; dd-mm-yyyy. In this unit, we represent a date constant as eight unsigned decimal digits in the format yyyymmdd.

3.3 Data Types

In a broad sense, a data type' defines a set of values, and the allowable operations on those values. Almost all programming languages explicitly include the notion of data type, though different languages may use different terminology. Most programming languages also allow the programmer to define additional data types, usually by combining multiple elements of other types and defining the valid operations of the new data type.

Example: A programmer might create a new data type named "Person" that specifies that data interpreted as Person would include a name and a date of birth.

Common data types may include:

1. Integers,
2. Floating-point numbers (decimals), and
3. Alphanumeric strings.

Example: In the Java programming language, the "int" type represents the set of 32-bit integers ranging in value from -2,147,483,648 to 2,147,483,647, as well as the operations that can be performed on integers, such as addition, subtraction, and multiplication. Colors, on the other hand, are represented by three bytes denoting the amounts each of red, green, and blue, and one

Notes

string representing that color's name; allowable operations include addition and subtraction, but not multiplication.

A data type can also be thought of as a constraint placed upon the interpretation of data in a type system, describing representation, interpretation and structure of values or objects stored in computer memory. The type system uses data type information to check of computer programs that access or manipulate the data.

Here are the most common data types:

char(size)	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
Varchar(size)	Variable-length character string. Max size is specified in parenthesis.
number(size)	Number value with a max number of column digits specified in parenthesis.
Date	Date value
number(size,d)	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

3.4 Schema Definition

It is the overall structure is called a database schema. Database schema is usually graphical presentation of the whole database. Tables are connected with external keys and key colums. When accessing data from several tables, database schema will be needed in order to find joining data elements and in complex cases to find proper intermediate tables. Some database products use the schema to join the tables automatically.

Database system has several schemas according to the level of abstraction. The physical schema describes the database design at physical level. The logical schema describes the database design at logical level. A database can also have sub-schemas (view level) that describes different views of database.

3.5 Basic Structure of SQL Queries

The general format of any SQL query is as follows:

SELECT *[DISTINCT] select-list*

FROM from-list

WHERE *qualification*

1. ***Select Clause:*** SELECT is a command from DML language which is used to select specific columns of the tables. Every SQL query starts with SELECT keyword and is followed by the list of the columns that forms the resulting relation.
2. ***From:*** This clause specifies the names of the tables from which data is to be retrieved. FROM is followed by a set of range values which uniquely identifies a table, when its name occurs more than once.
3. ***Where:*** WHERE keyword is used to specify a condition. All the tuples which satisfies the condition are selected.
4. ***Distinct:*** The use of this keyword is to prevent the duplication of rows, when this keyword is used all the tuples of the table are unique. There is no redundant data. The use of this keyword is optional.
5. ***Select-List:*** Collection of all the column names that user want in the new relation.

Notes

6. ***Qualification:*** It is a boolean expression representing a condition containing logical connectives and also comparison operators.

The following tables are used throughout the unit:

Employee

Eid	Ename	DNo	Esal	Age	Phone
101	John	2	35000	50	24578912
100	Harry	3	18000	29	33273120
107	Henry	7	22000	25	55809192
97	David	5	30000	41	23535135
108	Sam	1	25000	32	24532121
102	Henry	2	22000	35	24578290
120	Smith	4	20000	2	56408489
122	David	6	18000	25	24517635

Department

Dept_Managerid	DNo	Dname	Dlocation	PNo
108	1	Administration	Hyderabad	44
101	2	Accounts	Secunderabad	11
100	3	Packaging	Bombay	44
120	4	Research	Nellore	33
97	5	Accounts	Hyderabad	22
122	6	Operation	Pune	44
107	7	Packaging	Bangalore	55

Project

PNo	Pname	Pduration	Project_Managerid
11	A	9 Months	102
22	B	14 Months	97
33	C	16 Months	120
44	D	25 Months	108
55	E	9 Months	107

Example: Consider a simple query,

Query (a): Select the names and salaries of the employees whose salary is greater than 20000.

Solution:

SELECT DISTINCT E.ename, E.esal

FROM Employee E

WHERE E.esal > 20000.

Notes

The result of these SQL statements is a following relation.

Result using DISTINCT Keyword

Ename	Esal
John	35000
Henry	22000
David	30000
Sam	25000

It is quite interesting to note that this is same as using selection and projection operations in relational algebra.

$$\pi_{ename,\ esal}(\sigma_{esal>20000}\ \ (Employee))$$

Hence, an SQL query is similar to the expression containing selection, projection and cross product in relational algebra.

Notes Table does not contain all the tuples whose salary is greater than 20,000. The tuple with ename "Henry" is repeated twice but is displayed only once. This is due to the use of DISTINCT keyword. If this keyword is not used then one more tuple must be added to table i.e., if we write,

SELECT E.ename, E.esal

FROM Employee E

WHERE E.esal > 20000.

The resulting relation will be

Result without using 'DISTINCT' Keyword

Ename	Esal
John	35000
Henry	22000
David	30000
Sam	25000
Henry	22000

The omission of distinct keyword resulted in a vow (Henry, 22000) which is repeated twice. This is called as multiset of rows. It can be defined as a set consisting of unordered elements which may consist of many copies of same element but the total number of copies of the element is important. For example the set (1,2,1,1,) and (2,1,1,1) are same multiset. Both contains three copies of '1' and only one copy of '2'. But the multiset (1,2,2,1) is different. Number of copies of each element is an important criteria to differentiate between two multisets.

Rules to be Followed in Query Evaluation

Notes

Consider a simple query, Query

Query (b): Find the names of employees who are working in 'accounting' department.

Solution:

SELECT E.ename

FROM Employee E, Department D,

WHERE E.eid = D.Dept_managerid AND

D.Dname = 'Accounting'.

In order to solve this, follow the steps as shown

1. Calculate the cross product of the tables whose names are mentioned in the from-list

Eid	Ename	D.No.	Esal	Age	Phone	Dept. Manager id	d.no.	D.Name	D.Location	P.No.
101	John	2	35000	50	24578912	108	1	Administration	Hyderabad	44
101	John	2	35000	50	24578912	101	2	Accounts	Secunderabad	11
101	John	2	35000	50	24578912	97	5	Accounts	Hyderabad	22
97	Harry	5	30000	41	23535135	108	1	Administration	Hyderabad	44
97	Harry	5	30000	41	23535135	101	2	Accounts	Secunderabad	11
97	Harry	5	30000	41	23535135	97	5	Accounts	Hyderabad	22
10	Sam	1	25000	32	24532121	108	1	Administration	Hyderabad	44
108	Sam	1	25000	32	24532121	101	2	Accounts	Secunderabad	11
108	Sam	1	25000	32	24532121	97	5	Accounts	Hyderabad	22
102	Henry	2	22000	35	24578290	108	1	Administration	Hyderabad	44
102	Henry	2	22000	35	24578290	101	2	Accounts	Secunderabad	11
102	Henry	2	22000	35	24578290	97	5	Accounts	Hyderabad	22

2. Among the obtained rows from the cross product, reject the rows that do not satisfy the qualification condition.
3. The names of the columns that are not present in the select-list are deleted.
4. Remove the rows that have appeared twice when the distinct keyword is not used.

Now let us implement these steps an instance of employee table E

Eid	Ename	DNo	Esal	Age	Phone
101	John	2	35000	50	2457891
97	Harry	5	30000	41	23555135
108	Sam	1	25000	32	2453212
102	Henry	2	22000	35	24578290

An instance of department table D

Dept_Managerid	DNo	Dname	Dlocation	PNo
108	1	Administration	Hyderabad	44
101	2	Accounts	Secunderabad	11
97	5	Accounts	Hyderabad	22

Notes

Step I

Cross product of E and D

Step II

Apply qualification condition

E.eid = D.Dept_managerid AND

D.Dname = 'Accounts' to the cross product table and select second and sixth row.

Step III

Discard all the unwanted columns and the result contains only the ename field.

Result of Step-II

Hid	Ename	D.No.	Esal	Age	Phone	DepLManagerid	D.No.	D.Natne	D .Location	P.No.
101	John	2	35000	50	24578912	101	2	Accounts	Secunderabad	11
97	Harry	5	30000	41	23535135	97	5	Accounts	Hyderabad	22

Result of Step-III

Ename

John

Harry

This indicates that John and Harry are the employees whose department name is 'Accounts'.

Task In DBMS when you can use WHERE clause.

3.6 Creating Tables

The create table statement is used to create a new table. Here is the format of a simple create table statement:

create table "tablename"

("column1" "data type",

"column2" "data type",

"column3" "data type");

Format of create table if you were to use optional constraints:

create table "tablename"

("column1" "data type"

[constraint],

"column2" "data type"

[constraint],

Notes

"column3" "data type"

[constraint]);

[ ] = optional

Notes You may have as many columns as you'd like, and the constraints are optional.

Lab Exercise

```
create table employee
(first varchar(15),
last varchar(20),
age number(3),
address varchar(30),
city varchar(20),
state varchar(20));
```

To create a new table, enter the keywords create table followed by the table name, followed by an open parenthesis, followed by the first column name, followed by the data type for that column, followed by any optional constraints, and followed by a closing parenthesis. It is important to make sure you use an open parenthesis before the beginning table, and a closing parenthesis after the end of the last column definition.

Make sure you seperate each column definition with a comma. All SQL statements should end with a ";".

The table and column names must start with a letter and can be followed by letters, numbers, or underscores - not to exceed a total of 30 characters in length. Do not use any SQL reserved keywords as names for tables or column names (such as "select", "create", "insert", etc.).

Data types specify what the type of data can be for that particular column. If a column called "Last_Name", is to be used to hold names, then that particular column should have a "varchar" (variable-length character) data type.

3.7 DML Operations

Various DML (Data Manipulation Language) commands

1. SELECT used to select different columns
2. INSERT to put the data into tables
3. UPDATE to modify the data
4. DELETE to delete the data.

3.7.1 SELECT Command

SELECT command consists of expressions and strings. In the general form of basic SQL query, the select-list consists of:

1. Expressions and
2. Column name

Notes

SELECT expression AS column name

Where

Expression: It refers to mathematical or string expression, which is specified on column names and constants.

Column Name: It refers to the column's new name in the resultant query. It can also include aggregate functions/operators such as SUM, AVG, COUNT, MIN, MAX etc. It also allows the use of standard ready to use functions such as sqrt, mod etc.

Querys

Query (a): Find the names of all the employees who are working for "operation" department.

Solution:

SELECT E.ename

FROM Employee E, Department D

WHERE E.DNo = D.DNo AND

D.Dname = "operation". The answer is "David"

Query (b): Calculate the increment in the salary of the employees who are working on two different projects carried out at the same location.

Solution:

SELECT E.ename, E.esal + 1000 As salary

FROM Employee, Department D_1, Department D_2

WHERE D_1 .Dept_managerid = E.eid AND

D_1 .Dlocation = D_2.Dlocation AND

D_1.PNO <> D_2.PNO

This will result in a new relation when 'esaF coTumn is renamed as 'salary' by using AS

Result: Use of 'AS' clause

Ename	Salary
David	000
Sam	000

With select command we can select specific columns by specifying the names of the columns. But by using '*' with select command all the columns can be selected at once (no need to mention the names of all the columns).

Query (c): List all the information about the employees whose salary is greater than or equal to 20000.

Solution:

SELECT *

FROM Employee E

WHERE E.esal > = 20000.

Notes

The use of 'select *' is useful for those queries which are interactive but it is a poor style as it does not clearly mention the schema of the resulting relation. To determine the schema one has to refer the relation mentioned in FROM clause.

Result of "Select *"

Eid	Ename	DNo	Esal	Age	Phone
101	John	2	35000	50	24578912
100	Henry	7	22000	25	55809192
97	David	5	30000	41	23535135
108	Sam	1	25000	32	24532121
102	Henry	2	22000	35	24578290
120	Smith	4	20000	20	56408489

When strings are sorted alphabetically then we can apply comparison operators.

Collation

It is a mechanism that is used to compare the string characters to determine which characters are smaller (ASCII code) than the other characters in a particular string.

In addition to this, SQL provides another operator LIKE operator to perform pattern matching. It is of the form,

Scalar expression LIKE literal [Escape character] where,

Scalar expression = string value

Literal = '-' single character

= '%' zero or more character sequence!

The string '% ar %' results the set of strings which contains the characters 'ar' in them. The length of the resulting string must be atleast four characters.

To understand this, consider the following query

Query (d): List the names of the employees whose name start with 'H' and has 'r' as the third character.

Solution:

SELECT E.ename AS name, E.sal as salary

FROM Employee E

WHERE E.ename LIKE 'H-r%'.

This will result in a relation consisting of names of all the employees whose name start with H and third character is 'r'. The answer in this case is

Name	Salary
Harry	18000

Notes

Query (e): Find all the employees whose department name starts with 'pac'.

Solution:

SELECT *

FROM Employee E, Department D

WHERE E.eid = D.Dept_Managerid AND

D.Dname LIKE 'pac %'

3.7.2 Insert Command

The insert statement is used to insert or add a row of data into the table.

To insert records into a table, enter the key words insert into followed by the table name, followed by an open parenthesis, followed by a list of column names separated by commas, followed by a closing parenthesis, followed by the keyword values, followed by the list of values enclosed in parenthesis. The values that you enter will be held in the rows and they will match up with the column names that you specify. Strings should be enclosed in single quotes, and numbers should not.

insert into "tablename"

(first_column,...last_column)

values (first_value,...last_value);

In the example below, the column name first will match up with the value 'Luke', and the column name state will match up with the value 'Georgia'.

Example: insert into employee

(first, last, age, address, city, state)

values ('Luke', 'Duke', 45, '2130 Boars Nest',

'Hazard Co', 'Georgia');

Notes All strings should be enclosed between single quotes: 'string'

3.7.3 Update Command

Update Rows

The syntax for this command is

update tablename set colname1=colvalue where colname2=colvalue;

Example: update Student set Name = 'Ajay' where id = 2;

This command has updated the Name 'Rose' in Student table whose id is 2.

Notes

3.7.4 Delete Command

Delete Rows

The syntax for this command is-

delete from tablename where [search_conditions];

Example: delete from Student where id=1;

This statement is used to delete the row from Student table where the student id is 1.

Task Use update command in existing table.

3.8 DDL Commands for Creating and Altering

Various DDL (Data Definition Language) commands

1. CREATE to create a new table
2. ALTER to modify the structure of the table
3. DROP to delete the table from the database

Create Command

Create Table

This statement is used to create a table. The syntax for this command is create table tablename (colname1 datatype [constraint], colname2 datatype [constraint]);

Example: create table Student (id number(4) primary key, Name varchar2(20));

It creates the table Student which has two fields id i.e. Student id and Name i.e. the student name. The number and varchar2 are the data types of id and Name respectively. Field 'id' has the size 4 means it can take id up to 4 digits and same for Name, it can take the size upto 20 characters. And also added the constraint Primary key to the field 'id'.

Alter Command

Alter Table

This command is used to add, drop columns in a table. The syntax for this command is

alter table
tablename add colname1 datatype [constraint];
alter table tablename drop column colname1;

Example: alter table Student add DOB date

Notes

This command is used to add new field DOB in Student table. It's datatype is date. This is also used for drop column from the table. It will drop the DOB field by query given below:

Alter table Student drop column DOB;

Drop Command

The drop table command is used to delete a table and all rows in the table.

To delete an entire table including all of its rows, issue the drop table command followed by the tablename. drop table is different from deleting all of the records in the table.

Deleting all of the records in the table leaves the table including column and constraint information. Dropping the table removes the table definition as well as all of its rows.

drop table "tablename"

Example: drop table myemployees;

3.9 Set Operations

The SQL operations union, intersect, and except operate on relations and correspond to the relational-algebra operations U, n, and -. Like union, intersection, and set difference in relational algebra, the relations participating in the operations must be compatible; that is, they must have the same set of attributes.

```
select customer-name
from depositor
```

and the set of customers who have a loan at the bank, which can be derived by

```
select customer-name
from borrower
```

You will refer to the relations obtained as the result of the preceding queries as d and b, respectively.

The Union Operation

To find all customers having a loan, an account, or both at the bank, we write

```
(select customer-name
from depositor)
union
(select customer-name
from borrower)
```

The union operation automatically eliminates duplicates, unlike the select clause. Thus, in the preceding query, if a customer-say, Jones-has several accounts or loans (or both) at the bank, then Jones will appear only once in the result.

If we want to retain all duplicates, we must write union all in place of union:

```
(select customer-name from depositor)
union all (select Customer-name from borrower)
```

Notes

The number of duplicate tuples in the result is equal to the total number of duplicates that appear in both d and b. Thus, if Jones has three accounts and two loans at the bank, then there will be five tuples with the name Jones in the result.

The Intersect Operation

To find all customers who have both a loan and an account at the bank, we write

(select distinct customer-name from depositor)
intersect (select distinct customer-name from borrower)

The intersect operation automatically eliminates duplicates. Thus, in the preceding query, if a customer-say, Jones-has several accounts and loans at the bank, then Jones will appear only once in the result.

If we want to retain all duplicates, we must write intersect all in place of intersect:

(select customer-name from depositor)
intersect all (select customer-name from borrower)

The number of duplicate tuples that appear in the result is equal to the minimum number of duplicates in both d and b. Thus, if Jones has three accounts and two loans at the bank, then there will be two tuples with the name Jones in the result.

3.10 Aggregate Functions

Different aggregate operators that SQL support are,

1. ***Count:*** COUNT followed by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.
2. ***SUM:*** SUM followed by a column name returns the sum of all the values in that columns. If DISTINCT keyword is used then it will return the sum of all unique values in the columns.
3. ***AVG:*** AVG followed by a column name returns the average value of that column values. If DISTINCT keyword is used then it will return the average of distinct values only.
4. ***MAX:*** MAX followed by a column name returns the maximum value of that column.
5. ***MIN:*** Min followed by column name returns the minimum value of that column.

Queries Based on Aggregate Functions

Query (a): Find the sum of salaries of all the employees and also the minimum, maximum and average salary.

Solution:

SELECT SUM(E.esal) AS sum_salary, MAX(E.esal) AS Max_salary, MIN(E.esal) AS Min_salary, AVG([DISTINCT] E.esal) AS Average_salary

FROM Employee E.

This query calculates the total, minimum, maximum and average salaries and also renames the column names.

Notes

Query (b): List the number of employee in the company

Solution:

SELECT COUNT (*)

FROM Employee E.

Query (c): List the number of employees who are working on project number 44

Solution:

SELECT *COUNT(*)*

FROM Employee E, Department D

WHERE E.DNo = D.DNo AND

D.PNo = 44.

Query (d): Find the name and age of the eldest employee

Solution:

SELECT *E.ename, E.age*

FROM Employee E

WHERE E.age = (SELECT MAX(E2.age)

FROM *employees E2)*

(OR)

The above query can also be written as

SELECT *E.ename, E.age*

FROM Employee E

WHERE (SELECT MAX (E2.age)

FROM Employees E2 = E.age). Values

Task Discuss UNION operations.

3.11 Null Values

Null means nothing. Null values are those values which are assigned to an attribute if its value is unknown or is not applicable. Null values are used by DBMS when the user does not know the type information to be entered for a particular field, so the user doesn't enter any data and DBMS assigns that field a NULL value. A NULL value does not represent a zero or spaces it just means the absence of value for that field and this value can be inserted later. For example in the employee table it is not necessary that all the employees should have a phone number, so for the employer who does not have one yet, a NULL value can be assigned for that. When we insert a tuple (124, 'Kelly', 6, NULL, 26, NULL) in employer table then for the attribute e.sal and phone NULL values are assigned. Assignment of NULL value simply indicates that the value is not known or is inapplicable for that attribute. A field which is not declared as NOT NULL can have NULL values. But this inclusion of special type of value may result in the complications of other operations. NULL values are used to deal with the exceptional data or the data that is not

Notes

complete and any operation performed on null values may result into inconsistent data. In the following section we will discuss the problems that arises due to NULL value.

Comparison using NULL Values

When we compare valid values with NULL values boolean logic doesn't come handy i.e., we can't use a boolean logic which is two valued - TRUE or FALSE. With NULL values we have to use a then valued logic - TRUE or FALSE or UNKNOWN. For example, previous; we entered a NULL value for salary attribute of Kelly. Now if apply condition that-list all the employees whose salary > 20000, the evaluation of this condition for likely is UNKNOWN because salary is a NULL value for Kelly. Similarly, for other comparison operators (>, <, =, <>), the evaluation of the condition will always be unknown.

SQL provides is NULL comparison operator to compare to: NULL values i.e., to test whether a value is NULL or not. If we apply this operator to salary attribute

IS NULLE.esal

then it will return a TRUE to indicate that esal attribute is a null value.

Logical Connectives AND, OR and NOT

Use of logical connectives with NULL values becomes a lit complicated if we doesn't use three valued logic. For example consider the following query.

Query : List all the employees whose age is less than 40 whose salary is grater than 30000.

Solution:

SELECT *

FROM employee E

WHERE E.age<40ANDE.esal>30000

All the tuples that satisfy this condition are selected but what about the tuple we inserted with esal as NULL value? In this case, this condition will evaluate to unknown logical operators which involves at least one column whose value is assigned as NULL will always result in an unknown value. The following table will give you a better understanding of logical operators when used with null values. Point to note here is that we are using a three valued logic TRUE, FALSE or UNKNOWN i.e., the logical condition applied may evaluate to any one of them. (Unknown is used in case of NULL values).

Operation	Results and Reason
1. X AND Y * TRUE * FALSE * UNKNOWN	If both X and Y are true If either X or Y is false If either X or Y is unknown (null values)
2. X OR Y * TRUE * FALSE * UNKNOWN	If either of them is true. If either of them is false If one of the argument is, false and other is unknown
3. NOT X * TRUE * FALSE * UNKNOWN	If X is false If X is true If X is unknown

Notes

Impact on SQL Constructs

This unit deals with the impact of NULL values on the SQL constructs. To understand this impact, consider a simple student table.

Student Relation

Student_id	Std_name	Course_id	Class	Group
1	A	101	2	B
2	B	102	3	NULL
3	C	101	2	A
4	D	103	4	NULL
5	E	104	5	A
6	F	102	3	B
7	G	105	6	NULL

A Simple Query

Query: List all the names of students who belongs to group 'B'.

Solution:

```
SELECT    *
FROM      Student S
WHERE     S.group = 'B'
```

This solution will result in the set of tuples that satisfies the 'WHERE' condition and all other tuples that does not satisfy this condition are ignored in addition to these tuples. Tuples with NULL values are also ignored because for them the condition evaluates to false or unknown. This elimination of rows that resulted unknown, makes the queries that involves EXISTS and/ or UNIQUE much more simple, easy to understand and makes the evaluation of these queries (nested queries specially) much more easier.

We know that the comparison of any two fields with null values for equality is an unknown value. But when it comes to (=) equality operator, the two null value attributes are treated as equal. If a field contains two null values then that is considered as duplicate values. Two tuples are said to be duplicates if they hold the same value or if they hold the null values. So, the comparison of NULL values with the "=" operator always results in TRUE.

The result of all the arithmetic operators (+, -,%,/,*) results in an unknown value (NULL) if any one of the argument is a NULL value. Similarly with all the aggregate operators the result is NULL if these operators are applied to a NULL value. Aggregate functions simply delete the NULL values and then returns the result of aggregate operator i.e., SUM, AVG, MIN, MAX, COUNT(DISTINCT) simply delete/ignore the NULL values and returns the result of other NOT NULL tuples. Only exception in aggregate operator is count(*) which does not ignore/ delete. The NULL values, it counts them and then return the number of tuples in the table.

Disallowing NULL Values

The fields can take on NULL values if they are not declared as NOT NULL, we can restrict the insertion of null values for the field by declaring that field as NOT NULL. This means that the field cannot take null values. For the primary key constraint i.e., the field which is declared as primary key is also declared as NOT NULL. This declaration is implicit declaration done by DBMS.

Notes

Example: CREATE TABLE Student

(Sid INT NOT NULL,

Sname CHAR(10) NUT NULL,

Project VARCHAR(15),

Class INT,

PRIMARY KEY(eid))

In this declaration, creation of student table, Sid is the primary key hence it must be unique and it should not be NOT NULL. Project field indicates the project taken up by the student. This field can take NULL values as it is possible that a student is not interested in taking any project/not yet assigned a project. We restrict the insertion of NULL values in the same field by declaring it as NOT NULL.

Lab Exercise ***Conceptual Design:*** Create 3 tables name: Books, Orders and Customers and make ER Diagram between them.

3.12 Summary

- The Structured Query Language (SQL) is a 4th Generation Language (4GL) generally used for querying the database. Following is a consolidated list of SQL statements:
  - SELECT — Data retrieval statement
  - INSERT — Add rows
  - UPDATE — Update row
  - DELETE — Delete rows
  - CREATE — Create new tables/views
  - ALTER — Alter the schema or view
  - DROP — Delete the table
  - RENAME — Rename a table
  - COMMIT — Buffer to disk
  - ROLLBACK — Rollback the changes made
  - GRANT — Assign privileges to users
  - REVOKE — Remove the assigned privileges
- The basic commands that can be used under Oracle 8i environment were discussed. For example, @,/are some of the commands.
- Oracle9i SQL *PLUS offers a rich set of data types like integer, float, number, date, etc.
- The SELECT statements is used retrieve a set of rows from a specified table depending upon the WHERE clause.

Notes

- SQL supports many functions that can be used with dates, numbers, and character strings. You can also convert from one data type to another implicitly or explicitly. For example, TO_DATE, TO_CHAR, and so on.
- Another set of functions that are used for data manipulation are group functions like AVG, MAX, COUNT, WIN, etc.
- To join two or more tables we have equi join, self join, and outer (uses a + sign). To write certain complicated queries, use of subqueries makes things easier. A subquery is written in the where condition of the outer query along with operators like =, IN, ANY, ALL or EXISTS.
- SQL also supports set operations like UNION, DIFFERENCE, and MINUS. There a number of database objects that can be created and altered:
  - CREATE TABLE — Creating a new table
  - CREATE VIEW — Creating a new view
  - CREATE SEQUENCE — Creating an automatic sequence of numbers
  - INDEX — Creating an index on columns other than primary key for efficient data retrieval
- The two statements GRANT and REVOKE are important for assigning or removing privileges of a table or column or any database object.
- It is possible to set locks to tables in a multi-user environment for efficient security purpose. Finally, several additional examples were discussed on SQL.

3.13 Keywords

Creating table: To create a table, the create statement specifies the name of the table and the names and data types of each column of the table.

Data Definition Language (DDL): This part of SQL supports the creation, deletion and modification of tables. Integrity constraints can be defined on tables, either when the table is created or later. The DDL also provides commands for specifying access rights to tables. The commercial implementations also provide commands for creating and deleting indexes.

Data Manipulation Language (DML): This part of SQL allows users to pose queries, insert tuples, delete tuples and modify tuples (rows) in the database.

Select clause: SELECT is a command from DML language which is used to select specific columns of the tables.

SQL: SQL is the standard language for relational database management systems.

3.14 Self Assessment

Choose the appropriate answer:

1. SQL stands for:
   - (a) Systematic Query Language
   - (b) Semantic Query Language
   - (c) Structured Query Language
   - (d) Structured Queue Language

Notes

2. ANSI stands for:
   (a) American National Standards Institute
   (b) American National Systematic Institute
   (c) American Nation Standards Institute
   (d) Ahmedabad National Standards Institute
3. DML stands for:
   (a) Document Manipulation Language
   (b) Data Manipulation Language
   (c) Data Maintain Language
   (d) Database Manipulation Language

Fill in the blanks:

4. ..... can be used to create a table, index, or view.
5. The ..... supported by SQL depend on the particular implementation.
6. Database system has several schemas according to the level of .....
7. ..... keyword is used to specify a condition.
8. The ..... statement is used to insert or add a row of data into the table.
9. The drop table command is used to delete a table and ..... in the table.
10. Null means .....

3.15 Review Questions

1. Define query.
2. What is SQL? Explain its features.
3. Explain with examples different SQL commands used for creating and deleting relations.
4. Explain the basic structure of SQL query.
5. List some of the set operations supported by SQL. Give examples.
6. Explain different comparison and logical operators supported by SQL. Give examples.
7. Explain how to order the tuples in a table using SQL?
8. Explain group-by clause, aggregate functions and having clause available in SQL.
9. Explain with examples different SQL commands used for modifying the database.
10. Write a query to find the distinct customers and branch names of the branches situated in the city "Hyderabad" where the customers have taken the loans.

Answers: Self Assessment

1.	(c)	2.	(a)
3.	(b)	4.	data definition

Notes

5. data types
6. abstraction
7. WHERE
8. insert
9. all rows
10. nothing

3.16 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Unit 4: Advanced SQL

Notes

CONTENTS

Objectives

Introduction

4.1 Subqueries

4.2 Nested Subqueries

4.3 Complex Queries

4.4 Views

4.5 Joined Relations

4.5.1 Inner Join

4.5.2 Natural Join

4.5.3 Left Outer Join

4.5.4 Full Outer Join

4.6 Summary

4.7 Keywords

4.8 Self Assessment

4.9 Review Questions

4.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe subqueries and nested subqueries
- Explain complex queries
- Define views
- Discuss joined relations

Introduction

In this unit, we give you the details of some of the advanced features of Structured Query Language. We will discuss Assertions and Views, Triggers, Standard Procedure and Cursors. The concepts of embedded and dynamic SQL and SQLJ, which is used along with JAVA, are also been introduced. Some of the advanced features of SQL have been covered. We will provide examples in various sections rather than including a separate section of examples. The examples given here are in a SQL3 standard and will be applicable for any commercial database management system that supports SQL3 standards.

Notes

4.1 Subqueries

The expression following WHERE can be either a simple predicate as explained above or it can be a query itself! This part of the query following WHERE is called a Subquery.

A subquery, which in turn is a query can have its own subquery and the process of specifying subqueries can continue ad infinitum! More practically, the process ends once the query has been fully expressed as a SQL statement.

Subqueries can appear when using the comparison predicate, the IN predicate and when quantifiers are used.

Subqueries are similar to SELECT chaining. While SELECT chaining combines SELECTs on the same level in a query, however, subqueries allow SELECTs to be embedded inside other queries. They can perform several functions:

1. They can take the place of a constant.
2. They can take the place of a constant yet vary based on the row being processed.
3. They can return a list of values for use in a comparison.

Subqueries always appear in the HAVING clause or the WHERE clause of a query. A subquery may itself contain a WHERE clause and/or a HAVING clause, and, consequently.

Example: SELECT AVG(salary)FROM employee WHERE title = 'Programmer';

This statement will return the average salary for all employees whose title is equal to 'Programmer'

The HAVING clause allows you to specify conditions on the rows for each group - in other words, which rows should be selected will be based on the conditions you specify. The HAVING clause should follow the GROUP BY clause if you are going to use it.

HAVING clause syntax:

```
SELECT column1, SUM(column2)
FROM "list-of-tables"
GROUP BY "column-list"
HAVING "condition";
```

HAVING can best be described by example. Let's say you have an employee table containing the employee's name, department, salary, and age. If you would like to select the average salary for each employee in each department, you could enter:

```
SELECT dept, avg(salary)
FROM employee
GROUP BY dept;
```

But, let's say that you want to ONLY calculate & display the average if their salary is over 20000:

```
SELECT dept, avg(salary)
FROM employee
GROUP BY dept
HAVING avg(salary) > 20000;
```

Notes

4.2 Nested Subqueries

A query inside a query is called as nested query. Inner query is called as sub query. Sub query is usually present in WHERE or HAVING clause.

Consider the following example,

Querys

Query (a): Find the names of employees who are working in department number 6.

Solution:

```
SELECT    E.ename
FROM      Employee E
WHERE     E.eid IN (SELECT D.Dept_managerid
FROM      Department D
WHERE     D.DNo = 6)
```

This query returns "David"

These SQL statements can be read as,

Select employee name from table employee E such that E.eid is present in dept_managerid where department number is 6. The DBMS first solves the sub query,

```
SELECT    D.Dept_managerid
FROM      Department D
WHERE     D.DNo = 6
```

and retrieves the managerid of all the employees who are working for department number 6.

Result: **D.Dept_managerid**

122

Next, the DBMS checks the presence of this id in employee table. If there exists an id = 122 then it displays the result for that id.

Result: **ename**

David

The main query that contains the sub queries in called as outer query.

As already mentioned IN can also be replaced with NOT IN. In this case it checks for the tuples which are not present in the given relation. In order to find the employee who are not working in de number 6, we just have to replace IN by NOT IN and the whole query remains the same.

```
SELECT    Rename
FROM      Employee E
WHERE     E.eid NOT IN (SELECT D.Dept_managerid
FROM      Department D
WHERE     D.DNo = 6)
```

Query (b): Find the names of employees who are working on project C.

Notes

Solution:

```
SELECT    E.ename
FROM      Employee E
WHERE     E.eid IN (SELECT D.Dept_managerid
FROM      Department D
WHERE     D.PNo IN (SELECT P.PNo
FROM      Project P
WHERE     P.Pname = 'C')
```

This query is solved in a bottom-up approach. First, the second sub query is solved and all the project numbers are selected whose name is C. This output is fed as input to first sub query where in the department manager id is selected. Finally the names of employees are listed whose id is present in the relation.

Step by Step Procedure

1. PNo. is selected where pname is 'C' i.e. 33.
2. The first sub query checks whether this PNo is present in department or not. If it is present then its corresponding dept_managerid is selected i.e., 120.
3. The main query checks whether this 'id' is present in employee or not. If it is present then corresponding ename is retrieved i.e., Smith.

Task Discuss the purpose of HAVING clause.

4.3 Complex Queries

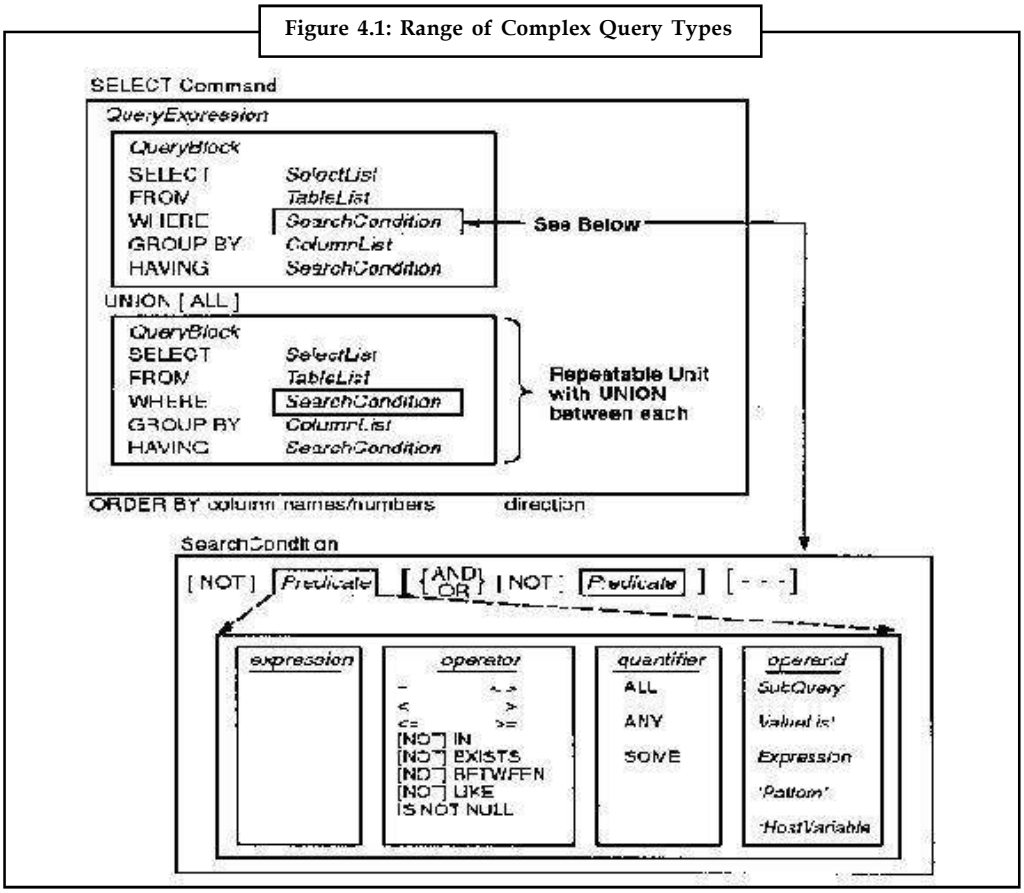
In addition to the simple queries shown in the previous section, you can create complex queries, which may contain more than one SELECT statement. At the highest level, a query is a SELECT statement, which consists of a query expression followed by an optional ORDER BY clause. At the next lower level, you can combine different query blocks into a single query expression with the UNION operator. Lower still, inside each query block is an optional search condition, which can contain predicates that incorporate subqueries. A subquery is always a single query block (SELECT) that can contain other subqueries but cannot contain a UNION. A query expression can contain a maximum of 16 query blocks from all sources, including UNION, subqueries, and the outer query block.

You can create a complex query by using the following:

1. UNION operator, which allows you to take the union of all rows returned by several query blocks in one SELECT statement.
2. Subqueries (also known as nested queries), which allow you to embed a query block within the search condition of an outer SELECT statement.
3. Special predicates, such as ANY, ALL, SOME, EXISTS, and IN, which allow you to compare the value of an expression with the value of special structures and subqueries.

Notes

Figure 4.1: Range of Complex Query Types



UNION Queries

A SELECT statement can consist of several query blocks connected by UNION or UNION ALL statements. Each individual SELECT statement returns a query result which is a set of rows selected from a specified table or tables. The union of these query results is presented as a table that consists of all rows appearing in one or more of the original query results.

If only the UNION statement is used, all duplicate rows are removed from the final set of rows. In this case, the maximum size of a tuple in the query result is given by the following formula:

$$(SelectListItems + 1)*2 + (SumListLengths) <= 4000$$

where,

SelectListItems is the number of items in the select list.

SumListLengths is the sum of the lengths of all the columns in the select list.

At compile time, *SumKeyLengths* is computed assuming columns of NULL and VARCHAR contain no data. At run time, the actual data lengths are assumed.

If the UNION ALL operator is used, duplicates are not removed. Candidates for duplicate removal are evaluated by comparing entire tuples, not just a single field. Only if two or more rows are entirely alike are the duplicates removed. In the case of the UNION ALL operator, the maximum size of a tuple in the query result is 3996 bytes, as it is for a non-UNION query expression. You cannot use LONG columns in a UNION statement.

Notes

Example: To find all customers having a loan, an account, or both at the bank, we write

```
(select customer-name
from depositor)
union
(select customer-name
from borrower)
```

The union operation automatically eliminates duplicates, unlike the select clause. Thus, in the preceding query, if a customer-say, Jones-has several accounts or loans (or both) at the bank, then Jones will appear only once in the result.

If we want to retain all duplicates, we must write union all in place of union:

```
(select customer-name
from depositor)
union all (select Customer-name
from borrower)
```

The number of duplicate tuples in the result is equal to the total number of duplicates that appear in both d and b. Thus, if Jones has three accounts and two loans at the bank, then there will be five tuples with the name Jones in the result.

4.4 Views

A view is a virtual table, which does not actually store data. But if it does not store any data, then what does it contain?

A view actually is a query and thus has a SELECT FROM WHERE ….. clause which works on physical table which stores the data. Thus, the view is a collection of relevant information for a specific entity.

Example: A student's database may have the following tables:

STUDENT (name, enrolment-no, dateofbirth)

MARKS (enrolment-no, subjectcode, smarks)

For the database above a view can be created for a Teacher who is allowed to view only the performance of the student in his/her subject, let us say MM-01.

```
CREATE VIEW SUBJECT-PERFORMANCE AS
(SELECT s.enrolment-no, name, subjectcode, smarks
FROM STUDENT s, MARKS m
WHERE s.enrolment-no = m.enrolment-no AND
subjectcode 'MM-01' ORDER BY s.enrolment-no;
```

A view can be dropped using a DROP statement as:

```
DROP VIEW SUBJECT-PERFORMANCE;
```

The table, which stores the data on which the statement of the view is written, is sometimes referred to as the base table. You can create views on two or more base tables by combining the data using joins. Thus, a view hides the logic of joining the tables from a user. You can also index the views too. This may speed up the performance. Indexed views may be beneficial for very large tables. Once a view has been created, it can be queried exactly like a base table.

Notes

Example: SELECT *

FROM STUDENT-PERFORMANCE

WHERE smarks >50

How the Views are Implemented?

There are two strategies for implementing the views. These are:

1. Query modification
2. View materialisation.

In the query modification strategy, any query that is made on the view is modified to include the view defining expression.

Example: Consider the view STUDENT-PERFORMANCE. A query on this view may be: The teacher of the course MM-0a wants to find the maximum and average marks in the course. The query for this in SQL will be:

SELECT MAX(smarks), AVG(smarks)
FROM SUBJECT-PERFORMANCE

Since SUBJECT-PERFORMANCE is itself a view the query will be modified automatically as:

SELECT MAX (smarks), AVG (smarks)
FROM STUDENT s, MARKS m

WHERE s.enrolment-no=m.enrolment-no AND subjectcode= "MM-01";

However, this approach has a major disadvantage. For a large database system, if complex queries have to be repeatedly executed on a view, the query modification will have to be done each time, leading to inefficient utilisation of resources such as time and space.

The view materialisation strategy solves this problem by creating a temporary physical table for a view, thus, materialising it. However, this strategy is not useful in situations where many database updates are made on the tables, which are used for view creation, as it will require suitable updating of a temporary table each time the base table is updated.

Can views be used for Data Manipulations?

Views can be used during DML operations like INSERT, DELETE and UPDATE. When you perform DML operations, such modifications need to be passed to the underlying base table. However, this is not allowed on all the views. Conditions for the view that may allow Data Manipulation are:

A view allows data updating, if it follows the following conditions:

1. If the view is created from a single table, then:
   (a) For INSERT operation, the PRIMARY KEY column(s) and all the NOT NULL columns must be included in the view.
   (b) View should not be defined using any aggregate function or GROUP BY or HAVING or DISTINCT clauses. This is due to the fact that any update in such aggregated attributes or groups cannot be traced back to a single tuple of the base table. For example, consider a view avgmarks (coursecode, avgmark) created on a base table

Notes

student(st_id, coursecode, marks). In the avgmarks table changing the class average marks for coursecode "MA 03" to 50 from a calculated value of 40, cannot be accounted for a single tuple in the Student base table, as the average marks are computed from the marks of all the Student tuples for that coursecode. Thus, this update will be rejected.

2. The views in SQL that are defined using joins are normally NOT updatable in general.
3. WITH CHECK OPTION clause of SQL checks the updatability of data from views, therefore, must be used with views through which you want to update.

Views and Security

Views are useful for security of data. A view allows a user to use the data that is available through the view; thus, the hidden data is not made accessible. Access privileges can be given on views. Let us explain this with the help of an example.

Consider the view that we have created for teacher-STUDENT-PERFORMANCE. We can grant privileges to the teacher whose name is 'ABC' as:

GRANT SELECT, INSERT, DELETE ON STUDENT-PERFORMANCE TO ABC WITH GRANT OPTION;

Notes The teacher ABC has been given the rights to query, insert and delete the records on the given view. Please also note s/he is authorised to grant these access rights (WITH GRANT OPTION) to any data entry user so that s/he may enter data on his/her behalf. The access rights can be revoked using the REVOKE statement as:

REVOKE ALL ON STUDENT-PERFORMANCE FROM ABC;

Task Create a new table with five column and create view on this table.

4.5 Joined Relations

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables. A JOIN is a means for combining fields from two tables by using values common to each.

SQL is relational database query language and as such, one of its most important features is its ability to retrieve information from several different related tables. In relational database terms, this process is called a join. The tables to be joined are named in the From clause of the Select with each table name separated by a comma. The relationships between the tables in a join are defined by the predicate in the Where clause.

SQL specifies four types of JOINs:

1. INNER
2. OUTER
3. LEFT
4. RIGHT

Notes

4.5.1 Inner Join

This is the simplest of all join operations.

Inner joins return all rows from multiple tables where the join condition is met. There must be a matching value in a field common to both tables. An Inner Join cannot be nested inside a Left Join or Right Join, it creates a new result table by combining column values of two tables based upon the join-predicate. The join condition determines whether both records are matched or not. If there is no match found, no records is returned.

loan inner join borrower on loan.loan-no = borrower.loan-no

The expression computes the theta join of the loan and the borrower relations, with join condition being loan.loan_no = borrower.loan_no. The attributes of the result consist of the attributes of the left-hand-side relation followed by the attributes of the right hand side of the relation.

Notes The attribute loan _no appears twice in the result. The first occurrence is from loan, and the second is from borrower.

Result of loan innter join borrower on loan.loan_no = borrower.loan_no

We rename the result relation of a join and the attributes of the result relation using as clause, as shown below:

loan inner join borrower on l oan.loan_no = borrower.loan_no
As lnbr (branch, loan_no, amount, cust, cust_loan_no)

The second occurrence of loan_no has been renamed as cust_loan_no. the ordering of the attributes in the result of the join is important for renaming.

Br_name	Loan_no	Loan_amount
K R CIRCLE	KL1	6000
SARASWATHIPURAM	SL1	7000
K R CIRCLE	KL2	4000

Loan relation

Cust_name	Loan_no
Padma	GL1
Vijaya	SL1
Shreya	KL2

Borrower relation

Br_name	Loan_no	Loan_amount	Cust_name	Loan_no
SARASWATHIPURAM	SL1	7000	Vijaya	SL1
K R CIRCLE	KL2	4000	Shreya	KL2

Notes

4.5.2 Natural Join

Natural join combines two tables based on their common columns i.e. columns with the same name. Therefore join condition is hidden and dependant on table structures at runtime. This obviously creates potential future danger-as soon as table structure changes, result can become unpredictable yet syntactically correct it offers a further specialization of equi-joins. The join predicate arises implicitly by comparing all columns in both tables that have the same column-name in the joined tables. The resulting joined table contains only one column for each pair of equally-named columns.

As previously noted in RELATIONAL ALGEBRA, this operation when performed, forces an equality on attributes, which are common in the relation specified. If we take a natural join of borrower and loan, then equality is forced on the attribute loan_no. loan natural join borrower

The only attribute common to loan and borrower is loan_no. the result of the expression is similar to the result of the inner join except that the attribute loan_no appears only once in the result of the natural join.

Br_name	Loan_no	Loan_amount	Cust_name
SARASWATHIPURAM	SL1	7000	Vijaya
K R CIRCLE	KL2	4000	Shreya

4.5.3 Left Outer Join

In **left outer join:** rows satisfying selection criteria from both joined tables are selected as well as all remaining rows from left joined table are being kept along with Nulls instead of actual right joined table values. or we can say it returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate). If the right table returns one row and the left table returns more than one matching row for it, the values in the right table will be repeated for each distinct row on the left table.

The LEFT OUTER JOIN expression is written as follows:

loan left outer join borrower on loan.loan_no = borrower.loan_no

Br_name	Loan_no	Loan_amount	Cust_name	Loan_no
SARASWATHIPURAM	SL1	7000	Vijaya	SL1
K R CIRCLE	KL2	4000	Shreya	KL2
K R CIRCKE	KL1	6000	Null	Null

4.5.4 Full Outer Join

The full outer join type is a combination of the left and right outer-join types. After the result of the inner join is computed, tuples form the left-hand-side relation that did not match with any from the right- hand-side are extended with nulls and are added to the result. Similarly, tuples from the right-hand-side relation that did not match with any tuples form the left-hand side relation are also extended with nulls, and are added to the result.

loan full outer join borrower using (loan_no)

Notes

The result of the expression is as follows:

Br_name	Loan_no	Loan_amount	Cust_name
SARASWATHIPURAM	SL1	7000	Vijaya
K R CIRCLE	KL2	4000	Shreya
Null	GL1	Null	Padma
K R CIRCLE	KL1	6000	Null

Find all customers who have either an account or a loan (but not both) at the bank

Select cust_name

From (depositor natural full outer join borrower)

Where account_no is null or loan_no is null

Task Role of natural join in DBMS.

Lab Exercise Create a table with 6 column and enter at least five values in the table and do this exercise:

1. select top 40% from table
2. select column1, column2 and column3 simultaneously from table (joint all)

4.6 Summary

- SQL also has a good programming level interfaces.
- The SQL supports a library of functions for accessing a database.
- These functions are also called the Application Programming Interface (API) of SQL.
- The advantage of using an API is that it provides flexibility in accessing multiple databases in the same program irrespective of DBMS, while the disadvantage is that it requires more complex programming.

4.7 Keywords

Full Outer Joins: The full outer join type is a combination of the left and right outer-join types.

Inner Joins: Inner joins return all rows from multiple tables where the join condition is met.

Natural Joins: Natural join combines two tables based on their common columns i.e. columns with the same name.

Nested Query: A query inside a query is called as nested query.

Subqueries: Subqueries are similar to SELECT chaining. While SELECT chaining combines SELECTs on the same level in a query, however, subqueries allow SELECTs to be embedded inside other queries.

Views: A view is a virtual table, which does not actually store data.

Notes

4.8 Self Assessment

Fill in the blanks:

1. You can combine different query blocks into a single query expression with the ..... operator.
2. A subquery is always a single query block ..... that can contain other subqueries but cannot contain a UNION.
3. A view can be dropped using a ..... statement.
4. ..... are useful for security of data.
5. ..... are used to query data from two or more tables, based on a relationship between certain columns in these tables.
6. An ..... cannot be nested inside a Left Join or Right Join.
7. ..... combines two tables based on their common columns.
8. Subqueries are similar to SELECT .....
9. The ..... clause should follow the GROUP BY clause.
10. A query inside a query is called as ..... query.

4.9 Review Questions

1. Create a table with five columns and apply subqueries concept on that table.
2. You already create a table in question no. 1, then apply SQL IN concept on that table with suitable example.
3. What do you mean by ORDER BY clause? Apply this clause with a suitable example.
4. Explain UNION clause in detail with the help of example.
5. How will you drop views? Explain with suitable example.
6. What is the purpose of joins in SQL? Explain inner join.
7. Create a table name student with five important column and apply order by and having clause on it.
8. What do you mean by complex queries?
9. How will you drop a view? Explain with example.
10. Distinguish between left and right joins.

Answers: Self Assessment

1.	union	2.	(Select)
3.	drop	4.	Views
5.	SQL joins	6.	inner join
7.	Natural join	8.	chaining
9.	having	10.	nested

4.10 Further Readings

Notes

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 5: Integrity Constraints

CONTENTS

Objectives

Introduction

5.1 Integrity Constraints

5.2 Authorization

5.3 DCL Commands

5.4 Embedded SQL

5.5 Dynamic SQL

5.6 Summary

5.7 Keywords

5.8 Self Assessment

5.9 Review Questions

5.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe integrity constraints
- Know authorization
- Explain DCL commands
- Describe embedded and dynamic SQL

Introduction

Sometimes a class type really represents a collection of individual components. Although this pattern can be modeled by an ordinary association, its meaning becomes much clearer if we use the notation for an aggregation. Database objects map capability between Java objects and a relational database (RDBMS) in a standard an highly extensible way, so that the objects themselves can be used in your application, removing the need to embed SQL code directly into your Java applications.

5.1 Integrity Constraints

Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

Notes

Besides the cell name, cell length and cell data type, there are other parameters i.e. other data constraints that can be passed to the DBA at cell creation time.

These data constraints will be connected to a cell by the DBA as flags. Whenever a user attempts to load a cell with data, the DBA will check the data being loaded into the cell against the data constraints defined at the time the cell was created. If the data being loaded fails any of the data constraint checks fired by the DBA, the DBA will not load the data into the cell, reject the entered record, and will flash an error message to the user.

These constraints are given a constraint name and the DBA stores the constraints with its name and instructions internally along with the cell itself.

The constraint can either be placed at the column level or at the table level.

Column Level Constraints: If the constraints are defined along with the column definition, it is called as a column level constraint. Column level constraint can be applied to anyone column at a time i.e. they are local to a specific column. If the constraint spans across multiple columns, the user will have to use table level constraints.

Table Level Constraints: If the data constraint attached to a specific cell in a table references the contents of another cell in the table then the user will have to use table level constraints. Table level constraints are stored as a part of the global table definition.

NULL Value Concepts

While creating tables, if a row lacks a data value for a particular column, that value is said to be null. Columns of any data types may contain null values unless the column was defined as not null when the table was created.

Principles of NULL Values

1. Setting a null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
2. A null value is not equivalent to a value of zero.
3. A null value will evaluate to null in any expression. e.g. null multiplied by 10 is null.
4. When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into that column.

Example: Create table client master with a not null constraint on columns client no, Name, address, address2.
NOT NULL as a column constraint:

```
CREATE TABLE client master
(client_no varchar2(6) NOT NULL,
name varchar2(20) NOT NULL,
address 1 varchar2(30) NOT NULL,
address2 varchar2(30) NOT NULL,
city varchar2(15), state varchar2(15), pin code number( 6),
remarks varchar2(60), bal_due number (10,2));
```

Notes

Primary Key Concepts

A primary key is one or more columns in a table used to uniquely identify each row in the table. Primary key values must not be null and must be unique across the column.

A multicolumn primary key is called a composite primary key. The only function that a primary key performs is to uniquely identify a row and thus if one column is used it is just as good as if multiple columns are used. Multiple columns i.e. (composite keys) are used only when the system designed requires a primary key that cannot be contained in a single column.

Example: **Primary Key as a Column Constraint:**

Create client_master where client_no is the primary key.

```
CREATE TABLE client master
(client_no varchar2(6) PRIMARY KEY,
name varchar2(20), add}-essl varchar2(30), address2 varchar2(30),
city varcbar2(15), state varchar2(15), pincode number(6),
remarks varchar2(60), bal_due number (10,2));
```

Primary Key as a Table Constraint:

Create a sales order details table where

Column Name	Data Type	Size	Attributes
S_order_no	varchar2	6	Primary Key
product_no	varchar2	6	Primary Key
qty_ordered	Number	8	
qty-disp	Number	8	
product_rate	Number	8,2	

```
CREATE TABLE sales order details

(s_order_no varchar2(6), product_no varchar2(6),
qty _ordered number(8), qty - disp number(8),
product_rate number(8,2),
PRIMARY KEY (s_order_no, product_no));
```

Unique Key Concepts

A unique key is similar to a primary key, except that the purpose of a unique key is to ensure that information in the column for each record is unique, as with telephone or driver's license numbers. A table may have many unique keys.

Example: Create Table client_master with unique constraint on column client_no

UNIQUE as a Column Constraint:

```
CREATE TABLE client master
(client_no varchar2( 6) CONSTRAINT cnmn - ukey UNIQUE,
name varchar2(20), address 1 varchar2(30), address2 varchar2(30),
city varchar2(15), state varchar2(l5), pincode number(6),
remarks varch_2(60), bal_due number(lO,2), partpaY311 char(l));
```

Notes

UNIQUE as a Table Constraint:

```
CREATE TABLE client master
(client_no varchar2(6), name varchar2(20),
addressl varchar2(30), address2 varchar2(30),
city varchar2(15), state varchar2(15), pincode number(6),
remarks varchar2(60), bal_due number(lO,2),
CONSTRAINT cnmn_ukey UNIQUE (client_no));
```

Default Value Concepts

At the time of cell creation a 'default value' can be assigned to it. When the user is loading a 'record' with values and leaves this cell empty, the DBA will automatically load this cell with the default value specified – The data type of the default value should match the data type of the column. You can use the default clause to specify any default value you want.

```
CREATE TABLE sales_order
(s_order_no varchar2(6) PRIMARY KEY,
s _order_date date, client_no varchar2(6),
dely _Addr varchar2(25), salesman _no varchar2(6),
dely_type char(l) DEFAULT 'F',
billed_yn char( l), dely_date date,
order_status varchar2(1 0))
```

Create sales_order table where:

Column Name	Data Type	Size	Attribute
S_order no	varchar2	6	Primary key
S_order date	Date		
Client_no	varchar2	6	
Dely_Addr	varchar2	25	
Salesman_no	varchar2	6	
Dely_type	char	I	Delivery: part (P)/Full (F) Default 'F'
Billed_yn	Char	I	
Dely_date	Date		
Order_status	varchar2	10	

Foreign Key Concepts

Foreign keys represent relationships between tables. A foreign key is a Column (or a group of columns) whose values-are derived from the primary key of the same or some other table.

The existence of a foreign key implies that the table with the foreign key is related to the – primary key table from which the foreign-key is derived. A foreign key must have a corresponding primary key value in the primary key table to have a meaning.

Example: The s_order_no column is the primary key of table sales_order. In table sales_order__details, s _order_no is a foreign key that references the s_order_.no values in table sales order.

Notes

The Foreign Key References constraint are as follows:

1. Rejects an INSERT or UPDATE of a value, if a corresponding value does not currently exist in the primary key table
2. Rejects a DEL_TE, if it would invalidate a REFERENCES constrain
3. Must reference a PRIMARY KEY or UNIQUE column(s) in primary key table
4. Will reference the PRIMARY KEY of the primary key table if no column or group of columns is specified in the constraint
5. Must reference a table, not a view or cluster;
6. Requires that you own the primary key table, have REFERENCE privilege on it, or have column-level REFERENCE privilege on the referenced colwnns in the primary key table;
7. Doesn't restrict how other constraints may reference the same tables;
8. Requires that the FOREIGN KEY column(s) and the CONSTRAINT column(s) have matching data types;
9. May reference the same table named in the CREATE TABLE statement;
10. Must not reference the same column more than once (in a single constraint).

Example: Create table sales_order _details with primary key as s_order_no and product_no and foreign key as s_order_no referencing column s_order_no in the sales order table.

FOREIGN KEY as a Column Constraint:

```
CREATE TABLE sales order details
( s_order_no varchar2(6) REFERENCES sales_order,
product_no varchar2(6),
qty _ordered number(8), qty - disp number(8), product_rate number(8,2),
PRIMARY KEY (s_order_no, product_no));
```

FOREIGN KEY as a Table Constraint:

```
CREATE TABLE sales order details
( s _order_no varchar2( 6),
product_no varchar2(6),
qty_ordered number(8), qty_disp number(8),
product_rate number(8,2),
PRIMARY KEY (s_order_no, product_no),
FOREIGN KEY (s_order_no) REFERENCES sales_order);
```

CHECK Integrity Constraints

Use the CHECK constraint when you need to enforce integrity rules that can be evaluated based on a logical expression. Never use CHECK constraints if the constraint can be defined using the not null, primary key or foreign key constraint.

Following are a few examples of appropriate CHECK constraints:

1. a CHECK constraint on the client no column of the client master so that no client no value starts with 'C'.

Notes

2. a CHECK constant on name column of the client master so that the name is entered in upper case.
3. a CHECK constraint on the city column of the client_master so that only the cities "BOMBAY", "NEW DELHI ", "MAPRAS" and "CALCUTTA" are allowed.

```
CREATE TABLE client master
(client_no varchar2(6) CONSTRAINT ck_clientno
CHECK ( client_no like 'C%'),
name varchar2(20) CONSTRAINT ck_cname
CHECK (name = upper(name»,
address I varchar2(30), address2 varchar2(30),
city varchar2( 15) CONSTRAINT ck _city
CHECK (city IN CNEWDELHI', 'BOMBAY', 'CALCUTTA', 'MADRAS')),
state varchar2(l5), pin code number(6),
remarks varchar2(60), bal- due number(10,2));
```

When using CHECK constraints, consider the ANSI I ISO standard which states that a CHECK constraint is violated only if the condition evaluates to False, True and unknown values do not violate a check condition. Therefore, make sure that a CHECK constraint that you define actually enforces the rule you need to enforce.

Example: Consider the following CHECK constraint for emp table:

```
CHECK ( sal > 0 or comm >= 0 )
```

At first glance, this rule may be interpreted as "do not allow a row in emp table unless the employee's salary is greater than 0 or the employee's commission is greater than or equal to "0".

Notes If a row is inserted with a null salary and a negative commission, the row does not violate the CHECK constraint because the entire check condition is evaluated as unknown. In this particular case, you can account for such violations by placing not null integrity constraint on both the sal and comm columns.

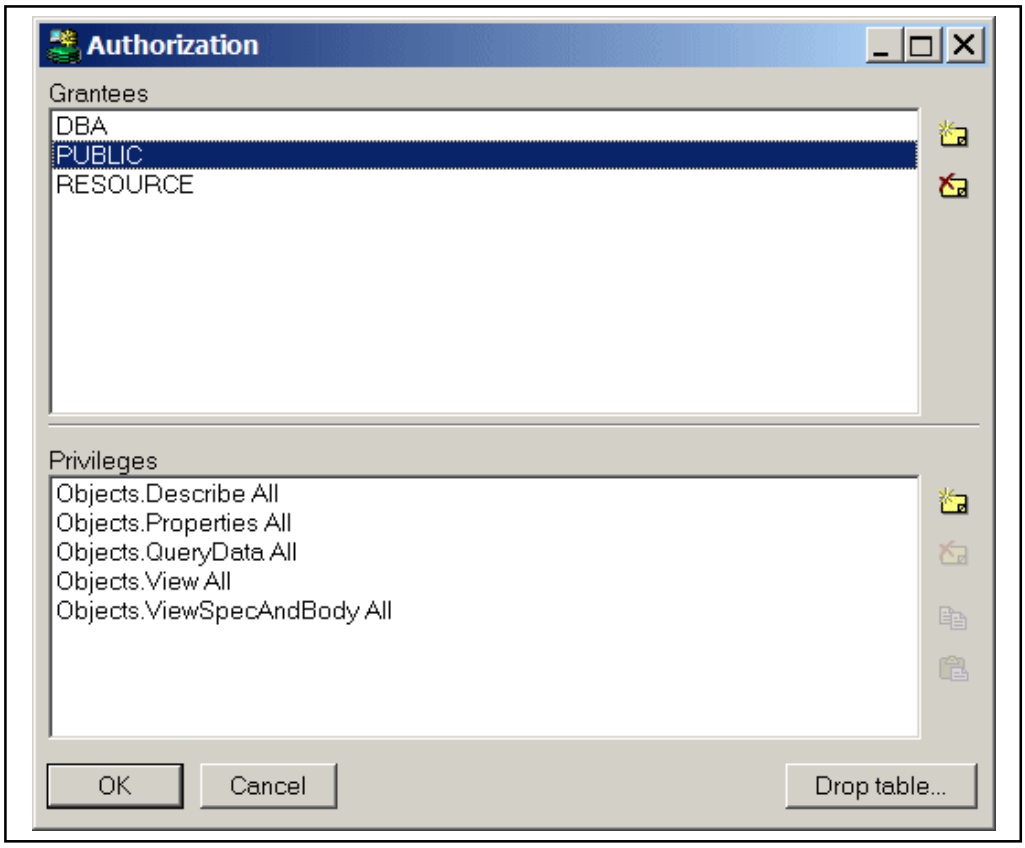
Task Explain primary key constraints.

5.2 Authorization

After installing PL/SQL Developer all users can use all PL/SQL Developer functionality, within the limits of the system privileges and object privileges that are granted to the Oracle user that is connected to the database.

Example: If the Oracle user does not have the create user system privilege, the PL/SQL Developer user can start the New user function in PL/SQL Developer, but will eventually get an "ORA-01031, insufficient privileges" error message from Oracle.

Notes



You can explicitly authorize all relevant PL/SQL Developer functionality to specific Oracle users and roles. In a development database you will allow all developers all functionality, in a test database you will typically not allow a user to alter objects, and in a production database you would typically disable all functions for most users that could alter the database or take up too much resources and would affect performance.

By granting PL/SQL Developer privileges to roles you can customize authorization for specific groups of people. You can make use of existing roles that implicitly map to a user group (such as DBA and RESOURCE) or you can create roles specifically for PL/SQL Developer user groups. To prevent all PL/SQL Developer users from accessing a specific database, you can simply not grant the System.Logon privilege to any user or role.

5.3 DCL Commands

Data control language (DCL) refers to the subgroup of SQL statements that controls access to database objects and data.

This sub-category of SQL statements is of particular interest to database administrators managing database user groups, and user IDs. DCL statements are used at the database level to control who can execute SQL statements, restrict what SQL statements users can execute, and to assign authorities to users so that they can execute a pre-defined set of SQL statements. Although user access to the database can also be administered at the operating system level or by using security plugins, DCL statements provide the most direct method for granting and revoking user privileges and authorities. Database administrators grant or revoke user privileges when a new user is added, a user is removed, a user's privileges are to be restricted or relaxed due to a change in security policy, or when special situations warrant a user being granted new privileges to execute a SQL statement.

Notes

In general DCL statements begin with one of the following keywords: GRANT, or REVOKE. There are different sub-categorizations of DCL statements based on the type of action to be granted or revoked. For example there are DCL statements related to statements, packages, and utilities. These statements generally all contain clauses referring to the name of a database privilege or authority, the name of the database object associated with the privilege if there is one, and the name of the user that will be modified. DCL statements can also be used to delegate the authority to grant and revoke specific privileges to other users.

DCL statements can be executed from a variety of interactive and application interfaces although they are most commonly executed in scripts or from DB2(R) tools that support SQL statement execution.

For security reasons it is important that privilege management be used to minimize the number of users that can modify the privileges in order to prevent data from being data accidentally or maliciously modified, retrieved, or lost. It follows therefore that to retrieve, insert, update, or delete data in a database users require particular authorities which should generally be restricted to the smallest sub-set of database users possible.

For the specific authorities required to execute a DCL statement, refer to the specific SQL Reference syntax topic for that statement.

GRANT

This command is used for gives access privileges to users for database. The syntax is:

GRANT dba to username;

REVOKE

This command is used for withdraws access privileges to users for database. The syntax is:

REVOKE permissions on tablename from username;

5.4 Embedded SQL

The SQL standard defines embeddings of SQL in a variety of programming languages, such as Pascal, PL/I, Fortran, C, and Cobol. A language in which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language constitute embedded SQL.

Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database. This embedded form of SQL extends the programmer's ability to manipulate the database even further. In embedded SQL, all query processing is performed by the database system. The result of the query is then made available to the program one tuple (record) at a time.

An embedded SQL program must be processed by a special preprocessor prior to compilation. Embedded SQL requests are replaced with host-language declarations and procedure calls that allow run-time execution of the database accesses. Then the resulting program is compiled by the host-language compiler. To identify embedded SQL requests to the preprocessor, we use the EXEC SQL statement; it has the form:

EXEC SQL <embedded SQL statement > END-EXEC

The exact syntax for embedded SQL requests depends on the language in which SQL is embedded. For instance, a semi-colon is used instead of END-EXEC when SQL is embedded in C or Pascal.

Notes

Declaring Variables and Exceptions

We place the statement SQL INCLUDE in the program to identify the place where the preprocessor should insert the special variables used for communication between the program and the database system. Variables of the host language can be used within embedded SQL statements, but they must be preceded by a colon (:) to distinguish them from SQL variables.

To write a relational query, we use the declare cursor statement. The result of the query is not yet computed. Rather, the program must use the open arid fetch commands (discussed later in this section) to obtain the result tuples.

Consider the banking schema. Assume that we have a host-language variable amount, and that we wish to find the names and cities of residence of customers who have more than amount dollars in any account. We can write this query as follows:

EXEC SQL

declare c cursor for
select customer-name, customer-city
.from deposit, customer
where deposit.customer-name = customer.customer-name and
deposit.balance > : amount

END-EXEC

The variable c in the preceding expression is called a cursor for the query. We use this variable to identify the query in the open statement, which causes the query to be evaluated, and in the fetch statement, which causes the values of one tuple to be placed in host-language variables.

The open statement for our sample query is as follows:

EXEC SQL open c END-EXCE

This statement causes the database system to execute the query and to save the results within a temporary relation. If the SQL query results in an error, the database system stores an error diagnostic in the SQL communication-area (SQLCA) variables, whose declarations are inserted by the SQL INCLUDE statement.

A series of fetch statements is executed to make tuples of the result available to the program. The fetch statement requires one host-language variable for each attribute of the result relation. For our example query, we need one variable to hold the customer-name value and another to hold the customer-city value. Suppose that those variables are en and cc, respectively. A tuple of the result relation is obtained by the statement:

EXEC SQL fetch c into : en, : cc END-EXEC

The program can then manipulate the variables en and cc using the feature of the host programming language.

The close statement must be used to tell the database system to delete the temporary relation that held the result of the query. For our example, this statement takes the form

EXEC SQL close c END-EXEC

Notes

Embedded SQL expressions for database modification (update, insert, and delete) don't return a result. Thus, they are somewhat simpler to express. A database-modification request takes the form

EXEC SQL < any valid update, insert, or delete> END-EXEC

Host-language variables, preceded by a colon, may appear in the SQL database modification expression. If an error condition arises in the execution of the statement, a diagnostic is set in the SQLCA.

Task Discuss the purpose of GRANT command in SQL

5.5 Dynamic SQL

The dynamic SQL component of SQL-92 allows programs to construct and submit SQL queries at run time. In contrast, embedded SQL statements must be completely present at compile time, and are compiled by the embedded SQL preprocessor. Using dynamic SQL, programs can create SQL queries as strings at run time (perhaps based on input from the user), and can either have them executed immediately, or have them prepared for subsequent use. Preparing a dynamic SQL statement compiles it, and subsequent uses of the prepared statement use the compiled version. The following is an example of the use of dynamic SQL from within a C program.

```
char * sqlprog = "update account set balance = balance * 1.05
where account-number = ?"
EXEC SQL prepare dynprog from : sqlprog;
char account [10] = "A-101";
EXEC SQL execute dynprog using : account;
```

The dynamic SQL program contains a ?, which is a place holder for a value that is provided when the SQL program is executed.

These are two main commands, PREPARE and EXECUTE, which we illustrate through a simple example:

```
char c_sqlstring[] = {"DELETE FROM Sailors WHERE rating>5"};
EXEC SQL PREPARE readytogo FROM : c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

The first statement declares the C variable c_sqlstring and initializes its value to the string representation of an SQL command. These second statement results in this string being parsed and compiled as an SQL command, with the resulting executable bound to the SQL variable readytogo. (Since readytogo is an SQL variable, just like a cursor name, it is not prefixed by a colon.) The third statement executes the command.

5.6 Summary

- Database objects allow fields to be defined that are calculated by any specified method (and not stored in the database).
- They allow referential integrity (the relationships between objects that must be maintained (e.g. invoice master/detail) to be defined in a database-independent way, they allow a set

Notes

of valid values to be defined for a field and verify field validation against that list automatically, they allow serial number fields to be automatically assigned correct values, and much more.

5.7 Keywords

Column Level Constraints: If the constraints are defined along with the column definition, it is called as a column level constraint.

Foreign Key: Foreign keys represent relationships between tables.

Primary Key: A primary key is one or more columns in a table used to uniquely identify each row in the table.

Table Level Constraints: If the data constraint attached to a specific cell in a table references the contents of another cell in the table then the user will have to use table level constraints.

5.8 Self Assessment

Fill in the blanks:

1. ..... ensure that changes made to the database by authorized users do not result in a loss of data consistency.
2. Setting a ..... is appropriate when the actual value is unknown.
3. A multicolumn primary key is called a ..... primary key.
4. The ..... of the default value should match the data type of the column.
5. By granting ..... privileges to roles you can customize authorization for specific groups of people.
6. ..... refers to the subgroup of SQL statements that controls access to database objects and data.
7. ..... command is used for gives access privileges to users for database.
8. The exact syntax for ..... requests depends on the language in which SQL is embedded.
9. A series of ..... is executed to make tuples of the result available to the program.
10. Embedded SQL expressions for database modification (update, insert, and delete) ..... a result.

5.9 Review Questions

1. Distinguish between primary key constraints and foreign key constraints.
2. What do you mean by NOT NULL concept in table creation? Explain
3. Explain authorization concept of database in detail.
4. If I want to search name "Amit Kumar" present in the table which one created by you. What is the query to display "Amit Kumar"? Explain.
5. Create table sales_order _details with primary key as s_order_no and product_no and foreign key as s_order_no referencing column s_order_no in the sales order table.

Notes

6. "DCL statements begin with one of the following keywords: GRANT, or REVOKE". Discuss the reason behind this.
7. "Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database." Explain
8. "REVOKE permissions on tablename from username" discuss the use of this command.
9. CREATE TABLE sales_order

   (s_order_no varchar2(6) PRIMARY KEY,
   s _order_date date, client_no varchar2(6),
   dely _Addr varchar2(25), salesman _no varchar2(6),
   dely_type char(l) DEFAULT 'F',
   billed_yn char(l), dely_date date,
   order_status varchar2(1 0))

   and also enter atleast two values in each column
10. What do you mean by default value?

Answers: Self Assessment

1.	Integrity constraints	2.	null value
3.	composite	4.	data type
5.	PL/SQL Developer	6.	Data control language (DCL)
7.	GRANT	8.	embedded SQL
9.	fetch statements	10.	don't return

5.10 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems,* 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 6: Relational Language and Database Design

CONTENTS

Objectives

Introduction

6.1 Relational Calculus

6.2 Tuple Relational Calculus

6.2.1 Syntax of TRC Queries

6.2.2 Semantics of TRC Queries

6.3 Domain Relational Calculus

6.4 Query-by-Example

6.5 Overview of Design Process

6.6 Entity-Relationship Model

6.7 Constraints

6.8 E-R Diagrams

6.9 ER Design Issues

6.9.1 Use of Entity Sets versus Attributes

6.9.2 Use of Entity Sets versus Relationship Sets

6.9.3 Binary versus n-ary Relationship Sets

6.9.4 Aggregation versus Ternary Relationships

6.10 Weak Entity Sets

6.11 Extended ER Features

6.11.1 Class Hierarchies

6.11.2 Aggregation

6.12 Summary

6.13 Keywords

6.14 Self Assessment

6.15 Review Questions

6.16 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Discuss relational calculus, tuple and domain relational calculus
- Explain entity relationship model
- Know the constraints
- Describe E-R diagrams and extended ER features

Introduction

Relational tables can be considered as sets. The rows of the tables can be considered as elements of the set. Operations that can be performed on sets can be done on relational tables. Relational Data operators are used to retrieve the required data from relational tables. Data is retrieved using queries and the queries are formulated using various data operators.

To be able to use a database to store data, it should be designed in an efficient manner. The first step in designing a database is data modeling. Data modeling enables a database designer to create a model that represents the way in which information is likely to be organized in the database. There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. This unit mainly focuses on data modeling using the Entity-Relationship approach. The basic techniques described here are applicable to the development of relational database applications.

6.1 Relational Calculus

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or declarative, in that it allows us to describe the set of answers without being explicit about how they should be computed. Relational calculus has had a bid influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).

The variant of the calculus that we present in detail is called the tuple relational calculus (TRC), variables in TRC take on tuples as values. In another variant, called the domain relational calculus (DRC), the variables range over field values. TRC has had more of an influence on SQL, while DRC has strongly influenced QBE.

6.2 Tuple Relational Calculus

A **tuple variable** is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of _elds. A tuple relational calculus query has the form {T | p (T)}, where T is a tuple variable and p(T) denotes a formula that describes T; we will shortly define formulas and queries rigorously. The result of this query is the set of all tuples t for which the formula p(T) evaluates to true with T = t. The language for writing formulas p(T) is thus at the heart of TRC and is essentially a simple subset of first-order logic. As a simple example, consider the following query.

Example: Find all sailors with a rating above 7.

$$\{ S \mid S \in \text{Sailors} \wedge S.\text{rating} > 7 \}$$

Notes

When this query is evaluated on an instance of the Sailors relation, the tuple variable S is instantiated successively with each tuple, and the test S.rating>7 is applied. The answer contains those instances of S that pass this test. On instance S3 of Sailors, the answer contains Sailors tuples with sid 31, 32, 58, 71, and 74.

6.2.1 Syntax of TRC Queries

We now define these concepts formally, beginning with the notion of a formula. Let Rel be a relation name, R and S be tuple variables, a an attribute of R, and b an attribute of S. Let op denote an operator in the set {<, >, =, $\leq$, $\geq$, $\neq$}. An atomic formula is one of the following:

1. R $\in$ Rel
2. R. a op S.b
3. R.a op constant, or constant op R.a

A formula is recursively defined to be one of the following, where p and q are themselves formula, and p (R) denotes a formula in which the variable R appears:

1. Any atomic formula
2. -p, p $\wedge$ q, p $\vee$ q, or p $\Rightarrow$ q
3. $\exists$ R(p(R), where R is a tuple variable
4. $\forall$ R(p(R), where R is a tupe variable

In the last two clauses above, the quantifiers $\exists$ and $\forall$ are said to bind the variable R. A variable is said to be free in a formula or subformula (a formula contained in a larger formula) if the (sub) formula does not contain an occurrence of a quantifier that binds it.

We observe that every variable in a TRC formula appears in a subformula that is atomic, and every relation schema specifies a domain for each field, this observation ensures that each variable in a TRC formula has a well-defined domain from which values for the variable are drawn. That is, each variable has a well-defined type, in the programming language sense. Informally, an atomic formula R 2 Rel gives R the type of tuples in Rel, and comparisons such as R.a op S.b and R.a op constant induce type restrictions on the field R.a. If a variable R does not appear in an atomic formula of the form R 2 Rel (i.e., it appears only in atomic formulas that are comparisons), we will follow the convention that the type of R is a tuple whose fields include all (and only) fields of R that appear in the formula.

We will not define types of variables formally, but the type of a variable should be clear in most cases, and the important point to note is that comparisons of values having different types should always fail. (In discussions of relational calculus, the simplifying assumption is often made that there is a single domain of constants and that this is the domain associated with each field of each relation.)

A TRC query is defined to be expression of of the form {T | p(T)}, where T is the only free variable in the formula p.

Task Write tuple relational calculus syntax.

Notes

6.2.2 Semantics of TRC Queries

What does a TRC query mean? More precisely, what is the set of answer tuples for a given TRC query? The answer to a TRC query {T | p(T)}, as we noted earlier, is the set of all tuples t for which the formula p(T) evaluates to true with variable T assigned the tuple value t. To complete this definition, we must state which assignments of tuple values to the free variables in a formula make the formula evaluate to true.

A query is evaluated on a given instance of the database. Let each free variable in a formula *F* be bound to a tuple value. For the given assignment of tuples to variables, with respect to the given database instance, *F* evaluates to (or simply 'is') true if one of the following holds:

1. *F* is an atomic formula $R \in Rel$, and *R* is assigned a tuple in the instance of relation *Rel*.
2. F is a comparison R.a op S.b, R.a op constant, or constant op R.a, and the tuples assigned to R and S have field values R.a and S.b that make the comparison true.
3. F is of the form –p and q is not true, or of the form $p \wedge q$, and both p and q are true, or of the form $p \vee q$, and one of them is true, or of the form $p \Rightarrow q$ and q is true whenever p is true.
4. F is the form $\exists$ R(p(R)), and there is some assignment of tuples to the free variables in p(R), including the variable R, that makes the formula p(R) true.
5. F is the form $\forall$ R(p(R)), and there is some assignment of tuples to the free variables in p(R) that makes the formula p(R) true no matter what tuple is assigned to R.

6.3 Domain Relational Calculus

A domain variable is a variable that ranges over the values in the domain of some attribute (e.g., the variable can be assigned an integer if it appears in an attribute whose domain is the set of intergers). A DRC query has the form $\{<x_1, x_2, \ldots, x_n> \mid p(<x_1, x_2, \ldots, x_n>)\}$, where each x_i is either a domain variable or a constant and $p(<x_1, x_2, \ldots, x_n>)\}$ denotes a DRC formula whose only free variables are the variables among the x_i, $1 < i < n$. The result of this query is the set of all tuples $<x_1, x_2, \ldots, x_n>$ for which the formula evaluates to true.

A DRC formula is defined in a manner that is very similar to the definition of a TRC formula. The main difference is that the variables are now domain variables. Let op denote an operator in the set $\{<,>,=, \leq, \geq, \neq\}$ and let X and Y be domain variables.

An atomic formula in DRC is one of the following:

1. $<x_1, x_2, \ldots, x_n> \in$ Rel, where Rel is a relation with n attributes, each x_i, $1 \leq i \leq n$ is either a variable or a constant.
2. X op Y
3. X op constant, or constant op X

A formula is recursively defined to one of the following, where p and q are themselves formulas, and p(X) denotes a formula in which the variable X appears:

1. Any atomic formula
2. -p, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
3. $\exists$ X(p(X), where X is a domain variable
4. $\forall$ X(p(X), where X is a domain variable

Notes

6.4 Query-by-Example

Query-by-Example (QBE) is another language for querying (and, like SQL, for creating and modifying) relational data. It is different from SQL, and from most other database query languages, in having a graphical user interface that allows users to write queries by creating example tables on the screen. A user needs minimal information to get started and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables.

A user writes queries by creating example tables. QBE uses domain variables, as in the DRC, to create example tables. The domain of a variable is determined by the column in which it appears, and variable symbols are prefixed with underscore (_) to distinguish them from constants. Constants, including strings, appear unquoted, in contrast to SQL. The fields that should appear in the answer are specified by using the command P., which stands for print. The fields containing this command are analogous to the target-list in the SELECT clause of an SQL query.

This unit introduce QBE through example queries involving just one relation. To print the names and ages of all sailors, we would create the following example table:

Sailor	sid	sname	rating	age
		P._N		P._A

A variable that appears only once can be omitted; QBE supplies a unique new name internally. Thus the previous query could also be written by omitting the variables N and A, leaving just P. in the sname and age columns. The query corresponds to the following DRC query, obtained from the QBE query by introducing existentially quantified domain variables for each field.

$$\{\langle N, A\rangle \mid \exists I, T(\langle I, N, T, A\rangle \in Sailors)\}$$

A large class of QBE queries can be translated to DRC in a direct manner. (Of course, queries containing features such as aggregate operators cannot be expressed in DRC.) This unit shall present DRC versions of several QBE queries. Although we will not define the translation from QBE to DRC formally, the idea should be clear from the examples; intuitively, there is a term in the DRC query for each row in the QBE query, and the terms are connected using^.

A convenient shorthand notation is that if we want to print all fields in some relation, we can place P. under the name of the relation. This notation is like the SELECT * convention in SQL. It is equivalent to placing a P. in every field:

Sailor	sid	sname	rating	age
P.				

Selections are expressed by placing a constant in some field:

Sailor	sid	sname	rating	age
P.			10	

Placing a constant, say 10, in a column is the same as placing the condition =10. This query is very similar in form to the equivalent DRC query

$$\{\langle I, N, 10, A\rangle \mid \langle I, N, 10, A\rangle \in Sailors\}$$

We can use other comparison operations (<, >,<=,>=, ¬) as well.

Notes

Example: We could say < 10 to retrieve sailors with a rating less than 10 or say ¬10 to retrieve sailors whose rating is not equal to 10. The expression ¬10 in an attribute column is the same as _= 10. As we will see shortly, ¬ under the relation name denotes (a limited form of) ¬∃ in the relational calculus sense.

6.5 Overview of Design Process

Our primary focus is the design of the database. The database design process can be divided into six steps:

Requirements Analysis

The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on the database, and what operations must be performed on the database. In other words, we must find out what the users want from the database. This process involves discussions with user groups, a study of the current operating environment, how it is expected to change an analysis of any available documentation on existing applications and so on.

Conceptual Database Design

The information gathered in the requirement analysis step is used to develop a high-level description of the data to be stored in the database, along with the conditions known to hold this data. The goal is to create a description of the data that matches both—how users and developers think of the data (and the people and processes to be represented in the data). This facilitates discussion among all the people involved in the design process i.e., developers and as well as users who have no technical background. In simple words, the conceptual database design phase is used in drawing ER model.

Logical Database Design

We must implement our database design and convert the conceptual database design into a database schema (a description of data) in the data model (a collection of high-level data description constructs that hide many low-level storage details) of the DBMS. We will consider only relational DBMSs, and therefore, the task in the logical design step is to convert the conceptual database design in the form of E-R Schema (Entity-Relationship Schema) into a relational database schema.

Schema Refinement

The fourth step in database design is to analyze the collection, of relations (tables) in our relational database schema to identify future problems, and to refine (clear) it.

Physical Database Design

This step may simply involve building indexes on some tables and clustering some tables, or it may involve redesign of parts of the database schema obtained from the earlier design steps.

Notes

Application and Security Design

Any software project that involves a DBMS must consider applications that involve processes and identify the entities.

Example: Users, user groups, departments, etc. We must describe the role of each entity in every process. As a security design, for each role, we must identify the parts of the database that must be accessible and the parts of the database that must not be accessible and we must take steps to ensure that these access rules are enforced. In general, our division of the design process into six steps are repeated until the design is satisfactorily known as tuning phase.

6.6 Entity-Relationship Model

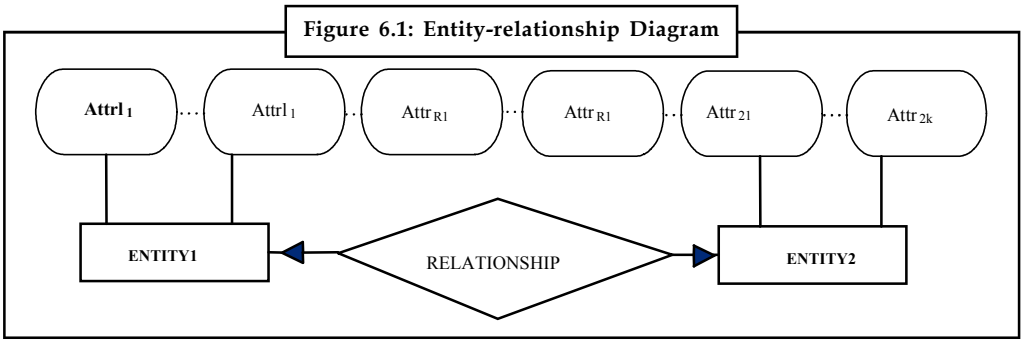
The entity-relationship (ER) data model allows us to describe the data involved in real-world enterprise in terms of objects (entities) and their relationships, and is widely used to develop an initial database design.

The ER model is important for its role in database design. It provides useful concepts that allows to change the detailed and informal description of what users want to a precise and formal description that can be implemented in a DBMS. Within the overall design process, the ER model is used in a phase called Conceptual database design.

Even though the ER model describes the physical database model, it is basically useful in the design and communication of the logical database model,

The overall logical structure of a database can be expressed graphically by an E-R diagram as follows:

Figure 6.1: Entity-relationship Diagram



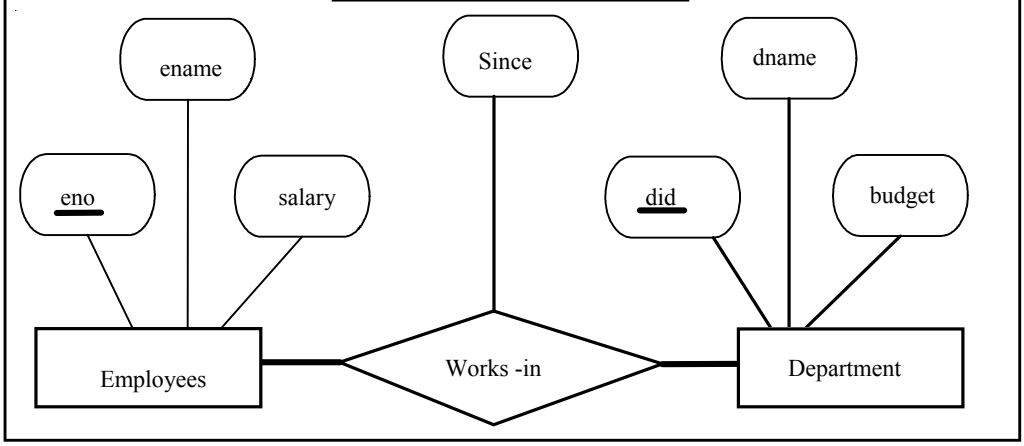
The E-R diagram is built up from the following components:

1. ***Rectangles:*** Which represent entity sets.
2. ***Diamonds:*** Which represent relationships among entity sets, which are connected to the rectangles by lines.
3. ***Ellipses:*** Which represent attributes, and are connected to the entities or relationship by lines.
4. ***Lines:*** Which link attributes to entity sets and entity sets to relationships.

The rectangles, diamonds and ellipses are labelled with the entity, relationships and attributes, respectively, that it represents.

Notes

Figure 6.2: E-R Diagram



Thus, an entity (rectangles) is a "thing" or "object" in the real world that is distinguishable from other objects, example, each employee is an entity and each department is an entity.

A relationship (diamond) is an association (connection) between different entities.

Example: Works-in a department, thus here, works-in is a relationship between two different entities.

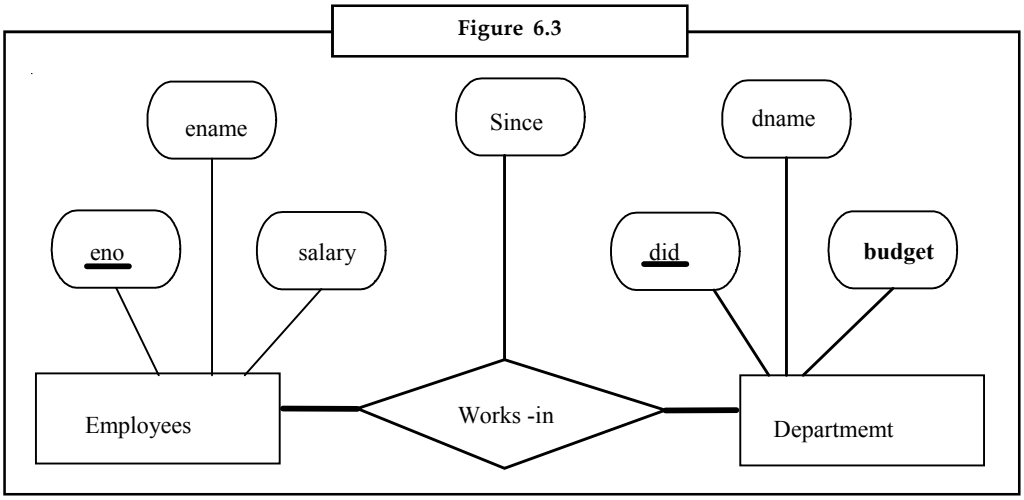
At last, each fields, i.e., employee, department and works-in has their own details, which are represented by attributes (ellipses), i.e., employee details are ename (employee_name), eno (employee_number) and salary, and, department details are dname (department_name), dno (department_number) and budget, and, works-in details are since (the starting date) of an employee in a department.

Task Which tool you used for your DBMS security?

6.7 Constraints

Consider the works-in relationship shown in the following figure.

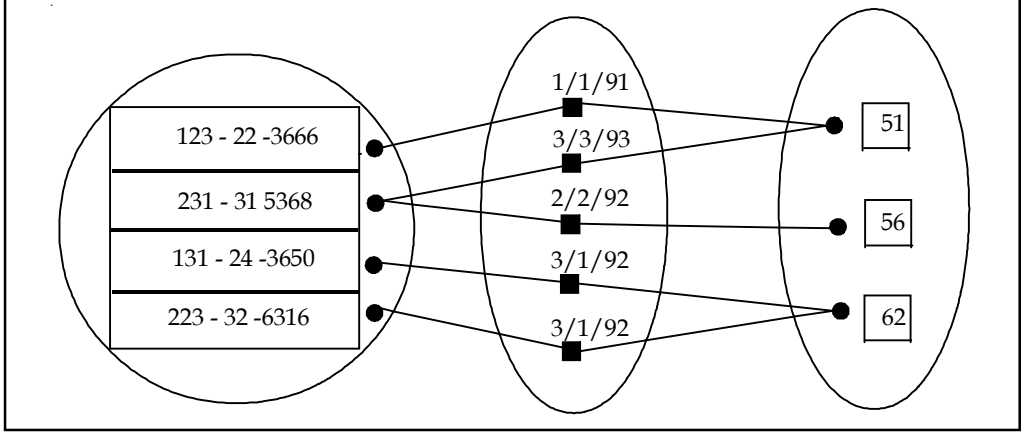
Figure 6.3



Notes

Here, we can note that, one employee can work in many departments, or, one department can have many employees. This can also be illustrated with the following Figure 6.4:

Figure 6.4

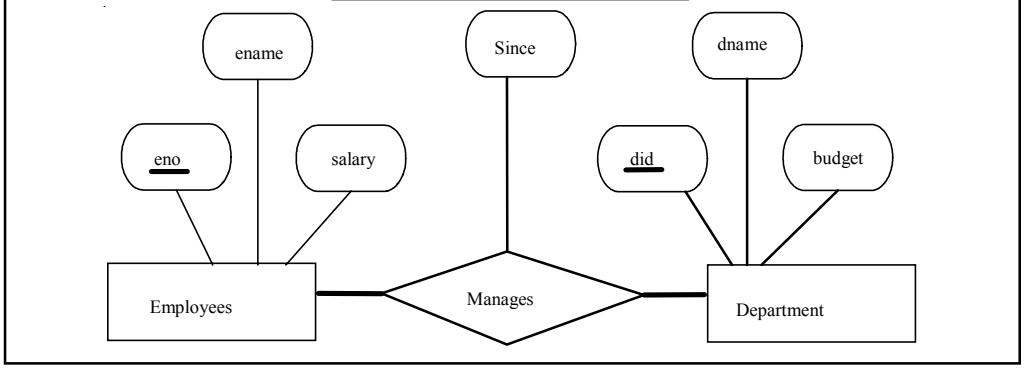


Here, employee 231-31-5368 has worked in Department 51 since, 3/3/93 and in Department 56 since 2/2/92. Thus one department can have many employees.

But, if we want to have only one employee in a department, then it is an example of a Key Constraint.

Example: Consider another relationship Manages between employees and department entities as in the following figure.

Figure 6.5



Here, each department can have only one manager. The restriction that each department can have only one manager is an example is an example of a Key Constraints This restriction is indicated in the above ER diagram by using an arrow from department to manages, such that a department can have only one manager. Therefore, as there is no arrow from employees entity to manages relationship, we can say that, many employees or one employee can manage department. But, as there is an arrow from department entity to manages relationship, we say that, the departments is managed by only one employee or there is only one manager for department(s), indicating many-to-one relationship.

Participation Constraint

The key constraint and its figure in the previous unit on manages relationship tells us that every department has one manager. This requirement is an example of a participation constraints i.e.,

Notes

the participation of the entity Department in the relationship manages is said to be total. If the participation is not total, then the participation is said to be partial. A partial participation example is the participation of the entity set Employees in the relationship Manages, thus every employee cannot be a manager of a department.

6.8 E-R Diagrams

Now we are in a position to write the ER diagram for the Company database which was introduced in the beginning of this unit. The readers are strictly advised to follow the steps shown in this unit to design an ER diagram for any chosen problem.

Step 1: Identify the Strong and Weak Entity Sets

After careful analysis of the problem we come to a conclusion that there are four possible entity sets as shown below:

1.	Employees	Strong Entity Set
2.	Departments	Strong Entity Set
3.	Projects	Strong Entity Set
4.	Dependents	Weak Entity Set

Step 2: Identify the Relevant Attributes

The next step is to get all the attributes that are most applicable for each entity set. Do this work by considering each entity set in mind and also the type of attributes. Next job is to pick the primary key for strong entity sets and partial key for weak entity sets.

Example: Following are the attributes:

1.	Employees	SSN. Name, Addr, DateOfBirth, Sex, Salary
2.	Departments	DNo. DName, DLocation
3.	Projects	PNo. PName, PLocation
4.	Dependents (weak)	DepName, DateOf Birth, Sex, Relationship

The underlined attributes are the primary keys and DepName is the partial key of Dependents. Also, DLocation may be treated as a multivalued attribute.

Step 3: Identify the Relationship Sets

In this step we need to find all the meaningful relationship sets among possible entity sets. This step is very tricky, as redundant relationships may lead to complicated design and in turn a bad implementation.

Example: Let us show below what the possible relationship sets are:

1.	Employees and Departments	WorksFor
2.	Employees and Departments	Manages
3.	Departments and Projects	Controls
4.	Projects and Employees	WorksOn

Notes

5. Dependents and Employees Has
6. Employees and Employees Supervises

Some problems may not have recursive relationship sets but some do have. In fact, our Company database has one such relationship set called Supervises. You can complete this step adding possible descriptive attributes of the relationship sets (Manages has StartDate and WorksOn has Hours).

Step 4: Identify the Cardinality Ratio and Participation Constraints

This step is relatively a simple one. Simply apply the business rules and your common sense. So, we write the structural constraints for our example as follows:

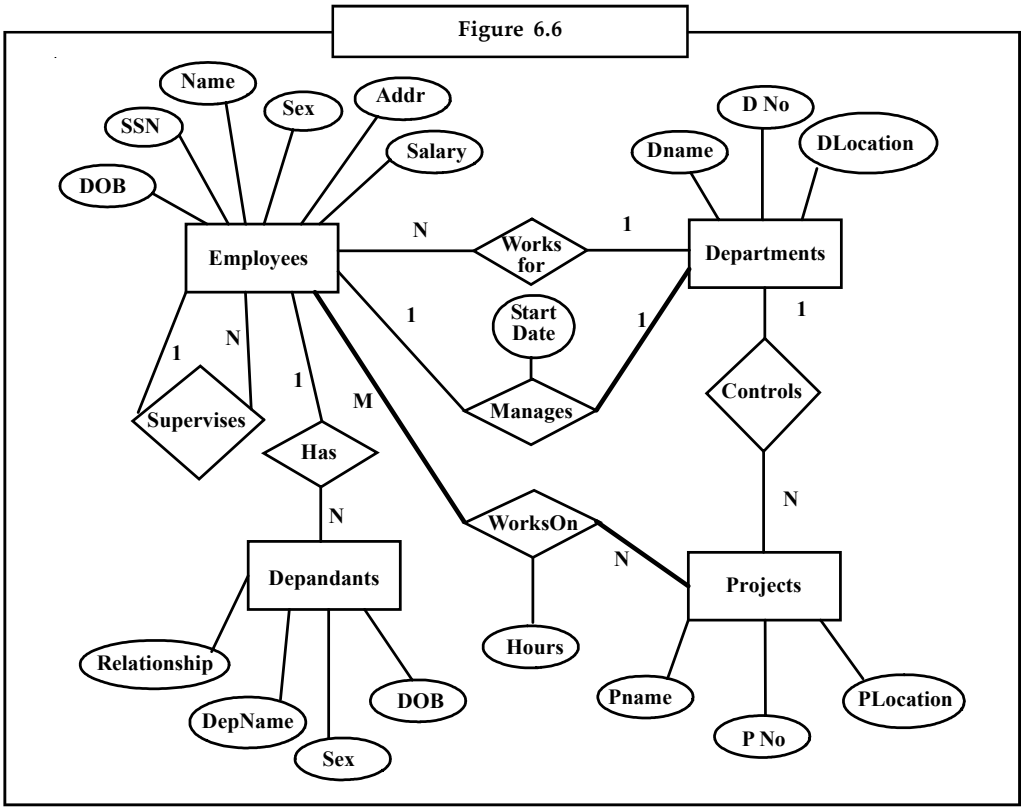
1. WorksFor N: 1 Total on either side
2. Manages 1: 1 Total on Employees and Partial on Departments side
3. Controls 1: N Total on either side
4. WorksOn M: N Total on either side
5. Has 1: M Total on Dependents and Partial on Employees

Step 5: Identify the IS-A and Has-A Relationship Sets

The last step is to look for "is-a" and "has-a" relationships sets for the given problem. As far as the Company database is concerned, there are no generalization and aggregation relationships in the Company database.

The complete single ER diagram by combining all the above five steps is shown in figure 6.6.

Figure 6.6



Notes

Task Discuss Entity-Diagram relationship.

6.9 ER Design Issues

The following are the ER design issues:

1. Use entry sets attributes
2. Use of Entity sets or relationship sets
3. Binary versus entry relationship sets
4. Aggregation versus ternary relationship.

6.9.1 Use of Entity Sets versus Attributes

Consider the entity set employee with attributes employee-name and telephone-number. It can easily be argued that a telephone is an entity in its own right with attributes telephone-number and location (the office where the telephone is located). If we take this point of view, the employee entity set must be redefined as follows:

1. The employee entity set with attribute employee-name.
2. The telephone entity set with attributes telephone-number and location.
3. The relationship set imp-telephone, which denotes the association between employees and the telephones that they have.

The first difference between these two definitions of an employee is that, every employee has precisely one telephone number associated with him. In the second case, however, the definition states that employees may have several telephone numbers (including zero) associated with them.

6.9.2 Use of Entity Sets versus Relationship Sets

In order to clarify whether an object is best expressed by an entity set or a relationship set, assume that a bank loan is modeled as an entity. An alternative is to model a loan as a relationship between customers and branches, with loan-number and amount as descriptive attributes. Each loan is represented by a relationship between a customer and a branch.

If every loan is held by exactly one customer and customer is associated with exactly one branch, the design where a loan is represented as a relationship, maybe satisfactory. However, with this design, we cannot represent conveniently a situation in which several customers hold a loan jointly. We must define a separate relationship for each holder of the joint loan. Then, we must replicate the values for the descriptive attributes loan-number and amount in each such relationship. Each such relationship must have the same value for the descriptive attributes loan-number and amount. Two problems arise as a result of the replication:

1. The data are stored multiple times, wasting storage space; and
2. Updates potentially leaves the date in an inconsistent state, where the values differ in two relationships for attributes that are supposed to have the same value.

Notes

6.9.3 Binary versus n-ary Relationship Sets

It is always possible to replace a no binary (n-ary, for n > 2) relationship set by a number of distinct binary relationship sets. For simplicity, consider the abstract ternary (n = 3) relationship set R, relating entity sets A, B and C. We replace the relationship set R by an entity set E, and create three relationship sets:

1. R_A, relating E and A
2. R_B relating E and B
3. R_C, relating E and C

If the relationship set R had any attributes, these are assigned to entity set E; otherwise, a special identifying attribute is created for E (since every entity set must have at least one attribute to distinguish members of the set). For each relationship (a., b, c.) in the relationship set R, we create a new entity e. in the entity set E. Then, in each of the three new relationship sets, we insert a relationship as follows

1. (ei, ai) in R_A
2. (ei, bi) in R_B
3. (ei, ci) in R_C

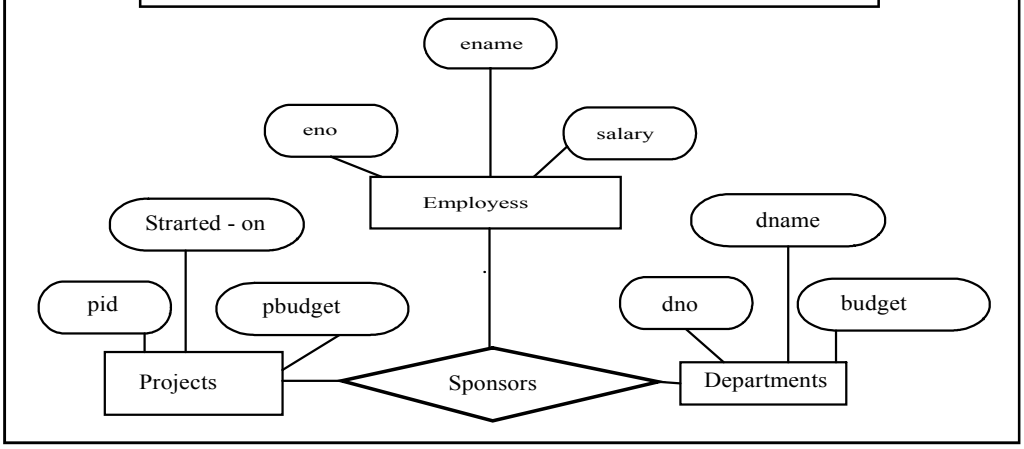
We can generalize this process in a straightforward manner n–ary relationship sets. Thus, conceptually, we can restrict the E-R model to include only binary relationship sets.

6.9.4 Aggregation versus Ternary Relationships

The choice between using aggregation OT a ternary relationship is mainly determined by the existence of relationship that relates a relationship set to an entity set (or second relationship set). The choice may also be guided by certain integrity constraints to we want to express.

Consider the constraint that each sponsorship (of a project by a department) be monitored by at most one employee. We cannot express this constraint in terms of the Sponsors2 relationship set. Also we can express the constraint by drawing an arrow from the aggregated relationship. Sponsors to the relationship Monitors. Thus, the presence of such a constraint serves as another reason for using aggregation rather than a ternary relationship set.

Figure 6.7: Using a Ternary Relationship Instead of Aggregation

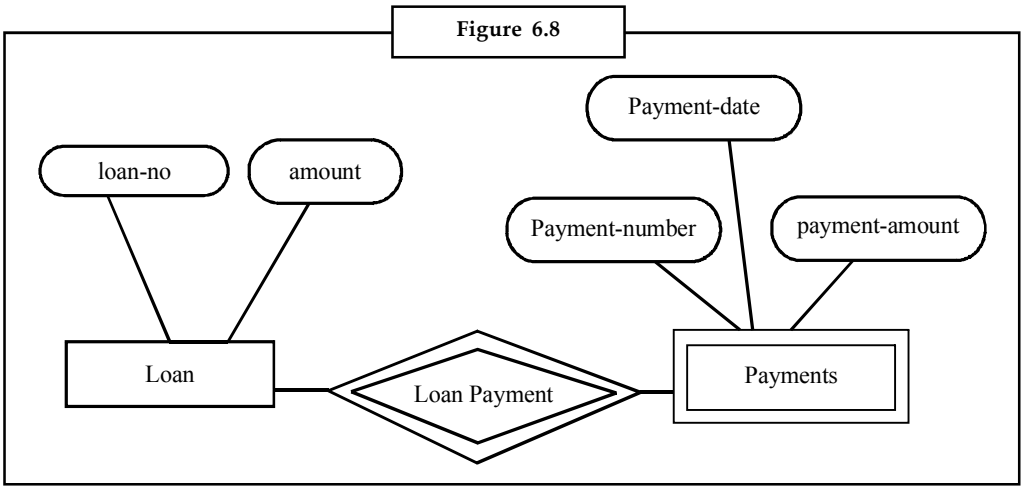


Notes

6.10 Weak Entity Sets

An entity set attributes that does not have a primary key within them, is termed as a weak entity set. As an example, consider the entity set Payment, which has the three attributes: payment - number, payment - date and payment - amount, illustrated with the following E-R diagram:

Figure 6.8



A weak entity set Payment is indicated in E-R diagram by a doubly outlined box, because each payment entity is district, payments for different loans may share the same payment number. Thus, this entity set does not have a primary key, it is a weak entity set.

In the above figure, the weak entity set Payment is dependent on the strong entity set Loan (An entity set that has a primary key is termed as a strong entity set), via the relationship loan-payment identified by a double outlined diamond.

The below figure illustrates the use of double lines to indicate total participation. The participation of the weak entity set Payment in the relationship loan-payment is total; it means that every payment must be related via loan-payment to some account. The arrow from loan-payment to loan indicates that each payment is for a single loan.

The discriminator of a weak entity set is underlined with a dashed, rather than a solid line. The discriminator of the weak entity set Payment is the attribute payment-number, which is formed by the primary key loan-number of the strong entity set Loan on which the weak entity set is dependent. That is, in this case of the weak entity set payment, its temporary primary key (discriminator) is {loan-number, payment-number}, where loan -number identifies the dominant entity of a Payment, and payment - number distinguishes payment entities within the same loan.

An entity set that has a primary key is termed as a strong entity set.

Example: The entity set loan in the above figure is a strong entity set with its primary key loan-number.

6.11 Extended ER Features

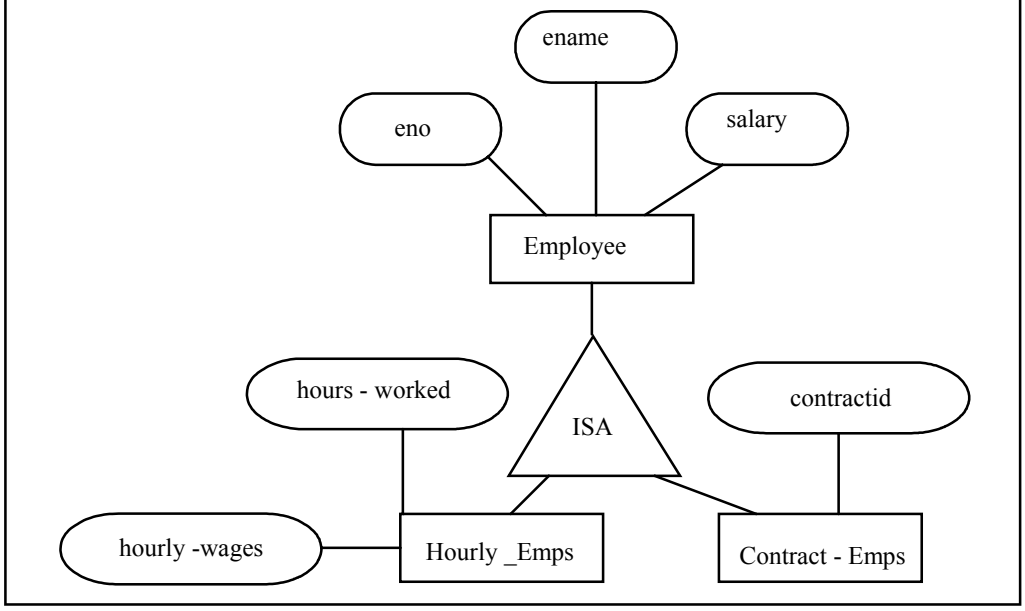
6.11.1 Class Hierarchies

To classify the entities in an entity set into subclass entity is known as class hierarchies. Example, we might want to classify Employees entity set into subclass entities Hourly-Emps entity set

Notes

and Contract-Emps entity set to distinguish the basis on which they are paid. Then the class hierarchy is illustrated as follows:

Figure 6.9: Class Hierarchy



This class hierarchy illustrates the inheritance concept. Where, the subclass attributes ISA (read as : is a) super class attributes; indicating the "is a" relationship (inheritance concept).Therefore, the attributes defined for a Hourly-Emps entity set are the attributes of Hourly-Emps plus attributes of Employees (because subclass can have superclass properties). Likewise the attributes defined for a Contract-Emps entity set are the attributes of Contract-Emps plus attributes of Employees.

Class Hierarchy based on Sub-super Set

1. ***Specialization:*** Specialization is the process of identifying subsets (subclasses) of an entity set (superclass) that share some special distinguishable characteristic. Here, the superclass (Employee) is defined first, then the subclasses (Hourly-Emps, Contract-Emps, etc.) are defined next.

   In short, Employees is specialized into subclasses.

2. ***Generalization:*** Generalization is the process of identifying (defining) some generalized (common) characteristics of a collection of (two or more) entity sets and creating a new entity set that contains (possesses) these common characteristics. Here, the subclasses (Hourly-Emps, Contract-Emps, etc.) are defined first, then the Superclass (Employee) is defined, next.

   In shortly, Hourly-Emps and Contract-Emps are generalized by Employees.

Class Hierarchy based on Constraints

1. ***Overlap constraints:*** Overlap constraints determine whether two subclasses are allowed to contain the same entity.

Notes

Example: Can Akbar be both an Hourly-Emps entity and a Contract-Emps entity?

The answer is, No.

Other example, can Akbar be both a Contract-Emps entity and a Senior-Emps entity (among them)?

The answer is, Yes. Thus, this is a specialisation hierarchy property. We denote this by writing "Contract-Emps OVERLAPS Senior-Emps".

2. ***Covering Constraints:*** Covering constraints determine whether the entities in the subclasses collectively include all entities in the superclass.

Example: Should every Employee be a Hourly-Emps or .Contract-Emps?

The Answer is, No. He can be a Daily-Emps.

Other example, should every Motor-vehicle (superclass) be a Bike (subclass) or a Car (subclass)?

The Answer is YES.

Thus generalization hierarchies property is that every instance of a superclass is an instance of a subclass.

We denote this by writing " Bikes and Cars COVER Motor-vehicles".

6.11.2 Aggregation

Aggregation allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship sets. That is, a relationship set in an association between entity sets. Sometimes we have to model a relationship between a collection of entities and relationships.

Example: Suppose that we have an entity set called Project and that each Project entity is sponsored by one or more departments. Thus, the sponsors relationship set captures this information but, a department that sponsors a project, might assign employees to monitor the sponsorship. Therefore, Monitors should be a relationship set that associates a sponsors relationship (rather than a Project or Department entity) with an Employees entity. However, again we have to define relationships to associate two or more entities.

Use of Aggregation

We use an aggregation, when we need to express a relationship among relationships. Thus, there are really two distinct relationships, Sponsors and Monitors, each with its own attributes.

Example: The Monitors relationship has an attribute until that records the ending date until when the employee is appointed as the sponsorship monitoring. Compare, this attribute with the attribute since of Sponsors, which is the starting date when the sponsorship took effect.

Task

Specialisation and generalisation are two important concepts of EER. What is your opinion?

Notes

To enable the readers for a better understanding of ER diagram we shall present few examples in this study. The reader should go through these examples carefully for a better and easier way of writing ER diagrams.

Case Study

Insurance Policy Management System

Consider a popular and common problem in the today's world: Insurance Policy Management System. The Software Requirements Specifications (SRS) for this problem can be stated as follows:

1. The Insurance Company has many branches with branchid, branch name, address or location, phone numbers, fax, etc.
2. In every branch there are different types of staff working. For example, there is a branch manager, field officers, development personnel, secretarial assistants, etc. It is necessary to keep track of staffed, staff name, address, position, salary, date of birth, etc.
3. Apart from the regular employees there are part-time staff called insurance agents who work on commission basis.
4. The Insurance Company must store the details of policy holders: policy holder's name, policy number, address, tenure, maturity amount, premium amount, etc. It is mandatory to mention the nominees in every policy.

With our knowledge of ER diagram studied so far, the various entity sets and the attributes can be identified as follows:

Step-1 and Step-2: Entity Sets and Attributes

1.	Branches	BranchID, BranchName, Addr, Phone
2.	Staff	StaffID, StaffName, Addr
3.	Customers	PolicyNo, Name, Addr, Age, Phone
4.	Policy (weak)	PolicyName, Tenure, MatAmount, PremAmount, Bonus

Sub Entity Sets of Staff

1.	Managers	Exp, Qualification
2.	DevOfficers	Qualification
3.	SectAssts	TypingSpeed
4.	FullTime	Salary, Position
5.	Agents	Type, Comm

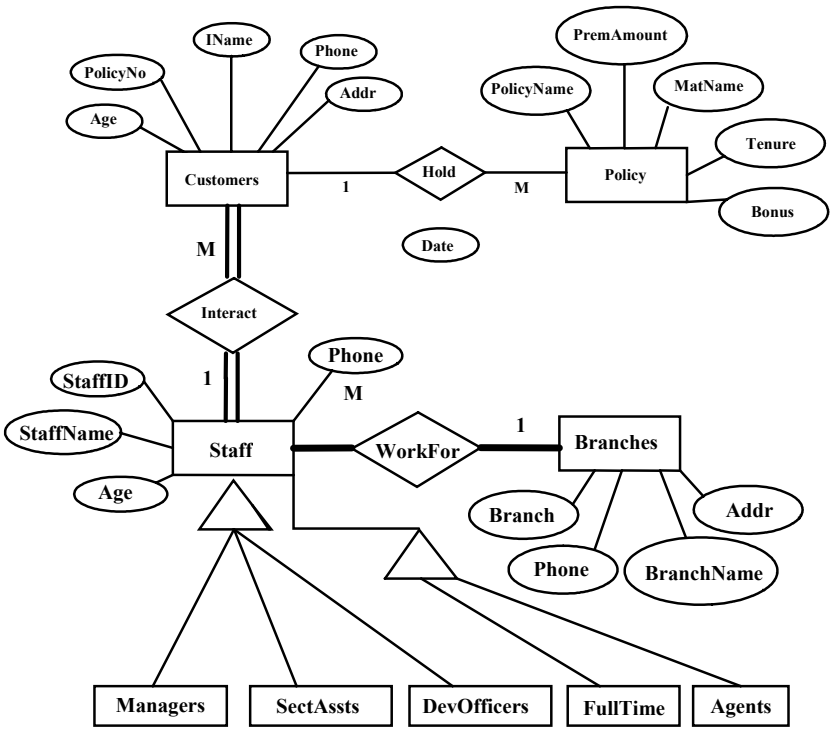
Policy is a weak entity, because without a customer or policy holder the insurance policy doesn't exist. Also, the entity set Policy does not have primary key, but only partial key (PolicyName). The name of the policy may be as follows: Endowment, Moneyback, Medical, etc.

Figure depicts the ER Diagram for Insurance Policy management with all the constraints incorporated.

Contd...

Notes

Figure 1: ER Diagram of COMPANY Database



Step-3: Relationship Sets

As per the requirements as stated already, the following relationship sets may be identified:

1.	WorksFor	Staff and Branches	N: 1	Total on either side
2.	Interacts	Staff and Customers	1:N	Total on either side
3.	Hold	Customers and Policy	1: N	Total on either side

4. Staff (Managers, SectAssts, DevOfficers) IS-A Relationship
5. Staff (FullTime, Agents) IS-A Relationship

6.12 Summary

Relational algebra is a procedural language. It provides a collection of operations to manipulate relations. It supports the notion of a query which is a request to retrieve information from a database. The relational algebra data operations and their corresponding operators are:

Basic Operations:

Selection	Select Operator (σ)
Projection	Project Operator (Π)
Cross-product	Cartesian product (×)
Union	Operator ($\cup$)
Set Difference	Difference Operator (–)

Notes

Additional Operations:

Intersection	Intersect Operator ($\cap$)
Join	Join Operator ($\bowtie$)
Division	Division operator ($\div$)
Rename	Rename operator (ρ)

Here, selection, projection, rename are unary operators and other operators are binary.

6.13 Keywords

Binary operations: Operations which operate on two relations.

ER model: The entity-relationship (ER) data model allows us to describe the data involved in real-world enterprise in terms of objects (entities) and their relationships, and is widely used to develop an initial database design.

Relational algebra: The Relational Algebra which is an algebraic notation, where queries are expressed by applying specialized operators to the relations.

Relational calculus: The Relational Calculus which is a logical notation, where queries are expressed by formulating some logical restrictions that the tuples in the answer must satisfy.

Unary operation: Operations which operates on only one relation.

6.14 Self Assessment

Choose the appropriate answer:

1. Relational calculus is an alternative to:
   (a) Relational algebra
   (b) Relational valuation
   (c) Related query
   (d) Relational calculation
2. QBE stands for
   (a) Quick-by-Example
   (b) Query-by-Example
   (c) Queue-by-Example
   (d) Query-by-Expansion

Fill in the blanks:

3. A ..... is a variable that takes on tuples of a particular relation schema as values.
4. A .....formula is defined in a manner that is very similar to the definition of a TRC formula.
5. The ER model is important for its role in .....
6. ..... represent entity sets.

Notes

7. An entity set attributes that does not have a primary key within them, is termed as a ..... set.

8. .....determine whether two subclasses are allowed to contain the same entity.

9. ..... determine whether the entities in the subclasses collectively include all entities in the superclass.

10. ..... allows us to indicate that a relationship set participates in another relationship sets.

6.15 Review Questions

1. Amazon.com has decided to reorganize its database. Information about users, books and sales are stored. Amazon stores as much information as possible on user activity, in order to analyze and improve its site. Below are few requirements:

   (a) A user has a unique id, name, password and a single email address. Amazon contacts users periodically by email, so it is important to know whether a user's email has been bouncing back messages and whether the user is willing to be spammed.

   (b) The last date of a user's visit is stored, so that Amazon can display to the user a list of items that are new to the site since his last visit.

   (c) Books have an ISDN number, title, author's name, publisher's name and cost.

   (d) For each sale, Amazon stores the date of sale, the items bought, the customer (that has to be a user), his/her address (street, number, city, state, country, zipcode), telephone number, and credit-card number.

   (e) Users who have bought at least one book, can place comments about every book (although it is not a book that he has bought at Amazon), by giving a rate to the book from 1 to 10.

   (f) Amazon stores, for each comment, the content of the comment and the percentage of users who were helped by this comment.

   (g) A book can be on a 'wish-list' of a user. This is a book that the user would like to buy at Amazon. Books from a wish-list can be bought for the user by himself, or by a friend. The friend has also to be a user of Amazon. Amazon wants to keep track of whether books on a wish-list were bought, and by whom they were bought.

   Draw an entity relationship diagram to model the information described above. Remember to put constraints, key attributes, etc. If you use the ISA relationship, state any covering and overlap constraints that hold. Make any necessary and logical assumptions. State any such assumptions clearly.

2. A Bank wants to computerize all of its transactions. It offers the following account types: Savings Bank (SB), Recurring Deposit (RD), Fixed Deposit (FD)

   The Bank also wishes to keep track of loans given to the customers. Identify the entities and its attributes with all possible relationships. Write the ER diagram and state clearly the assumptions that you make. The following assumptions may be considered:

   (a) A customer can have only one type of account.

   (b) Joint accounts are not allowed. Loans can be taken only when the customer has at least one of the account types.

Notes

3. Use an Entity-Relationship diagram to depict the information needs of a small computer business firm:

   (a) The employees of the company assemble different types of computers. For each employee a record is kept of his employee no., name, address, phone no., job title, and salary.

   (b) A record is also kept of each of the machines model, specs and name, and quantity on hand.

   (c) Each machine consists of a number of parts. An inventory must be kept of the parts in stock. For each part a record is kept of its name, price, and quantity on hand.

   (d) These parts are ordered from various suppliers. A record must be kept of the suppliers name, address, and phone number.

   (e) The assembled computers are sold to various customers. A record is kept of the customers name, address, and phone number. Some of these customers are credit customers and for these customers a record is kept of their credit limit.

4. Why are entity integrity and referential integrity important in a database?

5. Suppose that you are using the database composed of the two tables shown in table below:

Table name: DIRECTOR

DIR_NUM	DIR_LNAME	DIR_DOB
100	Broadway	12-Jan-65
101	Hollywoody	18-Nov-53
102	Goofy	21-Jun-62

Table name: PLAY

PLAY_CODE	PLAY_NAME	DIR_NUM
1001	Cat On a Cold, Bare Roof	102
1002	Hold the Mayo, Pass the Bread	101
1003	I Never Promised You Coffee	102
1004	Silly Putty Goes To Washington	100
1005	See No Sound, Hear No Sight	101
1006	Starstruck in Biloxi	102
1007	Stranger In Parrot Ice	101

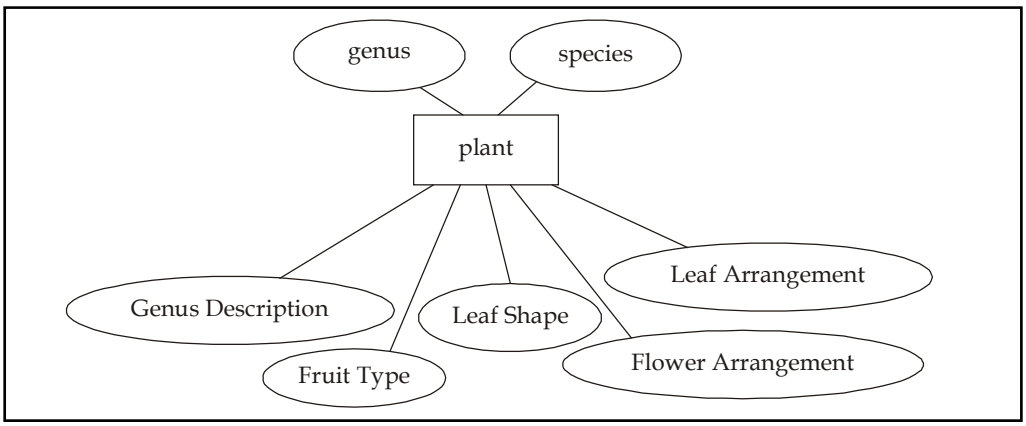
   (a) Identify the primary keys.

   (b) Identify the foreign keys.

   (c) Draw the entity relationship model.

Notes

6. Write all full functional dependencies for the ER diagram below. The genus and species of the plant determine the type of fruit, flower arrangement, leaf arrangement, and leaf shape.



Answers: Self Assessment

1.	(a)	2.	(b)
3.	tuple variable	4.	DRC
5.	database design	6.	Rectangles
7.	weak entity	8.	Overlap constraints
9.	Covering constraints	10.	Aggregation

6.16 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management*, 7th Edition.

Raghurama Krishnan, Johannes Gehrke,*Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 7: Relational Database Design

CONTENTS

Objectives

Introduction

7.1 Relational Database Design

7.2 Features of Relational Database

7.3 Atomic Domain and First Normal Form

7.4 Functional Dependencies

7.5 Multi-valued Dependencies

7.6 Join Dependencies

7.7 Rules about Functional Dependencies

7.8 Database Design Process

7.8.1 Logical Database Design

7.8.2 Entity Sets to Tables

7.9 Summary

7.10 Keywords

7.11 Self Assessment

7.12 Review Questions

7.13 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain relational database design
- Describe various features of relational database
- Know atomic domain and first normal form
- Describe functional and multivalued dependencies
- Explain join dependencies
- Describe database design process

Introduction

Relational database supports basic database operations in order to provide useful means for retrieving or manipulating data in tables. Because the relational model has its mathematical basis upon the relational theory (by thinking tables as sets or relations), the supported database operators conform to existing operators in relational algebra. In fact, a relational database software implementation, called DBMS, is said to have higher degree of relational completeness depending upon the extent to which relational algebra operators are supported.

Notes

7.1 Relational Database Design

The relational model was proposed by E. F. Codd in 1970. It deals with database management from an abstract point of view. The model provides specifications of an abstract database management system. To use the database management systems based on the relational model however, users do not need to master the theoretical foundations.

Codd defined the model as consisting of the following three components:

1. ***Data Structure:*** a collection of data structure types for building the database.
2. ***Data Manipulation:*** a collection of operators that may be used to retrieve, derive or modify data stored in the data structures.
3. ***Data Integrity:*** a collection of rules that implicitly or explicitly define a consistent database state or changes of states.

7.2 Features of Relational Database

A good database design has the following features:

1. ***Faithfulness:*** The design and implementation should be faithful to the requirements.
   (a) The use of constraints helps to achieve this feature.
2. ***Avoid Redundancy:*** Something is redundant if when hidden from view, you could still figure it out from other data. This value is important because redundancy.
   (a) Wastes space and
   (b) Leads to inconsistency.
3. ***Simplicity:*** Simplicity requires that the design and implementation avoid introducing more elements than are absolutely necessary – Keep it Simple (KIS).
   (a) This value requires designers to avoid introducing unnecessary intermediate concepts.
4. ***Right kind of element:*** Attributes are easier to implement but entity sets and relationships are necessary to ensure that the right kind of element is introduced.

7.3 Atomic Domain and First Normal Form

Please see section 7.10 in detail

Task DBMS vs. RDBMS.

7.4 Functional Dependencies

When a single constraint is established between two sets of attributes from the database it is called functional dependency. We consider a single universal relation scheme "A". A functional dependency denoted by XY, between two sets of attributes X and Y that are subset of universal relation "A" specifies a constraint on the possible tuples that can form a relation state of "A". The constraint is that, for any two tuples t_1 and t_2 in "A" that have $t_1(X) = t_2(X)$, we must also have $t_1(Y) = t_2(Y)$. It means that, if tuple t_1 and t_2 have same values for attributes X then $X \rightarrow Y$ to hold t_1 and t_2 must have same values for attributes Y.

Notes

Thus, FD X → Y means that the values of the Y component of a tuple in "A" depend on or is determined by the values of X component. In other words, the value of Y component is uniquely determined by the value of X component. This is functional dependency from X to Y (**but not Y to X**) that is, Y is functionally dependent on X.

The relation schema "A" determines the function dependency of Y on X (X → Y) when and only when:

1. If two tuples in "A", agree on their X value then
2. They must agree on their Y value.

Please note that if X → Y in "A", does not mean Y → X in "A".

This semantic property of functional dependency explains how the attributes in "A" are related to one another. A FD in "A" must be used to specify constraints on its attributes that must hold at all times.

Example: A FD state, city, place → pin-code should hold for any address in India. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes, for example, the FD pin-code→ area-code used to exist as a relationship between postal codes and telephone number codes in India, with the proliferation of mobile telephone, the FD is no longer true.

Consider a relation

STUDENT-COURSE (enrolno, sname, cname, classlocation, hours)

We know that the following functional dependencies (we identify these primarily from constraints, there is no thumb rule to do so otherwise) should hold:

1. ***enrolno → sname*** (the enrolment number of a student uniquely determines the student names alternatively, we can say that sname is functionally determined/dependent on enrolment number).
2. ***classcode → cname, classlocation*** (the value of a class code uniquely determines the class name and class location.
3. ***enrolno, classcode → Hours*** (a combination of enrolment number and class code values uniquely determines the number of hours and students study in the class per week (Hours).

These FDs can be optimised to obtain a minimal set of FDs called the canonical cover. However, these topics are beyond the scope of this course and can be studied by consulting further reading list.

Normalisation

The first concept of normalisation was proposed by Mr. Codd in 1972. Initially, he alone proposed three normal forms named first, second and third normal form. Later on, with the joint efforts of Boyce and Codd, a stronger definition of 3NF called Boyce-Codd Normal Form (BCNF) was proposed. All the said normal forms are based on the functional dependencies among the attributes of a relation. The normalisation process depends on the assumptions that:

1. A set of functional dependencies is given for each relation, and
2. Each relation has a designated primary key.

The normalisation process is based on the two assumptions /information above. Codd takes a relation schema through a series of tests to ascertain whether it satisfies a certain normal form.

Notes

The process proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as found necessary during analysis.

Therefore, normalisation is looked upon as a process of analysing the given relation schemas based on their condition (FDs and Primary Keys) to achieve the desirable properties:

1. *Firstly,* Minimizing redundancy, and
2. *Secondly,* Minimizing the insertion, deletion update anomalies.

Thus, the normalisation provides the database designer with:

1. A formal framework for analysing relation schemas.
2. A series of normal form tests that can be normalised to any desired degree.

The degree of normal forms to which a relation schema has been normalised through decomposition confirm the existence of additional properties that the relational schemas should possess. It could include any or both of two properties.

1. The lossless join and non-additive join property, and
2. The dependency preservation property.

Based on performance reasons, relations may be left in a lower normalisation status. It is not mandatory that the database designer must normalise to the highest possible normal form. The process of storing the join of higher normal form relations, as a base relation (which is in a lower normal form) is known as denormalisation).

7.5 Multi-valued Dependencies

In database modeling using the E-R Modeling technique, we usually face known difficulties that may arise when an entity has multivalued attributes. In the relational model, if all of the information about such entity is to be represented in one relation, it will be necessary to repeat all the information other than the multivalue attribute value to represent all the information. It will result in multi-tuples about the same instance of the entity in the relation and the relation having a composite key (the entity id and the mutlivalued attribute). This situation becomes much worse if an entity has more than one multivalued attributes and these values are represented in one relation by a number of tuples for each entity instance such that every value of one of the multivalued attributes appears with every value of the second multivalued attribute to maintain consistency. The multivalued dependency relates to this problem when more than one multivalued attributes exist. Let us consider the same through an example relation that represents an entity 'employee'.

emp (e#, dept, salary, job)

We have so far considered normalisation based on functional dependencies that apply only to single-valued facts. For example, e# → dept implies only one dept value for each value of e#. Not all information in a database is single-valued, for example, job in an employee relation may be the list of all projects that the employee is currently working on. Although e# determines the list of all the projects that an employee is working on, yet, e# → job is not a functional dependency.

The fourth and fifth normal forms deal with multivalued dependencies. Before discussing the 4NF and 5NF we will discuss the following example to illustrate the concept of multivalued dependency.

programmer (emp_name, projects, languages)

Notes

The above relation includes two multivalued attributes of the entity programmer -projects and languages. There are no functional dependencies.

The attributes projects and languages are assumed to be independent of each other. If we were to consider projects and languages as separate entities, we would have two relationships (one between employees and projects and the other between employees and programming languages). Both the above relationships are many-to-many relation, in the following sense:

1. One programmer could have several projects.
2. May know several programming languages, also
3. One project may be obtained by several programmers, and
4. One programming language may be known to many programmers.

The above relation is in 3NF (even in BCNF) with some disadvantages. Suppose a programmer has several projects (Proj_A, Proj_B, Proj_C, etc.) and is proficient in several programming languages, how should this information be represented? There are several possibilities.

emp_name	Projects	languages
DEV	Proj_A	C
DEV	Proj_A	JAVA
DEV	Proj_A	C++
DEV	Proj_B	C
DEV	Proj_B	JAVA
DEV	Proj_B	C++

emp_name	Projects	languages
DEV	Proj_A	NULL
DEV	Proj_B	NULL
DEV	NULL	C
DEV	NULL	JAVA
DEV	NULL	C++

emp_name	Projects	languages
DEV	Proj_A	C
DEV	Proj_B	JAVA
DEV	NULL	C++

Other variations are possible. Please note this is so as there is no relationship between the attributes 'projects' and programming 'languages'. All the said variations have some disadvantages. If the information is repeated, we face the problems of repeated information and anomalies as we did when second or third normal form conditions were violated. Without repetition, difficulties still exist with insertions, deletions and update operations. For example, in the first table we want to insert a new person RAM who has just joined the organisation and is proficient in C and JAVA. However, this information cannot be inserted in the first table as RAM has not been allotted to work on any project. Thus, there is an insertion anomaly in the first table. Similarly, if both Project A and Project B get completed on which DEV was working (so we delete all the tuples in the first table) then the information that DEV is proficient in C, JAVA, and C++ languages will also be lost. This is the deletion anomaly. Finally, please note that the information that DEV is working on Project A is being repeated at least three times. Also the information that DEV is proficient in JAVA is repeated. Thus, there is redundancy of information in the first tables that may lead to inconsistency on updating (update anomaly).

Notes

In the second and third tables above, the role of NULL values is confusing. Also the candidate key in the above relations is (emp name, projects, language) and existential integrity requires that no NULLs be specified. These problems may be overcome by decomposing a relation as follows:

emp_name	Projects
DEV	Proj_A
DEV	Proj_B

emp_name	languages
DEV	C
DEV	JAVA
DEV	C++

This decomposition is the concept of 4NF. Functional dependency A → B relates one value of A to one value of B while multivalued dependency A → → B defines a relationship where a set of values of attribute B are determined by a single value of A. Multivalued dependencies were developed to provide a basis for decomposition of relations like the one above. Let us define the multivalued dependency formally.

Task Normalization is a key concept of DBMS. Suggest.

Multivalued dependency: The multivalued dependency X → → Y is said to hold for a relation R(X, Y, Z) if, for a given set of value (set of values if X is more than one attribute) for attribute X, there is a set of (zero or more) associated values for the set of attributes Y and the Y values depend only on X values and have no dependence on the set of attributes Z.

Notes Whenever X → → Y holds, so does X → → Z since the role of the attributes Y and Z is symmetrical.

In the example given above, if there was some dependence between the attributes projects and language, for example, the language was related to the projects (perhaps the projects are prepared in a particular language), then the relation would not have MVD and could not be decomposed into two relations as above. However, assuming there is no dependence, emp_name → → projects and emp_name → → languages holds.

Trival MVD: A MVC X →→Y is called trivial MVD if either Y is a subset of X or X and Y together form the relation R.

The MVD is trivial since it results in no constraints being placed on the relation. If a relation like emp(eno, edpendent#) has a relationship between eno and edependent# in which eno uniquely determines the **values** of edependent#, the dependence of edependent# on eno is called a trivial MVD since the relation emp cannot be decomposed any further.

Therefore, a relation having non-trivial MVDs must have at least three attributes; two of them multivalued and not dependent on each other. Non-trivial MVDs result in the relation having some constraints on it since all possible combinations of the multivalued attributes are then required to be in the relation.

Let us now define the concept of MVD in a different way. Consider the relation R(X, Y, Z) having a multi-valued set of attributes Y associated with a value of X. Assume that the attributes Y and

Notes

Z are independent, and Z is also multi-valued. Now, more formally, $X \rightarrow\rightarrow Y$ is said to hold for R(X, Y, Z) if t_1 and t_2 are two tuples in R that have the same values for attributes X ($t_1[X] = t_2[X]$) then R also contains tuples t_3 and t_4 (not necessarily distinct) such that:

$t_1[X] = t_2[X] = t_3[X] = t_4[X]$
$t_3[Y] = t_1[Y]$ and $t_3[Z] = t_2[Z]$
$t_4[Y] = t_2[Y]$ and $t_4[Z] = t_1[Z]$

In other words if t_1 and t_2 are given by:

$t_1 = [X, Y_1, Z_1]$, and
$t_2 = [X, Y_2, Z_2]$

then there must be tuples t_3 and t_4 such that:

$t_3 = [X, Y_1, Z_2]$, and
$t_4 = [X, Y_2, Z_1]$

We are, therefore, insisting that every value of Y appears with every value of Z to keep the relation instances consistent. In other words, the above conditions insist that Y and Z are determined by X alone and there is no relationship between Y and Z since Y and Z appear in every possible pair and hence these pairings present no information and are of no significance. Only if some of these pairings were not present, there would be some significance in the pairings.

Notes If Z is single-valued and functionally dependent on X then $Z_1 = Z_2$. If Z is multivalue dependent on X then $Z_1 <> Z_2$.

The theory of multivalued dependencies is very similar to that for functional dependencies. Given D a set of MVDs, we may find D^+, the closure of D using a set of axioms. We do not discuss the axioms here. You may refer this topic in further readings.

We have considered an example of Programmer(Emp name, projects, languages) and discussed the problems that may arise if the relation is not normalised further. We also saw how the relation could be decomposed into P1(emp name, projects) and P2(emp name, languages) to overcome these problems. The decomposed relations are in fourth normal form (4NF), which we shall now define.

We now define 4NF. A relation R is in 4NF if, whenever a multivalued dependency $X \rightarrow\rightarrow Y$ holds, then either

1. The dependency is trivial
2. X is a candidate key for R.

The dependency $X \rightarrow\rightarrow ø$ or $X \rightarrow\rightarrow Y$ in a relation R (X, Y) is trivial, since they must hold for all R (X, Y). Similarly, in a trivial MVD $(X, Y) \rightarrow\rightarrow Z$ must hold for all relations R (X, Y, Z) with only three attributes.

If a relation has more than one multivalued attribute, we should decompose it into fourth normal form using the following rules of decomposition:

For a relation R(X,Y,Z), if it contains two nontrivial MVDs $X \rightarrow\rightarrow Y$ and $X \rightarrow\rightarrow Z$ then decompose the relation into R_1 (X,Y) and R_2 (X,Z) or more specifically, if there holds a non-trivial MVD in a relation R (X,Y,Z) of the form $X \rightarrow\rightarrow Y$, such that $X) \cap Y = \phi$, that is the set of attributes X and Y are disjoint, then R must be decomposed to R_1 (X,Y) and R_2 (X,Z), where Z represents all attributes other than those in X and Y.

Notes

Intuitively R is in 4NF if all dependencies are a result of keys. When multivalued dependencies exist, a relation should not contain two or more independent multivalued attributes. The decomposition of a relation to achieve 4NF would normally result in not only reduction of redundancies but also avoidance of anomalies.

7.6 Join Dependencies

Based on the discussion above, we know that the normal forms require that the given relation R if not in the given normal form should be decomposed in two relations to meet the requirements of the normal form. However, in some rare cases, a relation can have problems like redundant information and update anomalies, yet it cannot be decomposed in two relations without loss of information. In such cases, it may be possible to decompose the relation in three or more relations using the 5NF. But when does such a situation arise? Such cases normally happen when a relation has at least three attributes such that all those values are totally independent of each other.

The fifth normal form deals with join-dependencies, which is a generalisation of the MVD. The aim of fifth normal form is to have relations that cannot be decomposed further. A relation in 5NF cannot be constructed from several smaller relations.

A relation R satisfies join dependency *(R_1, R_2, ..., R_n) if and only if R is equal to the join of R_1, R_2, ..., R_n where R_i are subsets of the set of attributes of R.

A relation R is in 5NF if for all join dependencies at least one of the following holds:

1. **(R_1, R_2, ..., R_n) is a trivial join-dependency (that is, one of R_i is R)**
2. Every R_i is a candidate key for R.

An example of 5NF can be provided by the same above example that deals with emp_name, Projects and Programming languages with some modifications:

emp_name	Projects	Languages
DEV	Proj_A	C
RAM	Proj_A	JAVA
DEV	Proj_B	C
RAM	Proj_B	C++

The relation above assumes that any employee can work on any project and knows any of the three languages. The relation also says that any employee can work on projects Proj_A, Proj_B, Proj_C and may be using a different programming languages in their projects. No employee takes all the projects and no project uses all the programming languages and therefore all three fields are needed to represent the information. Thus, all the three attributes are independent of each other.

The relation above does not have any FDs and MVDs since the attributes emp_name, project and languages are independent; they are related to each other only by the pairings that have significant information in them. For example, DEV is working on Project A using C languague. Thus, the key to the relation is (emp_name, project, languague). The relation is in 4NF, but still suffers from the insertion, deletion, and update anomalies as discussed for the previous form of this relation. However, the relation therefore cannot be decomposed in two relations.

(emp_name, project), and

(emp_name, language)

Notes

Why?

Let us explain this with the help of a definition of join dependency. The decomposition mentioned above will create tables as given below:

Emp_project

emp_name	Projects
DEV	Proj_A
RAM	Proj_A
DEV	Proj_B
RAM	Proj_B

Emp_language

Emp_name	languages
DEV	C
RAM	JAVA
RAM	C++

On taking join of these relations on emp_name it will produce the following result:

emp_name	Projects	Languages
DEV	Proj_A	C
RAM	Proj_A	JAVA
RAM	Proj_A	C++
DEV	Proj_B	C
RAM	Proj_B	JAVA
RAM	Proj_B	C++

Since the joined table does not match the actual table, we can say that it is a lossy decomposition. Thus, the expected join dependency expression:

*((emp_name, project), (emp_name, language)) does not satisfy the conditions of lossless decomposition. Hence, the decomposed tables are losing some important information.

Can the relation 'Programmer' be decomposed in the following three relations?

(emp_name, project),

(emp_name, language) and

(Projects, language)

Please verify whether this decomposition in lossless or not. The join dependency in this case would be:

*((emp_name, project), (emp_name, language), (project, language))

and it can be shown that this decomposition is lossless.

Project-Join Normal Form

PJNF is defined using the concept of the join dependencies. A relation schema R having a set F of functional, multivalued, and join dependencies, is in PJNF (5 NF), if for all the join dependencies in the closure of F (referred to as F+) that are of the form

$*(R_1, R_2, \ldots, R_n)$,

Notes

1. $*(R_1, R_2, \ldots, R_n)$ is a trivial join dependency.
2. Every R_i is a superkey for R.

PJNF is also referred to as the Fifth Normal Form (5NF).

Let us first define the concept of PJNF from the viewpoint of the decomposition and then refine it later to a standard form.

Definition 1: A JD $*[R_1, R_2, \ldots, R_n]$ over a relation R is trivial if it is satisfied by every relation r(R).

The trivial JDs over R are JDs of the form $*[R_1, R_2, \ldots, R_n]$ where for some i the $R_i = R$.

Definition 2: A JD $*[R_1, R_2, \ldots, R_n]$ applies to a relation scheme R if $R = R_1 R_2 \ldots R_n$.

Definition 3: Let R be a relation scheme having F as the set of FDs and JDs over R. R will be in project-join normal form (PJNF) if for every JD $*[R_1, R_2, \ldots, R_n]$ which can be derived by F that applies to R, the following holds:

1. The JD is trivial
2. Every R_i is a super key for R.

For a database scheme to be in project-join normal form, every relation R in this database scheme should be in project-join normal form with respect to F.

Let us explain the above with the help of an example.

Example: Consider a relational scheme R = A B C D E G having the set of dependencies F = {*[A B C D, C D E, B D G], *[A B, B C D, A D], A → B C D E, B C → A G }. The R as given above is not in PJNF. Why? The two alternate keys to R are A and BC, so please note that the JD *[A B C D, C D E, B D G], does not satisfy the condition "Every R_i is a super key for R" as the two components of this JD viz., C D E and B D G, does not satisfy the condition.

However, if we decompose the R as $\{R_1, R_2, R_3\}$, where R_1 = A B C D, R_2 = C D E, and R_3 = B D G, then it is in PJNF with respect to F. Please note that in the example, the JD *[A B, B C D, A D] is implied by F and applies to R_1. Whereas, the FDs are trivial or have keys as the left side.

The definition of PJNF as given above is a weaker than the original definition of PJNF given by Fagin. The original definition ensures enforceability of dependencies by satisfying keys, in addition to elimination of redundancy. The final definition is:

Definition 4: Let R be a relation scheme having F as the set of FDs and JDs over R. R will be in project-join normal form (PJNF) if for every JD $*[R_1, R_2, \ldots, R_n]$ which can be derived by F that applies to R, is implied by the key FDs of R.

The following example demonstrates this definition.

Example: Consider a relation scheme R = A B C having the set of dependencies as F = {A→ B C, C → A B, *[A B, B C] }. Please note that the R is not in PJNF, although since A B and B C are the super keys of R, R satisfies the earlier definition of PJNF. But R does not satisfy the revised definition as given above.

Notes Since every multivalued dependency is also a join dependency, every PJNF schema is also in 4NF. Decomposing a relation scheme using the JDs that cause PJNF violations creates the PJNF scheme. PJNF may also be not dependency preserving.

Notes

Task Create join dependencies.

7.7 Rules about Functional Dependencies

Functional dependencies (FD) are type of constraint that is based on *keys*. A ***superkey*** is defined as in the relational schema *R* , where:

a subset *K* of *R* is a *subkey* of *R* if, in any legal relation *r(R)*, for all pairs, t_1 and t_2 in tuple *r* such that t_1 is not equal to t_2 then $t_1[K]$ is not equal to $t_2[K]$.

Or, no two rows (tuples) have the same value in the attribute(s) *K*, which is the key. Now, if there are two attributes (or sets of attributes) *A* and *B* that are legal in the relation schema *R*, we can have a functional dependency where

A implies *B*

for all pairs of tuples such that $t_1[A]$ is equal to $t_2[A]$ and$t_1[B]$ is equal to $t_2[B]$. This allows us to state that *K* is a superkey of *R* if *K* implies *R*. For example, in a relation that has names and social security numbers, whenever your Social Security number is the student ID, the name in that tuple can only contain your name. That is because your name is not unique, but your Social Security is. If I go to the Social Security Administration and search their database for the name "Gary Burt", the results is a large number of people. If I search of the social security number "123-45-6789", the result is one and only one person.

Another example is in the loan information that we looked at before:

Loan-info-schema = *(branch-name, loan-name, customer-name, amount)*

it can be shown that the *loan-number* implies both the *amount* and the *branch-name*.It does not imply the *customer-name* because there may be more than one person listed on the load, such as a husband and wife, or parent and child (when the parent co-signs the loan).

Functional dependencies:

1. Specify a set of constraints on a legal relation.
2. Test relations to see if they are legal.

Some relations are said to be trivial when they are satisfied by all relations:

7.8 Database Design Process

The database design process can be divided into six steps. The E-R model is most relevant to the first three steps.

1. ***Requirements Analysis:*** The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database.
2. ***Conceptual Database Design:*** The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the E-R model or a similar high-level data model.

Notes

3. ***Logical Database Design:*** We must choose a DBMS to implement our database design and convert the conceptual database design into a database schema in the data model of the chosen DBMS. Task here is to convert E-R schema into relational database schema.
4. ***Schema Refinement:*** The fourth step in database design is to analyze the collection of relations in our relational database schema to identify potential problems and to refine it.
5. ***Physical Database Design:*** In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.
6. ***Security Design:*** In this step, we identify different user groups and different roles played by various users (e.g., the development team for a product, the customer support representatives, and the product manager). For each role and user group, we must identify the parts of the database that they must be able to access and the parts of the database that they should not be allowed to access and take steps to ensure that they can access only the necessary parts.

Task Uses of Superkey in DBMS.

7.8.1 Logical Database Design

During logical design we transform the E-R diagrams that were developed during conceptual design into relational database schemas.

Transforming E-R diagrams to relations is a relatively straightforward process with a well-defined set of rules. In fact, many CASE tools can automatically perform many of the conversion steps. However, it is important that we understand the steps in this process for three reasons:

1. CASE tools often cannot model more complex data relationships such as ternary relationships and supertype/subtype relationships. For these situations we may have to perform the steps manually.
2. There are sometimes legitimate alternatives where we will need to choose a particular solution.
3. We must be prepared to perform a quality check on the results obtained with a CASE tool.

The ER model is convenient for representing an initial, high-level database design. Given an ER diagram describing a database, there is a standard approach to generating a relational database schema that closely approximates the ER design. We now describe how to translate an ER diagram into a collection of tables with associated constraints, i.e., a relational database schema.

7.8.2 Entity Sets to Tables

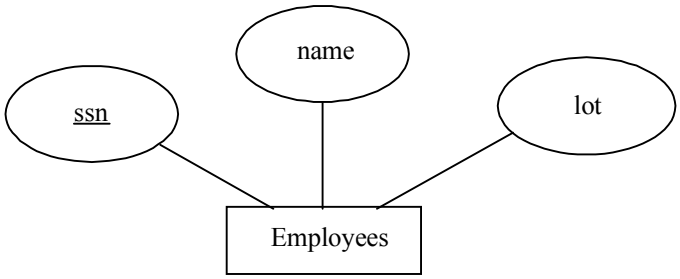
An entity set is mapped to a relation in a straightforward way: Each attribute of the entity set becomes an attribute of the table.

Notes Both the domain of each attribute and the (primary) key of an entity set.

Notes

Consider the Employees entity set with attributes ssn, name, and lot shown in Figure 7.1.

Figure 7.1: The Employees Entity Set



A possible instance of the Employees entity set, containing three Employees entities, is shown in Figure 7.2 in tabular format.

Figure 7.2: An Instance of the Employees Entity Set

ssn	name	lot
1	Raju	10
2	Santosh	20
3	Brahma	30

The following SQL statement captures the preceding information, including the domain constraints and key information:

```
CREATE TABLE Employees    (ssn     CHAR (11),
                          name     CHAR (30),
                          lot      INTEGER,
                          PRIMARY KEY (ssn) )
```

Relationship Sets (without Constraints) to Tables: A relationship set, like an entity set, is mapped to a relation in the relation model. To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship. Thus, the attributes of the relation include:

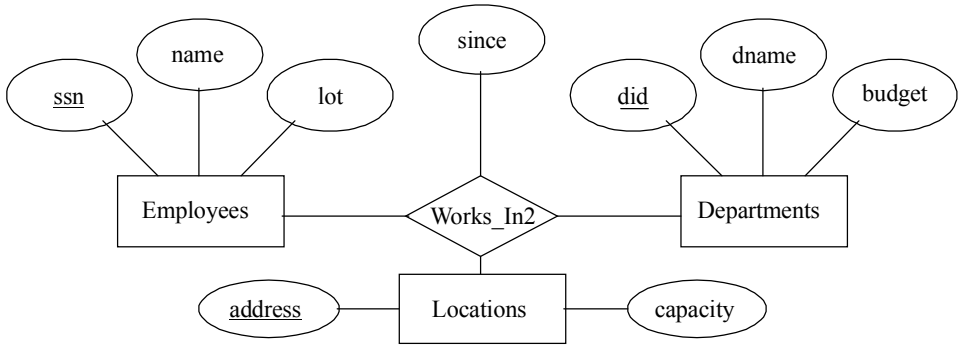
1. The primary key attributes of each participating entity set, as foreign key fields.
2. The descriptive attributes of the relationship set.

The set of non-descriptive attributes is a superkey for the relation. If there are no key constraints, this set of attributes is a candidate key.

Consider the Works_In2 relationship set shown in Figure 7.3. Each department has offices in several locations and we want to record the locations at which each employee works.

Notes

Figure 7.3: A Ternary Relationship Set



All the available information about the Works_In2 table is captured by the following SQL definition:

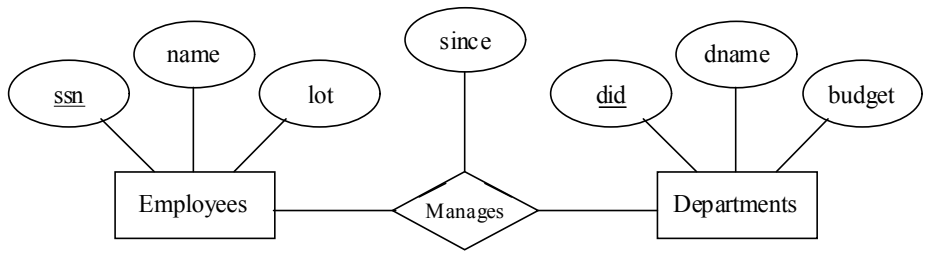
```
CREATE TABLE Works_In2    (ssn        CHAR(11),
                          did         INTEGER,
                          address     CHAR(20),
                          since       DATE,
                          PRIMARY     KEY(ssn, did, address),
                          FOREIGN     KEY(ssn) REFERENCES employees,
                          FOREIGN     KEY(address)REFERENCESLocations,
                          FOREIGN     KEY(did) REFERENCES Departments)
```

The address, did, and ssn fields cannot take on null values. Because these fields are part of the primary key for Works_In2, a NOT NULL constraint is implicit for each of these fields. These constraint ensures that these fields uniquely identify a department, an employee, and a location in each tuple of Works_In.

Translating Relationship Sets with Key Constrains: If a relationship set involves *n* entity sets and some *m* of them are linked via arrows in the ER diagram, the key for any one of these *m* entity sets constitutes a key for the relation to which the relationship set is mapped. Thus we have *m* candidate keys, and one of these should be designated as the primary key.

Consider the relationship set Manages shown in Figure 7.4. The table corresponding to Manages has the attributes ssn, did, since. However, because each department has at most one manager, no two tuples can have the same did value but differ on the ssn value. A consequence of this observation is that did is itself a key for Manages; indeed, the set did, ssn is not a key (because it is not minimal). The Manages relation can be defined using the following SQL statement:

Figure 7.4: Key Constraint on Manages



Notes

```
CREATE TABLE Manages    (ssn      CHAR(11),
                         Did      INTEGER,
                         since    DATE,
                         PRIMARY  KEY(did),
                         FOREIGN  KEY(ssn) REFERENCES Employees,
                         FOREIGN  KEY(did) REFERENCES Departments )
```

A second approach to translating a relationship set with key constraints is often superior because it avoids creating a distinct table for the relationship set.

This approach eliminates the need for a separate Manages relation, and queries asking for a department's manager can be answered without combining information from two relations. The only drawback to this approach is that space could be wasted if several departments have no managers. In this case the added fields would have to be filled with null values. The first translation (using a separate table for Manages) avoids this inefficiency, but some important queries require us to combine information from two relations, which can be a slow operation.

The following SQL statement, defining a Dept. Mgr relation that captures the information in both Departments and Manages, illustrates the second approach to translating relationship sets with key constraints:

```
CREATE TABLE Dept_Mgr   (did      INTEGER
                         dname    CHAR(20),
                         budget   REAL,
                         ssn      CHAR(11),
                         since    DATE,
                         PRIMARY  KEY(did),
                         FOREIGN  KEY(ssn)REFERENCES Employees)
```

Notes The *ssn* can take on *null* values.

Translating Relationship Sets with Participation Constraints: Consider the ER diagram in Figure 7.5, which shows two relationship sets, Manages and Works. In Every department is required to have a manager, due to the participation constraint, and at most one manager, due to the key constraint.

```
CREATE TABLE Dept_Mgr(  did      INTEGER
                        dname    CHAR(20),
                        budget   REAL,
                        ssn      CHAR(11) NOT NULL,
                        since    DATE,
                        PRIMARY KEY(did),
                        FOREIGN KEY (ssn)REFERENCESEmployees,
                        ON DELETE NO ACTION )
```

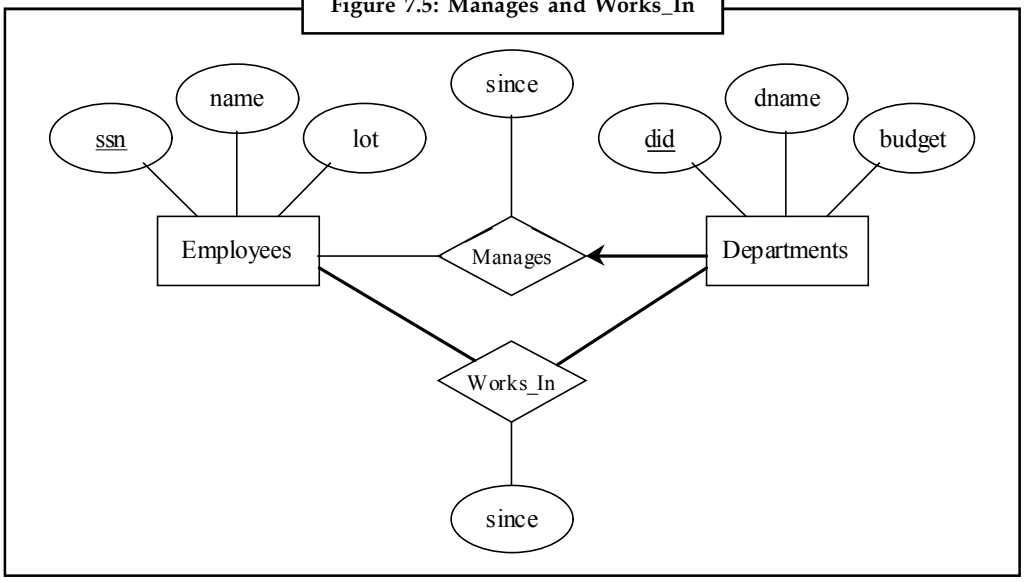
Table constraints and assertions can be specified using the null power of the SQL query language and are very expressive, but also very expensive to check and enforce.

Example: We cannot enforce the participation constraints on the Works_In relation without using these general constraint.

To ensure total participation of Departments in Works_In, we have to guarantee that every did value in Departments appears in a tuple of Works_In.

Notes

Figure 7.5: Manages and Works_In



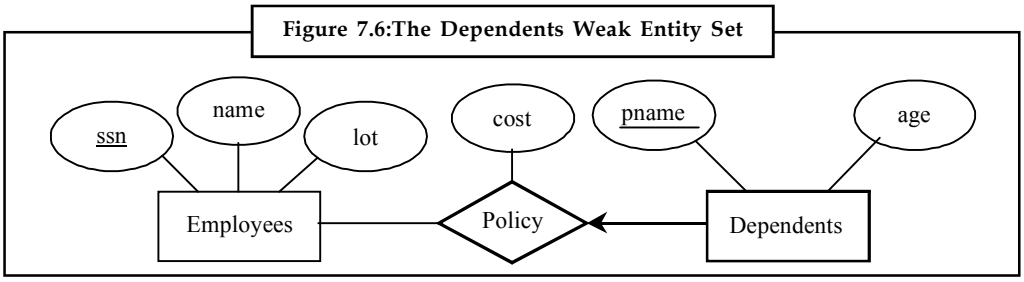
Translating Weak Entity Sets: A weak entity set always participates in a one-to-many binary relationship and has a key constraint and total participation. The weak entity has only a partial key. Also, when an owner entity is deleted, we want all owned weak entities to be deleted.

Consider the Dependents weak entity set shown in Figure 7.6, with partial key pname. A Dependents entity can be identified uniquely only if we take the key of the owning Employees entity and the pname of the Dependents entity and the Dependents entity must be deleted if the owning Employees entity is deleted.

We can capture the desired semantics with the following definition of the Dep-Policy relation:

```
CREATE TABLE Dep_Policy     (pname    CHAR(20),
                             age      INTEGER,
                             cost     REAL,
                             ssn      CHAR(11),
                             PRIMARY KEY(pname, ssn),
                             FOREIGN KEY (ssn) REFERENCES Employees,
                                      ONDELETECASCADE )
```

Figure 7.6:The Dependents Weak Entity Set

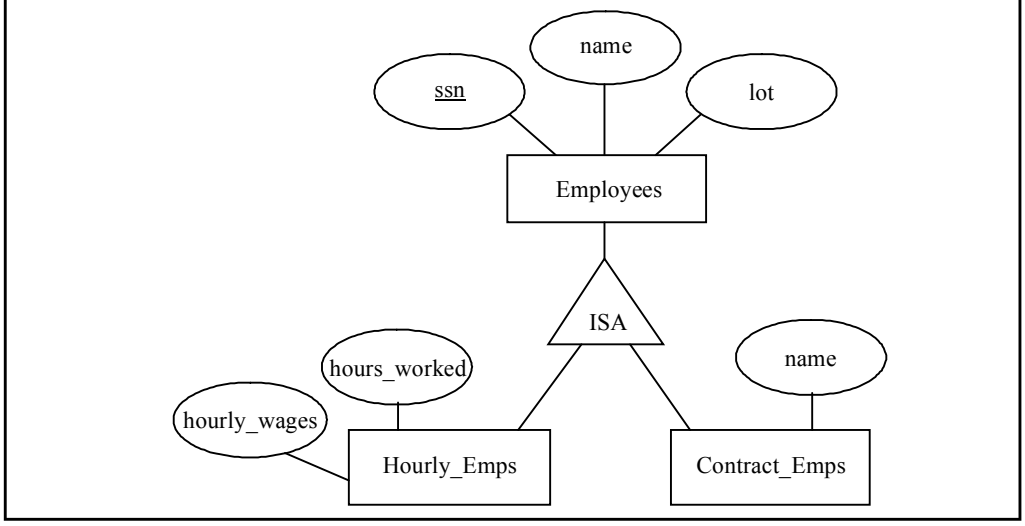


The primary key is pname, ssn, since Dependent is a weak entity.

Translating Class Hierarchies: We present the two basic approaches to handling ISA hierarchies by applying them to the ER diagram.

Notes

Figure 7.7: Class Hierarchy



1. We can map each of the entity sets Employees, Hourly_Emps, and Contract_Emps to a distinct relation. The relation for Hourly_Emps includes the hourly_wages and hours_worked attributes of Hourly_Emps. It also contains the key attributes of the superclass, which serve as the primary key for Hourly_Emps, as well as a foreign key referencing the superclass (Employees). For each Hourly_Emps entity, the value of the name and lot attributes are stored in the corresponding row of the superclass (Employees).

2. We can create just two relations, corresponding to Hourly_Emps and Contract_Emps. The relation for Hourly_Emps includes all the attributes of Hourly_Emps as well as all the attributes of Employees (i.e., ssn, name, lot, hourly_wages, hours_worked).

The first approach is general and is always applicable. Queries in which we want to examine all employees and do not care about the attributes specific to the subclasses are handled easily using the Employees relation.

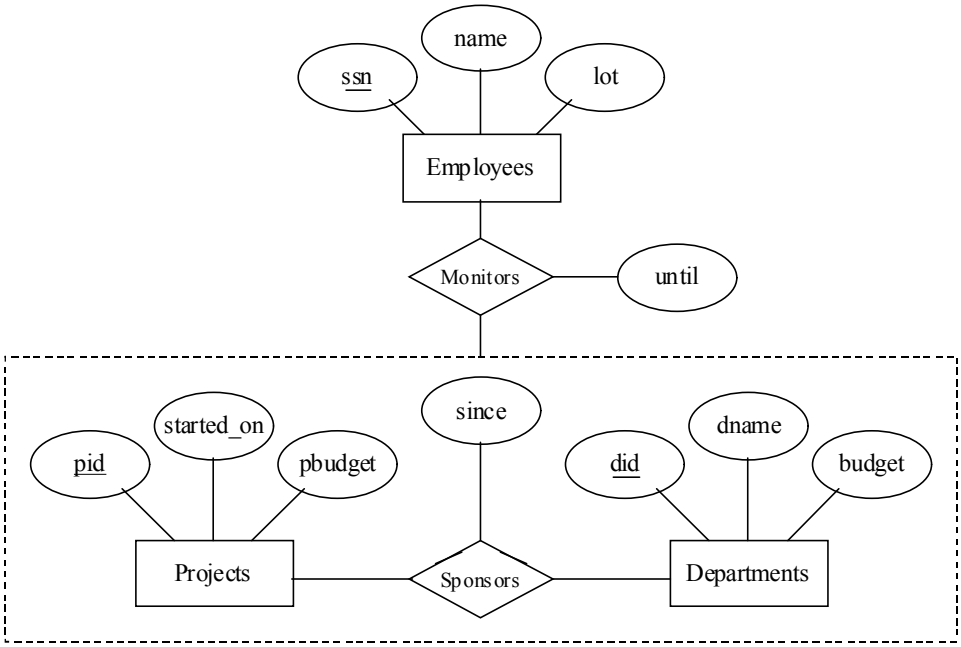
The second approach is not applicable if we have employees who are neither hourly employees nor contract employees, since there is no ways to store such employees. Also, if an employee is both an Hourly_Emps and a Contract_Emps entity, then the *name* and *lot* values are stored twice. This duplication can lead to some of the anomalies.

Translating ER Diagrams with Aggregation: Translating aggregation into the relational model is easy because there is not real distinction between entities and relationships in the relational model.

Consider the ER diagram shown in Figure 7.8. The Employees, Project, and Departments entity sets and the Sponsors relationship set are mapped as described in previous sections. For the Monitors relationship set, we create a relation with the following attributes: the key attributes of Employees (*ssn*), the key attributes of Sponsors (*did, pid*), and the descriptive attributes of Monitors (*until*).

Notes

Figure 7.8: Aggregation



Consider the Sponsors relation. It has attributes *pid*, *did*, and *since*, and in general we need it (in addition to Monitors) for two reasons:

1. We have to record the descriptive attributes (in our example, *since*) of the Sponsors relationship.
2. Not every sponsorship has a monitor, and thus some *pid*, *did* pairs in the Sponsors relation may not appear in the Monitors relation.

However, if Sponsors has no descriptive attributes and has total participation in Monitors, every possible instance of the Sponsors relation can be obtained by looking at the *pid*, *did*columns of the Monitors relation. Thus, we need not store the Sponsors relation in this case.

Task How many foreign key available on single table?

Case Study

Data Warehousing: A Strategic Tool

Datawarehouse is a massive independent business database system that is populated with data that has been extracted from a range of sources. The data is held separately from its origin and is used to help to improve the decision-making process.

Many traditional Databases are involved in recording day to day operational activities of the business, called online transaction processing (OLTP)., commonly implemented in Airline Bookings & Banking Systems, for fasters response and better control over data.

Contd...

Notes

After establishment of OLTP Systems, reports and summaries can be drawn for giving inputs to decision-making process and this process is called online analytical processing (OLAP).

For better customer relationships management strategy, the call centres and datawarehouses must be fully integrated. Datawarehouse works as a strategic tool for decision-support which requires lot of time for establishment, and needs to be updated with operational information on daily weekly or monthly basis.

Datawarehouse is used for proactive strategies formulation in critical & complex situations. A number of CRM vendors are advocating for single integrated customer database which includes call centre, web sites, branches and direct mail, but it lacks in analytical functioning of datawarehouse. This Database can't be expanded also, and carry decision support operations on call centre Database becomes slow & the query processing and inquiries handling operations also become slow & inefficient for agents dealing with customers.

Datawarehouse is must for identifying most profitable & loyal customers and those customers can be offered better customized services which increases the chances of additional profits.

Although call centre systems & datawarehouses are altogether different systems yet dependant on each other to fully exploit their potential respectively.

Questions

1. Explain OLTP & OLAP processes with their respective advantages.
2. How the response time in performing OLAP queries can be improved?
3. Explain the importance of regular updation of data in a datawarehouse.
4. Explain the role of datawarehousing in the functioning of a call centre.
5. "Datawarehouse works as a strategic tool for decision support". Comment.

Source: Management Information System by Dharmenda and Sangeeta Gupta

7.9 Summary

- As database is a collection of tables and tables are collection of fields and fields are collection of data items, so to design the database we have to follow certain rules on the data or information.
- To design an efficient database we have to rake all measures in the beginning, so these measures are taken according to rules.

7.10 Keywords

Foreign Key: A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data.

Functional Dependency: A functional dependency is a one-way relationship between two attributes (or two sets of attributes) A and B in a relation R such that at any given point of time, for each unique value of attribute A, only one value of attribute B is associated with it.

Normal Forms: Normalization is based on the concept of normal forms. A table is said to be in a particular normal form if it satisfies a certain set of constraints defined for that normal form.

Notes

7.11 Self Assessment

Fill in the blanks:

1. The relational model was proposed by .....
2. When a single constraint is established between two sets of attributes from the database it is called .....
3. The first concept of normalisation was proposed by Mr. Codd in .....
4. ..... is defined using the concept of the join dependencies.
5. Functional dependencies (FD) are type of constraint that is based on .....
6. The weak entity has only a ..... key.
7. The ..... is convenient for representing an initial, high-level database design.
8. An entity set is mapped to a relation in a .....way.
9. A ..... always participates in a one-to-many binary relationship and has a key constraint and total participation.
10. The ..... is trivial since it results in no constraints being placed on the relation.

7.12 Review Questions

1. Explain with examples the problems cause due to redundancy of data in tables or relations.
2. Define functional dependency. Give example.
3. Describe various features of relational database.
4. Describe with examples the techniques used for reducing the redundancy of data in tables.
5. Sketch with examples the differences between the Third Normal Form (3NF) and Boyce Codd Normal Form.
6. Explain why a relation table is subjected to advanced normalizations?
7. Define Multivalued Dependencies. Give Examples. Explain how are they eliminated?
8. What do you mean by join dependencies?
9. Consider the relation R (A, B, C, D, E) with functional dependencies

   $A \rightarrow B,C$

   $CD \rightarrow D$

   $B \rightarrow D$

   $E \rightarrow A$

   List the candidates key for R. Give a lossless dependency preserving decomposition of R. into 3 NF and explain it briefly.
10. Describe the rules of functional dependencies.

Notes

Answers: Self Assessment

1.	E. F. Codd	2.	functional dependency
3.	1972	4.	Project-Join Normal Form
5.	keys	6.	partial
7.	ER model	8.	straightforward
9.	weak entity set	10.	Multi Valued Dependencies

7.13 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 8: Normalization

CONTENTS

Objectives

Introduction

8.1 Normalization

8.2 First Normal Form

8.3 Second Normal Form

8.4 Third Normal Form

8.5 Boyce Codd Normal Form

8.6 Fourth Normal Form

8.7 Fifth Normal Form

8.8 Summary

8.9 Keywords

8.10 Self Assessment

8.11 Review Questions

8.12 Further Readings

Objectives

After studying this unit, you will be able to:

- Define normalization
- Describe first, second, third normal forms
- Knowing Boyce Codd normal form
- Explain fourth and fifth normal forms

Introduction

Normalization is the step-by-step decomposition of complex records into simple records. Through normalization a collection of a data in a record structure is replaced by successive record structures that are simpler and more manageable. Normalization is carried out mainly for four reasons:

1. To structure the data so that any pertinent relationship between entities can be represented.
2. To permit easy retrieval of data in response to query a report requests.
3. To simplify the maintenance of the data through updates, insertions and deletions.
4. To reduce the need to restructure or reorganize data when new application requirements arise.

Notes

8.1 Normalization

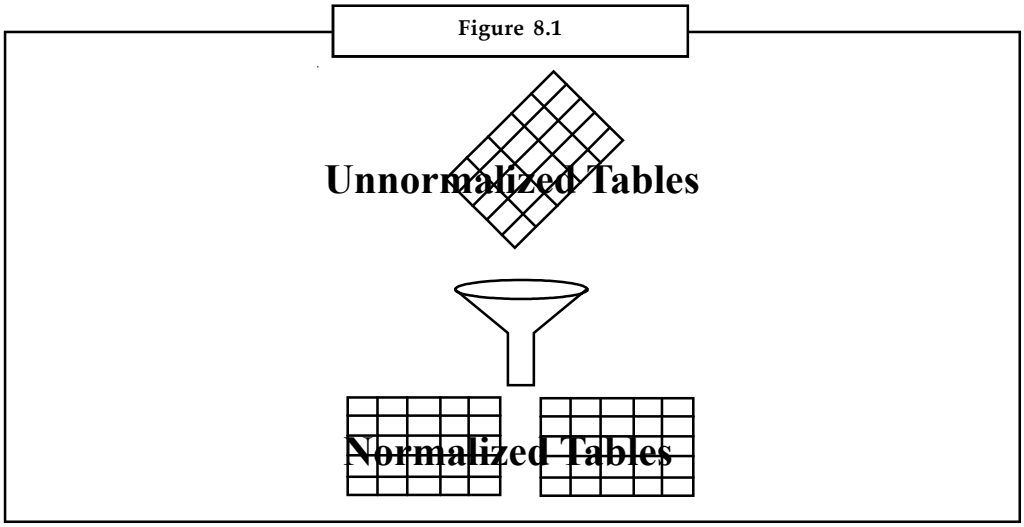
Normalization is typically a refinement process after identifying the entities, identifying their relationships, and defining the tables required in the database and the columns within each table. The starting point for the normalization process can be visualized as the universal relation. This relation contains all the attributes of interest, and is structured so that each tuple in the relation has single-valued element. Normalization results in the formation of tables that satisfy certain specified constraints and represent certain normal forms. Normal forms are table structures with minimum redundancy.

E.F. Codd defined the first, second and third normal forms. A revised version of third normal form is given by Boyce and Codd, which is known as Boyce-Codd normal form. Fagin defined the fifth normal form.

The process of splitting a relation to reduce the probability that anomalies will occur is known as decomposition. The key to going about decomposition is logical and methodical way. Normalization reduces redundancy-using principle of non-loss decomposition, which is the reduction of a table to smaller tables without any loss of information.

Normalization is the process of refining the design of relational tables to minimize data redundancy. Normalization consists of a series of rules that should be employed to make sure that the table is fully normalized by listing the functional dependencies and decomposing it into smaller, efficient tables. Normalization eliminates data maintenance anomalies, minimizes redundancy, and eliminates data inconsistency.

Figure 8.1



Normal Forms

Normalization is based on the concept of normal forms. A table is said to be in a particular normal form if it satisfies a certain set of constraints defined for that normal form.

These constraints usually apply to the attributes of the table and on the relationships among them.

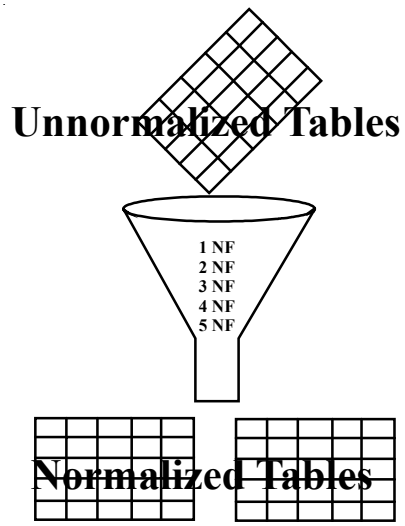
There are multiple levels of normal forms, each addresses a specific factor that could result in reducing data maintenance anomalies.

Notes

The various normal forms are:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. Boyce-Codd Normal Form
5. Fourth Normal Form
6. Fifth Normal Form

Figure 8.2



8.2 First Normal Form

The First Normal Form states that:

There should be no repeating groups in a column. In other words, columns in the table should have atomic values.

Notes **Atomic**

A column is said to be atomic if the values are indivisible units.

A relation R is said to be in the first normal form (1NF) if and only if every attribute contains atomic values only.

The table contains atomic values if there is one and only one data item for any given row and column intersection. Non-atomic values result in repeating groups. A repeating group is the reoccurrence of a data item or group of data items within records.

Notes

Example: Consider an Employee table with an attribute Dependents as shown below:

ID	Name	DeptNo	Sal	Mgr	Dependents
131	Ram	20	10000	134	Father, Mother, Sister
132	Kiran	20	7000	136	Wife, Son
133	Rajesh	20	5000	136	Wife
134	Padma	10	20000		Son, Daughter
135	Devi	30	3000	137	Father, Mother
136	Satish	20	6000		Father, Mother
137	V.V. Rao	30	10000		Wife, First Son, Second Son

Here, the dependents column has non-atomic values. To make the relation in INF, we have to convert the non-atomic values into atomic values as follows:

ID	Name	DeptNo	Sal	Mgr	Dependents
131	Ram	20	10000	134	Father
131	Ram	20	10000	134	Mother
131	Ram	20	10000	134	Sister
132	Kiran	20	7000	136	Wife
132	Kiran	20	7000	136	Son
133	Rajesh	20	5000	136	Wife
134	Padma	10	20000		Son
134	Padma	10	20000		Daughter
135	Devi	30	3000	137	Father
135	Defi	30	3000	137	Mother
136	Satish	20	6000		Father
137	V.V. Rao	30	10000		Wife
137	V.V. Rao	30	10000		First Son
137	V.V. Rao	30	10000		Second Son

The dependents column in the above table is having atomic values. Observe that for each dependent the other employee details such as ID, Name, Dept No, Sal and Mgr are repeated and will form as a repeating group. As per the definition of INF, the above relation is in INF. However, it is best practice to remove the repeating groups in the table.

Repeating groups refers to any set of columns whose values are related and repeat in the table. According to the rule of first normal form, the table should not have repeating groups of column values.

Notes

If there are such groups in the table, the table should be decomposed and the associated columns will form their own table while at the same time ensuring a link of this table with the original table (from where it was decomposed). Thus, the Employee relation can be divided into two relations namely Emp and Emp_Depend as follows:

Emp

ID	Name	DeptNo	Sal	Mgr
131	Ram	20	10000	134
132	Kiran	20	7000	136
133	Rajesh	20	5000	136
134	Padma	10	20000	
135	Devi	30	3000	137
136	Satish	20	6000	
137	V.V. Rao	30	10000	

Emp_Depend

S. No.	ID	Dependents
1	131	Father
2	131	Mother
3	131	Sister
4	132	Wife
5	132	Son
6	133	Wife
7	134	Son
8	134	Daughter
9	135	Father
10	135	Mother
11	136	Father
12	137	Wife
13	137	First Son
14	137	Second Son

Notes The SNo column is added in the Emp_Depend table because according to relational model, every table must have a unique identifier. In this table, ID cannot serve as a unique identifier and hence introduced SNo as primary key.

After first normal form, this table is divided into two tables: Customer and Customer Tran.

Notes

Table before First Normal Form Reports Table

Cust_ id	Name	Address	Acc_ id	Acc_ type	Min_ bal	Tran_ id	Tran_ type	Tan_ mode	Amount	Balance
001	Ravi	Hyd	994	SB	1000	14300		B/F	1000	1000
001	Ravi	Hyd	994	SB	1000	14301	Deposit	Bycash	1000	2000
001	Ravi	Hyd	994	SB	1000	14302	Withdrawal	ATM	500	1500
110	Tim	Sec'bad	340	CA	500	14303		B/F	3500	3500
110	Tim	Sec 'bad	340	CA	500	14304	Deposit	Payroll	3500	7000
110	Tim	Sec'bad	340	CA	500	14305	Withdrawal	ATM	1000	6000
420	Kavi	Vizag	699	SB	1000	14306		B/F	6000	6000
420	Kavi	Vizag	699	SB	1000	14307	Credit	Bycash	2000	8000
420	Kavi	Vizag	699	SB	1000	14308	Withdrawal	ATM	6500	1500

Tables after First Normal Form

Customer Table

Cust_id	Name	Address
001	Ravi	Hyd
110	Tim	Sec'bad
420	Kavi	Vizag

Customer_Tran Table

Tran_id	Cust_id	Acc_id	Acc_type	Min_bal	Tran_type	Tan_mode	Amount	Balance
14300	001	994	SB	1000		B/F	1000	1000
14301	001	994	SB	1000	Deposit	Bycash	1000	2000
14302	001	994	SB	1000	Withdrawal	ATM	500	1500
14303	110	340	CA	500		B/F	3500	3500
14304	110	340	CA	500	Deposit	Payroll	3500	7000
14305	110	340	CA	500	Withdrawal	ATM	1000	6000
14306	420	699	SB	1000		B/F	6000	6000
14307	420	699	SB	1000	Credit	Bycash	2000	8000
14308	420	699	SB	1000	Withdrawal	ATM	6500	1500

Since Cust_id, Name and Address form a repeating group and hence the Reports table is decomposed into Customer and Customer_Tran tables. (The primary key columns of each table are indicated in bold in figures).

The requirements for a table to be INF is exactly what relational database theory specifies as essential. That is, a table in a relational database must always be in INF.

8.3 Second Normal Form

A table in the first normal form may show redundancies due to partial dependencies. The Second Normal Form resolves partial dependencies.

The Second Normal Form states that

1. The table should be in 1st Normal form
2. Every non-key column must be fully functional dependent on the Primary key

Notes

Notes **Partially Dependent**

An attribute (column) is said to be partially dependent if its value can be determined by any one or more attributes of the primary key, but not all.

Each normal form is built upon the previous normal form. The first statement states that the prerequisite for second normal form is to have all its tables in first normal form.

The Fully Functional Dependency is for a given composite primary key (a primary key consisting of more than one attribute), each column attribute, which is not an attribute of the Primary key, should be dependent on each and every one of the primary key attributes.

If attributes dependent on only a part of the primary key attribute exist, they are removed and placed in a separate table where the primary key of the new table is the portion of the original key that they were dependent on.

A non-key attribute is fully functionally dependent on the primary key if it is functionally dependent on all the attributes comprising the primary key.

A relation R is said to be in the second normal form (2NF) if and only if it is in INF and every non-key attribute must be fully functional dependent on the Primary key.

The Customer Tran table is decomposed into three tables: Transaction, Customer Account and Accounts.

Tables before Second Normal Form

Customer Table

Cust_id	Name	Address
001	Ravi	Hyd
110	Tim	Sec 'bad
420	Kavi	Vizag

Customer_Tran Table

Tran_id	Cus_id	Acc_id	Acc_type	Min_bal	Tran_type	Tan_mode	Amount	Balance
14300	001	994	SB	1000		B/F	1000	1000
14301	001	994	SB	1000	Deposit	Bycash	1000	2000
14302	001	994	SB	1000	Withdrawal	ATM	500	1500
14303	110	340	CA	500		B/F	3500	3500
14304	110	340	CA	500	Deposit	Payroll	3500	7000
14305	110	340	CA	500	Withdrawal	ATM	1000	6000
14306	420	699	SB	1000		B/F	6000	6000
14307	420	699	SB	1000	Credit	Bycash	2000	8000
14308	420	699	SB	1000	Withdrawal	ATM	6500	1500

Notes

Tables after Second Normal Form

Customer Table

CusMd	Name	Address
001	Ravi	Hyd
110	Tim	Sec' bad
420	Kavi	Vizag

Customer_Accounts Table

Cust.id	Acc_id	Balance
001	994	1500
110	340	6000
420	699	1500

Accounts Table

Acc_id	Accjype	Min.bal
994	SB	1000
340	CA	500
699	SB	1000

Transaction Table

Tran_id	Acc_id	Tran_type	Tan_mode	Amount
14300	994		B/F	1000
14301	994	Deposit	Bycash	1000
14302	994	Withdrawal	ATM	500
14303	340		B/F	3500
14304	340	Deposit	Payroll	3500
14305	340	Withdrawal	ATM	1000
14306	699		B/F	6000
14307	699	Credit	Bycash	2000
14308	699	Withdrawal	ATM	6500

Acc_type and Min_bal columns of Customer_Account table are not fully functionally dependent on the primary key (dependent on acc_id), a new Accounts table is formed. Similarly, the Balance is dependent on Cust_id and Acc_id, but not fully functionally dependent on the primary key, resulting in a new Customer_Accounts table.

Task

FNF vs. SNF (First Normal Form vs. Second Normal Form)

Notes

8.4 Third Normal Form

A table the Second Normal form may show redundancies due to transitive dependencies.

The Third normal form resolves transitive dependencies. A transitive dependency arises when a non-key column is functionally dependent on another non-key column that in turn is functionally dependent on the primary key.

The Third Normal Form states that:

1. The table should be in 2nd Normal Form
2. The table should be devoid of transitive dependencies

Notes

Transitive Dependencies

Columns dependent on other columns that in turn are dependent on the primary key are said to be transitively dependent.

Foreign Key: A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data.

The first rule states that a prerequisite for Third normal form is that the table first satisfies the rules of the second normal form and in turn the first normal form.

A relation R is said to be in the third normal form (3NF) if and only if it is in 2NF and every non-key attribute must be non-transitively dependent on the Primary key.

3NF ensures that none of the non-key attributes are dependent upon another attribute which in turn is dependent on the primary key. Table shows the tables before and after third normal form. The Accounts table in the second normal form has a transitive dependency as follows:

Acc_id $\rightarrow$ Acc_type

Acc_type $\rightarrow$ Min_bal

This transitive dependency is resolved by decomposing the Accounts table into two tables: Acc_Detail and Product.

Tables before Third Normal Form

Customer_Accounts Table

Cust_id	Name	Address
001	Ravi	Hyd
110	Tim	Sec 'bad
420	Kavi	Vizag

Customer Table

Cust.id	Acc_id	Balance
001	994	1500
110	340	6000
420	699	1500

Notes

Accounts Table

Acc_id	Acc_type	Min.bal
994	SB	1000
340	CA	500
699	SB	1000

Transaction Table

Tran_.id	Acc_id	Tran_type	Tan_mode	Amount
14300	994		B/F	1000
14301	994	Deposit	Bycash	1000
14302	994	Withdrawal	ATM	500
14303	340		B/F	3500
14304	340	Deposit	Payroll	3500
14305	340	Withdrawal	ATM	1000
14306	699		B/F	6000
14307	699	Credit	Bycash	2000
14308	699	Withdrawal	ATM	6500

Tables after Third Normal Form

Customer Table

Cust_id	Name	Address
001	Ravi	Hyd
110	Tim	Sec'bad
420	Kavi	Vizag

Customer Accounts Table

Cust_id	Acc_id	Balance
001	994	1500
110	340	6000
420	699	1500

Acc_Detail Table

Acc_id	Acc_type
994	SB
340	CA
699	SB

Product Table

Acc_type	Min_bal
SB	1000
CA	500

Notes

Transaction Table

Tran_id	Acc_id	Tran_type	Tan_mode	Amount
14300	994		B/F	1000
14301	994	Deposit	Bycash	1000
14302	994	Withdrawal	ATM	500
14303	340		B/F	3500
14304	340	Deposit	Payroll	3500
14305	340	Withdrawal	ATM	1000
14306	699		B/F	6000
14307	699	Credit	Bycash	2000
14308	699	Withdrawal	ATM	6500

8.5 Boyce Codd Normal Form

The Boyce Codd Normal form is implemented in cases where the table has:

1. Multiple candidate keys
2. Composite candidate keys
3. Candidate keys that overlap Boyce-Codd normal form states that: Every functional dependency should be based on a candidate key for the table.

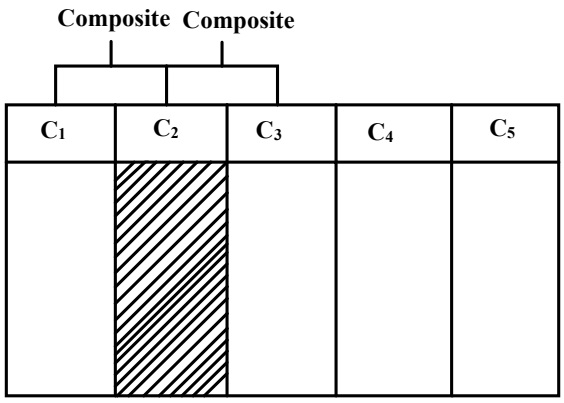
The rule states that, once all the functional dependencies in a table are listed, all columns can be dependent only on a candidate key of the table. In other words, in a functional dependency, the left hand side of the functional dependency should be a candidate key. If it is not, the table should be decomposed placing these functional dependencies along with a copy of their determinant in a new table.

If $A \rightarrow B$ is a functional dependency, and if B is not functionally dependent upon any subset of A, then A is said to be the determinant of B.

This definition says that a determinant is a column on which some of the columns are fully functional dependent.

BCNF – Addressing Overlapping Candidate Keys

Figure 8.3: Overlapping Candidate Keys



Notes

Table before Boyce Codd Normal Form

Loan Info Table

Branch Name	Customer Name	Loan Name	Amount
Delhi	Ajay	L-101	10,000
Vizag	Ram	L-230	15,000
Delhi	Sonali	L-101	10,000
Mumbai	Jagadish	L-420	1,00,000
Trivandum	Usha	L-500	5,000
Mysore	Pradeep	L-784	12,000

Table after Boyce Codd Normal Form

Borrower Table

Customer Name	Loan Name
Ajay	L-101
Ram	L-230
Sonali	L-101
Jagadish	L-420
Usha	L-500
Pradeep	L-784

Loan Table

Branch Name	Loan Name	Amount
Delhi	L-101	10,000
Vizag	L-230	15,000
Delhi	L-101	10,000
Mumbai	L-420	1,00,000
Trivandum	L-500	5,000
Mysore	L-784	12,000

3NF – BCNF Comparison

When a relation has more than one candidate key, data maintenance anomalies may result even though that relation is in 3NF.

Example: Consider the Student_Advisor relation as shown below:

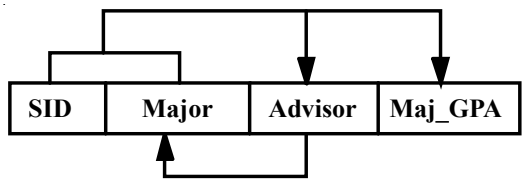
Relation in 3NF but not BCNF STUDENT ADVISOR

Notes

SID	Major	Advisor	Maj_GPA
123	Physics	Satish	4.0
123	English	Mahesh	3.3
456	Literature	Krishna	3.2
789	English	Praveen	3.7
123	Physics	Satish	3.5

Functional Dependencies in STUDENT_ADVISOR

Figure 8.4



This relation has the following attributes: SID (Student ID), Major, Advisor, and Maj_GPA. The primary key for this relation is the composite key consisting of SID and Major. Thus the two attributes Advisor and Maj_GPA are functionally dependent on this key. This reflects the constraint that although a given student may have more than one major, for each major a student has exactly one advisor and one GPA.

There is a second functional dependency in this relation. Major is functionally dependent on Advisor. That is, each Advisor advises in exactly one Major. Notice that this is not a transitive dependency. Rather, a key attribute (Major) is functionally dependent on a non-key attribute (Advisor).

Anomalies in Student_Advisor Relation

The Student_Advisor relation is clearly in 3NF, since there are no partial dependencies and no transitive dependencies. Nevertheless, because of the functional dependency between Major and Advisor, there are anomalies in this relationship. Consider the following scenarios:

Suppose that in Physics the advisor Satish is replaced by Rajesh. This change must be made in two rows in the table (update anomaly).

Suppose to insert a row with the information that Rajeev advises in Computer Science. This can be done only when at least one student majoring in Computer Science (may or may not be the case) is assigned Rajeev as an advisor (insert anomaly).

If student number 789 withdraws from school, there is a loss of information that Praveen advises in English (delete anomaly).

The anomalies in Student_Advisor result from the fact that there is a determinant (Advisor) that is not a candidate key in the relation. R.F. Boyce and E.F. Codd identified this deficiency and proposed a stronger definition of 3NF that remedies the problem.

Definition of Boyce-Codd Normal Form

A relation is in Boyce-Codd Normal Form (BCNF) if and only if every determinant in the relation is a candidate key.

Notes

Student_Advisor is not in BCNF because although the attribute Advisor is a determinant, it is not a candidate key (rather Major is functionally dependent on Advisor).

Converting a Relation to BCNF

A relation that is in 3NF (but not BCNF) can be converted to relations in BCNF as follows:

In the first step, the relation is modified so that the determinant in the relation that is not a candidate key becomes a component of the primary key of the revised relation. The attribute that is functionally dependent on that determinant becomes a non-key attribute. This is a legitimate restructuring of the original relation because of the functional dependency.

The result of applying this to Student_Advisor is given below:

The determinant Advisor becomes part of the composite primary key. The attribute Major (that was a part of the composite key in the old Student_Advisor relation) gets partitioned into a separate relation and becomes a non-key attribute in the new relation that is functionally dependent on the primary key of the new relation (Advisor).

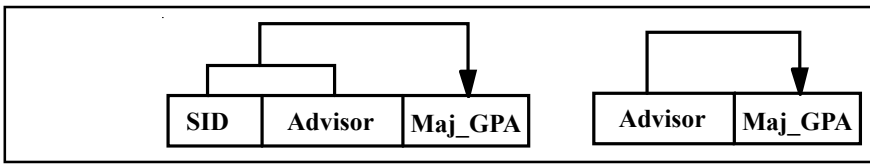
Therefore, these two relations are not only in 3NF, but also in BCNF, since there is only one candidate key (the primary key) in each relation.

Student

SID	Advisor	Maj_GPA
123	Satish	4.0
123	Mahesh	3.3
456	Krishna	3.2
789	Praveen	3.7
123	Satish	3.5

Advisor

Advisor	Major
Satish	Physics
Mahesh	English
Krishna	Literature
Praveen	English
Satish	Physics



The Comparison of 3NF and BCNF can therefore be summarized as follows:

If a relation has only one candidate key (which therefore becomes the primary key), then 3NF and BCNF are equivalent.

The two resulting relations Student and Advisor with sample data are shown. Then these are free of data maintenance anomalies.

Notes

Suppose that in Physics the advisor Satish is replaced by Rajesh. This change has to be made in only one row in the Advisor table (solving the update anomaly).

Suppose to insert a row with the information that Rajeev advises in Computer Science. This can be done conveniently by adding a row to the effect in Advisor table without necessarily associating it to a student, as was the case in the Student_Advisor table (resolving the insert anomaly).

If student number 789 withdraws from school, hence, delete the corresponding row from the Student relation without losing the information that Praveen advises in English. This may recall that this was the case in the older Student_Advisor relation, (resolving the delete anomaly).

8.6 Fourth Normal Form

The Fourth Normal form applies to tables with a special characteristic that does not arise often. This characteristic is called multi-valued dependency. A Multivalued dependency occurs when in a relational table containing at least three columns, one column has multiple rows whose values match a value of a single row of one of the other columns.

A multi valued dependency exists if, for each value of an attribute A, there exists a finite set of values of attribute B that are associated with A and a finite set of values of attribute C that are also associated with A. Attributes B and C are independent of each other.

4NF – Addressing Multi-valued Dependencies

BranchStaffClient

BranchNumber	StaffName	ClientName	ClientRequirement
B-41	Radha	Surya	A
B-41	Radha	Ravi	B
B-41	Smitha	Surya	B
B-41	Smitha	Ravi	A

Consider a table called Branch_Staff_Client, which depicts the various clients for a bank branch, the various staff who address the client's needs and each client's various requirements.

It is seen that each client's name gives rise to a list of staff names. In other words, the multiple staff names are dependent on a single client name. Similarly, each client's multiple requirements are dependent on a single clients name.

Therefore, Client name determines Staff name and Client name determines client requirements. But staff name and client requirements are independent of each other. This gives rise to the phenomenon of **multi-valued dependencies** that can be symbolically represented as:

Clientname $\rightarrow$ $\rightarrow$ StaffName

Clientname $\rightarrow$ $\rightarrow$ ClientRequirements

The Fourth Normal form states that:

The table should be in Boyce-Codd normal form and there should be no multi-valued dependencies.

The rule of 4NF states that the presence of multi-valued dependencies in the table should be eliminated. If a table has multi-valued dependencies, decompose the table and move the related

Notes

columns into separate tables along with a copy of the determinant. This copy serves as a foreign key to the original table.

Table before Fourth Normal Form

Branch_Staff Table

BranchNumber	StaffName	ClientName
B-41	Radha	Surya
B-41	Radha	Ravi
B-41	Smitha	Surya
B-41	Smitha	Ravi

Table after Fourth Normal Form

Branchjitaff Table

BranchNumber	ClientName	BranchNumber	StaffName
B-41	Surya	B-41	Radha
B-41	Ravi	B-41	Smitha

Task BCNF differ from 4th NF. Explain.

8.7 Fifth Normal Form

The Fifth Normal form is usually applied only for large relational data models. The Fifth Normal form states that the original table must be reconstructed from the tables into which it has been decomposed. 5NF allows decomposing a relation into three or more relations. Fagin defined the 5NF and according to Fagin's theorem, a relation (A, B, C) can be decomposed lossless in (A, B) and (A, C) if and only if $A \rightarrow \rightarrow B \mid C$ is true in (A, B, C).

The fifth normal form is based on the concept of join dependency. Join dependency means that a table, after it has been decomposed into three or more smaller tables, must be capable of being joined again on common keys to form the original table. The join dependency is more general form of multi-valued dependency.

A relation R satisfies join dependency (R_1, R_2…. R_n) if and only if R is equal to the join of R_1, R_2…. R_n where R_i are subsets of the set of attributes of R.

A relation R is in 5NF (or project-join normal form, PJNF) if for all join dependencies at least one of the following holds.

1. (R_1, R_2…. R_n) is a trivial join-dependency (that is, one of R_t is R)
2. Every R_i is a candidate key for R.

Notes

Definition of Fifth Normal Form

A table should be in fifth normal form (5NF) if and only if every join dependency in the table is implied by the candidate keys of the table.

Table before Fifth Normal Form

Dept.	Subject	Student
Comp. Sc.	CP1000	John Smith
Mathematics	MA1000	John Smith
Comp. Sc.	CP2000	Arun Kumar
Comp. Sc.	CP3000	Reena Rani
Physics	PHI 000	Raymond Chew
Chemistry	CH2000	Albert Garcia

Table after Fifth Normal Form

Dept-Subject

Dept.	Subject	Student
Comp. Sc.	CP1000	John Smith
Mathematics	MA1000	John Smith
Comp. Sc.	CP2000	Arun Kumar
Comp. Sc.	CP3000	Reena Rani
Physics	PHI 000	Raymond Chew
Chemistry	CH2000	Albert Garcia

Subject-Student

Subject	Student
CP1000	John Smith
MA 1000	John Smith
CP2000	Arun Kumar
CP3000	Reena Rani
PHI 000	Raymond Chew
CH2000	Albert Garcia

Dept-Student

Dept.	Student
Comp. Sc.	John Smith
Mathematics	John Smith
Comp. Sc.	Arun Kumar
Comp. Sc.	Reena Rani
Physics	Raymond Chew
Chemistry	Albert Garcia

The fifth normal form is also called Project-Join Normal Form (PJNF).

Notes

Case Study

Data Warehousing: A Strategic Tool

Data warehouse is a massive independent business database system that is populated with data that has been extracted from a range of sources. The data is held separately from its origin and is used to help to improve the decision-making process.

Many traditional Databases are involved in recording day to day operational activities of the business, called online transaction processing (OLTP), commonly implemented in Airline Bookings & Banking Systems, for fasters response and better control over data.

After establishment of OLTP Systems, reports and summaries can be drawn for giving inputs to decision-making process and this process is called online analytical processing (OLAP).

For better customer relationships management strategy, the call centres and data warehouses must be fully integrated. Data warehouse works as a strategic tool for decision-support which requires lot of time for establishment, and needs to be updated with operational information on daily weekly or monthly basis.

Data warehouse is used for proactive strategies formulation in critical & complex situations. A number of CRM vendors are advocating for single integrated customer database which includes call centre, web sites, branches and direct mail, but it lacks in analytical functioning of data warehouse. This Database can't be expanded also, and carry decision support operations on call centre Database becomes slow & the query processing and inquiries handling operations also become slow & inefficient for agents dealing with customers.

Data warehouse is must for identifying most profitable & loyal customers and those customers can be offered better customized services which increases the chances of additional profits.

Although call centre systems & data warehouses are altogether different systems yet dependant on each other to fully exploit their potential respectively.

Questions

1. Explain OLTP & OLAP processes with their respective advantages.
2. How the response time in performing OLAP queries can be improved?
3. Explain the importance of regular updation of data in a data warehouse.
4. Explain the role of data warehousing in the functioning of a call centre.
5. "Data warehouse works as a strategic tool for decision support". Comment.

Source: Dharmenda and Sangeeta Gupta, *Management Information System*

8.8 Summary

- The rules are provided by the normalisation where we decompose the databases to the minimum level so that there should not be any problems such as concurrency within the database.
- You can normalise the databases using various forms of normalisation such as 1st NF, 2nd NF, 3rd NF BCNF and 4NF, 5NF.

Notes

8.9 Keywords

Boyce-Codd Normal Form: A relation is in Boyce-Codd Normal Form (BCNF) if and only if every determinant in the relation is a candidate key.

Non Key Attribute: A non-key attribute is fully functionally dependent on the primary key if it is functionally dependent on all the attributes comprising the primary key.

Normalization: Normalization consists of a series of rules that should be employed to make sure that the table is fully normalized by listing the functional dependencies and decomposing it into smaller, efficient tables.

Transitive Dependency: A transitive dependency arises when a non-key column is functionally dependent on another non-key column that in turn is functionally dependent on the primary key.

8.10 Self Assessment

Fill in the blanks:

1. The fifth normal form deals with join-dependencies, which is a generalisation of the .....
2. Normalization is the process of refining the design of relational tables to minimize data .....
3. ..... is based on the concept of normal forms.
4. The Third normal form resolves ..... dependencies.
5. A ..... arises when a non-key column is functionally dependent on another non-key column that in turn is functionally dependent on the primary key.
6. ..... provide a method for maintaining integrity in the data.
7. A ..... dependency occurs when in a relational table containing at least three columns.
8. The ..... form is usually applied only for large relational data models.
9. Normal forms are table structures with .....
10. Normalization eliminates data maintenance anomalies, minimizes redundancy, and eliminates .....

8.11 Review Questions

1. Sketch with examples the differences between the Third Normal Form (3NF) and Boyce Codd Normal Form.
2. Explain the disadvantage of normalization.
3. Give a relation which is in 2 NF but not in 3 NF.
4. "Normalization is the process of refining the design of relational tables to minimize data redundancy". Explain
5. "A relation R is said to be in the first normal form (1NF) if and only if every attribute contains atomic values only". Discuss with the help of suitable example.

Notes

6. From the following given table.

Pet Id	Pet Name	Pet Type	Pet Age	Owner	Visit Date	Procedure
246	Rover	Dog	12	Sam Cook	Jan 13/2002	01 - Rabies Vaccination
					Mar 27/2002	10 - Examine And Treat Wound
					Apr 02/2002	05 - Heart Worm Test
298	Spot	Dog	2	Terry Kim	Jan 21/2002	08 - Tetanus Vaccination
					Mar 10/2002	05 - Heart Worm Test
341	Morris	Cat	4	Sam Cook	Jan 23/2001	01 - Rabies Vaccination
					Jan 13/2002	01 - Rabies Vaccination
519	Tweedy	Bird	2	Terry Kim	Apr 30/2002	20 - Annual Check Up
					Apr 30/2002	12 - Eye Wash

Find:

UNF:

1NF:

2NF:

3NF:

7. Give the current Normal Form and Normalize to 3rd if necessary.

(a) Property (**PropertyID**, Address, City, State, Zip, Description, Price, {Type, Tdescription}, *OfficeNbr*)

Type → Tdescription

(b) PC (**PC#**, Location)

PCAccess (**PC#**, **EmpID**, AccessType)

MgrPCAccess (**PC#**, **EmpID**, **OfficeNbr**, AccessType)

PC# is only unique within an office.

(c) OfficeEmp (**OfficeNbr**, Oaddress, **EmpID**, Ename, Eaddress)

OfficeNbr → Oaddress

EmpID → Ename, Eaddress

(c) Registration (**Course#**, CourseName, **Student#**, StudentName, Grade, Major)

(d) Sale (**Order#**, Date, Clerk#, ClerkName)

(e) Student (**StudentID**, Name, Address, Major, Minor)

Only 1 major and minor per student

(f) Student (**StudentID**, Name, Address, {Major}, {Minor})

Multiple majors and minors per student

Notes

(g) Owner (**OwnerSSN**, Name, Address, City, State, Zip)

PctOwned (**PropertyID**, **OwnerSSN**, PctOwned)

Property (**PropertyID**, Address, City, State, Zip, Description, Price, OfficeNbr)

(h) PCAccess (**PC#**, **EmpID**, AccessType, AccessDescription)

Answers: Self Assessment

1.	MVD	2.	redundancy
3.	Normalization	4.	transitive
5.	transitive dependency	6.	Foreign keys
7.	Multivalued	8.	Fifth Normal
9.	minimum redundancy	10.	data inconsistency

8.12 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management*, 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 9: Transaction Management

CONTENTS

Objectives

Introduction

9.1 Concept of a Transaction

9.2 Transaction State

9.3 Implementation of Atomicity and Durability

9.4 Concurrent Execution

9.5 Serializability

9.6 Recoverability

9.7 Implementation of Isolation

9.8 Testing for Serializability

9.9 Lock based Protocols

9.9.1 Two-phase Locking (2 PL) Protocol

9.9.2 Strict Two-phase Locking (Strict 2PL) Protocol

9.10 Timestamp based Protocol

9.11 Validation Based Protocols

9.12 Deadlock Handling

9.12.1 Deadlock Prevention

9.12.2 Deadlock Recovery

9.13 Insert and Delete Operation

9.14 Weak Level of Consistency

9.14.1 Degree-two Consistency

9.14.2 Weak Levels of Consistency in SQL

9.15 Summary

9.16 Keywords

9.17 Self Assessment

9.18 Review Questions

9.19 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- State the transaction concept
- Explain the idea of serializability and recoverability
- Discuss lock based protocols and deadlock handling
- Define weak level of consistency

Introduction

The concept of transaction is the foundation for concurrent execution of transactions in a DBMS and recovery from system failure in a DBMS. A user writes data access/updates programs in terms of the high level query language supported by the DBMS. To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program or transaction as a series of read and write of database objects.

To read a database object, it is first brought into main memory (in some frame of the buffer pool) from disk, and then its value is copied into a program. This is done by read operation.

To write a database object, in-memory copy of the object is first modified and then written to disk. This all is done by write operation.

Usually, Database objects are the units in which program reads or writes information. The units could be pages, records and so on, but this is dependents on the DBMS. Therefore, we can also consider a database to be "a fixed collection of independent objects."

A transaction is an execution of a user program and is seen by the DBMS as a series or list of actions. The actions that can be executed by a transaction includes the reading and writing of database.

A transaction is a unit of program execution that accesses and possibly updates various data items.

9.1 Concept of a Transaction

A user writes data access/update programs in terms of the high-level query and update language supported by the DBMS. To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program, or transaction, as a series of reads and writes of database objects:

1. To read a database object, it is first brought into main memory (specifically, some frame in the buffer pool) from disk, and then its value is copied into a program variable.
2. To write a database object, an in-memory copy of the object is first modified and then written to disk.

Database 'objects' are the units in which programs read or write information. The units could be pages, records, and so on, but this is dependent on the DBMS and is not central to the principles underlying concurrency control or recovery. In this unit, we will consider a database to be a fixed collection of independent objects. When objects are added to or deleted from a database, or there are relationships between database objects that we want to exploit for performance, some additional issues arise.

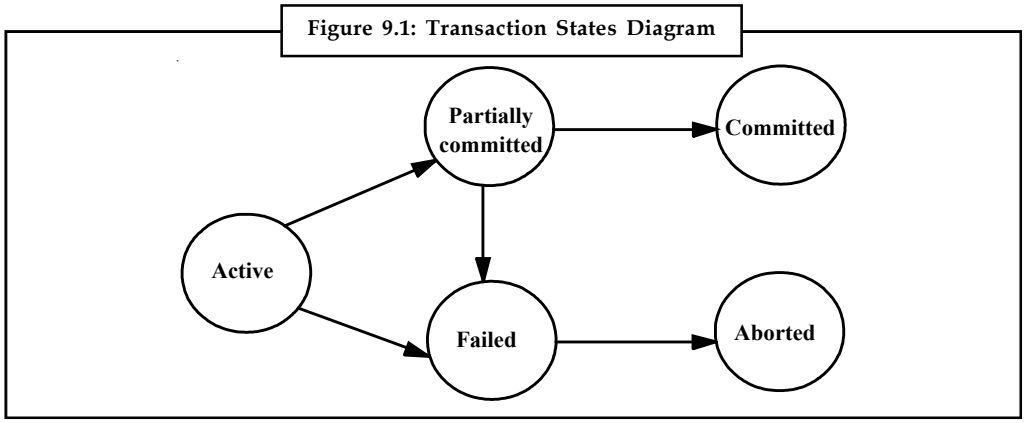
Notes

There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures:

1. Users should be able to regard the execution of each transaction as **atomic**: either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (say, when a system crash occurs).
2. Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency**, and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.
3. Users should be able to understand a transaction without considering the effect of other concurrently executing transactions, even if the DBMS interleaves the actions of several transactions for performance reasons. This property is sometimes referred to as **isolation**: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
4. Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called **durability**.

9.2 Transaction State

Figure 9.1: Transaction States Diagram



A transaction must be in one of the following states:

1. ***Active State:*** This is the initial state of a transaction, the transaction stays in this state while it is starting execution.
2. ***Partially Committed State:*** This transaction state occurs after the final (last) statement of the transaction has been executed.
3. ***Failed State:*** This transaction state occurs after the discovery that normal execution can no longer proceed.
4. ***Aborted State:*** This transaction state occurs after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
5. ***Committed State:*** This transaction state occurs after the successful completion of the transaction. The transaction states diagram corresponding to a transaction is shown above.

A transaction starts in the active state. When it finishes its final (last) statement, it enters the partially committed state. At this point, the transaction has completed its execution, but there is

Notes

still a possibility that it may be aborted, since the actual output may still, be temporarily residing in main memory, and thus a hardware failure may stop its successful completion.

When the database system writes out successfully the output information to disk, then the transaction enters the committed state.

A transaction may also enters the failed state from the active state or from the partially committed state after the system determines that the transaction can no longer proceed with its normal execution because of hardware failure or logical errors. Such a transaction must be rolled back and the database has been restored to its state prior to the start of the transaction. Then, it is known as the aborted state. At this state, the system has two options as follows:

1. ***Restart the Transaction:*** It can restart the transaction, but only if the transaction was aborted as a result of some hardware failure or software error. A restarted transaction is considered to be a new transaction.
2. ***Kill the Transaction:*** It can kill the transaction because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad.

Task When a transaction enter into the failed state?

9.3 Implementation of Atomicity and Durability

Transactions can be incomplete for three kinds of reasons. First, a transaction can be **aborted**, or terminated unsuccessfully, by the DBMS because some anomaly arises during execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed anew. Second, the system may crash (e.g., because the power supply is interrupted) while one or more transactions are in progress. Third, a transaction may encounter an unexpected situation (for example, read an unexpected data value or be unable to access some disk) and decide to abort (i.e., terminate itself).

Of course, since users think of transactions as being atomic, a transaction that is interrupted in the middle may leave the database in an inconsistent state. Thus a DBMS must find a way to remove the effects of partial transactions from the database, that is, it must ensure transaction atomicity: either all of a transaction's actions are carried out, or none are. A DBMS ensures transaction atomicity by *undoing* the actions of incomplete transactions. This means that users can ignore incomplete transactions in thinking about how the database is modified by transactions over time. To be able to do this, the DBMS maintains a record, called the *log*, of all writes to the database. The log is also used to ensure durability: If the system crashes before the changes made by a completed transaction are written to disk, the log is used to remember and restore these changes when the system restarts.

The DBMS component that ensures atomicity and durability is called the recovery manager.

9.4 Concurrent Execution

Transaction-processing systems usually allow multiple transactions to run concurrently (at the same time) known as concurrent execution of transactions. The DBMS interleaves (inter changes) the actions of different transactions to improve performance, but not all interleavings should be allowed. Because allowing multiple transactions 16 update data concurrently causes several good reasons as well as several complications with the consistency of the database, which are illustrated as follows:

Notes

Motivation for Concurrent Execution

The schedule, involving two transactions shown in the following figure represents an interleaved execution of the two transactions.

T_1	T_2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

First, while one transaction is waiting for a page to be read in from disk, the CPU can process another transaction This is because I/O activity can be done in parallel with CPU activity in a computer. Overlapping I/O and CPU activity reduces the amount of time and increases system throughput which is the average number of transactions completed in a given time.

Second, interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short transaction could get stuck behind a long transaction, leading to unpredictable delays in response, time, or average time taken to complete a transaction.

9.5 Serializability

A serializable schedule over a set S of committed transactions is a schedule whose effect on any consistent database is guaranteed to be identical to that of some complete serial schedule over S. That is, even-though the actions of transactions are interleaved, the result of executing transactions serially in different orders may produce different results.

9.6 Recoverability

Unfortunately, the timestamp protocol presented above permits schedules that are not recoverable, as illustrated by the schedule in Table below. If $TS(T_1) = 1$ and $TS(T_2) = 2$, this schedule is permitted by the timestamp protocol (with or without the Thomas Write Rule). The timestamp protocol can be modified to disallow such schedules by buffering all write actions until the transaction commits. In the example, when T_1 wants to write A, WTS(A) is updated to reflect this action, but the change to A is not carried out immediately; instead, it is recorded in a private workspace, or buffer. When T_2 wants to read A subsequently, its timestamp is compared with WTS(A), and the read is seen to be permissible. However, T_2 is blocked until T_1 completes. If T_1 commits, its change to A is copied from the buffer; otherwise, the changes in the buffer are discarded. T_2 is then allowed to read A.

T_1	T_2
W(A)	
	R(A)
	W(B)
	Commit

This blocking of T_2 is similar to the effect of T_1 obtaining an exclusive lock on *A*! Nonetheless, even with this modification the timestamp protocol permits some schedules that are not permitted by 2PL; the two protocols are not quite the same.

Notes

Because recoverability is essential, such a modification must be used for the timestamp protocol to be practical. Given the added overheads this entails, on top of the (considerable) cost of maintaining read and write timestamps, timestamp concurrency control is unlikely to beat lock-based protocols in centralized systems. Indeed, it has mainly been studied in the context of distributed database systems.

Task If any transaction encounter unexpected situation then transaction complete of incomplete.

9.7 Implementation of Isolation

Users are responsible for ensuring transaction consistency. That is, the user who submits a transaction must ensure that when run to completion by itself against a 'consistent' database instance, the transaction will leave the database in a 'consistent' state. For example, the user may (naturally!) have the consistency criterion that fund transfers between bank accounts should not change the total amount of money in the accounts. To transfer money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent in a global sense, even though the new account balance may satisfy any integrity constraints with respect to the range of acceptable account balances. The user's notion of a consistent database is preserved when the second account is credited with the transferred amount. If a faulty transfer program always credits the second account with one dollar less than the amount debited from the first account, the DBMS cannot be expected to detect inconsistencies due to such errors in the user program's logic.

The isolation property is ensured by guaranteeing that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions one after the other in some serial order. For example, if two transactions T_1 and T_2 are executed concurrently, the net effect is guaranteed to be equivalent to executing (all of) T_1 followed by executing T_2 or executing T_2 followed by executing T_1. (The DBMS provides no guarantees about which of these orders is effectively chosen.) If each transaction maps a consistent database instance to another consistent database instance, executing several transactions one after the other (on a consistent initial database instance) will also result in a consistent final database instance.

Database consistency is the property that every transaction sees a consistent database instance. Database consistency follows from transaction atomicity, isolation, and transaction consistency.

9.8 Testing for Serializability

Example: The schedule shown in the following figure is serializable.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	Commit
Commit	

Notes

A Serializable Schedule

Even though the actions of equivalent to first running T_1 and T_2 are interleaved, the result of this schedule is entirely and then running T, entirely. Actually, Ts read and write of B is not influenced by T_2's actions on B, and the net effect is the same if these actions are the serial schedule First T_1, then T_2. This schedule is also serializable schedule if first T_2, then T_1. Therefore, if T, and T_2 are submitted concurrently to a DBMS, either of these two schedules could be chosen as first.

A DBMS might sometimes execute transactions which is not a serial execution i.e., not serializable.

This can happen for two reasons as follows:

1. The DBMS might use a concurrency control method that ensures the executed schedule itself.
2. SQL gives programmers the authority to instruct the DBMS to choose non-serializable schedule.

9.9 Lock based Protocols

A DBMS must be able to ensure that only serializable, recoverable, schedules are allowed, and that no actions of committed transactions are lost, while undoing aborted transactions. So, a DBMS uses a locking protocol to achieve this.

Locking Protocol: A locking protocol is a set of rules to be followed by each transaction to ensure that, even-though actions of several transactions might be interleaved, the net effect is same as executing all transactions in some serial order.

9.9.1 Two-phase Locking (2 PL) Protocol

The two-phase locking protocol ensures serializability. This protocol requires that each transaction issues lock and unlock requests in two phases as follows:

1. ***Growing Phase:*** A transaction may obtain locks but may not release any lock.
2. ***Shrinking Phase:*** A transaction may release locks, but may not obtain any new locks.

Initially, a transaction is in the growing phase. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests. Thus, the transaction acquires lock and releases lock as needed. A modification of two-phase locking is called the Strict Two-Phase Locking Protocol.

9.9.2 Strict Two-phase Locking (Strict 2PL) Protocol

The strict two-phase locking protocol, requires, that, in addition to locking being two-phase, all exclusive-mode lock. (Exclusive lock is the mode of locking to provide exclusive use of the data - item to one and only one transaction) taken by a transaction, must be held until that transaction commits successfully, preventing any other transaction from reading the data.

Thus, the most widely used locking protocol is the Strict Two-Phase Locking Protocol, also called as Strict 2PL, has following two rules:

RULE 1: If a transaction T wants to read an object, it first requests a Shared lock [which is the mode of locking to ensure that the data-item does not undergoes incorrect any modifications while it is locked in this mode on the object]. But, if a transaction T wants to modify an object, it first requests an exclusive lock on the object.

Notes

RULE 2: All locks held by a transaction are released when the transaction is completed.

The locking protocol can have two cases as follows:

1. If two transactions access completely different parts of the database, then they proceed without interruption on their ways.
2. If two transactions access same object of the database, then their actions are ordered serially i.e., all actions of locked transaction are completed first, then this lock is released and the other transaction can now proceed.

Thus, the locking protocol only helps the second case i.e., interleaving of transactions.

Moreover, the shared lock on database object A is denoted as S(A) and the exclusive lock on database object A is denoted as X(A).

Example: Suppose that both transactions T_1 and T_2 read the same data objects A and B perform the following operations:

1. T_1 deducts $100 from account A.
2. T_2 reads accounts of A and B and adds 6% interest to each.
3. T_1 adds $100 to account B.

This example, may use interleaving of transactions and produce incorrect results. But, if the Strict 2PL Protocol is used, such interleaving is disallowed and performs both transactions as follows:

First T would obtain an exclusive lock on A and then read and write A. Then, T_2 would request a lock A, but, this request cannot be granted until T_1 releases its exclusive lock on A, and the DBMS therefore suspends T_2 as follows:

T_1	T_2
X(A)	
R(A)	
A: = A – 100;	
W(A)	

Now, T_1 proceeds to obtain an exclusive lock on B reads and writes B, then finally commits and the locks on A and B are released. Now, T_2's lock request is granted and it proceeds. As a result the Strict 2PL Protocol results in a serial execution of the two transactions as follows:

T_1	T_2
X(A)	
R(A)	
A: = A-100;	
W(A)	
X (B)	
R(B)	
B: = B + 100;	
W (B)	
Commit	

Contd...

Notes

	X(A)
	R(A)
	A: = A + 0.06 A;
	W(A)
	X(B)
	R(B)
	B: = B + 0.06B;
	W(B)
	Commit

Schedule Illustrating Strict 2PL with Serial Execution

Moreover, the following figure illustrates a schedule following Strict 2PL with interleaved actions, where shared lock on A can only read A, but cannot write A and both T_1 and T_2 can hold shared lock at the same time whereas an exclusive lock on B and C can both read and write the respective B and C but an exclusive lock cannot be held at the same time, instead an exclusive lock must hold each data item serially one after the other.

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	X(B)
	R(B)
	W(B)
	Commit
X(C)	
R(C)	
W(C)	
Commit	

Task Process to test serializability.

9.10 Timestamp based Protocol

In lock-based concurrency control, conflicting actions of different transactions are ordered by the order in which locks are obtained, and the lock protocol extends this ordering on actions to transactions, thereby ensuring serializability. In optimistic concurrency control, a timestamp ordering is imposed on transactions, and validation checks that all conflicting actions occurred in the same order.

Timestamps can also be used in another way: each transaction can be assigned a timestamp at startup, and we can ensure, at execution time, that if action a_i of transaction T_i conflicts with action a_j of transaction T_j, *ai* occurs before a_j if $TS(T_i) < TS(T_j)$. If an action violates this ordering, the transaction is aborted and restarted.

Notes

To implement this concurrency control scheme, every database object O is given a **read timestamp** $RTS(O)$ and a **write timestamp** $WTS(O)$. If transaction T wants to read object O, and $TS(T) < WTS(O)$, the order of this read with respect to the most recent write on O would violate the timestamp order between this transaction and the writer. Therefore, T is aborted and restarted *with a new, larger timestamp*. If $TS(T) > WTS(O)$, T reads O, and $RTS(O)$ is set to the larger of $RTS(O)$ and $TS(T)$.

Notes There is a physical change the change to $RTS(O)$ to be written to disk and to be recorded in the log for recovery purposes, even on reads. This write operation is a significant overhead.

Observe that if T is restarted with the same timestamp, it is guaranteed to be aborted again, due to the same conflict. Contrast this behavior with the use of timestamps in 2PL for deadlock prevention: there, transactions were restarted with the *same* timestamp as before in order to avoid repeated restarts. This shows that the two uses of timestamps are quite different and should not be confused.

Next, let us consider what happens when transaction T wants to write object O:

1. If TS(T) < RTS(O), the write action conflicts with the most recent read action of O, and T is therefore aborted and restarted.
2. If TS(T) < WTS(O), a naive approach would be to abort T because its write action conflicts with the most recent write of O and is out of timestamp order. It turns out that we can safely ignore such writes and continue. Ignoring outdated writes is called the Thomas Write Rule.
3. Otherwise, T writes O and WTS(O) is set to TS(T).

The Thomas Write Rule

We now consider the justification for the Thomas Write Rule. If $TS(T) < WTS(O)$, the current write action has, in effect, been made obsolete by the most recent write of O, which *follows* the current write according to the timestamp ordering on transactions. We can think of T's write action as if it had occurred immediately *before* the most recent write of O and was never read by anyone.

If the Thomas Write Rule is not used, that is, T is aborted in case (2) above, the timestamp protocol, like 2PL, allows only conflict serializable schedules. (Both 2PL and this timestamp protocol allow schedules that the other does not.) If the Thomas Write Rule is used, some serializable schedules are permitted that are not conflict serializable, as illustrated by the schedule in table below. Because T_2's write follows.

A Serializable Schedule that is not Conflict Serializable

T_1	T_2
R(A)	
	W(A)
	Commit
W(A)	
Commit	

Notes

T_1's read and precedes T_1's write of the same object, this schedule is not conflict serializable. The Thomas Write Rule relies on the observation that T_2's write is never seen by any transaction and the schedule in table above is therefore equivalent to the serializable schedule obtained by deleting this write action, which is shown in table below.

A Conflict Serializable Schedule

T_1	T_2
R(A)	
	Commit
W(A)	
Commit	

9.11 Validation based Protocols

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low. Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state. A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead. A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for monitoring the system.

We assume that each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order,

1. ***Read phase:*** During this phase, the system executes transaction T_i. It reads the values of the various data items and stores them in variables local to T_i. It performs all write operations on temporary local variables, without updates of the actual database.
2. ***Validation phase:*** Transaction T_i performs a validation test to determine whether it can copy to the database the temporary local variables that hold the results of write operations without causing a violation of serializability.
3. ***Write phase:*** If transaction T_i succeeds in validation (step 2), then the system applies the actual updates to the database. Otherwise, the system rolls back T_i.

Each transaction must go through the three phases in the order shown. However, all three phases of concurrently executing transactions can be interleaved.

To perform the validation test; we need to know when the various phases of transactions T_i took place. We shall, therefore, associate three different timestamps with transaction T_i:

1. Start(T_i), the time when T_i started its execution.
2. Validation(T_i), the time when T_i finished its read phase and started its validation phase.
3. Finish(T_i), the time when T_i finished its write phase.

We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation(T_i). Thus, the value TS(T_i) = Validation(T_i) and, if TS(T_j) < TS(T_k), then any produced schedule must be equivalent to a serial schedule in which transaction T_j appears before transaction T_k. The reason we have chosen Validation(T_i), rather than Start(T_i), as the timestamp of transaction T_i is that we can expect faster response time provided that conflict rates among transactions are indeed low.

Notes

The validation test for transaction T_j requires that, for all transactions T_i with $TS(T_i) < TS(T_j)$, one of the following two conditions must hold:

1. $Finish(T_i) < Start(T_j)$ Since T_i completes its execution before T_j started, the serializability order is indeed maintained.
2. The set of data items written by T_i does not intersect with the set of data items read by T, and T_i completes its write phase before T_j starts its validation phase ($Start(T_j) < Finish(T_i) < Validation(T_j)$). This condition ensures that the writes of T_i and T_j do not overlap. Since the writes of T_i do not affect the read of T_j, and since T_j cannot affect the read of T_i the serializability order is indeed maintained.

Schedule 5, a schedule produced by using validation.

T_{14}	T_{15}
read(B)	read(B) B := B – 50 read (A) A := A + 50
read (A) (validate) display (A + B)	(validate) write(B) write (A)

As an illustration, consider again transactions T_{14} and T_{15}. Suppose that $TS(T_{14}) < TS(T_{15})$. Then, the validation phase succeeds in the schedule 5 in above table.

Notes The writes to the actual variables are performed only after the validation phase of T_{15}. Thus, T_{14} reads the old values of B and A, and this schedule is serializable.

The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed. However, there is a possibility of starvation of long transactions, due to a sequence of conflicting short transactions that cause repeated restarts of the long transaction. To avoid starvation, conflicting transactions must be temporarily blocked, to enable the long transaction to finish.

This validation scheme is called the optimistic concurrency control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end. In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conflict is detected, even though there is a chance that the schedule may be conflict serializable.

Task Discuss write phase in validation based protocols.

9.12 Deadlock Handling

Consider the following two transactions:

T_1:	write (X) write(Y)	T_2:	write(Y) write(X)

Notes

A schedule with deadlock is given as:

T 1	T 2
lock-X on X write (X)	
	lock-X on Y write (X) wait for **lock-X** on X
wait for **lock-X** on Y	

A system is deadlocked if a set of transactions in the set is waiting for another transaction in the set.

9.12.1 Deadlock Prevention

Deadlock prevention protocols ensure that the system will never enter into a deadlock state. The basic prevention strategies are:

1. The strategies require that each transaction lock all its data items before it begins execution (pre declaration).
2. They impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Two common techniques for deadlock prevention are wait-die and wound-wait. In both the wait-die and wound-wait schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and deadlock is hence avoided.

These schemes use transaction timestamps for the prevention of deadlock.

Wait-die scheme — non-preemptive

1. Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
2. A transaction may die several times before acquiring the needed data item

Wound-wait scheme — preemptive

1. Older transaction wounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
2. May be fewer rollbacks than wait-die scheme.

Timeout-based Schemes

The timeout-based schemes have the following characteristics:

1. A transaction waits for a lock only for a specified amount of time. After that, the wait times out and transaction are rolled back. Thus deadlocks are prevented.
2. Simple to implement; but starvation may occur. Also it is difficult to determine the good value of timeout interval.

Notes

Deadlock Detection

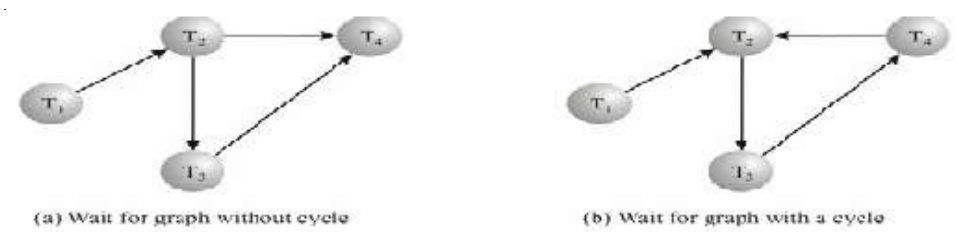
Deadlocks can be detected using a wait-for graph, which consists of a pair G = (V,E),

1. V is a set of vertices (all the transactions in the system)
2. E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.

If $T_i \rightarrow T_j$ is in E, then there is a directed edge from T_i to T_j, implying that T_i is waiting for T_j to release a data item. When T_i requests a data item currently being held by T_j, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when T_j is no longer holding a data item needed by T_i.

The system is in a state of deadlock if the wait-for graph has a cycle. You must make use of a deadlock-detection algorithm periodically to look for such cycles.

Figure 9.2: Deadlock Detection



9.12.2 Deadlock Recovery

When a deadlock is detected, some transaction will have to be rolled back to break the deadlock. Selecting that transaction, as a victim will incur minimum cost. Rollback will determine the distance the transaction needs to be rolled back. Total rollback aborts the transaction and then restarts it. The more effective method of deadlock recovery is to rollback transaction only as far as necessary to break the deadlock. Starvation occurs if the same transaction is always selected as a victim. You can include the number of rollbacks in the cost factor calculation in order to avoid starvation.

9.13 Insert and Delete Operation

If two-phase locking is used:

1. A delete operation may be performed only if the transaction deleting the tuple has an exclusive lock on the tuple to be deleted.
2. A transaction that inserts a new tuple into the database is given an X-mode lock on the tuple.

Insertions and deletions can lead to the phantom phenomenon.

1. A transaction that scans a relation (e.g., find all accounts in Bombay) and a transaction that inserts a tuple in the relation (e.g., insert a new account at Bombay) may be in conflict with each other, despite the fact that the two transactions are not accessing any tuple in common.
2. If only tuple locks are used, non-serialisable schedules may be the result: the scan than transaction may not see the new account, yet may be serialised before the insert transaction.

The transaction scanning the relation reads the information that indicates what tuples the relation contains, while a transaction inserting a tuple updates the same information.

Notes

The information should be locked.

One solution to such a problem may be:

1. Associating a data item with the relation helps represent the information about the tuples the relation contains.
2. Transactions scanning the relation acquires a shared lock in the data item.
3. Transactions inserting or deleting a tuple acquires an exclusive lock on the data item.

> *Notes* Locks on the data item do not conflict with locks on individual tuples.

The above mentioned protocol provides very low concurrency for insertions/deletions. Index locking protocols provide higher concurrency while preventing the phantom phenomenon, (by requiring locks on certain index buckets).

Index Locking Protocol: Every relation must have at least one index. Access to a relation must be made only through one of the indices on the relation. A transaction T_i that performs a lookup must lock all the index buckets that it accesses, in S-mode. A transaction T_i may not insert a tuple t_i into a relation r without updating all indices to r. T_i must perform a lookup on every index to find all index buckets that could have possibly contained a pointer to tuple t_i, had it existed already, and obtain locks in X-mode on all these index buckets. T_i must also obtain locks in X-mode on all index buckets that it modifies. The rules of the two-phase locking protocol must be observed for index locking protocols to be effective.

9.14 Weak Level of Consistency

The protocols such as strict two phase locking protocol restricts concurrency while transactions are being execution. Can we allow more concurrency by compromising on correctness/accurateness, which now needs to be ensured by database programmers rather than by the DBMS? We can operate on weak levels of consistency using the following mechanism:

9.14.1 Degree-two Consistency

Degree-two consistency differs from two-phase locking in that S-locks may be released at any time, and locks may be acquired at any time, however:

1. X-locks must be held till the transaction has ended
2. Serialisability is not guaranteed. The programmer must ensure that no erroneous database state will occur.

One of the degree two-consistency level protocols is Cursor stability. It has the following rules:

1. For reads, each tuple is locked, read, and lock is immediately released
2. X-locks are held till end of transaction.

Notes

9.14.2 Weak Levels of Consistency in SQL

SQL allows non-serialisable executions:

1. ***Serialisable*** is the default.
2. ***Repeatable*** read allows only committed records to be read, and repeating a read should return the same value (so read locks should be retained). (However, the phantom phenomenon need not be prevented) that is, T_1 may see some records inserted by T_2, but may not see others inserted by T_2.
3. ***Read committed*** same as degree two consistency, but most systems implement it as cursor-stability.
4. ***Read uncommitted*** allows even uncommitted data to be read. This is the level, which has almost no restriction on concurrency but will result in all sorts of concurrency related problems.

Task Use of wound-wait scheme.

Lab Exercise Create a table and apply implicit transactions on that.

9.15 Summary

- As transaction is the smallest unit of a work.
- It plays an important role in a Database Management System.
- In this unit we cover Properties of a transaction, Basic operations on transaction, State of transaction etc. and cover Concepts of Concurrency Control and various problems which occur on Concurrency such as Lost Updata problem, Dirty read problem etc. and cover concepts about serializability along with a technique to find whether given transaction are serial or not.

9.16 Keywords

A transaction: This is a unit of program execution that accesses and possibly updates various data items.

Aborted state: This transaction state occurs after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Active state: This is the initial state of a transaction, the transaction stays in this state while it is starting execution.

Committed state: This transaction state occurs after the successful completion of the transaction.

Complete schedule: A schedule that contains either an abort or a commit statement for each transaction whose actions are listed in schedule, then it is called a complete schedule. A complete transaction must also contain all the actions of every transaction that appears in it.

Partially committed state: This transaction state occurs after the final (last) statement of the transaction has been executed.

Notes

9.17 Self Assessment

Fill in the blanks:

1. Database ..... are the units in which programs read or write information.
2. Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk known as .....
3. ..... is the initial state of a transaction.
4. ..... state occurs after the discovery that normal execution can no longer proceed.
5. Transaction-processing systems usually allow multiple transactions to run concurrently (at the same time) known as .....
6. Users are responsible for ensuring transaction .....
7. The ..... locking protocol ensures serializability.
8. ..... protocols ensure that the system will never enter into a deadlock state.
9. When a deadlock is detected, some transaction will have to be rolled back to break the .....
10. A transaction is an execution of a user program and is seen by the DBMS as a ..... or list of actions.

9.18 Review Questions

1. What are the properties of transaction? Explain briefly.
2. Write short notes on transactions and schedules.
3. Explain lock-based concurrency control in detail.
4. What are ACID properties? Explain.
5. Why we need concurrent execution of transactions?
6. Describe the strict two-phase locking (strict 2PL) protocol.
7. What transaction characteristics are under programmer control in SQL? Discuss various access modes and isolation levels.
8. Explain transaction state in detail.
9. Describe concurrent execution. Also explain motivation for concurrent execution.
10. Explain Thomas Write Rule.

Answers: Self Assessment

1.	'objects'	2.	durability
3.	Active state	4.	Failed
5.	concurrent execution of transactions	6.	consistency
7.	two-phase	8.	Deadlock prevention
9.	deadlock	10.	series

9.19 Further Readings

Notes

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 10: Datalog and Recursion

CONTENTS

Objectives

Introduction

10.1 Datalog and Recursion

10.2 Evaluation of Datalog Program

10.3 Recursive Queries and Negation

10.4 Modeling Complex Data Semantics

10.5 Specialization

10.6 Generalization

10.7 Summary

10.8 Keywords

10.9 Self Assessment

10.10 Review Questions

10.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Know datalog and recursion
- Describe datalog program evaluation
- Explain specialization and generalization

Introduction

While relational algebra can express many useful operations on relations, there are some computations that cannot be written as an expression of relational algebra. A common kind of operation on data that we cannot express in relational algebra involves an infinite, recursively defined sequence of similar expressions.

10.1 Datalog and Recursion

We now define a relation called Components that identifies the components of every part. Consider the following program, or collection of rules:

Components (Part, Subpart): Assembly (Part, Subpart, Qty).

Components (Part, Subpart): Assembly (Part, Part2, Qty),

Components (Part2, Subpart):

Notes

These are rules in Datalog, a relational query language inspired by Prolog, the wellknown logic programming language; indeed, the notation follows Prolog. The first rule should be read as follows:

For all values of Part, Subpart, and Qty,

if there is a tuple <Part, Subpart, Qty> in Assembly,

then there must be a tuple <Part, Subpart> in Components.

The second rule should be read as follows:

For all values of Part, Part2, Subpart, and Qty,

if there is a tuple <Part, Part2, Qty> in Assembly and

a tuple <Part2, Subpart> in Components,

then there must be a tuple <Part, Subpart> in Components.

The part to the right of the:- symbol is called the body of the rule, and the part to the left is called the head of the rule. The symbol:- denotes logical implication; if the tuples mentioned in the body exist in the database, it is implied that the tuple mentioned in the head of the rule must also be in the database.

Notes The body could be empty; in this case, the tuple mentioned in the head of the rule must be included in the database.

Therefore, if we are given a set of Assembly and Components tuples, each rule can be used to infer, or deduce, some new tuples that belong in Components. This is why database systems that support Datalog rules are often called deductive database systems.

Each rule is really a template for making inferences: by assigning constants to the variables that appear in a rule, we can infer specific Components tuples. For example, by setting Part=trike, Subpart=wheel, and Qty=3, we can infer that <trike, wheel> is in Components. By considering each tuple in Assembly in turn, the first rule allows us to infer that the set of tuples obtained by taking the projection of Assembly onto its first two fields is in Components.

Figure 10.1: Components Tuples Obtained by Applying the Second Rule Once

part	*subpart*
trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

The second rule then allows us to combine previously discovered Components tuples with Assembly tuples to infer new Components tuples. We can apply the second rule by considering the cross-product of Assembly and (the current instance of) Components and assigning values to the variables in the rule for each row of the cross-product, one row at a time. Observe how the

Notes

repeated use of the variable Part2 prevents certain rows of the cross product from contributing any new tuples; in effect, it specifies an equality join condition on Assembly and Components. The tuples obtained by one application of this rule are shown in Figure 10.1. (In addition, Components contains the tuples obtained by applying the first rule; these are not shown.)

The tuples obtained by a second application of this rule are shown in Figure 10.2. Note that each tuple shown in Figure 10.1 is reinferred. Only the last two tuples are new.

Figure 10.2: Components Tuples Obtained by Applying the Second Rule Twice

part	*subpart*
trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube
trike	rim
trike	tube

Applying the second rule a third time does not generate any additional tuples. The set of Components tuples shown in Figure 10.2 includes all the tuples that can be inferred using the two Datalog rules defining Components and the given instance of Assembly. The components of a trike can now be obtained by selecting all Components tuples with the value trike in the first field.

Each application of a Datalog rule can be understood in terms of relational algebra. The first rule in our example program simply applies projection to the Assembly relation and adds the resulting tuples to the Components relation, which is initially empty. The second rule joins Assembly with Components and then does a projection. The result of each rule application is combined with the existing set of Components tuples using union.

The only Datalog operation that goes beyond relational algebra is the repeated application of the rules defining Components until no new tuples are generated. This repeated application of a set of rules is called the fixpoint operation.

We conclude this section by rewriting the Datalog de_nition of Components in terms of extended SQL, using the syntax proposed in the SQL:1999 draft and currently supported in IBM's DB2 Version 2 DBMS:

```
WITH RECURSIVE Components(Part, Subpart) AS
        (SELECTA1.Part, A1.Subpart FROM Assembly A1)
        UNION
        (SELECTA2.Part, C1.Subpart
        FROM Assembly A2, Components C1
        WHERE A2.Subpart = C1.Part)

SELECT * FROM Components C2
```

The WITH clause introduces a relation that is part of a query definition; this relation is similar to a view, but the scope of a relation introduced using WITH is local to the query definition. The

Notes

RECURSIVE keyword signals that the table (in our example, Components) is recursively defined. The structure of the definition closely parallels the Datalog rules. Incidentally, if we wanted to find the components of a particular part, for example, *trike*, we can simply replace the last line with the following:

```
SELECT * FROM Components C2
WHERE C2.Part = 'trike'
```

10.2 Evaluation of Datalog Program

We classify the relations in a Datalog program as either output relations or input relations. Output relations are defined by rules (e.g., Components), and input relations have a set of tuples explicitly listed (e.g., Assembly). Given instances of the input relations, we must compute instances for the output relations. The meaning of a Datalog program is usually defined in two different ways, both of which essentially describe the relation instances for the output relations. Technically, a query is a selection over one of the output relations (e.g., all Components tuples C with C.part = trike). However, the meaning of a query is clear once we understand how relation instances are associated with the output relations in a Datalog program.

The first approach to defining what a Datalog program means is called the least model semantics and gives users a way to understand the program without thinking about how the program is to be executed. That is, the semantics is declarative, like the semantics of relational calculus, and not operational like relational algebra semantics. This is important because the presence of recursive rules makes it difficult to understand a program in terms of an evaluation strategy.

The second approach, called the least fixpoint semantics, gives a conceptual evaluation strategy to compute the desired relation instances. This serves as the basis for recursive query evaluation in a DBMS. More efficient evaluation strategies are used in an actual implementation, but their correctness is shown by demonstrating their equivalence to the least fixpoint approach. The fixpoint semantics is thus operational and plays a role analogous to that of relational algebra semantics for non-recursive queries.

Least Model Semantics

We want users to be able to understand a Datalog program by understanding each rule independently of other rules, with the meaning: If the body is true, the head is also true. This intuitive reading of a rule suggests that given certain relation instances for the relation names that appear in the body of a rule, the relation instance for the relation mentioned in the head of the rule must contain a certain set of tuples. If a relation name R appears in the heads of several rules, the relation instance for R must satisfy the intuitive reading of all these rules. However, we do not want tuples to be included in the instance for R unless they are necessary to satisfy one of the rules defining R. That is, we only want to compute tuples for R that are supported by some rule for R.

To make these ideas precise, we need to introduce the concepts of models and least models. A model is a collection of relation instances, one instance for each relation in the program, that satisfies the following condition. For every rule in the program, whenever we replace each variable in the rule by a corresponding constant, the following holds:

1. If every tuple in the body (obtained by our replacement of variables with constants) is in the corresponding relation instance,
2. Then the tuple generated for the head (by the assignment of constants to variables that appear in the head) is also in the corresponding relation instance.

Notes

Observe that the instances for the input relations are given, and the definition of a model essentially restricts the instances for the output relations.

Consider the rule

Components(Part, Subpart) :- Assembly(Part, Part2, Qty),

Components(Part2, Subpart).

Suppose that we replace the variable Part by the constant wheel, Part2 by tire, Qty by 1, and Subpart by rim:

Components(wheel, rim) :- Assembly(wheel, tire, 1),

Components(tire, rim).

Let A be an instance of Assembly and C be an instance of Components. If A contains the tuple <wheel, tire, 1> and C contains the tuple <tire, rim>, then C must also contain the tuple <wheel, rim> in order for the pair of instances A and C to be a model. Of course, the instances A and C must satisfy the inclusion requirement illustrated above for every assignment of constants to the variables in the rule: If the tuples in the rule body are in A and C, the tuple in the head must be in C.

Safe Datalog Programs

Consider the following program:

Complex Parts(Part) :- Assembly(Part, Subpart, Qty), Qty > 2.

According to this rule, complex part is defined to be any part that has more than two copies of any one subpart. For each part mentioned in the Assembly relation, we can easily check if it is a complex part. In contrast, consider the following program:

Price Parts(Part,Price) :- Assembly(Part, Subpart, Qty), Qty > 2.

This variation seeks to associate a price with each complex part. However, the variable Price does not appear in the body of the rule. This means that an infinite number of tuples must be included in any model of this program! To see this, suppose that we replace the variable Part by the constant trike, SubPart by wheel, and Qty by 3. This gives us a version of the rule with the only remaining variable being Price:

Price Parts(trike,Price) :- Assembly(trike, wheel, 3), 3 > 2.

Now, any assignment of a constant to Price gives us a tuple to be included in the output relation Price Parts. For example, replacing Price by 100 gives us the tuple Price Parts(trike,100). If the least model of a program is not finite, for even one instance of its input relations, then we say the program is unsafe.

Database systems disallow unsafe programs by requiring that every variable in the head of a rule must also appear in the body. Such programs are said to be range restricted, and every range-restricted Datalog program has a finite least model if the input relation instances are finite.

Notes

10.3 Recursive Queries and Negation

Consider the following rule:

Big(Part) :- Assembly(Part, Subpart, Qty), Qty > 2,

not Small(Part).

Small(Part) :- Assembly(Part, Subpart, Qty), not Big(Part).

These two rules can be thought of as an attempt to divide parts (those that are mentioned in the first column of the Assembly table) into two classes, Big and Small. The first rule defines Big to be the set of parts that use at least three copies of some subpart and that are not classified as small parts. The second rule defines Small as the set of parts that are not classified as big parts.

If we apply these rules to the instance of Assembly shown in Figure 10.3, trike is the only part that uses at least three copies of some subpart. Should the tuple <trike> be in Big or Small? If we apply the first rule and then the second rule, this tuple is in Big. To apply the first rule, we consider the tuples in Assembly, choose those with Qty > 2 (which is just <trike>), discard those that are in the current instance of Small (both Big and Small are initially empty), and add the tuples that are left to Big. Therefore, an application of the first rule adds <trike> to Big. Proceeding similarly, we can see that if the second rule is applied before the first, <trike> is added to Small instead of Big!

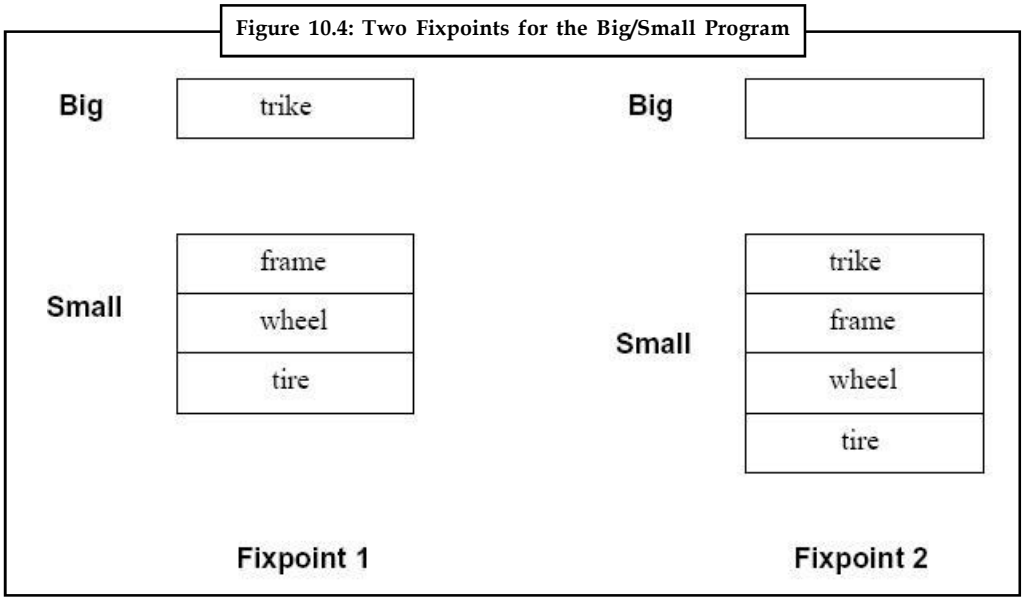
Figure 10.3: An instance of assembly

part	*subpart*	*qty*
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

This program has two fixpoints, neither of which is smaller than the other, as shown in Figure 10.4. The first fixpoint has a Big tuple that does not appear in the second fixpoint; therefore, it is not smaller than the second fixpoint. The second fixpoint has a Small tuple that does not appear in the first fixpoint; therefore, it is not smaller than the first fixpoint. The order in which we apply the rules determines which fixpoint is computed, and this situation is very unsatisfactory. We want users to be able to understand their queries without thinking about exactly how the evaluation proceeds.

Notes

Figure 10.4: Two Fixpoints for the Big/Small Program



The root of the problem is the use of not. When we apply the first rule, some inferences are disallowed because of the presence of tuples in Small. Parts that satisfy the other conditions in the body of the rule are candidates for addition to Big, and we remove the parts in Small from this set of candidates. Thus, some inferences that are possible if Small is empty (as it is before the second rule is applied) are disallowed if Small contains tuples (generated by applying the second rule before the first rule). Here is the difficulty: If not is used, the addition of tuples to a relation can disallow the inference of other tuples. Without not, this situation can never arise; the addition of tuples to a relation can never disallow the inference of other tuples.

Task

"Each application of a Datalog rule can be understood in terms of relational algebra". Discuss.

Range-Restriction and Negation

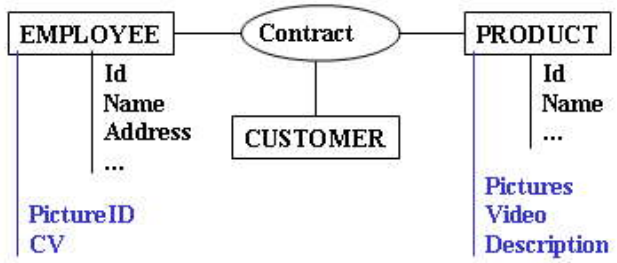
If rules are allowed to contain not in the body, the definition of range-restriction must be extended in order to ensure that all range-restricted programs are safe. If a relation appears in the body of a rule preceded by not, we call this a negated occurrence. Relation occurrences in the body that are not negated are called positive occurrences. A program is range-restricted if every variable in the head of the rule appears in some positive relation occurrence in the body.

10.4 Modeling Complex Data Semantics

Data modelling, using a specific data model type, and as a unique activity during information system design, is commonly attributed to Charles Bachman (1969) who presented the Data Structure Diagram as one of the first, widely used data models for network database design. Several alternative data model types were proposed shortly thereafter, perhaps the best known of which are shown in Figure 10.5.

Notes

Figure: 10.5: An Administrative DB with Media Data



(a) ER model of multiple media DB

EMPLOYEE <Id, Name, Address, PictureID, CV, ...>
CUSTOMER <Id, Name, Address, Rating, ...>
PRODUCT <Id, Name, Pictures, Video, Description, ...>
Contract <EId, CId, PId, date, quantity, cost, ... >

(b) Relational model of multiple media DB

The relational model was quickly criticized for being 'flat' in the sense that all information is represented as a set of tables with atomic cell values. The definition of well-formed relational models requires that complex attribute types (hierarchic, composite, multi-valued, and derived) be converted to atomic attributes and that relations be normalized. Inter-entity (inter-relation) relationships are difficult to visualize in the resulting set of relations, making control of the completeness and correctness of the model difficult. The relational model maps easily to the physical characteristics of electronic storage media, and as such, is a good tool for design of a physical database.

The entity-relationship approach to modelling, had two primary objectives: first to visualize inter-entity relationships and second to separate the DB design process into two phases:

1. Record, in an ER model, the entities and inter-entity relationships required "by the enterprise", i.e. by the owner/user of the information system or application. This phase and its resulting model should be independent of the DBMS tool that is to be used for realizing the DB.
2. Translate the ER model to the data model supported by the DBMS to be used for implementation.

This two-phase design supports data independence, i.e. the possibility of DB structure modification at the physical level without requiring changes to the enterprise or user view of the DB content.

10.5 Specialization

We define specialization as a restriction on the extension of a process: a process p_1 is a specialization of a process p_0 if every instance of p_1 is also an instance of p_0, but not necessarily vice versa. This definition can be restated to take into account frame of reference; there are two cases to consider:

1. Both processes are described using the same frame of reference. In this case the extensions of the processes are described in the same terms and can be compared directly. Thus p_1 is a specialization of p_0 if and only if the extension of p_1 as described using the given frame of reference is a subset of the extension of p_0 as similarly described.

Notes

2. The processes are described using different frames of reference, but there exists a "common" frame of reference (which is a refinement of both of these). In this case, p_1 is a specialization of p_0 if and only if the refinement of p_1 is a specialization of the refinement of p_0 under the common frame of reference. Thus this second case is reduced to the first by means of refinement.

We propose that one useful way to operationalize this notion of specialization is in terms of a set of transformations for any particular process representation, which, when applied to a process description, produce a description of a specialization of that process. The two part definition of specialization suggests that two sorts of transformations will be needed:

A specializing transformation is an operation which, when applied to a process described using a given representation and a given frame of reference, results in a new process description under that representation and frame of reference corresponding to a specialization of the original process. Specializing transformations change the extension of a process while preserving the frame of reference.

In contrast, a refining transformation is an operation which changes the frame of reference of a process while preserving its extension, producing a process description of the same process under a different frame of reference.

For each type of transformation there is a related inverse type: a generalizing transformation acts on a process description to produce a generalization of the original process and is thus the inverse of a specializing transformation. Similarly, an abstracting transformation is the inverse of the refining transformation, producing a new description of the same process under a frame of reference for which the original frame is a refinement.

Given that the refining/abstracting transformations preserve the extension of a process, it follows from our definition of process specialization that a specializing transformation composed with refining/abstracting transformations in any sequence produces a specialization. The analogous statement holds for generalizing transformations.

A set of refining/abstracting transformations is said to be complete if for any process p described under a frame of reference, the description of that process under any other frame of reference can be obtained by applying to p a finite number of transformations drawn from the set.

A set of specializing transformations is said to be locally complete if for any frame of reference and any process p described using that frame of reference, then any specialization of p described under that frame of reference can be obtained by applying to p a finite number of transformations drawn from the set. This corresponds to the first part of the definition of process specialization given above.

There is also a notion of completeness corresponding to the second part of the definition. A set of specializing transformations and refining/abstracting transformations is said to be globally complete if for any process p, any specialization of p for which a common frame of reference exists can be obtained by applying to p a finite number of transformations drawn from the set.

Proposition: Let A be a complete set of refining/abstracting transformations and S be a locally complete set of specializing transformations. Then A [[union]] S is globally complete.

Proof: Consider a process p_0 and a specialization p_1 for which a common frame of reference exists. Since A is complete, one can apply a finite sequence of transformations from A to p_0 to produce its refinement in the common frame of reference. By local completeness one can then apply specializing transformations to produce the refinement of p_1 (since it is a specialization of the refinement of p_0 by assumption). Finally, by the completeness of A one can transform the refinement of p_1 into p_1.

Notes

10.6 Generalization

A generalization hierarchy is a structured grouping of entities that share common attributes. It is a powerful and widely used method for representing common characteristics among entities while preserving their differences. It is the relationship between an entity and one or more refined versions. The entity being refined is called the *supertype* and each refined version is called the *subtype.*

Generalization hierarchies should be used when (1) a large number of entities appear to be of the same type, (2) attributes are repeated for multiple entities, or (3) the model is continually evolving. Generalization hierarchies improve the stability of the model by allowing changes to be made only to those entities germane to the change and simplify the model by reducing the number of entities in the model.

Creating a Generalization Hierarchy

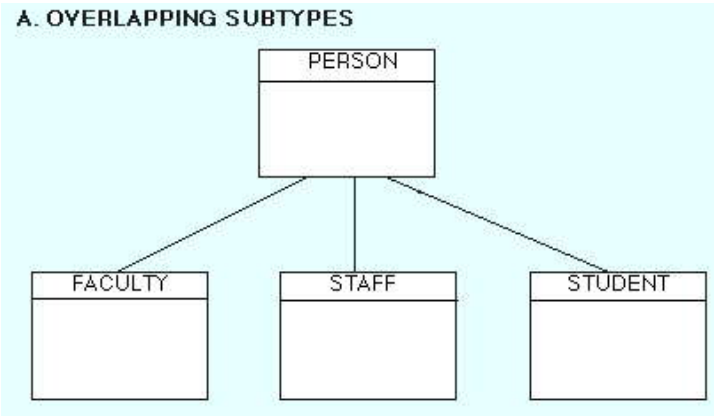
To construct a generalization hierarchy, all common attributes are assigned to the supertype. The supertype is also assigned an attribute, called a discriminator, whose values identify the categories of the subtypes. Attributes unique to a category, are assigned to the appropriate subtype. Each subtype also inherits the primary key of the supertype. Subtypes that have only a primary key should be eliminated. Subtypes are related to the supertypes through a one-to-one relationship.

Types of Hierarchies

A generalization hierarchy can either be overlapping or disjoint. In an overlapping hierarchy an entity instance can be part of multiple subtypes. For example, to represent people at a university you have identified the supertype entity PERSON which has three subtypes, FACULTY, STAFF, and STUDENT. It is quite possible for an individual to be in more than one subtype, a staff member who is also registered as a student.

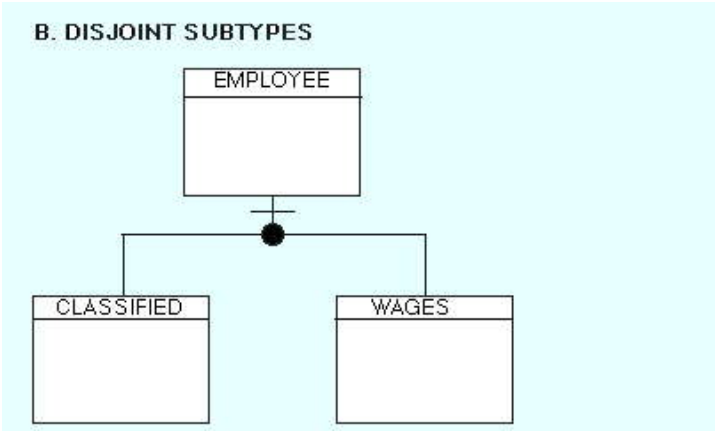
Example: In a disjoint hierarchy, an entity instance can be in only one subtype. For example, the entity EMPLOYEE, may have two subtypes, CLASSIFIED and WAGES. An employee may be one type or the other but not both. Figure 10.6 shows A) overlapping and B) disjoint generalization hierarchy.

Figure 10.6



Contd...

Notes



Task

"Specializing transformations change the extension of a process while preserving the frame of reference". Explain

Rules

The primary rule of generalization hierarchies is that each instance of the supertype entity must appear in at least one subtype; likewise, an instance of the subtype must appear in the supertype.

Subtypes can be a part of only one generalization hierarchy. That is, a subtype can not be related to more than one supertype. However, generalization hierarchies may be nested by having the subtype of one hierarchy be the supertype for another.

Subtypes may be the parent entity in a relationship but not the child. If this were allowed, the subtype would inherit two primary keys.

10.7 Summary

- The objective of data modeling is to design a data structure for a database fitting as good as possible with some relevant world, often related to an organization with some information need.
- In general there is some relationship between a data model and a part of the existing world, but it is also possible that a data model has a relationship with some imaginary and abstract world.

10.8 Keywords

Datalog Program: Datalog program as either output relations or input relations.

Data modeling: Data modeling, using a specific data model type, and as a unique activity during information system design.

Specialization: specialization as a restriction on the extension of a process.

Notes

10.9 Self Assessment

Fill in the blanks:

1. ..... can be a part of only one generalization hierarchy.
2. A relational query language inspired by .....
3. Each application of a Datalog rule can be understood in terms of .....
4. The ..... semantics is thus operational and plays a role analogous to that of relational algebra semantics for non-recursive queries.
5. A model is a collection of relation .....
6. ..... disallow unsafe programs by requiring that every variable in the head of a rule must also appear in the body.
7. ..... is a powerful and widely used method for representing common characteristics among entities while preserving their differences.
8. ..... unique to a category, are assigned to the appropriate subtype.
9. A generalization hierarchy can either be overlapping or .....
10. In an ..... hierarchy an entity instance can be part of multiple subtypes.

10.10 Review Questions

1. What do you mean by generalization?
2. Explain hierarchies types in generalization.
3. Describe least model semantics.
4. What do you mean by safe datalog program? Explain
5. "A model is a collection of relation instances, one instance for each relation in the program". Discuss
6. Describe negation and recursive queries in detail.
7. "A refining transformation is an operation which changes the frame of reference of a process while preserving its extension, producing a process description of the same process under a different frame of reference". Explain
8. "A generalization hierarchy is a structured grouping of entities that share common attributes". Explain
9. Describe the rule of generalization hierarchy.
10. Write short note on datalog.

Answers: Self Assessment

1.	Subtypes	2.	Prolog
3.	relational algebra	4.	fixpoint
5.	instances	6.	Database systems

Notes

7. Generalization
8. Attributes
9. disjoint
10. overlapping

10.11 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 11: Recovery System

CONTENTS

Objectives

Introduction

11.1 Introduction to Crash Recovery

11.1.1 Stealing Frames and Forcing Pages

11.1.2 Recovery - Related Steps during Normal Execution

11.1.3 Overview of ARIES

11.2 Failure Classification

11.3 Storage Structure

11.4 Recovery and Atomicity

11.5 Log Based Recovery

11.6 Recovery with Concurrent Transactions

11.7 Buffer Management

11.8 Failure with Loss of Non-volatile Storages

11.9 Summary

11.10 Keywords

11.11 Self Assessment

11.12 Review Questions

11.13 Further Readings

Objectives

After studying this unit, you will be able to:

- Realise the concept of crash recovery
- Discuss recovery and atomicity
- Explain log based recovery
- State the concept of buffer management

Introduction

The database system must take actions in advance to ensure that the atomicity and durability properties of transactions as a computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, information may be lost. Are preserved. An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed

Notes

before the failure. The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a crash.

11.1 Introduction to Crash Recovery

A transaction may fail because of hardware or a software failure. It is the responsibility of the recovery manager to handle such failure and ensure 'atomicity' and 'durability'. It attains atomicity by undoing the uncommitted transactions. It also attains durability by retaining the committed transaction results even after system crashes.

Under normal execution, transaction manager takes care of serializability by providing locks when requested. It writes the data to the disk in order to avoid loss of data after the system crash.

11.1.1 Stealing Frames and Forcing Pages

1. ***Steal Approach:*** The changes made on an object '0' by a transaction are written onto the disk even before the transaction is committed. This is because another transaction wants a page to be loaded and buffer manager finds replacing frame with object '0' as optimal.
2. ***Force Approach:*** All the objects in buffer pool are forced to disk after the transaction is committed.

The simplistic implementation of recovery management is to use no-steal-force approach. With no steal, the data will not be written until a transaction is committed, hence there is no need of an undo operation and force approach enables us to write data to the disk after committing, hence we need not perform redo operation.

Though these approaches are simple, they have certain disadvantages. No steal approach requires a large buffer pool. Force approach involves expensive I/O costs.

If an object is frequently modified, then if needs to be written onto the disk very frequently involve expensive I/O operation.

Hence steal and no-force approach is implemented by recovery management. Using this techniques the page is not written onto disk when the modifying transaction is still active. And it does-not force a page to be written onto disk, when transaction commits.

11.1.2 Recovery - Related Steps during Normal Execution

The recovery manager, stores the modifications made onto a storage which does not react to system failures. Such storage is called stable storage.

The modifications made to the data is called log. Recovery manager loads the log onto the stable storage before the new changes are made.

If a transaction is aborted, then log enables recovery manager to undo the operations and redo the operations if it is committed.

No force approach does not write data into the disk after the transaction is committed. If a transaction is committed just before a crash, then the modifications made by transaction will not be loaded onto the disk. This modified data is reloaded from the stable storage.

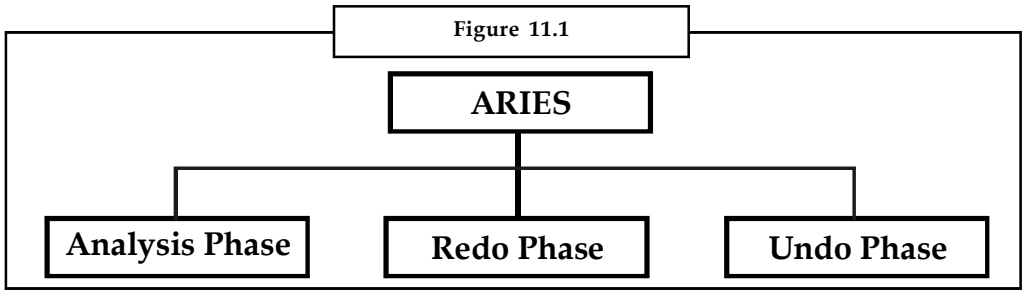
Steal approach enables to write data onto the disk before committing. If a crash occurs before committing, then all the data modified onto the disk must be undone. This is done with help of log.

Notes

The time and efforts needed to recover is proportional to number of changes made. A process called check pointing is used to minimize the time needed to recover.

11.1.3 Overview of ARIES

Figure 11.1



ARIES is, an algorithm for recovering from crash, that uses no-force, steal approach.

1. ***Analysis Phase:*** It analyses the buffer pool to identify the active transactions and dirty pages.
2. ***Undo Phase:*** If the modified data is loaded into disk before a transaction commits, then it must undo the modification in case of crash.
3. ***Redo Phase:*** It must restore the data which it was before the crash. This is done if the data modified by committed transaction is not loaded onto the disk.
4. ***Rollback:*** All log records are stored in a linked list and to operate rollback, the linked list is accessed in reverse order.

11.2 Failure Classification

There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. The simplest type of failure is one that does not result in the loss of information in the system. The failures that are more difficult tom deal with are those that result in loss of information. Various types of failure are:

Transaction Failure

There are two types of errors that may cause a transaction to fail:

1. ***Logical error:*** The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.
2. ***System error:*** The system has entered an undesirable state (for example, deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be executed at a later time.

System Crash

There is a hardware malfunction, or a bug in the database soft ware or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of nonvolatile storage remains intact, and is not corrupted.

Notes

The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption. Well-designed systems have numerous internal checks, at the hardware and the software level, that bring the system to a halt when there is an error. Hence, the fail-stop assumption is a reasonable one.

Disk Failure

A disk block loses its content as a result of either a head crash or failure during a data transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as tapes, are used to recover from the failure.

To determine how the system should recover from failures, we need to identify the failure modes of those devices used for storing data. Next, we must consider how these failure modes affect the contents of the database. We can then propose algorithms to ensure database consistency and transaction atomicity despite failures. These algorithms, known as recovery algorithms, have two parts:

1. Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures.
2. Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability.

Task Sometime your system not running properly its logical or system error.

11.3 Storage Structure

The various data items in the database may be stored and accessed in a number of different storage media. To understand how to ensure the atomicity and durability properties of a transaction, we must gain a better understanding of these storage media and their access methods.

Storage Types

Storage media can be distinguished by their relative speed, capacity, and resilience to failure, and classified as volatile storage or nonvolatile storage. We review these terms, and introduce another class of storage, called stable storage.

Stable Storage Implementation

To implement stable storage, we need to replicate the needed information in several nonvolatile storage media (usually disk) with independent failure modes, and to update the information in a controlled manner to ensure that failure during data transfer does not damage the needed information.

RAID systems guarantee that the failure of a single disk (even during data transfer) will not result in loss of data. The simplest and fastest form of RAID is the mirrored disk, which keeps two copies of each block, on separate disks. Other forms of RAID offer lower costs, but at the expense of lower performance.

RAID systems, however, cannot guard against data loss due to disasters such as fires or flooding. Many systems store archival backups of tapes off-site to guard against such disasters. However, since tapes cannot be carried off-site continually, updates since the most recent time that tapes

Notes

were carried off-site could be lost in such a disaster. More secure systems keep a copy of each block of stable storage at a remote site, writing it out over a computer network, in addition to storing the block on a local disk system. Since the blocks are output to a remote system as and when they are output to local storage, once an output operation is complete, the output is not lost, even in the event of a disaster such as a fire or flood. We study such remote' backup systems

In this section, we discuss how storage media can be protected from failure during data transfer. Block transfer between memory and disk storage can result in:

1. ***Successful completion:*** The transferred information arrived safely at its destination.
2. ***Partial failure:*** A failure occurred in the midst of transfer, and the destination block has incorrect information.
3. ***Total failure:*** The failure occurred sufficiently early during the transfer that the destination block remains intact.

We require that, if a data-transfer failure occurs, the system detects it and invokes a recovery procedure to restore the block to a consistent state. To do so, the system must maintain two physical blocks for each logical database block; in the case of mirrored disks, both blocks are at the same location; in the case of remote backup, one of the blocks is local, whereas the other is at a remote site. An output operation is executed as follows:

1. Write the information onto the first physical block.
2. When the first write completes successfully, write the same information onto the second physical block.
3. The output is completed only after the second write completes successfully.

During recovery, the system examines each pair of physical blocks. If both are the same and no detectable error exists, then no further actions are necessary. (Recall that errors in a disk block, such as a partial write to the block, are detected by storing a checksum with each block.) If the system detects an error in one block, then it replaces its content with the content of the other block. If both blocks contain no detectable error, but they differ in content, then the system replaces the content of the first block with the value of the second. This recovery procedure ensures that a write to stable storage either succeeds completely (that is, updates all copies) or results in no change.

The requirement of comparing every corresponding pair of blocks during recovery is expensive to meet. We can reduce the cost greatly by keeping track of block writes that are in progress, using a small amount of nonvolatile RAM. On recovery, only blocks for which writes were in progress need to be compared.

The protocols for writing out a block to a remote site are similar to the protocols for writing blocks to a mirrored disk system

We can extend this procedure easily to allow the use of an arbitrarily large number of copies of each block of stable storage. Although a large number of copies reduces the probability of a failure to even lower than two copies-do, it is usually reasonable to simulate stable storage with only two copies.

Data Access

The database system resides permanently on nonvolatile storage (usually disks), and is partitioned into fixed-length storage units called blocks. Blocks are the units of data transfer to and from disk, and may contain several data items. We shall assume that no data item spans two

Notes

or more blocks. This assumption is realistic for most data-processing applications, such as our banking example.

Transactions input information from the disk to main memory, and then output the information back onto the disk. The input and output operations are done in block units. The blocks residing on the disk are referred to as physical blocks; the blocks residing temporarily in main memory are referred to as buffer blocks. The area of memory where blocks reside temporarily is called the disk buffer.

Block movements between disk and main memory are initiated through the following two operations:

1. input(B) transfers the physical block B to main memory.
2. output(B) transfers the buffer block B to the disk, and replaces the appropriate physical block there.

Each transaction T_i has a private work area in which copies of all the data items accessed and updated by T_i are kept. The system creates this work area when the transaction is initiated; the system removes it when the transaction either commits or aborts. Each data item X kept in the work area of transaction T_i is denoted by X_i. Transaction T_i interacts with the database system by transferring data to and from its work area to the system buffer. We transfer data by these two operations:

1. ***read(X)*** assigns the value of data item X to the local variable X_i. It executes this operation as follows:
   (a) If block B x on which X resides is not in main memory, it issues input(B x).
   (b) It assigns to X_i the value of X from the buffer block.
2. ***write(X)*** assigns the value of local variable X_i to data item X in the buffer block. It executes this operation as follows:
   (a) If block B x on which X resides is not in main memory, it issues input(Bx)
   (b) It assigns the value of X_i to X in buffer B x.

Notes Both operations may require the transfer of a block from disk to main memory. They do not, however, specifically require the transfer of a block from main memory to disk.

A buffer block is eventually written out to the disk either because the buffer manager needs the memory space for other purposes or because the database system wishes to reflect the change to B on the disk. We shall say that the database system performs a force-output of buffer B if it issues an output(B).

When a transaction needs to access a data item X for the first time, it must execute read (X). The system then performs all updates to X on X_i. After the transaction accesses X for the final time, it must execute write(X) to reflect the change to X in the database itself.

The output(B x) operation for the buffer block B x on which X resides ,does not need to take effect immediately after write(X) is executed, since the block Bx may contain other data items that are still being accessed. Thus, the actual output may take place later. Notice that, if the system crashes after the write(X) operation was executed but before output(Bx) was executed, the new value of X is never written to disk and, thus, is lost.

Notes

Task Disk failure reasons.

11.4 Recovery and Atomicity

Consider again our simplified banking system and transaction T_i that transfers $50 from account A to account B, with initial values of A and B being $1000 and $2000, respectively. Suppose that a system crash has occurred during the execution of T_i, after output (B A) has taken place, but before output(B B) was executed, where B A and B B denote the buffer blocks on which A and B reside. Since the memory contents were lost, we do not know the fate of the transaction; thus, we could invoke one of two possible recovery procedures:

1. ***Execute*** T_i**.** This procedure will result in the value of A becoming $900, rather than $950. Thus, the system enters an inconsistent state.
2. ***Do not execute*** T_i**.** The current system state has values of $950 and $2000 for A and B, respectively. Thus, the system enters an inconsistent state.

In either case, the database is left in an inconsistent state, and thus this simple recovery scheme does not work. The reason for this difficulty is that we have modified the database without having assurance that the transaction will indeed commit. Our goal is to perform either all or no database modifications made by T_i. However, if T_i performed multiple database modifications, several output operations may be required, and a failure may occur after some of these modifications have been made, but before all of them are made.

To achieve our goal of atomicity, we must first output information describing the modifications to stable storage, without modifying the database itself. As we shall see, this procedure will allow us to output all the modifications made by a committed transaction, despite failures.

11.5 Log Based Recovery

Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures. Recovery algorithms have two parts:

1. Actions taken during normal transaction processing is to ensure that enough information exists to recover from failures,
2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

While modifying the database, without ensuring that the transaction will commit, may leave the database in an inconsistent state. Let us consider an example transaction T_1 that transfers ₹1000/- from account X to account Y; goal is either to perform all database modifications made by T_1 or none at all. T_1 modifies x by subtracting ₹1000/- and modifies Y by adding ₹ 1000/-. A failure may occur after one of these modifications has been made, but before all of them are made. To ensure consistency despite failures, we have several recovery mechanisms.

A log is maintained on a stable storage media. The log is a sequence of log records, and maintains a record of update activities on the database. When transaction T_i starts, it registers itself by writing a

<T_i **start**>log record

Before T_i executes **write**(X), a log record <T_i, X, V_1, V_2> is written, where V_1 is the value of X before the write (undo value), and V_2 is the value to be written to X *(redo value).*

Notes

1. Log record notes that T_i has performed a write on data item ***X***. ***X*** had value V_1 before the write, and will have value V_2 after the write.
2. When T_i finishes it last statement, the log record **<T_i commit>** is written. We assume for now that log records are written directly to a stable storage media (that is, they are not buffered).

Two approaches for recovery using logs are:

1. Deferred database modification.
2. Immediate database modification.

Deferred Database Modification

The deferred database modification scheme records all the modifications to the log, but defers all the writes to after partial commit. Let us assume that transactions execute serially, to simplify the discussion.

A transaction starts by writing <T_i *start*> record to log. A write(X) operation results in a log record <T_i, X, V > being written, where V is the new value for X. The write is not performed on X at this time, but is deferred. When T_i partially commits, <T_i commit> is written to the log. Finally, the log records are read and used to actually execute the previously deferred writes. During recovery after a crash, a transaction needs to be redone if both <T_i start> and<T_i commit> are there in the log. Redoing a transaction T_i (redoT_i) sets the value of all data items updated by the transaction to the new values. Crashes can occur while

1. The transaction is executing the original updates, or
2. While recovery action is being taken.

Example: Transactions T_1 and T_2 (T_1 executes before T_2):

T_1: read (X)	T_2: read (Z)
X= X"1000	Z= Z" 1000
Write (X)	write (Z)
read (Y)	
Y= Y + 1000	
write (Y)	

The following figure shows the log as it appears at three instances of time (Assuming that initial balance in X is 10,000/- Y is 8,000/- and Z has 20,000/-):

Figure 11.1

<T_1 start>	<T_1 start>	<T_1 start>
<T_1, X 9000>	<T_1, X 9000>	<T_1, X 9000>
<T_1, Y 9000>	<T_1, Y 9000>	<T_1, Y 9000>
<T_1, Commit>	<T_1, Commit>	
<T_2 start>	<T_2 start>	
<T_2 , Z 19000>	<T_2 , Z 19000>	
<T_2, Commit>		
(a)	(b)	(c)

Notes

If log on stable storage at the time of crash as per (a) (b) and (c) then in:

(a) No redo action needs to be performed.

(b) redo(T_1) must be performed since <T_1 commit> is present

(c) redo(T_2) must be performed followed by redo(T_2) since

<T_1 commit> and <T_2 commit> are present.

Notes This sequence of redo operation as suggested in (c) any number of times, it will still bring the value of X, Y, Z to consistent redo values. This property of the redo operation is called idempotent.

Immediate Database Modification

The immediate database modification scheme allows database updates on the stored database even of an uncommitted transaction. These updates are made as the writes are issued (since undoing may be needed, update logs must have both the old value as well as the new value). Updated log records must be written before database item is written (assume that the log record is output directly to a stable storage and can be extended to postpone log record output, as long as prior to execution of an output (Y) operation for a data block Y all log records corresponding to items Y must be flushed to stable storage).

Output of updated blocks can take place at any time before or after transaction commit. Order in which blocks are output can be different from the order in which they are written.

Example:

Log	Write operation	Output
<T_1 start>		
<T_1, X, 10000, 9000>		
T_1, Y, 8000, 9000		
	X = 9000	Output Block of X
	Y = 9000	Output Block of Y
<T_1 commit>		
<T_2 start>		
<T_2, Z, 20,000, 19,000>		
	Z= 19000	Output Block of Z
<T_2 commit>		

The recovery procedure in such has two operations instead of one:

1. ***undo***(T_i) restores the value of all data items updated by T_i to their old values, moving backwards from the last log record for T_i ,

2. ***redo***(T_i) sets the value of all data items updated by T_i to the new values, moving forward from the first log record for T_i.

Both operations are idempotent, that is, even if the operation is executed multiple times the effect is the same as it is executed once. (This is necessary because operations may have to be re-executed during recovery).

Notes

When recovering after failure:

1. Transaction T_i needs to be undone if the log contains the record

   <T_i start>, but does not contain the record <T_i commit>.

2. Transaction T_i needs to be redone if the log contains both the record <T_i **start**> and the record <T_i **commit**>.

Undo operations are performed first, then redo operations.

Example: Consider the log as it appears at three instances of time.

(a)	(b)	(c)
<T_1 start>	<T_1 start>	<T_1 start>
<T_1, X 10000, 9000>	<T_1, X 10000, 9000>	<T_1, X 10000, 9000>
<T_1, Y 8000, 9000>	<T_1, Y 8000, 9000>	<T_1, Y 8000, 9000>
	<T_1, Commit>	<T_1, Commit>
	<T_2 start>	<T_2 start>
	<T_2 , Z 20000, 19000>	<T_2 , Z 20000, 19000>
		<T_2, Commit>

Recovery actions in each case above are:

1. ***undo (T_1):*** Y is restored to 8000 and X to 10000.
2. ***undo (T_2) and redo (T_1):*** Z is restored to 20000, and then X and Y are set to 9000 and 9000 respectively.
3. ***redo (T_1) and redo (T_2):*** X and Y are set to 9000 and 9000 respectively. Then Z is set to 19000

Checkpoints

The following problem occurs during recovery procedure:

Searching the entire log is time-consuming as we are not aware of the consistency of the database after restart. Thus, we might unnecessarily redo transactions, which have already output their updates to the database.

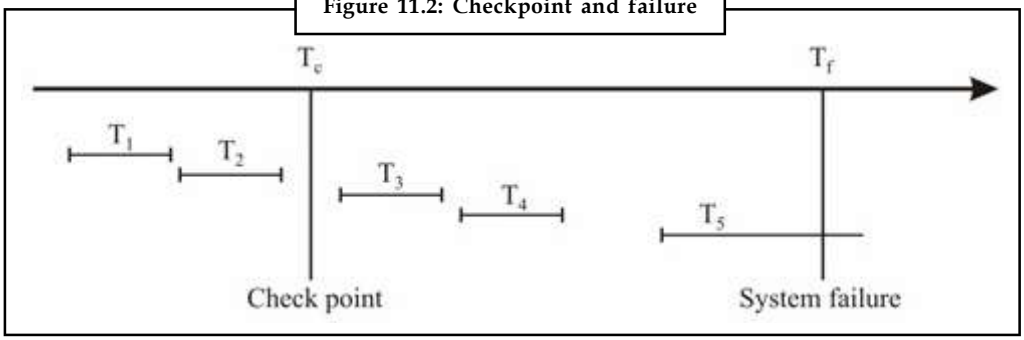
Thus, we can streamline recovery procedure by periodically performing check pointing. Check pointing involves:

1. Output of all the log records currently residing in the non-volatile memory onto stable storage.
2. Output all modified buffer blocks to the disk.
3. Write a log record < checkpoint> on a stable storage.

During recovery we need to consider only the most recent transactions that started before the checkpoint and is not completed till, checkpoint and transactions started after check point. Scan backwards from end of log to find the most recent **<checkpoint>** record. Continue scanning backwards till a record <T_i **start**> is found. Need only consider part of the log following above **star**t record. The earlier part of the log may be ignored during recovery, and can be erased whenever desired. For all transactions (starting from T_i or later) with no <T_i **commit**>, execute **undo** (T_i). (Done only in case immediate modification scheme is used). Scanning forward in the log, for all transactions starting from T_i or later with a <T_i **commit**>, execute **redo**(T_i).

Notes

Figure 11.2: Checkpoint and failure



1. T_1 and T_2 can be ignored (updates already output to disk due to checkpoint)
2. T_3 and T_4 are redone.
3. T_5 is undone.

Task When you can use check points?

11.6 Recovery with Concurrent Transactions

We can modify log-based recovery schemes to allow multiple transactions to execute concurrently. All transactions share a single disk buffer and a single log. A buffer block can have data items updated by one or more transactions. We assume concurrency control using strict two-phase locking; logging is done as described earlier. The checkpointing technique and actions taken on recovery have to be changed since several transactions may be active when a checkpoint is performed.

Checkpoints are performed as before, except that the checkpoint log record is now of the form

< checkpoint L>

where L is the list of transactions active at the time of the checkpoint. We assume no updates are in progress while the checkpoint is carried out. When the system recovers from a crash, it first does the following:

1. Initialises undo-list and redo-list to empty
2. Scans the log backwards from the end, stopping when the first <checkpoint L> record is found.

For each log record found during the backward scan:

1. If the record contains <T_i commit>, add T_i to *redo-list.*
2. If the record contains <T_i start>, then if T_i is not in *redo-list*, add T_i to *undo-list*
3. For every T_i in *L*, if T_i is not in *redo-list*, add T_i to *undo-list.*

At this point undo-list consists of incomplete transactions, which must be undone, and redo-list consists of finished transactions that must be redone.

Recovery now continues as follows:

Scan log backwards from most recent record, stopping when <T_i start> records have been encountered for every T_i in undo-list. During the scan, perform undo for each log record that belongs to a transaction in undo-list. Locate the most recent <checkpoint L> record. Scan log

Notes

forwards from the <checkpoint L> record till the end of the log. During the scan, perform redo for each log record that belongs to a transaction on redo-list.

SQL does not have very specific commands for recovery but, it allows explicit COMMIT, ROLLBACK and other related commands.

11.7 Buffer Management

When the database is updated, a lot of records are changed in the buffers allocated to the log records, and database records. Although buffer management is the job of the operating system, however, some times the DBMS prefer buffer management policies of their own.

Log Record Buffering

Log records are buffered in the main memory, instead of being output directly to a stable storage media. Log records are output to a stable storage when a block of log records in the buffer is full, or a log force operation is executed. Log force is performed to commit a transaction by forcing all its log records (including the commit record) to stable storage. Several log records can thus be output using a single output operation, reducing the I/O cost.

The rules below must be followed if log records are buffered:

1. Log records are output to stable storage in the order in which they are created.
2. Transaction T_i enters the commit state only when the log record <T_i commit> has been output to stable storage.
3. Before a block of data in the main memory is output to the database, all log records pertaining to data in that block must be output to a stable storage.

These rules are also called the write-ahead logging scheme.

Database Buffering

The database maintains an in-memory buffer of data blocks, when a new block is needed, if the buffer is full, an existing block needs to be removed from the buffer. If the block chosen for removal has been updated, even then it must be output to the disk. However, as per write-ahead logging scheme, a block with uncommitted updates is output to disk, log records with undo information for the updates must be output to the log on a stable storage. No updates should be in progress on a block when it is output to disk. This can be ensured as follows:

1. Before writing a data item, the transaction acquires exclusive lock on block containing the data item.
2. Lock can be released once the write is completed. (Such locks held for short duration are called latches).
3. Before a block is output to disk, the system acquires an exclusive latch on the block (ensures no update can be in progress on the block).

A database buffer can be implemented either, in an area of real main-memory reserved for the database, or in the virtual memory. Implementing buffer in reserved main-memory has drawbacks. Memory is partitioned before-hand between database buffer and applications, thereby, limiting flexibility. Although the operating system knows how memory should be divided at any time, it cannot change the partitioning of memory.

Notes

Database buffers are generally implemented in virtual memory in spite of drawbacks. When an operating system needs to evict a page that has been modified, to make space for another page, the page is written to swap space on disk. When the database decides to write buffer page to disk, buffer page may be in swap space, and may have to be read from swap space on disk and output to the database on disk, resulting in extra I/O, Known as dual paging problem. Ideally when swapping out a database buffer page, the operating system should handover the control to the database, which in turn outputs page to database instead of to swap space (making sure to output log records first) dual paging can thus be avoided, but common operating systems do not support such functionality.

11.8 Failure with Loss of Non-volatile Storages

Recall that the responsibility for the atomicity and durability properties of ACID transaction lies in the recovery component of the DBMS. For recovery purpose it is necessary to distinguish between two types of storage:

1. Volatile storage
2. Non-volatile storage

Volatile Storage: Volatile storage such as main memory, whose state is lost in the event of a system crash or power outage.

Non-volatile storage: Non-volatile storage such as magnetic disks or tapes, whose contents persists across such events.

The recovery subsystem is relied upon to ensure correct operation in the presence of three different types of failures.

1. ***Transaction Failure:*** When a transaction that is in progress reaches a state from which it cannot successfully commit, all updates that it made must be removed from the database in order to preserve the atomicity property. This is known as transaction rollback.
2. ***System Failure:*** If the system fails in a way that causes the loss of volatile memory contents. Recovery must ensure that:
   (a) The updates of all transactions that had committed prior to the crash are reflected in the database.
   (b) All updates of other transactions are removed from the database.
3. ***Media Failure:*** In the event that data is lost corrupted on the non-volatile storage (e.g., due to a disk head crash) then the on-line version of data is lost. In this case the database must be restored from an archival version of the database and brought upto date using operation logs.

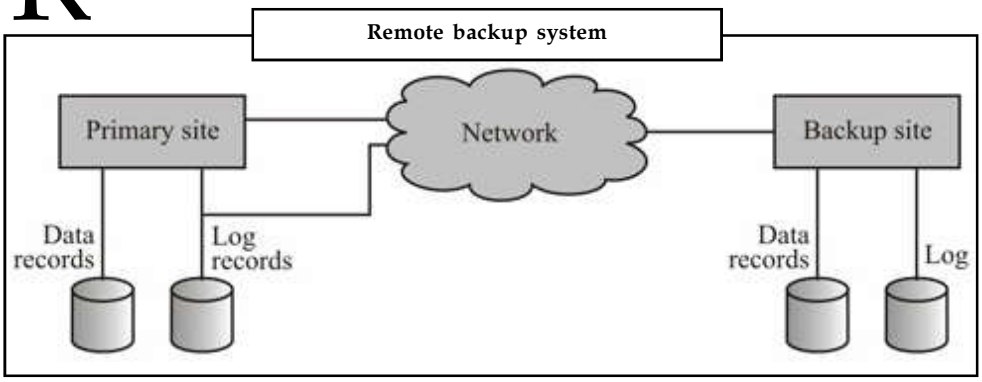
Task Suppose buffer is full after that what the process to store data.

Notes

Case Study

Remote Backup Systems

Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.



Detection of Failure: Backup site must be able to detect when the primary site has failed. To distinguish primary site failure from link failure, we would need to maintain several communication links between the primary and remote backup.

Transfer of Control: To take over control, backup site first performs recovery using its copy of the database and all the log records it has received from the primary site.

Thus, completed transactions are redone and incomplete transactions are rolled back. When the backup site takes over processing, it becomes the new primary site in order to transfer control back to the old primary site. When it recovers the database, the old primary site receive redo logs from the old backup and apply all updates locally.

Time to recover: To reduce delay in takeover, backup site periodically proceses the redo log records (in effect, performing recovery from previous database state), performs a checkpoint, and can then delete earlier parts of the log.

Hot-Spare configuration permits very fast takeover, backup continually processes redo log record as they arrive, applying the updates locally. When failure of the primary site is detected, the backup rolls back incomplete transactions, and is ready to process new transactions.

Alternative to remote backup are distributed databases with replicated data. Remote backup is faster and cheaper, but has a lower tolerance level for detecting failure.

The hot-spare configuration ensures durability of updates by delaying transaction committed until update is logged at backup. You can avoid this delay by permitting lower degrees of durability.

One-safe: A transaction commits in this scheme as soon as transaction's commit log record is written at the primary site. The problem in such a case is that, updates may not arrive at backup site before it takes over.

Two-very-safe: A transaction commits when transaction's commit log record is written at both sites primary and backup. Although this reduces the availability of the data base since transactions cannot commit if either site fails.

Contd...

Notes

Two-safe: This commit protocol proceeds if both the primary and backup site are active. However, if only the primary is active, the transaction commits as soon as its commit log record is written at the primary site. This protocol provides better availability than two-very-safe, as it avoids problems of lost transactions in one-safe.

11.9 Summary

- The recovery mechanism is needed in database system to take care of failures.
- One Formatted: Centered can use either the log based or page based recovery schemes.
- Buffer management is an important issue for DBMS as it affects the process of recovery.
- Remote backup systems allow a copy of complete data.

11.10 Keywords

Deferred database modification: The deferred database modification scheme records all the modifications to the log, but defers all the writes to after partial commit.

Disk failure: A disk block loses its content as a result of either a head crash or failure during a data transfer operation.

Immediate database modification: The immediate database modification scheme allows database updates on the stored database even of an uncommitted transaction.

Log records: Log records are buffered in the main memory, instead of being output directly to a stable storage media.

Recovery: Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures.

Storage media: Storage media can be distinguished by their relative speed, capacity, and resilience to failure, and classified as volatile storage or nonvolatile storage.

System crash: There is a hardware malfunction, or a bug in the database soft ware or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt.

11.11 Self Assessment

Fill in the blanks:

1. A transaction may fail because of hardware or a ..... failure.
2. The modifications made to the data is called .....
3. ..... systems guarantee that the failure of a single disk will not result in loss of data.
4. .....failure occurred in the midst of transfer, and the destination block has incorrect information.
5. RAM stands for .....
6. A database buffer can be implemented either, in an area of real main-memory reserved for the database, or in the .....
7. Non-volatile storage such as .....

Notes

8. The recovery manager, stores the modifications made onto a storage which does not react to system failures. Such storage is called .....
9. .....analyses the buffer pool to identify the active transactions and dirty pages.
10. ..... failure occurred sufficiently early during the transfer that the destination block remains intact.

11.12 Review Questions

1. Define recovery.
2. Describe ARIES.
3. What do you mean by transaction failure?
4. Distinguish between system crash and disk failure.
5. How will you implement stable-storage? Explain.
6. Describe log based recovery in detail.
7. Distinguish between deferred database modification and immediate database modification with the help of a suitable example.
8. Write short notes on:
   (a) Log record buffering
   (b) Database buffering
   (c) Check points
9. Distinguish between volatile and non-volatile storage.
10. Explain remote backup system.

Answers: Self Assessment

1.	software	2.	log
3.	RAID	4.	Partial
5.	Random Access Memory	6.	virtual memory
7.	magnetic disks or tapes	8.	stable storage
9.	Analysis phase	10.	Total

11.13 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management*, 7th Edition.

Notes

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 12: Query Processing and Optimization

CONTENTS

Objectives

Introduction

12.1 Query Processing: An Overview

12.1.1 Optimisation

12.1.2 Measure of Query Cost

12.2 Selection Operation

12.2.1 File Scan

12.2.2 Index Scan

12.2.3 Implementation of Complex Selections

12.2.4 Disjunction

12.2.5 Negation

12.3 Sorting

12.3.1 Create Sorted Partitions

12.3.2 Merging Partitions (N-way Merge)

12.3.3 Cost Analysis

12.4 Join Operation

12.4.1 Nested-loop Join

12.4.2 Block Nested-loop Join

12.4.3 Indexed Nested-loop Join

12.4.4 Merge-join

12.4.5 Hybrid Merge-join

12.4.6 Hash-join

12.4.7 Complex Joins

12.5 Evaluation of Expression

12.6 Creation of Query Evaluation Plans

12.7 Transformation of Relational Expressions

12.8 Estimating Statistics of Expression Results

12.9 Choice of Evaluation Plan

Contd...

Notes

12.10 Summary

12.11 Keywords

12.12 Self Assessment

12.13 Review Questions

12.14 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe query processing
- Realise selection operation and join operation
- State the creation of query evaluation plan
- Explain the choice of evaluation plan

Introduction

The Query Language – SQL is one of the main reasons of success of RDBMS. A user just needs to specify the query in SQL that is close to the English language and does not need to say how such query is to be evaluated. However, a query needs to be evaluated efficiently by the DBMS. But how is a query-evaluated efficiently? This unit attempts to answer this question. The unit covers the basic principles of query evaluation, the cost of query evaluation, the evaluation of join queries, etc. in detail. It also provides information about query evaluation plans and the role of storage in query evaluation and optimisation.

12.1 Query Processing: An Overview

Figure 12.1 shows the processing of an query.

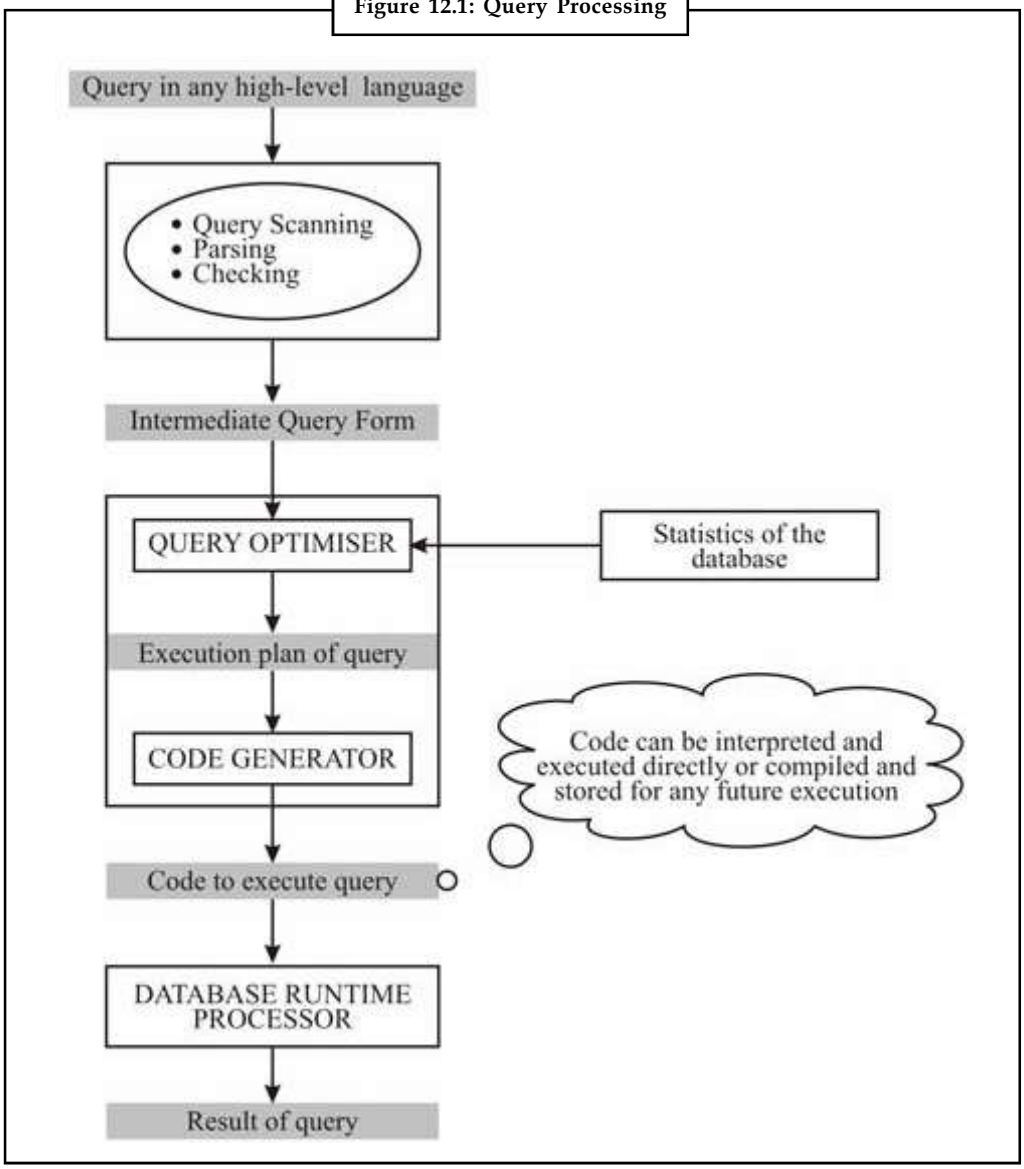
In the first step Scanning, Parsing, and Validating is done to translate the query into its internal form. This is then further translated into relational algebra (an intermediate query form). Parser checks syntax and verifies relations. The query then is optimised with a query plan, which then is compiled into a code that can be executed by the database runtime processor.

You can define query evaluation as the query-execution engine taking a query-evaluation plan, executing that plan, and returning the answers to the query. The query processing involves the study of the following concepts:

1. How to measure query costs?
2. Algorithms for evaluating relational algebraic operations.
3. How to evaluate a complete expression using algorithms on individual operations?

Notes

Figure 12.1: Query Processing



12.1.1 Optimisation

A relational algebra expression may have many equivalent expressions.

Example: $\sigma_{(salary < 5000)} (\pi_{salary} (EMP))$ is equivalent to $\pi_{salary} (\sigma_{salary <5000} (EMP))$.

Each relational algebraic operation can be evaluated using one of the several different algorithms. Correspondingly, a relational-algebraic expression can be evaluated in many ways.

An expression that specifies detailed evaluation strategy is known as evaluation-plan, for example, you can use an index on salary to find employees with salary < 5000 or we can perform complete relation scan and discard employees with salary $\geq$ 5000. The basis of selection of any of the scheme will be the cost.

Notes

Query Optimisation: Amongst all equivalent plans choose the one with the lowest cost. Cost is estimated using statistical information from the database catalogue, for example, number of tuples in each relation, size of tuples, etc.

Thus, in query optimisation we find an evaluation plan with the lowest cost. The cost estimation is made on the basis of heuristic rules.

Task Criteria for a new query creation in DBMS.

12.1.2 Measure of Query Cost

Cost is generally measured as total elapsed time for answering the query. There are many factors that contribute to time cost. These are disk accesses, CPU time, or even network communication.

Typically disk access is the predominant cost as disk transfer is a very slow event and is also relatively easy to estimate. It is measured by taking into account the following activities:

Number of seeks	×	average-seek-cost
Number of blocks read	×	average-block-read-cost
Number of blocks written	×	average-block-written-cost.

Notes The cost for writing a block is higher than the cost for reading a block.

This is due to the fact that the data is read back after being written to ensure that the write was successful. However, for the sake of simplicity we will just use number of block transfers from disk as the cost measure. We will also ignore the difference in cost between sequential and random I/O, CPU and communication costs. The I/O cost depends on the search criteria i.e., point/range query on an ordering/other fields and the file structures: heap, sorted, hashed. It is also dependent on the use of indices such as primary, clustering, secondary, B+ tree, multilevel, etc. There are other cost factors also, these may include buffering, disk placement, materialisation, overflow/free space management etc.

12.2 Selection Operation

The selection operation can be performed in a number of ways. Let us discuss the algorithms and the related cost of performing selection operations.

12.2.1 File Scan

These are the search algorithms that locate and retrieve records that fulfil a selection condition in a file. The following are the two basic files scan algorithms for selection operation:

1. ***Linear search***: This algorithm scans each file block and tests all records to see whether they satisfy the selection condition.

   The cost of this algorithm (in terms of block transfer) is defined as:

   Cost of searching records satisfying a condition = Number of blocks in a database = N_b.

   Cost for searching a key attribute value = Average number of block transfer for locating the value (on an average, half of the file needs to be traversed) so it is = $N_b/2$.

Notes

Linear search can be applied regardless of selection condition or ordering of records in the file, or availability of indices.

2. ***Binary search***: It is applicable when the selection is an equality comparison on the attribute on which file is ordered. Assume that the blocks of a relation are stored continuously then, cost can be estimated as:

   Cost = Cost of locating the first tuple by a binary search on the blocks + Sequence of other blocks that continue to satisfy the condition.

$$= [\log_2 (N_b)] + \frac{\text{Average number of tuples with the same value}}{\text{Blocking factor (Number of tuples in a block) of the relation}}$$

   These two values may be calculated from the statistics of the database.

12.2.2 Index Scan

Search algorithms that use an index are restricted because the selection condition must be on the search-key of the index.

1. (a) ***Primary index-scan for equality***: This search retrieves a single record that satisfies the corresponding equality condition. The cost here can be calculated as:

   Cost = Height traversed in index to locate the block pointer +1(block of the primary key is transferred for access).

   (b) ***Hash key:*** It retrieves a single record in a direct way thus, cost in hash key may also be considered as Block transfer needed for finding hash target +1

2. ***Primary index-scan for comparison***: Assuming that the relation is sorted on the attribute(s) that are being compared, (< , > etc.), then we need to locate the first record satisfying the condition after which the records are scanned forward or backward as the condition may be, displaying all the records. Thus cost in this case would be:

   Cost = Number of block transfer to locate the value in index + Transferring all the blocks of data satisfying that condition.

> *Notes* We can calculate roughly (from the cost point of view) the number of blocks satisfying the condition as:
>
> Number of values that satisfy the condition × Average number of tuples per attribute value/Blocking factor of the relation.

3. ***Equality on clustering index to retrieve multiple records***: The cost calculations in this case are somewhat similar to that of algorithm (primary index-scan for comparison).

   (a) *Equality on search-key of secondary index:* Retrieves a single record if the search-key is a candidate key.

   Cost = Cost of accessing index + 1.

   It retrieves multiple records if search-key is not a candidate key.

   Cost = Cost of accessing index + Number of records retrieved (It can be very expensive).

   Each record may be on a different block, thus, requiring one block access for each retrieved record (this is the worst case cost).

Notes

(b) *Secondary index, comparison*: For the queries of the type that use comparison on secondary index value $\geq$ a value, then the index can be used to find first index entry which is greater than that value, scan index sequentially from there till the end and also keep finding the pointers to records.

For the $\leq$ type query just scan leaf pages of index, also keep finding pointers to records, till first entry is found satisfying the condition.

In either case, retrieving records that are pointed to, may require an I/O for each record.

Notes Linear file scans may be cheaper if many records are to be fetched.

12.2.3 Implementation of Complex Selections

Conjunction: Conjunction is basically a set of AND conditions.

1. ***Conjunctive selection using one index:*** In such a case, select any algorithm given earlier on one or more conditions. If possible, test other conditions on these tuples after fetching them into memory buffer.
2. ***Conjunctive selection using multiple-key index***: Use appropriate composite (multiple-key) index if they are available.
3. ***Conjunctive selection by intersection of identifiers*** requires indices with record pointers. Use corresponding index for each condition, and take the intersection of all the obtained sets of record pointers. Then fetch records from file if, some conditions do not have appropriate indices, test them after fetching the tuple from the memory buffer.

Task Discuss the uses of hash key.

12.2.4 Disjunction

Specifies a set of OR conditions.

Disjunctive selection by union of identifiers is applicable if all conditions have available indices, otherwise use linear scan. Use corresponding index for each condition, take the union of all the obtained sets of record pointers, and eliminate duplicates, then fetch data from the file.

12.2.5 Negation

Use linear scan on file. However, if very few records are available in the result and an index is applicable on attribute, which is being negated, then find the satisfying records using index and fetch them from the file.

12.3 Sorting

Now we need to take a look at sorting techniques that can be used for calculating costing. There are various methods that can be used in the following ways:

1. Use an existing applicable ordered index (e.g., B+ tree) to read the relation in sorted order.

Notes

2. Build an index on the relation, and then use the index to read the relation in sorted order. (Options 1&2 may lead to one block access per tuple).
3. For relations that fit in the memory, techniques like quicksort can be used.
4. For relations that do not fit in the memory, external sort-merge is a good choice.

Let us go through the algorithm for External Sort-Merge.

12.3.1 Create Sorted Partitions

Let i be 0 initially.

Repeat steps (1) to (4) until the end of the relation:

1. Read M blocks of relation into the memory. (Assumption M is the number of available buffers for the algorithm).
2. Sort these buffered blocks using internal sorting.
3. Write sorted data to a temporary file – temp (i)
4. i = i + 1;

Let the final value of i be denoted by N;

Notes Each temporary file is a sorted partition.

12.3.2 Merging Partitions (N-way Merge)

// We assume (for now) that N < M.

// Use N blocks of memory to buffer temporary files and 1 block to buffer output.

Read the first block of each temporary file (partition) into its input buffer block;

Repeat steps (1) to (5) until all input buffer blocks are empty;

1. Select the first record (in sort order) among all input buffer blocks;
2. Write the record to the output buffer block;
3. **If** the output buffer block is full then write it to disk and empty it for the next set of data. This step may be performed automatically by the Operating System;
4. Delete the record from its input buffer block;
5. **If** the buffer block becomes empty **then** read the next block (if any) of the temporary file into the buffer.

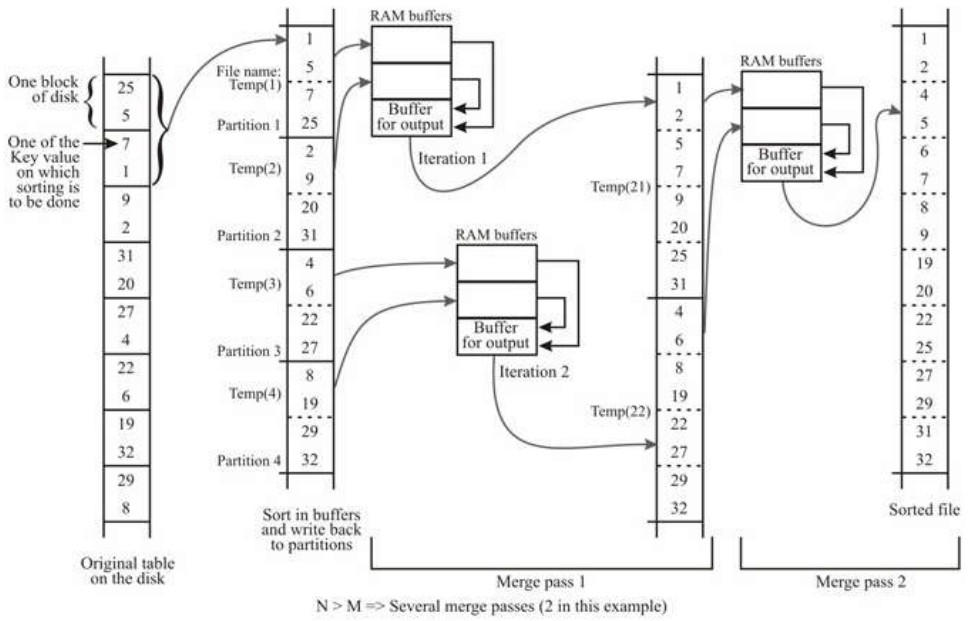
If N ≥ M, several merge passes are required, in each pass, contiguous groups of M × 1 partitions are merged and a pass reduces the number of temporary files temp (i) by a factor of M –1. For example, if M=11 and there are 90 temporary files, one pass reduces the number of temporary files to 9, each temporary file begin 10 times the size of the earlier partitions.

Repeated passes are performed till all partitions have been merged into one.

Figure 12.2 shows an example for external sorting using sort-merge.

Notes

Figure 12.2: External Merge-sort Example



12.3.3 Cost Analysis

Cost Analysis is may be performed, according to the following activities:

1. Assume the file has a total of Z blocks.
2. Z block input to buffers + Z block output – for temporary file creation.
3. Assuming that $N \geq M$, then a number of merge passes are required
4. Number of merge passes = $[\log_{M-1}(Z/M)]$. Please note that of M buffers 1 is used for output.
5. So number of block transfers needed for merge passes $=2\times Z\ ([\log_{M-1}(Z/M)])$ as all the blocks will be read and written back of the buffer for each merge pass.
6. Thus, the total number of passes for sort-merge algorithm $= 2Z + 2Z\ ([\log_{M-1}(Z/M)]) = 2Z \times ([\log_{M-1}(Z/M)]+1)$.

Task Where you can use negation? Discuss.

12.4 Join Operation

There are number of algorithms that can be used to implement joins:

1. Nested-loop join
2. Block nested-loop join
3. Indexed nested-loop join

Notes

4. Merge-join
5. Hash-join
6. Complex join.

The choice of join algorithm is based on the cost estimate. Let us use the following information to elaborate the same:

MARKS (enroll no, subject code, marks): 10000 rows, 500 blocks

STUDENT (enroll no, name, dob): 2000 rows, 100 blocks

12.4.1 Nested-loop Join

Let us show the algorithm for the given relations.

To compute the theta or equi-join

```
for each tuple s in STUDENT
{
        for each tuple m in MARKS
        {
                test s.enrolno = m.enrolno to see if they satisfy the join
                condition if they do, output joined tuple to the result.
        };
};
```

1. In the nested loop join there is one outer relation and one inner relation.
2. It does not use or require indices. It can be used with any kind of join condition. However, it is expensive as it examines every pair of tuples in the two relations.
3. If there is only enough memory to hold one block of each relation, the number of disk accesses can be calculated as:

For each tuple of STUDENT, all the MARKS tuples (blocks) that need to be accessed.

However, if both or one of the relations fit entirely in the memory, block transfer will be needed only once, so the total number of transfers in such a case, may be calculated as:

= Number of blocks of STUDENT + Number of blocks of MARKS

= 100 + 500 = 600.

If only the smaller of the two relations fits entirely in the memory then use that as the inner relation and the bound still holds.

Cost for the worst case:

Number of tuples of outer relation × Number of blocks of inner relation + Number of blocks of outer relation.

2000 × 500 + 100 = 1,000,100 with STUDENT as outer relation.

There is one more possible bad case when MARKS is on outer loop and STUDENT in the inner loop. In this case, the number of Block transfer will be:

10000 × 100 + 500 = 1,000,500 with MARKS as the outer relation.

Notes

12.4.2 Block Nested-loop Join

This is a variant of nested-loop join in which a complete block of outer loop is joined with the block of inner loop.

The algorithm for this may be written has:

```
for each block s of STUDENT
{
        for each block m of MARKS
        {
                for each tuple si in s
                {
                        for each tuple mi in m
                        {
                                Check if (si and mi) satisfy the join condition
                                if they do output joined tuple to the result
                        };
                };
        };
};
```

Worst case of block accesses in this case = Number of Blocks of outer relation (STUDENT) ×Number of blocks of inner relation (MARKS) + Number of blocks of outer relation (STUDENT).

Best case: Blocks of STUDENT + Blocks of MARKS

Number of block transfers assuming worst case:

100 × 500 + 100 = 50,100 (much less than nested-loop join)

Number of block transfers assuming best case:

400 + 100 = 500 (same as with nested-loop join)

Improvements to Block Nested-loop Algorithm

The following modifications improve the block Nested method:

Use M – 2 disk blocks as the blocking unit for the outer relation, where M = memory size in blocks.

Use one buffer block to buffer the inner relation.

Use one buffer block to buffer the output.

This method minimizes the number of iterations.

12.4.3 Indexed Nested-loop Join

Index scans can replace file scans if the join is an equi-join or natural join, and an index is available on the inner relation's join attribute.

For each tuple si in the outer relation STUDENT, use the index to look up tuples in MARKS that satisfy the join condition with tuple si.

In a worst case scenarios, the buffer has space for only one page of STUDENT, and, for each tuple in MARKS, then we should perform an index lookup on MARKS index.

Notes

Worst case: Block transfer of STUDENT+ number of records in STUDENT * cost of searching through index and retrieving all matching tuples for each tuple of STUDENT.

If a supporting index does not exist than it can be constructed as and when needed.

If indices are available on the join attributes of both STUDENT and MARKS, then use the relation with fewer tuples as the outer relation.

Example: Compute the cost for STUDENT and MARKS join, with STUDENT as the outer relation. Suppose MARKS has a primary B+-tree index on enroll no, which contains 10 entries in each index node. Since MARKS has 10,000 tuples, the height of the tree is 4, and one more access is needed to the actual data. The STUDENT has 2000 tuples. Thus, the cost of indexed nested loops join as:

$$100 + 2000 \times 5 = 10{,}100 \text{ disk accesses}$$

12.4.4 Merge-join

The merge-join is applicable to equi-joins and natural joins only. It has the following process:

1. Sort both relations on their join attribute (if not already sorted).
2. Merge the sorted relations to join them. The join step is similar to the merge stage of the sort-merge algorithm, the only difference lies in the manner in which duplicate values in join attribute are treated, i.e., every pair with same value on join attribute must be matched.

Figure 12.3: Sample Relations for Computing Join

STUDENT

Enrol no	Name	-----
1001	Ajay	-----
1002	Aman	-----
1005	Raman	-----
1100	Raman	-----
------	------	------

MARKS

Enrol no	Subject code	Marks
1001	MM-01	55
1001	FM-02	75
1002	MA-03	90
1005	CB-04	75
-------	------	-------

The number of block accesses:

Each block needs to be read only once (assuming all tuples for any given value of the join attributes fit in memory). Thus number of block accesses for merge-join is:

Blocks of STUDENT + Blocks of MARKS + The cost of sorting if relations are unsorted.

12.4.5 Hybrid Merge-join

This is applicable only when the join is an equi-join or a natural join and one relation is sorted and the other has a secondary B+-tree index on the join attribute.

The algorithm is as follows:

Merge the sorted relation with the leaf entries of the B+-tree. Sort the result on the addresses of the unsorted relation's tuples. Scan the unsorted relation in physical address order and merge with the previous results, to replace addresses by the actual tuples. Sequential scan in such cases is more efficient than the random lookup method.

Notes

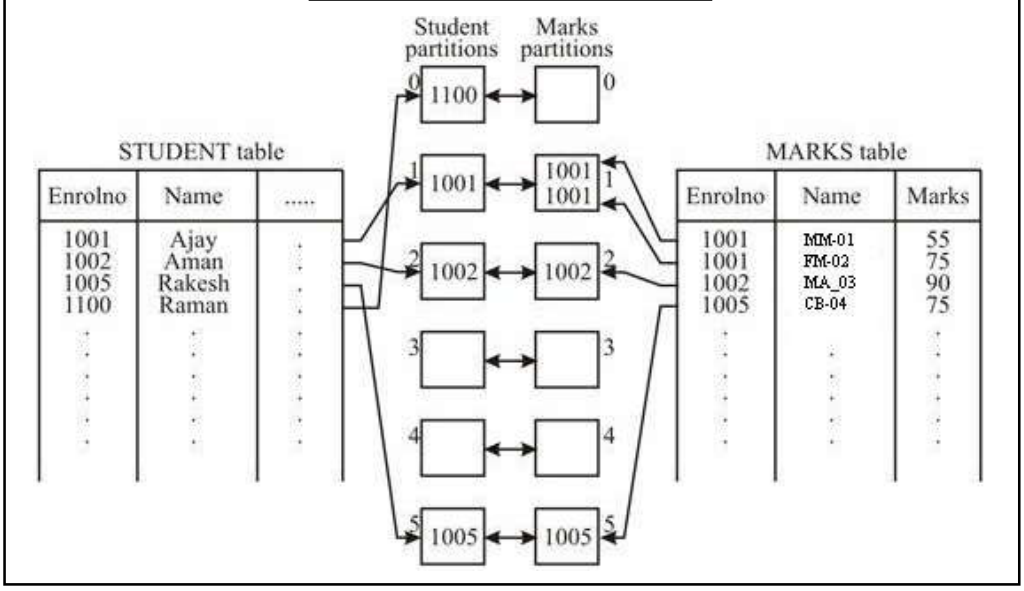
12.4.6 Hash-join

This is applicable to both the equi-joins and natural joins. A hash function h is used to partition tuples of both relations, where h maps joining attribute (enroll no in our example) values to {0, 1, ..., n-1}.

The join attribute is hashed to the join-hash partitions. In the example of Figure 12.4 we have used mod 100 function to hashing, and n = 100.

Error!

Figure 12.4: A Hash-join Example



Once the partition tables of STUDENT and MARKS are made on the enrolment number, then only the corresponding partitions will participate in the join as:

A STUDENT tuple and a MARKS tuple that satisfy the join condition will have the same value for the join attributes. Therefore, they will be hashed to equivalent partition and thus can be joined easily.

Algorithm for Hash-join

The hash-join of two relations r and s is computed as follows:

1. Partition the relation r and s using hashing function h. (When partitioning a relation, one block of memory is reserved as the output buffer for each partition).
2. For each partition si of s, load the partition into memory and build an in-memory hash index on the join attribute.
3. Read the tuples in ri from the disk, one block at a time. For each tuple in ri locate each matching tuple in si using the in-memory hash index and output the concatenation of their attributes.

In this method, the relation s is called the build relation and r is called the probe relation. The value n (the number of partitions) and the hash function h is chosen in such a manner that each si should fit in to the memory. Typically n is chosen as Number of blocks of s/Number of

Notes

memory buffers] *f (M) where f is a "fudge factor", typically around 1.2. The probe relation partitions ri need not fit in memory.

Average size of a partition si will be less than M blocks using the formula for n as above thereby allowing room for the index. If the build relation s is very huge, then the value of n as given by the above formula may be greater than M" 1 i.e., the number of buckets is > the number of buffer pages. In such a case, the relation s can be recursively partitioned, instead of partitioning n ways, use M – 1 partitions for s and further partition the M – 1 partitions using a different hash function. You should use the same partitioning method on r. This method is rarely required, as recursive partitioning is not needed for relations of 1GB or less for a memory size of 2MB, with block size of 4KB.

Cost calculation for Simple Hash-join

1. Cost of partitioning r and s: all the blocks of r and s are read once and after partitioning written back, so cost 1 = 2 (blocks of r + blocks of s).
2. Cost of performing the hash-join using build and probe will require at least one block transfer for reading the partitions

   Cost 2 = (blocks of r + blocks of s)
3. There are a few more blocks in the main memory that may be used for evaluation, they may be read or written back. We ignore this cost as it will be too less in comparison to cost 1 and cost 2.

   Thus, the total cost = cost 1 + cost 2

   = 3 (blocks of r + blocks of s)

Cost of Hash-Join requiring Recursive Partitioning

The cost of partitioning in this case will increase to number of recursion required, it may be calculated as:

Number of iterations required = ([$\log_{M-1}$ (blocks of s) ×1)

Thus, cost 1 will be modified as:

= 2 (blocks of r + blocks of s) × ([$\log_{M-1}$ (blocks of s)] ×1)

The cost for step (2) and (3) here will be the same as that given in steps (2) and (3) above.

Thus, total cost = 2(blocks of r + blocks of s) ([$\log_{M-1}$(blocks of s) – 1]) + (blocks of r + blocks of s).

Because s is in the inner term in this expression, it is advisable to choose the smaller relation as the build relation. If the entire build input can be kept in the main memory, n can be set to 1 and the algorithm need not partition the relations but may still build an in-memory index, in such cases the cost estimate goes down to (Number of blocks r + Number of blocks of s).

Handling of Overflows

Even if s is recursively partitioned hash-table overflow can occur, i.e., some partition si may not fit in the memory. This may happen if there are many tuples in s with the same value for join attributes or a bad hash function.

Notes

Partitioning is said to be skewed if some partitions have significantly more tuples than the others. This is the overflow condition. The overflow can be handled in a variety of ways:

1. ***Resolution (during the build phase):*** The overflow partition s is further partitioned using different hash function. The equivalent partition of r must be further partitioned similarily.
2. ***Avoidance (during build phase):*** Partition build relations into many partitions, then combines them.

However, such approaches for handling overflows fail with large numbers of duplicates. One option of avoiding such problems is to use block nested-loop join on the overflowed partitions.

Let us explain the hash join and its cost for the natural join STUDENT ⋈ MARKS

Assume a memory size of 25 blocks ⇒ M=25;

SELECT build s as STUDENT as it has less number of blocks (100 blocks) and r probe as MARKS (500 blocks).

Number of partitions to be created for STUDENT = (block of STUDENT/M)* fudge factor (1.2) = (100/25) × 1.2 = 4.8

Thus, STUDENT relation will be partitioned into 5 partitions of 20 blocks each. MARKS will also have 5 partitions of 100 blocks each. The 25 buffers will be used as –20 blocks for one complete partition of STUDENT plus 4 more blocks for one block of each of the other 4 partitions. One block will be used for input of MARKS partitions.

The total cost = 3(100+500) = 1800 as no recursive partitioning is needed.

Hybrid Hash-join

This is useful when the size of the memory is relatively large, and the build input is larger than the memory. Hybrid hash join keeps the first partition of the build relation in the memory. The first partition of STUDENT is maintained in the first 20 blocks of the buffer, and not written to the disk. The first block of MARKS is used right away for probing, instead of being written and read back. Thus, it has a cost of 3(80 + 400) + 20 +100 = 1560 block transfers for hybrid hash-join, instead of 1800 with plain hash-join.

Hybrid hash-join is most useful if M is large, such that we can have bigger partitions.

12.4.7 Complex Joins

A join with a conjunctive condition can be handled, either by using the nested loop or block nested loop join, or alternatively, the result of one of the simpler joins (on a few conditions) can be computed and the final result may be evaluated by finding the tuples that satisfy the remaining conditions in the result.

A join with a disjunctive condition can be calculated either by using the nested loop or block nested loop join, or it may be computed as the union of the records in individual joins.

Task Where you can use merge join? Explain

Notes

12.5 Evaluation of Expression

Before we discuss the evaluation of a query expression, let us briefly explain how a SQL query may be represented. Consider the following student and marks relations:

STUDENT (enrolno, name, phone)

MARKS (enrolno, subjectcode, grade)

To find the results of the student(s) whose phone number is '1129250025', the following query may be given.

SELECT enrolno, name, subjectcode, grade

FROM STUDENT s, MARKS m

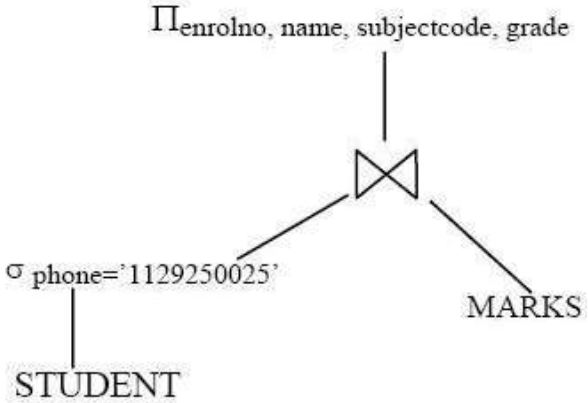
WEHRE s.enrolno=m.enrolno AND phone= '1129250025'

The equivalent relational algebraic query for this will be:

$$\pi_{\text{enrolno, name, subjectcode}} (\sigma_{\text{phone='1129250025'}} (\text{STUDENT}) \bowtie \text{MARKS})$$

This is a very good internal representation however, it may be a good idea to represent the relational algebraic expression to a query tree on which algorithms for query optimisation can be designed easily. In a query tree, nodes are the operators and relations represent the leaf. The query tree for the relational expression above would be:

Figure 12.5: A Sample Query Tree



In the previous section, we have seen the algorithms for individual operations. Now let us look at the methods for evaluating an entire expression. In general we use two methods:

1. Materialisation
2. Pipelining.

Materialisation

Evaluate a relational algebraic expression from the bottom-up, by explicitly generating and storing the results of each operation in the expression. For example, in Figure 12.5 compute and store the result of the selection operation on STUDENT relation, then take the join of this result with MARKS relation and then finally compile the projection operation.

Materialised evaluation is always possible though; the cost of writing/reading results to/from disk can be quite high.

Notes

Pipelining

Evaluate operations in a multi-threaded manner, (i.e., passes tuples output from one operation to the next parent operation as input) even as the first operation is being executed. In the previous expression tree, it does not store (materialise) results instead, it passes tuples directly to the join. Similarly, does not store results of join, and passes tuples directly to the projection. Thus, there is no need to store a temporary relation on a disk for each operation. Pipelining may not always be possible or easy for sort, hash-join.

One of the pipelining execution methods may involve a buffer filled by the result tuples of lower level operation while, records may be picked up from the buffer by the higher level operation.

Complex Joins

When an expression involves three relations then we have more than one strategy for the evaluation of the expression. For example, join of relations such as:

STUDENT $\bowtie$ MARKS $\bowtie$ SUBJECTS

may involve the following three strategies:

Strategy 1: Compute STUDENT $\bowtie$ MARKS; use result to compute result $\bowtie$ SUBJECTS

Strategy 2: Compute MARKS $\bowtie$ SUBJECTS first, and then join the result with STUDENT

Strategy 3: Perform the pair of joins at the same time. This can be done by building an index of enroll no in STUDENT and on subject code in SUBJECTS.

For each tuple m in MARKS, look up the corresponding tuples in STUDENT and the corresponding tuples in SUBJECTS. Each tuple of MARKS will be examined only once.

Strategy 3 combines two operations into one special-purpose operation that may be more efficient than implementing the joins of two relations.

12.6 Creation of Query Evaluation Plans

Generation of query-evaluation plans for an expression involves several steps:

1. Generating logically equivalent expressions using equivalence rules
2. Annotating resultant expressions to get alternative query plans
3. Choosing the cheapest plan based on estimated cost.

The overall process is called cost based optimisation.

The cost difference between a good and a bad method of evaluating a query would be enormous. We would therefore, need to estimate the cost of operations and statistical information about relations. For example a number of tuples, a number of distinct values for an attribute etc. Statistics helps in estimating intermediate results to compute the cost of complex expressions.

12.7 Transformation of Relational Expressions

Two relational algebraic expressions are said to be equivalent if on every legal database instance the two expressions generate the same set of tuples (order of tuples is irrelevant).

Notes

Let us define certain equivalence rules that may be used to generate equivalent relational expressions.

Equivalence Rules

1. The conjunctive selection operations can be equated to a sequence of individual selections. It can be represented as:

   $$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. The selection operations are commutative, that is,

   $$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last of the sequence of projection operations is needed, the others can be omitted.

   $$\pi_{\text{attriblist1}}(\pi_{\text{attriblist2}}(\pi_{\text{attriblist3}} \ldots\ldots(E) ..) = \pi_{\text{attriblist1}}(E)$$

4. The selection operations can be combined with Cartesian products and theta join operations.

   $$\sigma_{\theta 1}(E_1 \times E_2) = E_1 \bowtie_{\theta 1} E_2$$

   and

   $$\sigma_{\theta 2}(E_1 \bowtie_{\theta 1} E_2) = E_1 \bowtie_{\theta 2 \wedge \theta 1} E_2$$

5. The theta-join operations and natural joins are commutative.

   $$E1 \bowtie_{\theta} E2 = E2 \bowtie_{\theta} E1$$

6. The Natural join operations are associative. Theta joins are also associative but with the proper distribution of joining condition:

   $$(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$$

7. The selection operation distributes over the theta join operation, under conditions when all the attributes in selection predicate involve only the attributes of one of the expressions (E1) being joined.

   $$\sigma_{\theta 1}(E1 \bowtie_{\theta} E2) = (\sigma_{\theta 1}(E1)) \bowtie_{\theta} E2)$$

8. The projections operation distributes over the theta join operation with only those attributes, which are present in that relation.

   $$\pi_{\text{attrlist1} \cup \text{attrlist2}}(E1 \bowtie_{\theta} E2) = (\pi_{\text{attrlist1}}(E1) \bowtie_{\theta} \pi_{\text{attrlist2}}(E2))$$

9. The set operations of union and intersection are commutative. But set difference is not commutative.

   $E1 \cup E2 = E2 \cup E$ and similarly for the intersection.

10. Set union and intersection operations are also associative.

    $(E1 \cup E2) \cup E3 = E1 \cup (E2 \cup E3)$ and similarly for intersection.

11. The selection operation can be distributed over the union, intersection, and set-differences operations.

    $$\sigma_{\theta 1}(E1 - E2) = ((\sigma_{\theta 1}(E1) - (\sigma_{\theta 1}(E2))$$

Notes

12. The projection operation can be distributed over the union.

$$\pi_{attriblist1}(E1 \cup E2) = \pi_{attriblist1}(E1) \cup \pi_{attriblist1}(E2)$$

The rules as above are too general and a few heuristics rules may be generated from these rules, which help in modifying the relational expression in a better way. These rules are:

1. Combining a cascade of selections into a conjunction and testing all the predicates on the tuples at the same time:

$$\sigma_{\theta 2}(\sigma_{\theta 1}(E)) \text{ convert to } \sigma_{\theta 2 \wedge \theta 1}(E)$$

2. Combining a cascade of projections into single outer projection:

$$\pi_4(\pi_3(\ldots\ldots(E))) = \pi_4(E)$$

3. Commutating the selection and projection or vice-versa sometimes reduces cost
4. Using associative or commutative rules for Cartesian product or joining to find various alternatives.
5. Moving the selection and projection (it may have to be modified) before joins. The selection and projection results in the reduction of the number of tuples and therefore may reduce cost of joining.
6. Commuting the projection and selection with Cartesian product or union.

Task What do you mean by pipelining? Discuss

12.8 Estimating Statistics of Expression Results

Let us explain use of some of these rules with the help of an example. Consider the query for the relations:

STUDENT (enrollno, name, phone)

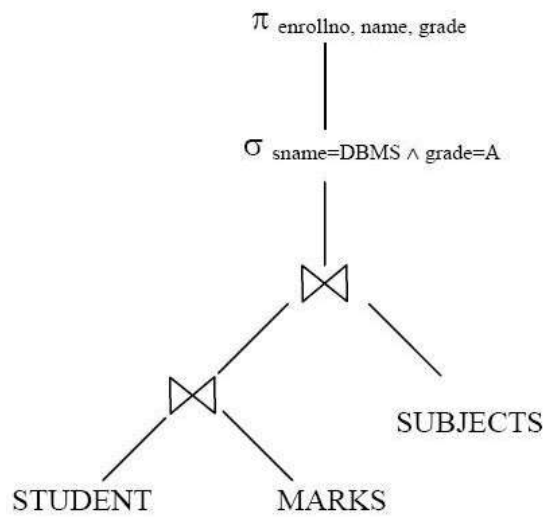
MARKS (enrollno, subjectcode, grade)

SUBJECTS (subjectcode, sname)

Consider the query: Find the enrolment number, name, and grade of those students who have secured an A grade in the subject DBMS. One of the possible solutions to this query may be:

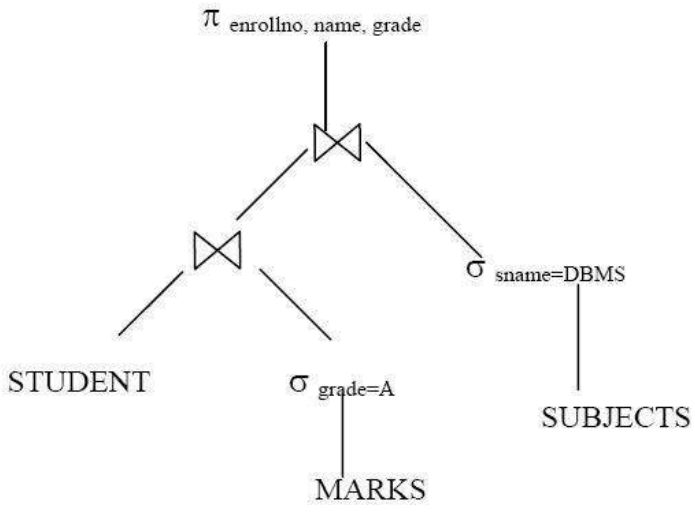
Notes

Figure 12.6



The selection condition may be moved to the join operation. The selection condition given in the Figure 12.6 is: sname = DBMS and grade = A. Both of these conditions belong to different tables, as s name is available only in the SUBJECTS table and grade in the MARKS table. Thus, the conditions of selection will be mapped accordingly as shown in the Figure 12.7. Thus, the equivalent expression will be:

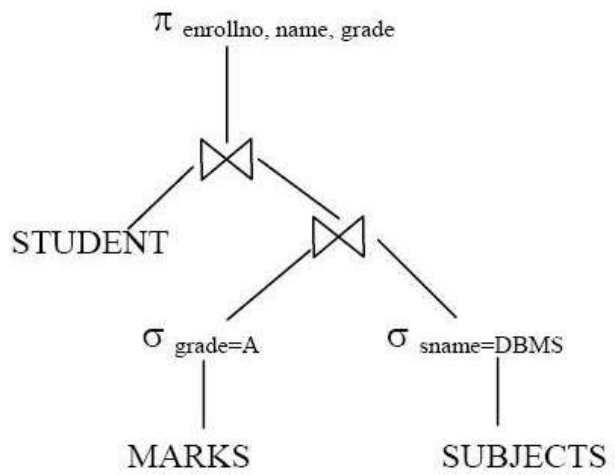
Figure 12.7



The expected size of SUBJECTS and MARKS after selection will be small so it may be a good idea to first join MARKS with SUBJECTS. Hence, the associative law of JOIN may be applied.

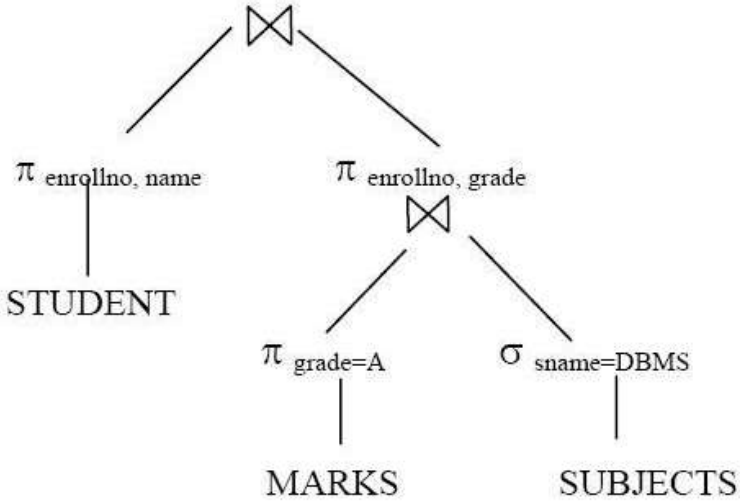
Notes

Figure 12.8



Finally projection may be moved inside. Thus the resulting query tree may be:

Figure 12.9



Obtaining Alternative Query Expressions

Query optimisers use equivalence rules to systematically generate expressions equivalent to the given expression. Conceptually, they generate all equivalent expressions by repeatedly executing the equivalence rules until no more expressions are to be found. For each expression found, use all applicable equivalence rules and add newly generated expressions to the set of expressions already found. However, the approach above is very expensive both in time space requirements. The heuristics rules given above may be used to reduce cost and to create a few possible but good equivalent query expression.

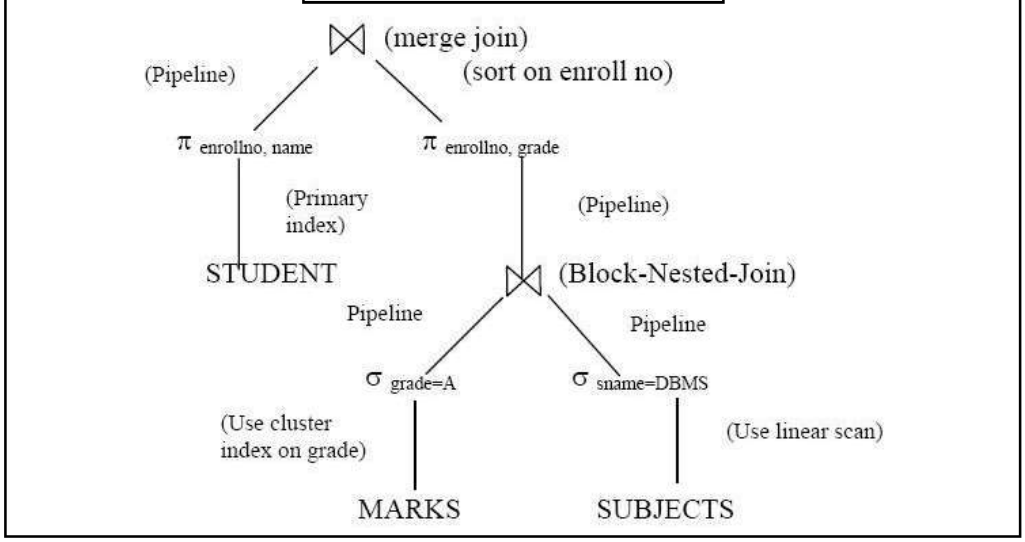
Notes

12.9 Choice of Evaluation Plan

Let us first define the term Evaluation Plan.

An evaluation plan defines exactly which algorithm is to be used for each operation, and how the execution of the operation is coordinated. For example, Figure 12.10 shows the query tree with evaluation plan.

Figure 12.10: Query Evaluation Plan



Choice of Evaluation Plans

For choosing an evaluation technique, we must consider the interaction of evaluation techniques. Please note that choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. For example, merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for a higher level aggregation. A nested-loop join may provide opportunity for pipelining. Practical query optimisers incorporate elements of the following two broad approaches:

1. Searches all the plans and chooses the best plan in a cost-based fashion.
2. Uses heuristic rules to choose a plan.

12.10 Summary

- In this unit, you have study query processing and evaluation.
- A query in a DBMS is a very important operation, as it needs to be efficient.
- Query processing involves query parsing, representing query in alternative forms, finding the best plan of evaluation of a query and then actually evaluating it.
- The major query evaluation cost is the disk access time.
- In this unit, we have discussed the cost of different operations in details.
- However, an overall query cost will not be a simple addition of all such costs.

Notes

12.11 Keywords

Index scan: Search algorithms that use an index are restricted because the selection condition must be on the search-key of the index.

Indexing: A database index is a data structure that improves the speed of operations on a database table.

Join: Join operation is considered as the real power behind the relational database implementations

Query cost: Cost is generally measured as total elapsed time for answering the query.

12.12 Self Assessment

Choose the appropriate answer:

1. RDBMS stands for:
   (a) Relative Database Management System
   (b) Relational Database Management System
   (c) Relative Document Management System
   (d) Relational Document Management System
2. CPU stands for
   (a) Central Processing Unit (b) Control Processing Unit
   (c) Central Power Unit (d) Central Power Utilization

Fill in the blanks:

3. Cost is generally measured as total elapsed time for answering the .....
4. The I/O cost depends on the .....
5. ..... is applicable when the selection is an equality comparison on the attribute on which file is ordered.
6. ..... search retrieves a single record that satisfies the corresponding equality condition.
7. Index scans can replace file scans if the join is an .....
8. The ..... is applicable to equi-joins and natural joins only.
9. A join with a ..... can be calculated either by using the nested loop or block nested loop join
10. ..... use equivalence rules to systematically generate expressions equivalent to the given expression.

12.13 Review Questions

1. Describe various differences between DBMS and RDBMS.
2. What do you mean by query processing? Explain
3. How will you measure query cost? Explain.

Notes

4. Differentiate between linear search and binary search.
5. What do you mean by sorting? Explain how will you create sorted partition.
6. Write short notes on:
   (a) Indexed nested-loop join
   (b) Merge-join
   (c) Hash-join
7. How will you calculate cost for simple hash-join? Explain
8. Explain complex join with the help of suitable example. If you need to create new table then create with virtual data.
9. Explain various rule of equivalence which one used in generation of equivalent relational expressions.
10. Explain various factors apply to choosing of evaluation plan.

Answers: Self Assessment

1.	(b)	2.	(a)
3.	query	4.	search criteria
5.	Binary search	6.	Primary index-scan for equality
7.	equi-join	8.	merge-join
9.	disjunctive condition	10.	Query optimisers

12.14 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 13: Parallel Databases

CONTENTS

Objectives
Introduction
13.1 Parallel Database
13.2 I/O Parallelism
13.2.1 Horizontal Partitioning
13.2.2 Vertical Partitioning
13.3 Inter-query Parallelism
13.4 Intra-query Parallelism
13.5 Inter-operation and Intra-operation Parallelism
13.6 Summary
13.7 Keywords
13.8 Self Assessment
13.9 Review Questions
13.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe parallel database
- Define I/O parallelism
- Explain inter- and intra-query parallelism
- Realise intra- and inter-operation parallelism

Introduction

In today's world, Client-Server applications that rely on a database on the server as a data store while servicing requests from multiple clients are quite commonplace. Most of these applications use a Relational Database Management System (RDBMS) as their data store while using an object oriented programming language for development. This causes a certain inefficiency as objects must be mapped to tuples in the database and vice versa instead of the data being stored in a way that is consistent with the programming model. The "impedance mismatch" caused by having to map objects to tables and vice versa has long been accepted as a necessary performance penalty.

13.1 Parallel Database

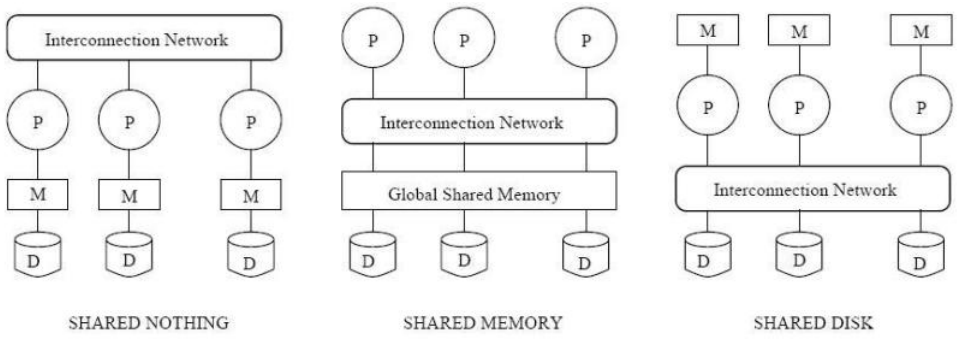
A parallel database system is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries. Although data may be stored in a distributed fashion in such a system, the distribution is governed solely by performance considerations.

Notes

Architectures of Parallel Database

The basic idea behind parallel databases is to carry out evaluation steps in parallel whenever possible, in order to improve performance. There are many opportunities for parallelism in a DBMS; databases represent one of the most successful instances of parallel computing.

Figure 13.1: Physical Architectures for Parallel Database Systems



Three main architectures have been proposed for building parallel DBMSs. In a shared-memory system, multiple CPUs are attached to an interconnection network and can access a common region of main memory. In a shared-disk system, each CPU has a private memory and direct access to all disks through an interconnection network.

In a shared-nothing system, each CPU has local main memory and disk space, but no two CPUs can access the same storage area; all communication between CPUs is through a network connection. The three architectures are illustrated in Figure 13.1.

The shared memory architecture is closer to a conventional machine, and many commercial database systems have been ported to shared memory platforms with relative ease. Communication overheads are low, because main memory can be used for this purpose, and operating system services can be leveraged to utilize the additional CPUs.

Although this approach is attractive for achieving moderate parallelism a few tens of CPUs can be exploited in this fashion memory contention becomes a bottleneck as the number of CPUs increases. The shared-disk architecture faces a similar problem because large amounts of data are shipped through the interconnection network.

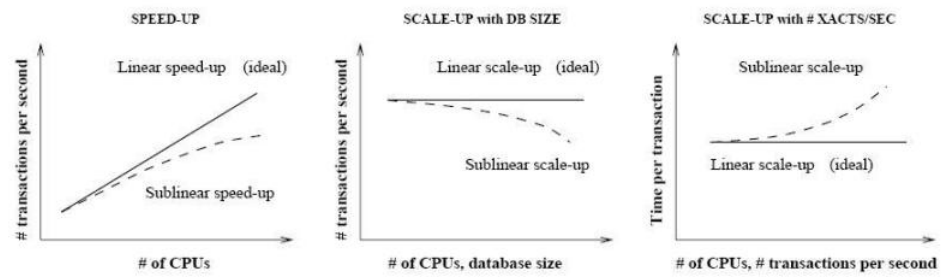
The basic problem with the shared-memory and shared-disk architectures is interference:

As more CPUs are added, existing CPUs are slowed down because of the increased contention for memory accesses and network bandwidth. It has been noted that even an average 1 percent slowdown per additional CPU means that the maximum speedup is a factor of 37, and adding additional CPUs actually slows down the system; a system with 1,000 CPUs is only 4 percent as effective as a single CPU system! This observation has motivated the development of the shared-nothing architecture, which is now widely considered to be the best architecture for large parallel database systems.

The shared-nothing architecture requires more extensive reorganization of the DBMS code, but it has been shown to provide linear speed-up, in that the time taken for operations decreases in proportion to the increase in the number of CPUs and disks, and linear scale-up, in that performance is sustained if the number of CPUs and disks are increased in proportion to the amount of data. Consequently, ever-more powerful parallel database systems can be built by taking advantage of rapidly improving performance for single CPU systems and connecting as many CPUs as desired.

Notes

Figure 13.2: Speed-up and Scale-up



Speed-up and scale-up are illustrated in Figure 13.2. The speed-up curves show how, for a fixed database size, more transactions can be executed per second by adding CPUs. The scale-up curves show how adding more resources (in the form of CPUs) enables us to process larger problems. The first scale-up graph measures the number of transactions executed per second as the database size is increased and the number of CPUs is correspondingly increased. An alternative way to measure scale-up is to consider the time taken per transaction as more CPUs are added to process an increasing number of transactions per second; the goal here is to sustain the response time per transaction.

Task Parallel database vs. distributed data base.

13.2 I/O Parallelism

Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with Decision Support Systems (DSS) and Data warehouses. Symmetric Multiprocessing (SMP), clustered systems, and Massively Parallel Systems (MPP) gain the largest performance benefits from parallel execution because statement processing can be split up among many CPUs on a single Oracle system. You can also implement parallel execution on certain types of Online Transaction Processing (OLTP) and hybrid systems.

Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time. An example of this is when 12 processes handle 12 different months in a year instead of one process handling all 12 months by itself. The improvement in performance can be quite high.

Parallel execution helps systems scale in performance by making optimal use of hardware resources. If your system's CPUs and disk controllers are already heavily loaded, you need to alleviate the system's load or increase these hardware resources before using parallel execution to improve performance.

Some tasks are not well-suited for parallel execution.

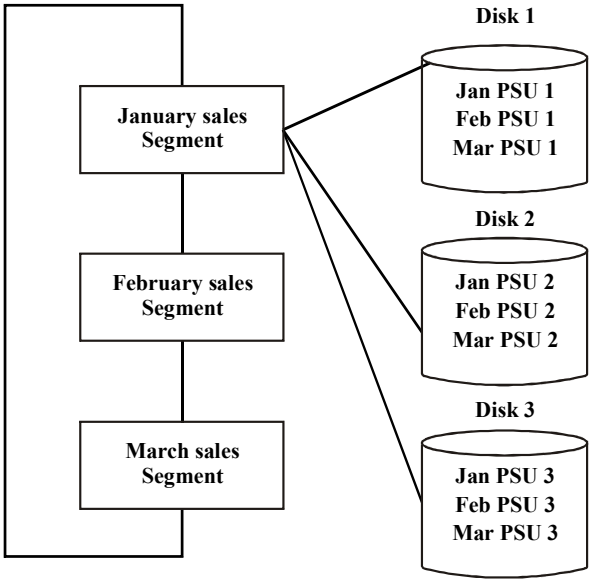
Example: Many OLTP operations are relatively fast, completing in mere seconds or fractions of seconds, and the overhead of utilizing parallel execution would be large, relative to the overall execution time.

There are various ways in which partitioning will be done which are as follows:

1. Horizontal partitioning
2. Vertical partitioning

Notes

Figure 13.3: Sales-monthly Segmentation



13.2.1 Horizontal Partitioning

Horizontal partitioning a fact table speed up queries without indexing, by minimizing the set of data to be scanned. It is not feasible to partition a fact table into segments (virtual object like table) as we could not assure of each segment to be of same size as others, because the number of transaction within the business at given point will vary it may between year to year.

Since business transaction volumes will be higher in seasons like festivals and occasions as compared to the rest of year.

So we have to think all possible ways while partitioning the fact data at the same time we also have to consider the requirements for manageability of the data warehouse.

Partitioning by Time into Equal Segments

This is a standard form of partitioning. Here partition of fact table is done on a time period basis, where each time period represents a significance retention period within the business. Like if maximum queries are on month-to-date values, then the partition is done into monthly segments or if maximum query period is fortnight-to-date, consider partition into fortnightly segments but we should consider the number of tables should not exceed the limit.

We can reuse table partitions by removing all the data from the partitions, by round robin by the warehouse manager. It is better to consider that number of partitions will store transactions over a busy period of the business.

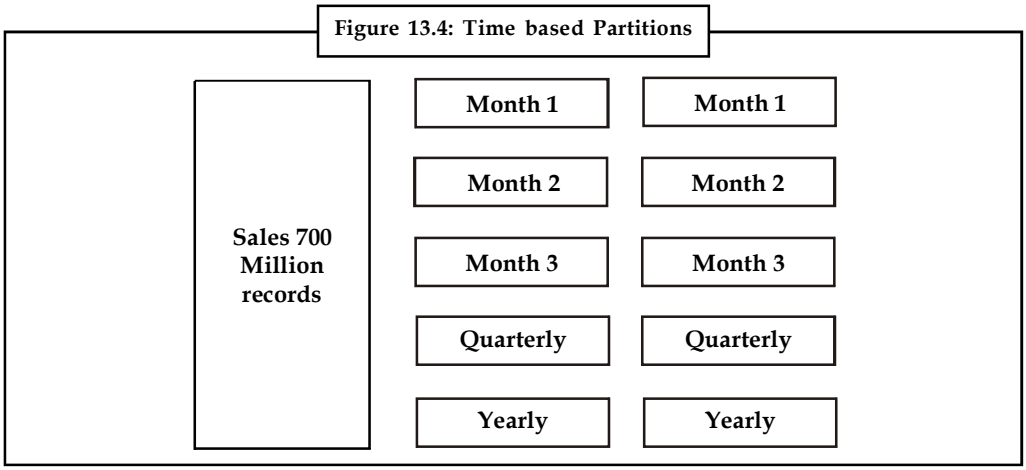
Partitioning by Time into Different-sized Segments

This type of partitions is done where previous data is accessed infrequently. This will be as set of small partitions for current data, large partition for active data and even larger partition for inactive data.

Notes

Example: For last three months including current month the partition will be three monthly, one quarterly partition for the previous quarter and for the remaining year (i.e. six months) one half-yearly partition like this these partitions would be created for each year.

Figure 13.4: Time based Partitions



The advantage of this is that all the information will be available online and the number of tables is kept relatively small which reduces operation costs. This method is used where the business requires data dipping recent history and data mining aged history.

Disadvantage of this technique is that the data must be physically repartitioned at the start of the new month or at least at the start of each quarter.

Partitioning on a Different Dimension

This technique shows that the partitioning can be done by product place, region, product or any other dimension other than time.

Example: Consider a sale of products is structured into different regions and partitioned on region-by-region basis.

While partitioning in this technique it is essential that we must first determine that the basis for partitioning may not tend to be changed in future. Because if it need to be changed men the entire fact table would have to be repartitioned. In above example, if the region with in the business changes then the fact table and partitions has to be changes. So it is better to partition those tables in the time basis where you can't assure that the situation may change in future.

Partition by Size of Table

Sometimes in data warehouse it may not be clear basis for partitioning that fact tables like time or any dimension. In that case you can consider partitioning the fact table on a size basis i.e., on the limitation of the table size, it should be predetermined. If any of table exceeds that size then a new table partition is created.

In some data warehouse in ATM banking or cellular operator services, we could see that business operates for all the days and all the time (i.e., 24 hours a day and seven days a week), where there is no operational concept of the end of business clay since the transaction may occur at any time. So it is not proper to split transactions on a daily/weekly/monthly basis. Like these areas we can go for this technique, which is like as transactions are loaded into the data warehouse, we

Notes

create a new table partition when a predetermined size is reached. It is somewhat complex and requires meta data to identify what data is stored in each partition.

Partitioning Dimensions

In some data warehouses dimensions may contain large number of entries that may need to be partitioned in the same as a fact table.

Example: Consider a large dimension which varies depending on time and you got to store all the changes and variations of that dimension for business logics, that dimension may become quite large.

It may affect query response time. In these situations the dimension should be partitioned depending on the level in hierarchy. In general this technique is appropriate are unusual. So, always check in the dimension table may not contain embedded facts that are causing unnecessary rows to be added.

Using Round-robin Partitions

In this technique where the data warehouse is holding the complete historical complement of information, a new partition is creating archiving and reusing the old partition for the latest data.

Meta data is used to allow user to access tools to refer to the appropriate partition. Warehouse manager creates a specific table names such as sales_last_month or sales_moth_to_date, which represents the data content of a physical partition.

Table management facility is easy in the data warehouse with this technique by allowing the system to refer to the same physical table partitions. The information can change periodically but this can be managed by using appropriate meta data.

13.2.2 Vertical Partitioning

In this partitioning data splits vertically and the process is as shown in figure below. This process takes two forms; they are normalization and row splitting.

Normal means making 'right', here it is concern to data placing in right position. Normalization allows common fields to be collapsed into single rows by which reducing space usage.

Removal of redundancy is done through normalization of a table. Let us take PDPL company produces drugs they keep track of daily sales in the form of a table which consist of p_id, qty, rate, sale_on, vendor_id, vendorName, loc, region. Sample of data as given below:

P_id	qty	rate	Sale_on	Vendor_id	vendorName	Loc	Region
898	5	9.90	24-JAN-03	1111	Radha Krishna	GopalPatnam	Vizag
987	4	89.00	24-1AN-03	2222	Raj shekhar	ReinBazar	Hyd
342	6	44.30	27-JAN-03	1111	Radha Krishna	GopalPatnam	Vizag
433	2	300.00	29-JAN-03	1111	Radha Krishna	GopalPatnam	Vizag

Notes

To make non-redundant data in the above table we construct two tables first consists of Vendor information and secondly the product sale information:

Vendor

Vendor_id	vendorName	Loc	Region
1111	Radha Krishna	GopalPatnam	Vizag
2222	Raj shekhar	ReinBazar	Hyd

Product-sale

P_id	qty	rate	Sale_on	Vendor_id
898	5	9.90	24-JAN-03	1111
987	4	89.00	24-1AN-03	2222
342	6	44.30	27- JAN-03	1111
433	2	300.00	29-JAN-03	1111

Splitting is like to split less used column information out from a frequently accessed fact table in data warehouse. Row splitting is to speed access to the large table by reducing its size and the other data is still maintained and can be accessed separately.

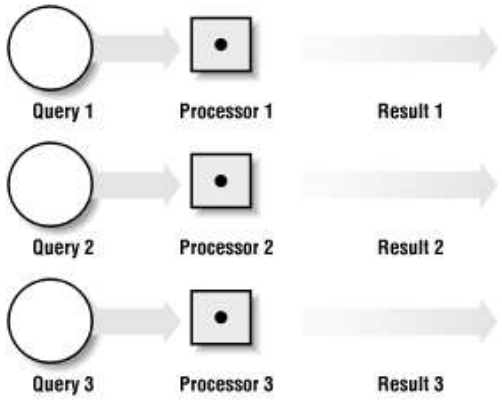
Vertical partition has to be done when there is no requirements to perform major join operations between the two partitions.

Task Discuss SMP and DSS.

13.3 Inter-query Parallelism

Inter-query parallelism is the ability to use multiple processors to execute several independent queries simultaneously. Figure 13.5 illustrates inter-query parallelism, showing how three independent queries can be performed simultaneously by three processors. Inter-query parallelism does not provide speedup, because each query is still executed by only one processor.

Figure 13.5: Inter-query Parallelism



Notes

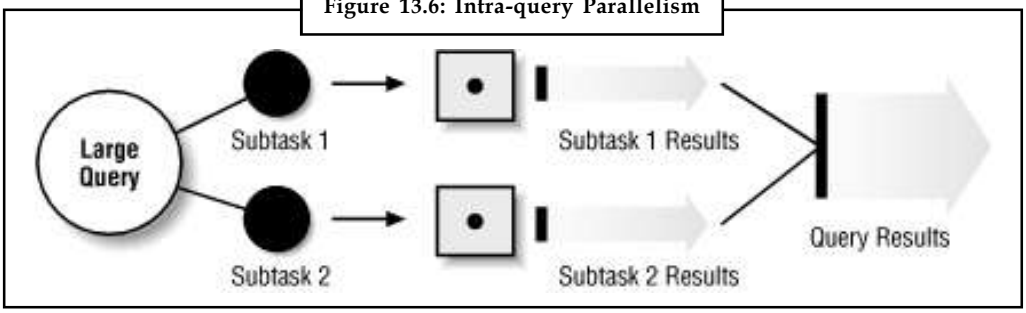
In Online Transaction Processing (OLTP) applications, each query is independent and takes a relatively short time to execute. As the number of OLTP users increases, more queries are generated. Without inter-query parallelism, all queries will be performed by a single processor in a time-shared manner. This slows down response time. With inter-query parallelism, queries generated by OLTP users can be distributed over multiple processors. Since the queries are performed simultaneously by multiple processors, response time remains satisfactory.

13.4 Intra-query Parallelism

Intra-query parallelism is the ability to break a single query into subtasks and to execute those subtasks in parallel using a different processor for each. The result is a decrease in the overall elapsed time needed to execute a single query. Intra-query parallelism is very beneficial in Decision Support System (DSS) applications, which often have complex, long-running queries. As DSS systems have become more widely used, database vendors have been increasing their support for intra-query parallelism.

Figure 13.6 shows how one large query may be decomposed into two subtasks, which then are executed simultaneously using two processors. The results of the subtasks then are merged to generate a result for the original query. Intra-query parallelism is useful not only with queries, but also with other tasks such as data loading, index creation, and so on.

Figure 13.6: Intra-query Parallelism



13.5 Inter-operation and Intra-operation Parallelism

To illustrate intra-operation parallelism and inter-operation parallelism, consider the following statement:

```
SELECT * FROM employees ORDER BY employee_id;
```

The execution plan implements a full scan of the employees table followed by a sorting of the retrieved rows based on the value of the employee_id column. For the sake of this example, assume the last_name column is not indexed. Also assume that the degree of parallelism for the query is set to four, which means that four parallel execution servers can be active for any given operation.

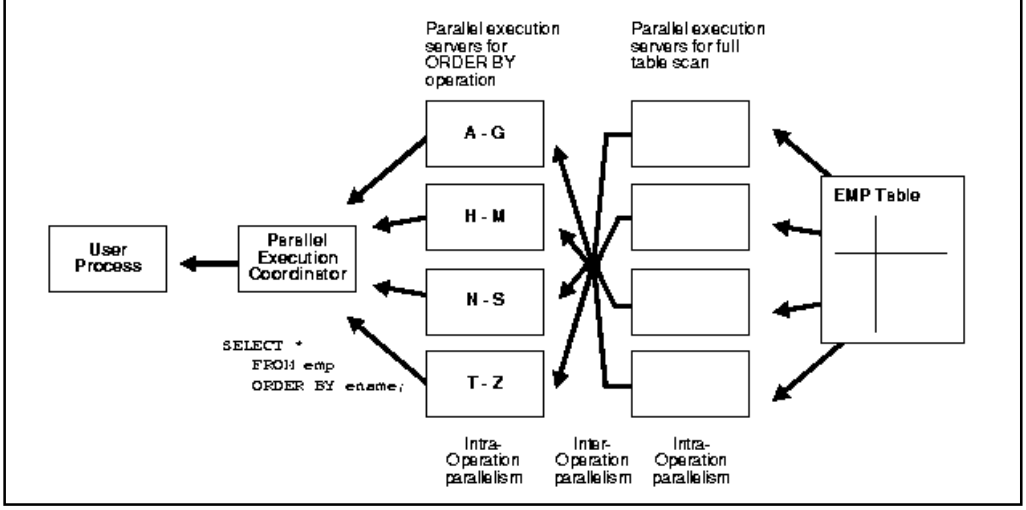
Each of the two operations (scan and sort) performed concurrently is given its own set of parallel execution servers. Therefore, both operations have parallelism. Parallelization of an individual operation where the same operation is performed on smaller sets of rows by parallel execution servers achieves what is termed intra-operation parallelism. When two operations run concurrently on different sets of parallel execution servers with data flowing from one operation into the other, we achieve what is termed inter-operation parallelism.

Due to the producer/consumer nature of the Oracle server's operations, only two operations in a given tree need to be performed simultaneously to minimize execution time.

Notes

Figure 13.7 illustrates the parallel execution of our sample query.

Figure 13.7: Inter-operation Parallelism and Dynamic Partitioning



As you can see from Figure 13.7, there are actually eight parallel execution servers involved in the query even though the degree of parallelism is four. This is because a parent and child operator can be performed at the same time (inter-operation parallelism).

Also note that all of the parallel execution servers involved in the scan operation send rows to the appropriate parallel execution server performing the sort operation. If a row scanned by a parallel execution server contains a value for the ename column between A and G, that row gets sent to the first ORDER BY parallel execution server. When the scan operation is complete, the sorting processes can return the sorted results to the coordinator, which in turn returns the complete query results to the user.

Example: **Parallel Query Intra- and Inter-operation**

As an example of parallel query with intra- and inter-operation parallelism, consider a more complex query:

SELECT /*+ PARALLEL(employees 4) PARALLEL(departments 4) USE_HASH(employees)

ORDERED */

MAX(salary), AVG(salary)

FROM employees, departments

WHERE employees.department_id = departments.department_id

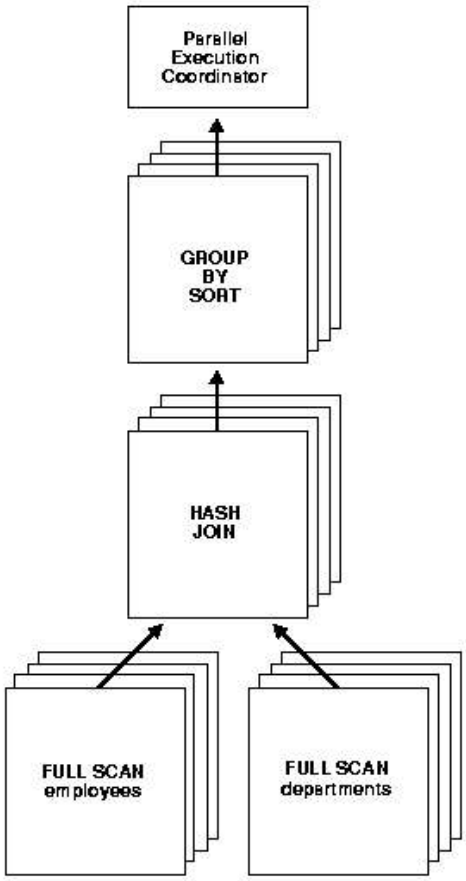
GROUP BY employees.department_id;

Notes The query to force the join order and join method, and to specify the degree of parallelism (DOP) of the tables employees and departments. In general, you should let the optimizer determine the order and method.

The query plan or data flow graph corresponding to this query is illustrated in Figure 13.8.

Notes

Figure 13.8: Data Flow Diagram for Joining Tables



Given two sets of parallel execution servers SS1 and SS2, the execution of this plan will proceed as follows: each server set (SS1 and SS2) will have four execution processes because of the PARALLEL hint in the query that specifies the DOP. In other words, the DOP will be four because each set of parallel execution servers will have four processes.

Slave set SS1 first scans the table employees while SS2 will fetch rows from SS1 and build a hash table on the rows. In other words, the parent servers in SS2 and the child servers in SS2 work concurrently: one in scanning employees in parallel, the other in consuming rows sent to it from SS1 and building the hash table for the hash join in parallel. This is an example of inter-operation parallelism.

After SS1 has finished scanning the entire table employees (that is, all granules or task units for employees are exhausted), it scans the table departments in parallel. It sends its rows to servers in SS2, which then perform the probes to finish the hash-join in parallel. After SS1 is done scanning the table departments in parallel and sending the rows to SS2, it switches to performing the GROUP BY in parallel. This is how two server sets run concurrently to achieve inter-operation parallelism across various operators in the query tree while achieving intra-operation parallelism in executing each operation in parallel.

Task Differentiate inter- and intra-operation.

Notes

Another important aspect of parallel execution is the re-partitioning of rows while they are sent from servers in one server set to another. For the query plan in figure, after a server process in SS1 scans a row of employees, which server process of SS2 should it send it to? The partitioning of rows flowing up the query tree is decided by the operator into which the rows are flowing into. In this case, the partitioning of rows flowing up from SS1 performing the parallel scan of employees into SS2 performing the parallel hash-join is done by hash partitioning on the join column value. That is, a server process scanning employees computes a hash function of the value of the column employees.employee_id to decide the number of the server process in SS2 to send it to. The partitioning method used in parallel queries is explicitly shown in the EXPLAIN PLAN of the query.

Notes The partitioning of rows being sent between sets of execution servers should not be confused with Oracle's partitioning feature whereby tables can be partitioned using hash, range, and other methods.

13.6 Summary

- Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware.
- These new designs provide impressive speedup and scale-up when processing relational database queries.

13.7 Keywords

Horizontal Partitioning: Horizontal partitioning a fact table speed up queries without indexing, by minimizing the set of data to be scanned.

Inter-query Parallelism: Inter-query parallelism is the ability to use multiple processors to execute several independent queries simultaneously.

Intra-query Parallelism: Intra-query parallelism is the ability to break a single query into subtasks and to execute those subtasks in parallel using a different processor for each.

OLTP: Online Transactional Processing

Parallel Database: Parallel database system is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries.

13.8 Self Assessment

Fill in the blanks:

1. ..... main architectures have been proposed for building parallel DBMSs.
2. MPP stands for .....
3. ..... helps systems scale in performance by making optimal use of hardware resources.
4. ..... parallelism does not provide speedup, because each query is still executed by only one processor.

Notes

5. In ..... applications, each query is independent and takes a relatively short time to execute.
6. Intra-query parallelism is very beneficial in ..... applications, which often have complex, long-running queries.
7. The basic problem with the shared-memory and shared-disk architectures is .....
8. The ..... architecture requires more extensive reorganization of the DBMS code.
9. SMP stands for .....
10. ..... is used to allow user to access tools to refer to the appropriate partition.

13.9 Review Questions

1. What do you mean by parallel database?
2. Distinguish between parallel database and distributed database.
3. Why parallel database more critical than object oriented databases? Explain
4. "Decision support system used in parallel database." Explain
5. Explain horizontal partitioning.
6. Distinguish between horizontal and vertical partitioning.
7. Explain intra-query concept in detail.
8. Distinguish between intra-operation and inter-operation parallelism.
9. What do you mean by inter-query parallelism?
10. Write short note on "partition by size of table".

Answers: Self Assessment

1.	Three	2.	massively parallel systems
3.	Parallel execution	4.	Inter-query
5.	Online Transaction Processing (OLTP)	6.	Decision Support System (DSS)
7.	interference	8.	shared-nothing
9.	Symmetric multiprocessing	10.	Meta data

13.10 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Notes

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net

Notes

Unit 14: Application Development and Administration

CONTENTS

Objectives

Introduction

14.1 Database and the Web

14.2 Web Interface to Databases

14.2.1 Server Side Database Communication with CGI

14.2.2 Chains of Communication

14.2.3 Using Perl 5 and the DBI Module to Communicate with Databases

14.2.4 The DBI Module

14.2.5 The DBI API

14.2.6 Getting the Pieces

14.2.7 Running CGI Applications on a Single Station Local Area Network

14.3 Data Administrator's Role and Functions

14.4 Accessing Database through Web

14.5 Performance Tuning

14.6 Summary

14.7 Keywords

14.8 Self Assessment

14.9 Review Questions

14.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss the web interface to databases
- Identify data administrator role and functions
- Know how to access database through web
- Explain performance tuning

Introduction

What is a Database?

Once upon a time, in the primitive and barbarian days before computers, the amount of information shepherded by a group of people could be collected in the wisdom and the stories

Notes

of its older members. In this world, storytellers, magicians, and grandparents were considered great and honored storehouses for all that was known.

Apparently, and according to vast archeological data, campfires were used (like command-line middleware) by the younger members of the community to access the information stored in the minds of the elders using API's such as:

```
public String TellUsAboutTheTimeWhen(String s);.
```

And then of course, like a sweeping and rapidly-encompassing viral infection, came agriculture, over-production of foodstuffs, and the origins of modern-day commerce.

Dealing with vast storehouses of wheat, rice, and maize became quite a chore for the monarchs and emperors that developed along with the new economy. There was simply too much data to be managed in the minds of the elders (who by now were feeling the effects of hardware obsolescence as they were being pushed quietly into the background).

And so, in order to store all the new information, humanity invented the technology of writing. And though great scholars like Aristotle warned that the invention of the alphabet would lead to the subtle but total demise of the creativity and sensibility of humanity, data began to be stored in voluminous data repositories, called books.

As we know, eventually books propogated with great speed and soon, whole communities of books migrated to the first real "databases", libraries.

Unlike previous versions of data warehouses (people and books), that might be considered the australopithecines of the database lineage, libraries crossed over into the modern-day species, though they were incredibly primitive of course.

Specifically, libraries introduced "standards" by which data could be stored and retrieved.

After all, without standards for accessing data, libraries would be like my closet, endless and engulfing swarms of chaos. Books, and the data within books, had to be quickly accessible by anyone if they were to be useful.

In fact, the usefulness of a library, or any base of data, is proportional to its data storage and retrieval efficiency. This one corollary would drive the evolution of databases over the next 2000 years to its current state.

Thus, early librarians defined standardized filing and retrieval protocols. Perhaps, if you have ever made it off the web, you will have seen an old library with its cute little indexing system (card catalog) and pointers (Dewey decimal system).

And for the next couple thousand years libraries grew, and grew, and grew along with associated storage/retrieval technologies such as the filing cabinet, colored tabs, and three ring binders.

All this until one day about half a century ago, some really bright folks including Alan Turing, working for the British government were asked to invent an advanced tool for breaking German cryptographic "Enigma" codes.

14.1 Database and the Web

The Web is the cornerstone of electronic commerce. Many organizations offer products through their Web sites, and customers can place orders by visiting a Web site. For such applications a URL must identify more than just a file, however rich the contents of the file; a URL must provide an entry point to services available on the Web site.

It is common for a URL to include a form that users can fill in to describe what they want. If the requested URL identifies a form, the Web server returns the form to the browser, which displays

Notes

the form to the user. After the user fills in the form, the form is returned to the Web server, and the information filled in by the user can be used as parameters to a program executing at the same site as the Web server.

The use of a Web browser to invoke a program at a remote site leads us to the role of databases on the Web. The invoked program can generate a request to a database system. This capability allows us to easily place a database on a computer network, and make services that rely upon database access available over the Web. This leads to a new and rapidly growing source of concurrent requests to a DBMS, and with thousands of concurrent users routinely accessing popular Web sites, new levels of scalability and robustness are required.

The diversity of information on the Web, its distributed nature, and the new uses that it is being put to lead to challenges for DBMSs that go beyond simply improved performance in traditional functionality. For instance, we require support for queries that are run periodically or continuously and that access data from several distributed sources. As an example, a user may want to be notified whenever a new item meeting some criteria (e.g., a Peace Bear Beanie Baby toy costing less than $15) is offered for sale at one of several Web sites. Given many such user profiles, how can we efficiently monitor them and notify users promptly as the items they are interested in become available? As another instance of a new class of problems, the emergence of the XML standard for describing data leads to challenges in managing and querying XML data.

14.2 Web Interface to Databases

14.2.1 Server Side Database Communication with CGI

So far in this tutorial we have remained pretty airy. We have learned lots of "database theory" and "abstract" SQL communication protocols.

But what you probably came to learn was how you could actually put a real-live database on the web.

Well, to be fair, understanding all that abstract stuff that we have discussed so far is necessary for you to be able to do what you want to do.

Unfortunately, putting your database on the web turns out to be a pretty complex feat involving all sorts of technologies. So, there are a bunch of things you needed to learn in order to get the whole kit-and-caboodle operational.

However don't worry, in this part, we are going to put it all together and get your data webified.

Nevertheless, before we get into the nitty gritty lets recap a bit and also provide an overview for what we are going to do in this part.

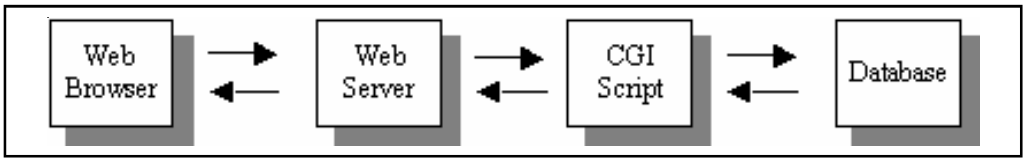
14.2.2 Chains of Communication

A Web Database follows the client-server database model. A Database Engine sits on some central computer somewhere and serves data to multiple web-based clients (perhaps lots of customers using Netscape Navigator).

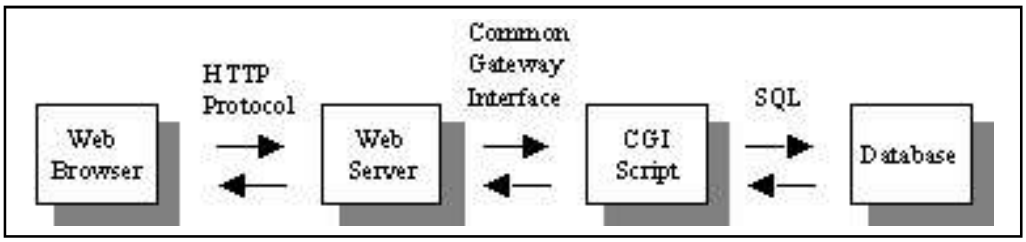
Because we are dealing with web-based clients however, we must also have a Web Server that handles requests from web-browsers and then forwards them to the Database. Likewise, the web server will wait for the database to respond and then pass on that response to the waiting web browsers. The whole interaction is much simpler as a picture.

Notes

The CGI Script will be responsible for understanding what the Web Server is saying and will also be responsible for knowing how to talk to the Database. This type of CGI script is seriously multilingual.



Of course, the CGI script can be written in any computer programming language and can use all sorts of inventive or proprietary methods to talk to the Database.



14.2.3 Using Perl 5 and the DBI Module to Communicate with Databases

However, for purposes of demonstration, in this part we will show an example of writing a CGI script in Perl 5, using the DBI (Database Independent Interface) module supplied for free with the normal Perl 5 distribution.

Task What are the uses of XML? Discuss

As an example, our DBI-aware Perl 5 CGI Script will be used to access an MS Access Database on a Windows 95 system using a local web server called *Sambar*.

Of course, as we will see in just a bit, the fact that we use Perl 5 and the DBI module means, that the code we develop here will work on UNIX, NT or Macintosh without any modifications.

It will also work "out of the box" for most commercial and public domain databases available for the web including Oracle, Sybase, Informix, Postgres, DB2, etc.

Finally, you should know that you need not use Sambar as your Web Server. Any Web Server that can handle CGI will do. I am just using Sambar because it is free and it is what I happened to download. In retrospect, I would probably recommend Apache since it is available on so many platforms.

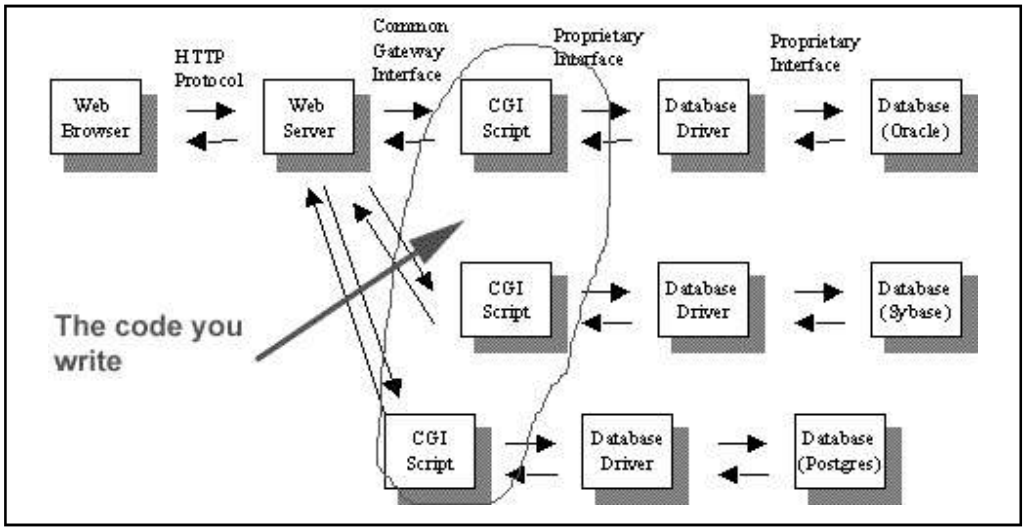
In short, the beauty of Perl and of the DBI module is that they are both designed to be highly cross platform and non-proprietary.

Notes

14.2.4 The DBI Module

The DBI Module is an incredibly useful tool because it abstracts away the need to learn how to communicate with every type of database out there. Consider what a pain it would be if you had to write a CGI script for every proprietary database engine.

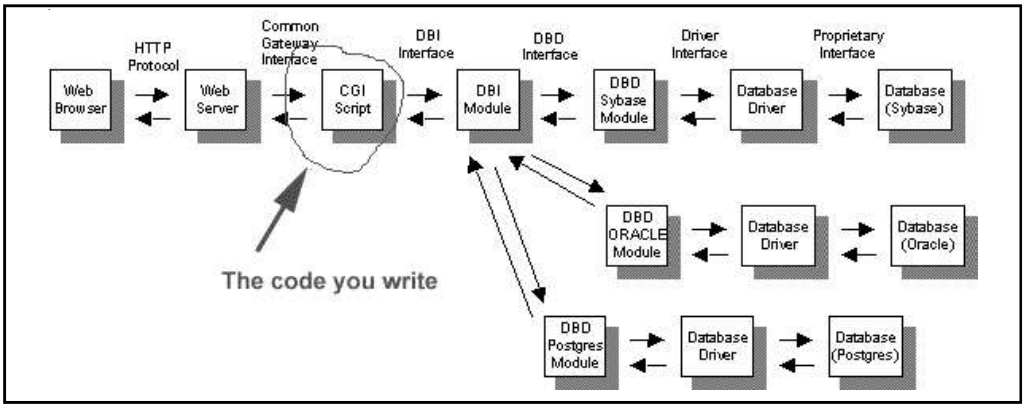
Now imagine how much extra work you would have if the management decided to change database vendors on you after you had already written your code!



What the DBI module does for you is to provide a single interface to send SQL commands to any proprietary database. Essentially, you write all your code to a generic standard, and move it from database to database without having to re-code your application.

The secret to the DBI module is the extensive library of DBD (Database Dependent) modules that come with the standard distribution of Perl 5. These DBD modules provide the code that hooks up DBI to a proprietary database driver.

Of course, since these DBD modules are already written for almost any Database you would consider using, you needn't do any of that hard work. All you need to worry about is satisfying the requirements of the DBI interface. Everything else is handled behind the scenes.



It looks pretty complex but believe me, it isn't. And it makes your life as a programmer incredibly easy. That is, although the communication takes place through so many actors, you, as the programmer, need not bother with the details of that communication. You talk to the DBI module, and the DBI module handles the multitude of proprietary database blue meanies by itself.

Notes

Fortunately, dozens of great module writers have done all that hard work for you. You'll see just how easy it is when we start going over code!

14.2.5 The DBI API

So how do you use the DBI module?

Well, as with any Perl 5 module, you simply use the "USE" keyword.

Once, the DBI module is loaded, you can then instantiate a database connection. Establishing a connection involves creating a "database handle" object by using DBI to connect to a database given a data source name, database user, database password, and a database driver.

Consider the following example in which we connect to a Database with the data source name of "MyCompany" using the username "selena", password "12mw_l", and the ODBC driver. Notice that the ODBC driver name is tagged to the beginning of the Database name.

```
        use DBI;
$dbHandle = (DBI->connect('DBI:ODBC:MyCompany',
         "selena",
         '12mw_l'));
```

We'll show you how to setup a database name etc. when we cover Access later. However, you can imagine that regardless of what database you use on what system, you will be able to define these values and use them.

Once you have created a database handle object, you can do things with the object. Most likely, you will be sending SQL statements to the database via the database handle. To do so, you create a statement handle object by calling the prepare() method on the database handle object and then call the execute() method on the statement handle object. Consider the following code:

```
        use DBI;
$dbHandle = (DBI->connect('DBI:ODBC:MyCompany',
         "selena",
         '12mw_l'));
$sql = "SELECT * FROM Employees";
$statementHandle = $dbHandle->prepare($sql);
$statementHandle->execute() ||
        die $statementHandle->errstr;
```

Once the sql has been sent to the database, the DBI module will store the results. To get to the results, you can use any number of useful statement handle methods. One of the most common methods is the fetchall_arrayref() method that returns all the returned database rows in a reference to an array of references to rows. Consider the following:

```
        use DBI;
$dbHandle = (DBI->connect('DBI:ODBC:MyCompany',
        "selena",
        '12mw_l'));
```

Notes

```
$sql = "SELECT * FROM Employees";
$statementHandle = $dbHandle->prepare($sql);
$statementHandle->execute() ||
        die $statementHandle->errstr;
$arrayRef = $statementHandle->fetchall_arrayref;
```

Of course, once you have the reference to the array, you can dereference everything to access the data.

Finally, when you have massaged all the data, you close the database connection with the database handle object's disconnect() method:

```
$dbh->disconnect();
```

At the end of this part, I have included several examples of CGI script using DBI so you can see how you might build an application around the methods. Further, fantastic documentation for DBI can be found at http://www.hermetica.com/. However, I will also include a cursory API reference summary (based upon the DBI documentation by Tim Bunce) of the most used methods here so you can get a feel for what is available.

Table 14.1: General DBI Class Methods

Name	Usage	Description
connect()	$dbHandle = DBI-> connect (DBI:$driverName:$dataSource, $usrName, $passwrd); $dbHandle = DBI-> connect ($dataSource, $usrName, $passwrd, $driverName);	This method establishes a connection to the specified Data Source and returns a database handle object. Note that if you define the driver name as the fourth parameter, you need not tag the DBI:$driverName along with the $dataSource name.
available_drivers ()	@drivers = DBI-> available_drivers(); @drivers = DBI-> available_drivers ($warnFlag);	As you can see, this method returns an array of drivers that the module can see. It gets the list by searching through the @INC array for DBD modules. If you pass a true value as a parameter, warnings about hidden drivers will be quelled.
data_sources()	@dataSources = DBI-> data_sources($driver);	Returns a list of the data sources available for a given driver if the driver supports the method.
trace()	DBI->trace ($level); DBI->trace ($level, $file);	Provides tracing functionality such that the $file is appended to with trace information. A Level 0 rating disables tracing and a level 2 tracing provides a detailed trace.
neat()	$neatvalue = DBI::neat($value, $maxLength);	Formats the specified value such that strings are quoted, undefined values are replaced with "undef", unprintable characters are replaced with a ".", and string will be truncated and ended with "..." if longer than $maxLength
neat_list()	$delimitedList = DBI::neatList(\@listRef, $maxlength, $delimiter);	Formats an array by calling neat() on each element and forming a string by joining the elements with the specified delimiter.
dump_results()	$dbRows = DBI::dump_results ($statementHandle, $maxLength, $lineSeparator, $fieldSeparator, $fileHandle);	Gets all the rows from the statement handle object, calls neat_list() on each row and prints the results to $filehandle that is <STDOUT> by default.

Contd...

Notes

General Handle Methods

Name	Usage	Description
err()	$result = $handle->err();	Gets the error code returned from the database.
errstr()	$result = $handle->errstr();	Gets the error message returned from the database.
state()	$state = $handle->state();	Gets the SQLSTATE error code.
trace()	$handle->trace ($level); $handle->trace ($level, $file);	Provides tracing functionality such that the $file is appended to with trace information. A Level 0 rating disables tracing and a level 2 tracing provides a detailed trace.

Database Handle Methods

Name	Usage	Description
prepare()	$statementHandle = dataSourceHandle -> prepare($sql);	Prepares a statement for execution.
do()	$rowCount = $databaseHandle -> do($sql);	Prepares and executes an SQL statement and returns the number of rows returned.
commit()	$rowCount = $databaseHandle -> commit();	Makes permanent the last set of database changes if supported.
rollback()	$rowCount = $databaseHandle -> rollback()	Undoes uncommitted changes if supported.
disconnect()	$rowCount = $databaseHandle - >disconnect()	Disconnects from the database.
ping()	$rowCount = $databaseHandle - > ping()	Tries to figure out if the database server is still available.
quote()	$sql = $databaseHandle - > quote($string);	Spiffys up a string for an SQL statement.

Statement Handle Methods

Name	Usage	Description
execute()	$value = $statementHandle -> execute();	Executes a prepared statement.
fetchrow_arrayRef ()	$arrayRef = $statementHandle -> fetchrow_arrayref();	Gets the next row of data as a reference to an array holding the column values.
fetchrow_array()	@array = $statementHandle -> fetchrow_array();	Gets the next row of data as an array.
fetchrow_hashref( )	$hashRef = $statementHandle -> fetchrow_hashRef();	Gets the next row of data in which the keys to the hash reference are the column names and the values are the column values.
fetchall_arrayref()	$reference = $statementHandle -> fetchall_arrayref()	Gets all the rows as references in a referenced array.

Notes There are all sorts of other more complex methods such as binding and error handling, but you should consult the documentation and the DBI tutorial referenced above. These topics are a bit beyond the scope of this tutorial.

14.2.6 Getting the Pieces

So now that we understand what it is that we are building, let's continue by assembling the tools and raw material.

Notes

As I said before, we are going to demonstrate this process on a Windows 95 machine. I chose to demonstrate on a Windows box because I know that even if you are a UNIX developer, you have access to a Windows machine that you can use to develop a test site. Of course, most likely, you develop on a Windows system. Either way, everything we go over can be performed on any platform, just with different software packages. I will try to point out where things might be different as we go along.

Installing Perl

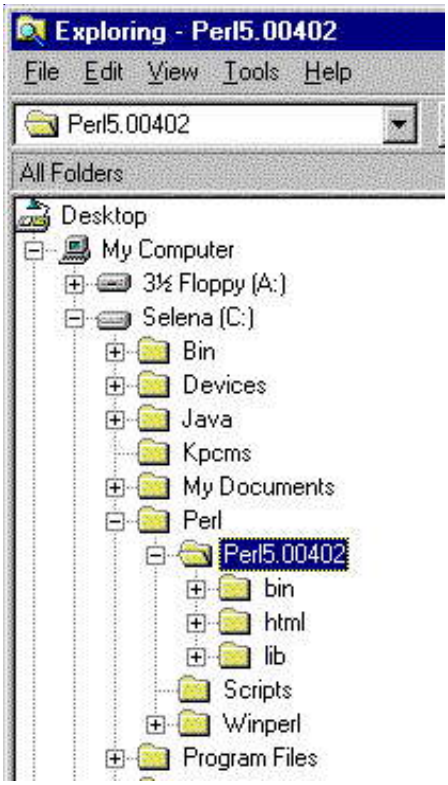
The first thing you will need to do is download Perl 5. You can do this easily, by going to www.perl.com . There are two versions of Perl available and the differences are explained at www.perl.com. However, I use Gurusamy Sarathy's binary version of Perl for extra module support.

If you are using UNIX, you can download the .tar.gz

Notes If you are using UNIX it is almost assuredly already installed and/or installable by your systems administrator. You probably should not do it yourself.

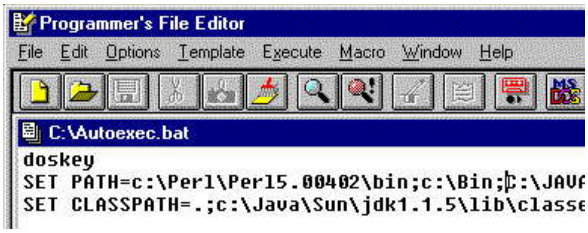
If you are using Windows or Macintosh, download the ZIP file. In any case, unzip utilities for all the platforms are available at www.shareware.com.

When you have downloaded the zipped file, you should uncompress it on your local hard drive. On Windows, I tend to extract it into a directory like c:\Perl\Perl5.004\. But the directory you use does not really matter much.



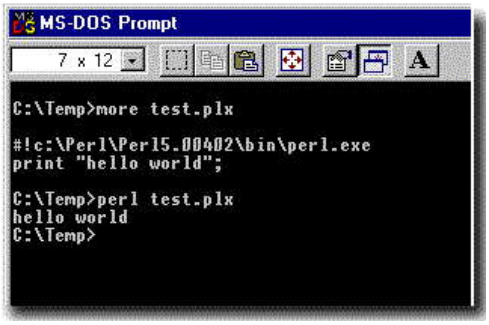
Notes

Notes Windows and UNIX users, you might want to add the Perl executable to your path, but it is not necessary for this tutorial. In case, you want to add it to your path, here is an example of the line I use in my autoexec.bat file. In UNIX you would typically define the PATH in .login, though it is better to install Perl in a standard directory that is already defined in your PATH such as "/usr/local/bin".



Okay, once Perl is installed, try running a simple Perl program to make sure everything is hunky dory. Here is the program I use:

```
#!c:\Perl\Perl5.00402\bin\perl.exe
print "hello world";
```



Installing a Web Server: Sambar

Now that you have Perl installed and running, it is time to download a Web Server that you can use locally to test CGI scripts.

But how can I run a web server on my computer that is not hooked up to the web?

Well, you "can" run a web server locally, but it will only be useful for testing and development. That is, you will create a network composed of one computer and then use your web browser to access the web server!

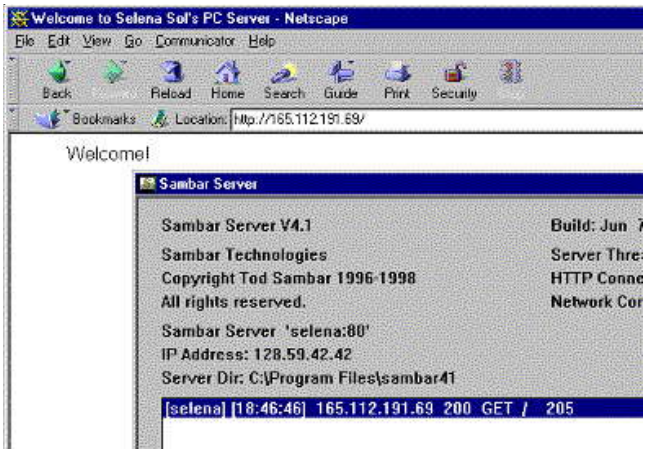
You can easily pick up a free web server for any operating system you are using. For UNIX and Windows I recommend Apache. You can also use the Sambar Web Server for Windows. In the case of this tutorial, I downloaded Sambar from www.sambar.com because it is so incredibly easy to install. However, which Web Server you choose will not make a difference for this tutorial. They will all work like Sambar for our purposes.

Once you download the self-extracting executable file from www.sambar.com, you run the setup program and the server is installed. There is not much else to it. The server is fairly featureless, but for testing purposes that can be just what the doctor ordered.

Once that is done, try running Sambar! You should see the status window. Once that is done, you can try connecting to your personal web server. Just use your IP address in the location field in

Notes

Netscape or Internet Explorer or use "localhost". You can see in the following image that my Netscape document request is being registered and handled by Sambar.



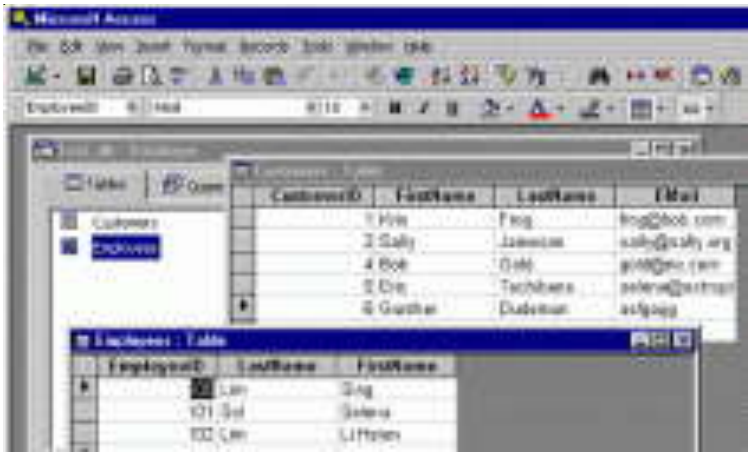
14.2.7 Running CGI Applications on a Single Station Local Area Network

You still with me? Okay the next thing we need to do is try running some CGI applications with your new server. To do so, simply dump a simple CGI script in the cgi-bin directory under the sambar directory. Here is one I use:

Setting up a Sample Database

Next, you need to make sure that you have set up a sample database to test on. In the previous parts we have introduced MSAccess which is a good database to practice with since it is fairly lightweight and user-friendly. Of course, as we have said before, what database you use will not really matter. I am just using MSAccess to demonstrate.

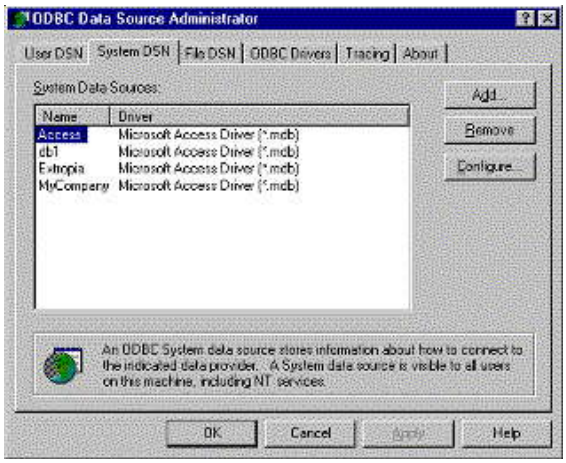
Go ahead and setup a database with two tables: Employees and Customers with some sample data in each. In fact, here are the tables I am using for your reference:



When you are done with data entry, you'll need to register your database with your operating system. On Windows, you simply need to use the 32 bit ODBC Control panel.

Notes

The "Add Wizard" will walk you right through assigning a system ODBC name to the Access database you setup. Notice that I made four of them: "Access", "db1", "Extropia", and "MyCompany".



Putting it all together with a DBI-AwareCGI Script

I have prepared a little DBI script for us to walk through. You can copy this over to your system, change the setup variables and it should run on your system as well. The code of this should all be straight forward and you should have all read the Perl tutorials at Web Ware, so I won't spend much time documenting the code.

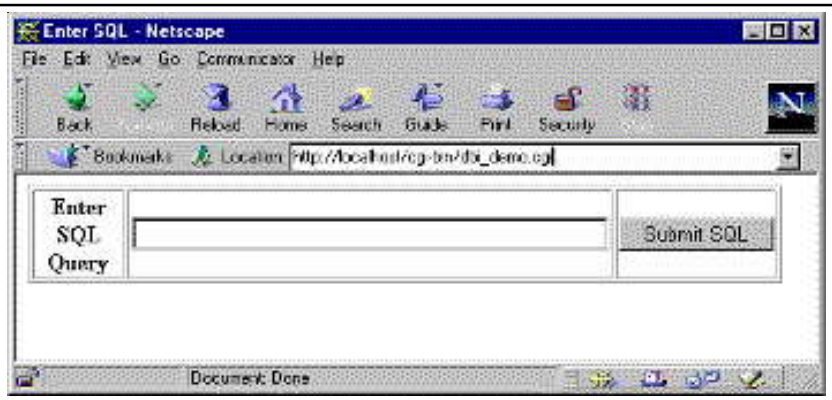
```
        # First we identify the location of
        # the Perl interpreter. You will need
        # to change this line to reflect the
        # location of Perl on your system.
#!c:\Perl\Perl5.00402\bin\perl.exe
        # Next we will tell Perl that we are
        # going to USE the DBI and CGI
        # modules.
        #
        # We will use the CGI module mainly
        # to handle incoming form data.
```

Notes

```
        #
        # The CGI::CARP module has a nice
        # feature called "fatalsToBrowser"
        # that sends error messages to the
        # web browser window so that the
        # user does not get a meaningless
        # 500 Server Error message.
use DBI;
use CGI;
use CGI::Carp qw(fatalsToBrowser);
        # Setup some implementation
        # specific variables. $dbname
        # is going to be the Data
        # Source name that you
        # assigned to your database in
        # the 32Bit ODBC Control
        # Panel plus the DBI:ODBC
        # pre-tag. I did not setup
        # any security on my database
        # since it is only for local
        # testing, so we can leave
        # those values blank.
$dbName = "DBI:ODBC:MyCompany";
$dbUserName = "";
$dbPassword = "";
        # We will create a new CGI
        # object and use it to send
        # out the HTTP header and
        # to parse out the incoming
        # form variables "requestType"
        # and "sql". You will see
        # where those form variables
        # come from in just a bit.
$dataIn = new CGI;
$dataIn->header();
```

Notes

```
$requestType = $dataIn->param('requestType');
$sql = $dataIn->param('sql');
        # Next we will check to see
        # if there is a value coming
        # in from a form for the
        # variable "sql". If there
        # is no value, then we know
        # that the user has not yet
        # seen the submission form.
        # In that case, we will send
        # them the HTML form so they
        # can actually submit a value
        # for "sql". The following
        # screen shot shows you what
        # will be returned.
```



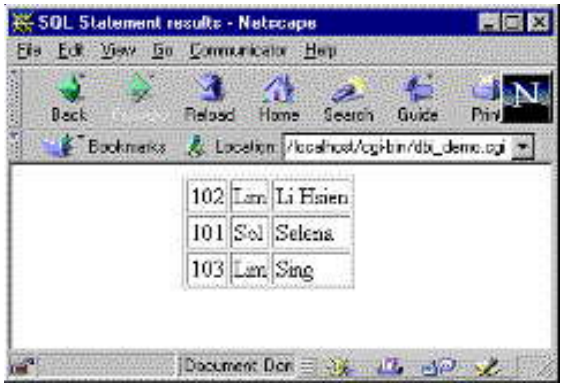

```
if ($sql eq "")
{ print qq!
<HTML>
<HEAD>
<TITLE>Enter
SQL</TITLE>
</HEAD>
<BODY BGCOLOR = "#FFFFFF"
                                        TEXT = "#000000">
<FORM METHOD = "POST"
        ACTION = "dbi_demo.cgi">
```

Notes

```
<TABLE BORDER = "1">
<TR>
<TH>Enter SQL Query</TH>
<TD>
<INPUT TYPE = "TEXT"
SIZE = "40"
NAME = "sql">
</TD>
 <TD>
<INPUT TYPE = "SUBMIT"
NAME = "requestType"
VALUE = "Submit SQL">
 </TD>
</TR>
</TABLE>
</FORM>
</BGODY>
</HTML>!;
exit; }
# If there was a value for $sql, we know
# that the user has already seen the
# HTML form and has submitted some
# SQL for us to process. In that case,
# we will open a connection to the
# database, execute the SQL, gather
# the result set and display it
# to the user. In the case of a
# simple SELECT, we will display
# the results in an HTML table.
# If, on the other hand, the SQL
# was a DELETE, MODIFY or INSERT,
# we will let them know that the
# operation was successful.
#
# Notice that you need to do a lot
```

Notes

```
# of dereferencing with the returned
# rows :)
```




```
else
        {
        $dbh = DBI->connect($dbName,
                $dbUserName,
                $dbPassword);
        $dataObject = $dbh->prepare($sql);
        $dataObject->execute();
        @dbRows = $dataObject->
                fetchall_arrayref();
        if ($sql =~ /^SELECT/i)
                {
                print qq!
                <HTML>
                <HEAD>
                <TITLE>SQL Statement
                        Results</TITLE>
                </HEAD>
                <BODY BGCOLOR = "#FFFFFF"
                        TEXT = "#000000">
                <CENTER>
                <TABLE BORDER = "1">!;
                foreach $rowReference (@dbRows)
                        {
                        foreach $columnReference
```

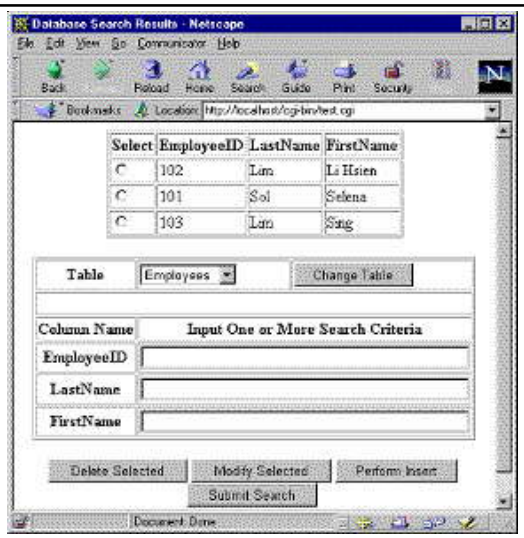
Notes

```
                    (@$rowReference)
                {
                print qq!<TR>!;
                foreach $column
                    (@$columnReference)
                {
                print qq!<TD>
                    $column
                    </TD>\n!;
                }
            print qq!</TR>!;
            }
        }
        print qq!
</TABLE>
</CENTER>
</BODY>
</HTML>!;
exit;
}
else
{
        print qq~Your SQL Query has been
            processed, please hit the
            back button and submit a
            SELECT to see the changes!~;
        }
}
```

Well, however simple that application is, it should be enough to get you started. You will have to design your own application logic for a more useful database tool, but all the components for a full featured application are demonstrated here.

As it so happens, I wrote a more complex application that you can use if you'd like. This application handles inserts, selects, updates and deletes in a much more user-friendly way. If you want to see the code, click here. Otherwise check out the screen shot of the interface.

Notes



Task Discuss something about DBI.

14.3 Data Administrator's Role and Functions

Data administrator is one who oversees the data integrity of the database. They do not create objects and they may not have privileges to do anything other than "view" the data and report any data discrepancies to the database administrator (one who administers the entire database). The data administrator can check data in the tables to make sure there are no errors, can check to see if tables that need an index have one, are all programs valid–stuff like that. The database administrator is usually busy with fixing problems, creating tablespaces, tables, looking at error messages. In a nutshell, the Data Administrator performs data integration tasks.

The following considerations apply to the Data Administrator's role:

1. Users who are assigned to the Data Administrator role have unrestricted access to all business data within their scope, even if they belong to a business role that has restricted settings.
2. Assigning a member to the Data Administrator role does not guarantee explicit permissions to the staging and application database to perform all necessary tasks for data integration. For more information, see Performance Point Server Data Integration Help.
3. For security reasons, work performed by Data Administrator role members should be carefully reviewed, because those users can browse databases. We recommend that Data Administrator role members do not have full database permissions unless it is necessary.
4. Members of the Data Administrator role cannot create or delete applications or model sites or manage role membership.

On the Data Administrator Role page, users who currently belong to the role are listed in the User ID column next to the appropriate application or model site. Except for the Global Administrator role, which has system-wide scope, Planning Server administrator roles have either an application scope or a model-site scope. Application scope permissions apply for all model sites in an application. Model-site scope permissions apply only for the specific model site.

Notes

You must belong to the User Administrator role to add users to or remove users from the Data Administrator role. Before you can add a user to any Planning Server role, the user must first be added to Planning Server from the Users page.

To add a user to the Data Administrator role

1. On the Data Administrator Role page, select the scope of the role.

   To add a user who has application scope permissions, under Data Administrator Role: Applications, select the appropriate application.

   To add a user who has model-site scope permissions, under Data Administrator Role: Model Sites, select the appropriate application in the drop-down list, and then select the appropriate model site.

2. Click Add/Remove Users: The Add or Remove Users: Data Administrator Role dialog box opens.
3. In the text box next to the Add button, type the user ID of the person whom you want to add to the role. The user ID must be entered in the domain\user name format.
4. Click Add.
5. When you are finished adding users, click Close.

To remove a user from the Data Administrator role

1. On the Data Administrator Role page, select the scope of the role.

   To remove a user who has application scope permissions, under Data Administrator Role: Applications, select the appropriate application.

   To remove a user who has model-site scope permissions, under Data Administrator Role: Model Sites, select the appropriate application in the drop-down list, and then select the appropriate model site.

2. Click Add/Remove Users. The Add or Remove Users: Data Administrator Role dialog box opens.
3. Select the check box next to the user who you want to remove from the role.
4. Click Remove.
5. Click Close.

Data administrators view data from the business perspective and must have an understanding of the business to be truly effective. Their expertise allows them to define, describe, organize and categorize data. DAs build models of the data that are a based upon data entity relationships and the business rules that govern them. Data administrators provide the framework that allows disparate business units within an organization to share timely and accurate data. Because Data Administration maps the business rules into constraints that can be enforced by the database and turns raw data into information stored in tables, the Data Administrator acts as the intermediary between the business and Database Administration units.

14.4 Accessing Database through Web

Creating a Web Database Application

How do we create a web database? Well, we would like to demonstrate this with the help of a very simple example. To implement the example, we use Active Sever Pages (ASP), which is one of the old popular technologies with the backend as Ms-Access. ASP is a very easy technology to

Notes

use. Although, Microsoft Access does not support websites with a lot of traffic, it is quite suitable for our example. A web database makes it very convenient to build a website.

Let us show the process of creating a web database. We would need to follow the following steps to create a student database and to make it accessible as a web database.

1. The first step is to create a database using Ms-Access with the following configuration:

Student-id	Text (10)	Primary Key
Name	Text (25)	
Phone	Text (12)	

   Now, you would need to enter some meaningful data into the database and save it with the name students.mdb.

2. Put your database online by using ftp to transfer students.mdb to the web server on which you are allowed access. Do not put the file in the same directory in which your web site files are stored, otherwise, the entire databases may be downloaded by an unauthorised person. In a commercial set up it may be better to keep the data on the Database server. This database then can be connected through a Data Source Name (DSN) to the website. Let us now build the required interface from ASP to the Database. A simple but old method may be with connecting ASP using ActiveX Data Object (ADO) library. This library provides ASP with the necessary functionality for interacting with the database server.

3. The first and most basic thing we need to do is to retrieve the contents of the database for display. You can retrieve database records using ADO Recordset, one of the objects of ADO.

   Dim recordsettest

   Set recordsettest = Server.CreateObject("ADODB.Recordset")

   The commands given above create a variable (recordsettest) to store new Recordset object using the Server object's CreateObject method.

4. Now fill this Recordset with records from the database with its Open method. Open takes two parameters:

   (a) The table name that contains the records to be fetched and

   (b) The connection string for the database.

Now, the name of the table is straight forward, as we would obviously, create a table with a name. However, the connection string is slightly more complex. Since the ADO library is capable of connecting to many database servers and other data sources, the string must tell Recordset not only where to find the database (the path and file name) but also how to read the database, by giving the name of its database provider.

A database provider is a software that has a very important role. It allows ADO to communicate with the given type of database in a standard way. ADO the provider for MS-Access, SQL Server, Oracle, ODBC database servers etc. Assuming that we are using the provider Jet.OLEDB to connect to an Access database, the connection string would be:

Provider = Microsoft.Jet.OLEDB.version; Data Source = ~\student\student.mdb

A connection string for MS SQL may be like:

Provider = SQLOLEDB; Data Source = servername; Initial Catalog = database name;

User Id = username; Password = password.

Notes

So a sample code could be:

```
Dim recordsettest
Dim db_conn 'The database connection string
recordsettest= Server.CreateObject ("ADODB.Recordset")
db_conn = "Provider=Microsoft.Jet.OLEDB.version; Data Source= ~\student\student.mdb
recordsetest.Open "student", db_conn
```

However, since many ASP pages on the site will require such a string, it is common to place the connection string in an application variable in the global workspace.

```
<SCRIPT LANGUAGE= "VBScript" RUNAT= "Server">
        Sub Application_OnStart ( )
                Dim db_conn
        db_conn= "Provider" = Microsoft.Jet.OLEDB.version;
        Data Source= ~\student\student.mdb Application ("db_conn") = db_conn
        End Sub
</SCRIPT>
```

The code to retrieve the contents of the student table will now include only the code for Recordset.

```
Dim recordsettest
Set recordsettest = Server.CreateObject("ADODB.Recordset'')
recordsettest.Open "Student", Application("db_conn")
```

Thus, we have established the connection and can get information from the MS-Access database. But another problem remains. How do we display the results?

1. Now, we can access recordsets, like of the database table, the result sets with the data row as one database record. In our example, we have put the contents of the student table into the recordsetrest object. An opened Recordset keeps track of the current record. Initially the first record is the current record. A MoveNext method of the Recordset object moves the pointer to the next record in the set, if any. An EOF property of the Recordset will become true at the end of Recordset. Thus, to display student id and name, you may write the following code:

```
Do While Not recordsettest.EOF
    Response. Write "<li> "& recordsettest ("students-id") &"
    Response.Write ''<p> "& recordsettest ("name") &" </p> </li>
    recordsetest(''name)
    recordsettest.MoveNext
Loop
If recordsettest.BOF Then
Response.Write "<p>No students data in the database. </p>"
```

Notes The BOF property checks if the file is empty.

2. Once you have completed the task then you must be close the recordset as:
   Recordsettest.Close

   This command sets free the connection to the database. As these connections may not be very large in numbers, therefore, they need not be kept open longer than it is necessary.

Notes

Thus, the final code may look like:

```
<html>
<head>
<title> Student Data</title>
</head>
<body>
<o1>
<%

Dim recordsettest

Set recordsettest = Server.CreateObject("ADODB.Recordset'')

recordsettest.Open "Student", Application("db_conn")

Do While Not recordsettest.EOF

    Response. Write "<li> "& recordsettest ("students-id") &"

    Response.Write ''<p> "& recordsettest ("name") &" </p> </li>

    recordsetest(''name)

    recordsettest.MoveNext

Loop

If recordsettest.BOF Then

    Response.Write "<p>No students data in the database. </p>"

Recordsettest.Close
```

Save this file on the web server. Now, test this program by storing suitable data in the database. This application should produce the simple list of the student. However, you can create more complex queries to data using SQL. Let us try to explain that with the help of an example.

Task Data administrator play important role in DBMS. Suggest.

Example: Get the list of those students who have a phone number specified by you. More than one student may be using use this phone number as a contact number.

Notes The actual implementation you would need to create more complex queries.

Please also note that the only change we need to perform in the programming given above is, in the Open statement where we need to insert SQL command, such that on opening the recordset only the required data from the database is transferred to it. Thus, the code for this modified database access may be:

```
<html>

<head>

<title> Student Data</title>
```

Notes

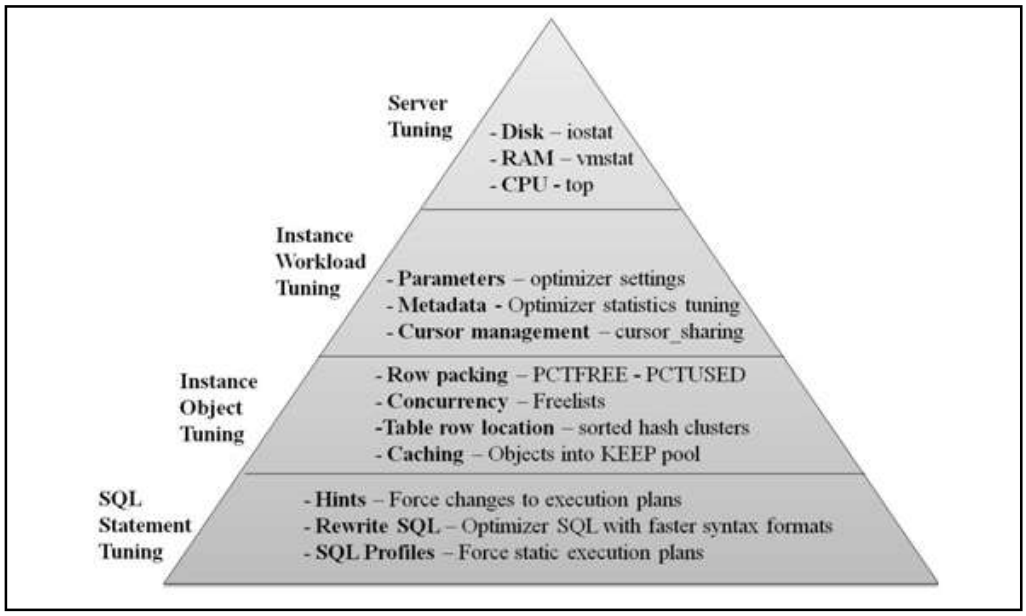
```
</head>
<body>
<o1>
<%
Dim recordsettest
Set recordsettest = Server.CreateObject("ADODB.Recordset'')
recordsettest.Open "SELECT student-id, name FROM Student
        WHERE phone = " & Request ("phone") ", Application("db_conn")
Do While Not recordsettest.EOF
        Response.Write "<li> "& recordsettest ("students-id") &"
        Response.Write "<p> "& recordsettest ("name") &" </p> </li>
        records  etest(''name)
        recordsettest.MoveNext
Loop
If recordsettest.BOF Then
        Response.Write "<p>No students data in the database. </p>"
Recordsettest.Close
```

Notes You can build on more complex queries. You may also refer to more advanced ASP versions and connections.

14.5 Performance Tuning

Performance tuning is the one area in which the Oracle DBA probably spends most of his or her time. If you're a DBA helping developers to tune their SQL, you can improve performance by suggesting more efficient queries or table- and index-organization schemes. If you're a production DBA, you'll be dealing with user perceptions of a slow database, batch jobs taking longer and longer to complete, and so on. Performance tuning focuses primarily on writing efficient SQL, allocating appropriate computing resources, and analyzing wait events and contention in the system.

Notes



Oracle tuning involves the following steps, with each step getting more specific and targeted:

1. ***Server & network tuning:*** This is always the first step, as not amount of tuning will help a poor server environment.

2. ***Instance tuning:*** Tuning the Oracle SGA is the next step, and all of the Oracle initialization parameters must be reviewed to ensure that the database has been properly configured for it's workload. In some cases, a database may have a bi-modal workload (online vs. batch) and the instance parms are adjusted as-needed during this step.

3. ***Object tuning:*** This step of performance tuning looks at the setting for Oracle tables and indexes. Table and index settings such as PCTFREE, PCTUSED, and FREELISTS can have a dramatic impact on Oracle performance.

4. ***SQL tuning:*** This is last step in tuning, and the most time-consuming tuning operation because there can be many thousands of individual SQL statements that access the Oracle database. If you have carefully optimized the workload as a whole from step 2, there you will only need to tune "outlier" SQL statements. Within this step, there are sub-steps:

   (a) *Remove unnecessary large-table full-table scans:* In this tuning step you evaluate the SQL based on the number of rows returned by the query. Standard b-tree indexes can be added to tables, and bitmapped and function-based indexes can also eliminate full-table scans.

   (b) *Cache small-table full-table scans*: In this step we ensure that a dedicated data buffer is available for the rows.

   (c) *Verify optimal index usage*: This step is critical because you may have "missing" indexes in your database, causing excessive I/O.

   (d) *Materialize your aggregations and summaries for static tables*: One features of the Oracle SQLAccess advisor is recommendations for new indexes and suggestions for materialized views.

Task I want to connect databases to my web application? Discuss.

Notes

Lab Exercise Create a student database with important columns like, Student name, Roll No, Course Name, Subject Name, Year etc.. and also make a web page with all necessary fields to display all information on the page.

14.6 Summary

This unit covers the features of database for web, Server Side Database Communication with CGI, Chains of Communication, using Perl 5 and the DBI Module to Communicate With Databases, DBI Module and API, Getting the Pieces, Running CGI Applications on a Single Station Local Area Network along with JDBC.

14.7 Keywords

Chains of communication: A Web Database follows the client-server database model. A Database Engine sits on some central computer somewhere and serves data to multiple web-based clients.

Database administrator: Data administrator is one who oversees the data integrity of the database.

DBI module: The DBI Module is an incredibly useful tool because it abstracts away the need to learn how to communicate with every type of database out there. Consider what a pain it would be if you had to write a CGI script for every proprietary database engine.

URL: Uniform Resource Locator

14.8 Self Assessment

Choose the appropriate answer:

1. URL stands for:
   (a) Uniform Resource Location
   (b) Uniform Resource Locator
   (c) Uniform Region Locator
   (d) Uniform Region Location
2. CGI stands for
   (a) Common Gateway Interface
   (b) Communication Gateway Interface
   (c) Common Gateway interchange
   (d) Communication Gateway Interchange
3. ODBC stands for
   (a) Open Database Communication
   (b) Open Database Connectivity
   (c) Open Database Connection
   (d) Open Document Connectivity

Notes

Fill in the blanks:

4. ..... performs data integration tasks.
5. Data administrators ..... from the business perspective and must have an understanding of the business to be truly effective.
6. ..... is the one area in which the Oracle DBA probably spends most of his or her time.
7. The Web is the cornerstone of .....
8. A Web Database follows the ..... database model.
9. The use of a ..... to invoke a program at a remote site leads us to the role of databases on the Web.
10. The ..... is an incredibly useful tool because it abstracts away the need to learn how to communicate with every type of database out there.

14.9 Review Questions

1. Explain about database for web.
2. What is chains of communication?
3. Using Perl 5 and the DBI module to communicate with databases.
4. Explain the DBI Module and API.
5. Define the role of administrator and also the process of adding and deleting a user to the role of data administrator.
6. How will you access database through web? Explain with the help of suitable example.
7. Create a web page and also create database connect both web page and database.
8. How will you modify table with the help of web pages? Explain
9. Explain general DBI class methods in detail.
10. How will you install Perl? Explain step by step.

Answers: Self Assessment

1.	(b)	2.	(a)
3.	(c)	4.	Data Administrator
5.	view data	6.	Performance tuning
7.	electronic commerce	8.	client-server
9.	Web browser	10.	DBI Module

14.10 Further Readings

Books

C.J. Date, *Introduction to Database Systems*, Pearson Education.

Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.

Notes

Martin Gruber, *Understanding SQL*, BPB Publication, New Delhi

Peter Rob & Carlos Coronel, *Database Systems Design, Implementation and Management,* 7th Edition.

Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.

Silberschatz, Korth, *Database System Concepts*, 5th Edition, McGraw Hill.

SIlberschatz-Korth-Sudarshan, *Database System Concepts*, 4th Edition, Tata McGraw Hill

Vai Occardi, *Relational Database: Theory & Practice*, BPB Publication, New Delhi

Online links

www.en.wikipedia.org

www.webopedia.com

www.web-source.net