CS6302 DATABASE MANAGEMENT SYSTEMS

UNIT I INTRODUCTION TO DBMS 10

File Systems Organization - Sequential, Pointer, Indexed, Direct - Purpose of Database System- Database System Terminologies-Database characteristics- Data models – Types of data models – Components of DBMS- Relational Algebra. LOGICAL DATABASE DESIGN: Relational DBMS - Codd's Rule - Entity-Relationship model - Extended ER Normalization – Functional Dependencies, Anomaly- 1NF to 5NF- Domain Key Normal Form – Denormalization

UNIT II SQL & QUERY OPTIMIZATION 8

SQL Standards - Data types - Database Objects- DDL-DML-DCL-TCL-Embedded SQL-Static Vs Dynamic SQL - QUERY OPTIMIZATION: Query Processing and Optimization - Heuristics and Cost Estimates in Query Optimization.

UNIT III TRANSACTION PROCESSING AND CONCURRENCY CONTROL 8

Introduction-Properties of Transaction- Serializability- Concurrency Control – Locking Mechanisms- Two Phase Commit Protocol-Dead lock.

UNIT IV TRENDS IN DATABASE TECHNOLOGY 10

Overview of Physical Storage Media – Magnetic Disks – RAID – Tertiary storage – File Organization – Organization of Records in Files – Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing - Introduction to Distributed Databases- Client server technology- Multidimensional and Parallel databases- Spatial and multimedia databases- Mobile and web databases- Data Warehouse-Mining- Data marts.

UNIT V ADVANCED TOPICS 9

DATABASE SECURITY: Data Classification-Threats and risks – Database access Control – Types of Privileges –Cryptography- Statistical Databases.- Distributed Databases-Architecture-Transaction Processing-Data Warehousing and Mining-Classification-Association rules-Clustering-Information Retrieval- Relevance ranking-Crawling and Indexing the Web- Object Oriented Databases-XML Databases.

TOTAL: 45 PERIODS

TEXT BOOK:

1. Ramez Elmasri and Shamkant B. Navathe, “Fundamentals of Database Systems”, Fifth Edition, Pearson Education, 2008.

REFERENCES:

1. Abraham Silberschatz, Henry F. Korth and S. Sudharshan, “Database System Concepts”, Sixth Edition, Tata Mc Graw Hill, 2011.
2. C.J.Date, A.Kannan and S.Swamynathan, “An Introduction to Database Systems”, Eighth Edition, Pearson Education, 2006.
3. Atul Kahate, “Introduction to Database Management Systems”, Pearson Education, New Delhi, 2006.
4. Alexis Leon and Mathews Leon, “Database Management Systems”, Vikas Publishing House Private Limited, New Delhi, 2003.
5. Raghu Ramakrishnan, “Database Management Systems”, Fourth Edition, Tata Mc Graw Hill, 2010.
6. G.K.Gupta, “Database Management Systems”, Tata Mc Graw Hill, 2011.
7. Rob Cornell, “Database Systems Design and Implementation”, Cengage Learning, 2011.

UNIT I - INTRODUCTION TO DBMS 10

File Systems Organization - Sequential, Pointer, Indexed, Direct - Purpose of Database System-Database System Terminologies-Database characteristics- Data models – Types of data models – Components of DBMS- Relational Algebra. LOGICAL DATABASE DESIGN: Relational DBMS - Codd's Rule - Entity-Relationship model - Extended ER Normalization – Functional Dependencies, Anomaly- 1NF to 5NF- Domain Key Normal Form – Denormalization

----

UNIT I INTRODUCTION TO DBMS

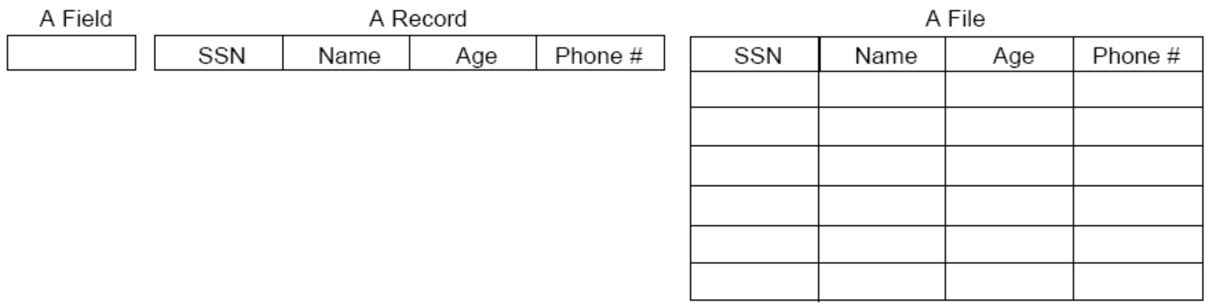
File Systems Organization -Sequential, Pointer, Indexed, Direct -Purpose of Database System-Database System Terminologies-Database characteristics-Data models –Types of data models –Components of DBMS-Relational Algebra. LOGICAL DATABASE DESIGN: Relational DBMS -Codd's Rule-Entity-Relationship model -Extended ER Normalization –Functional Dependencies, Anomaly-1NF to 5NF-Domain Key Normal Form –Denormalization.

1. File Systems Organization -Sequential, Pointer, Indexed, Direct

A file is a collection or set (ordered or unordered) of data elements stored on storage media. A system software module is responsible for managing (reading, processing, deleting, etc.) a file.

Logical Structure of a file

- **Field**: Smallest (fixed) indivisible logical unit of a file. A field holds a part of some data value. **Record**: A set of logically related fields. Cardinality of this set may be fixed or variable, i.e., a record size may be fixed or variable. A file, therefore, is a collection of logically related records.



A record can be of fixed or variable size. Most commonly fixed size records are used. The file structure has been illustrated here in a tabular form but note that a file is not just a table its records can be organized in many different ways each way defines a unique organization.

Operations on file Create: Create a new file.

- **Write**: Write to a file
- **Read**: Read a part of a file
- **Rewind**: Go to the beginning of the file
- **Delete**: Remove a file from the system.
- **Close**: Close a file (at the end of manipulation).
- **Open**: Open a file for processing.
- **Modify**: Modify a field value.

There are basically three categories of file organizations (a) Sequential organization, (b) organization using Index, and (c) Random organization. We begin with sequential category. Files of these categories are called sequential files.

Sequential file

In this organization records are written consecutively when the file is created. Records in a sequential file can be stored in two ways. Each way identifies a file organization.

• **Pile file**: Records are placed one after another as they arrive (no sorting of any kind).
• **Sorted file**: Records are placed in ascending or descending values of the primary key.

File Reorganization:

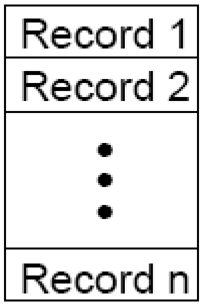
In file reorganization all records, which are marked to be deleted are deleted and all inserted records are moved to their correct place (sorting). File reorganization steps are:

• read the entire file (all blocks) in RAM.
• remove all the deleted records.
• write all the modified blocks at a different place on the disk.

Inserting a record: To insert a record, it is placed at the end of the file. No need to sort (ascending or descending order) the file. However, the file may have duplicate records.

Deleting or modifying a record: This will require fetching the block containing the record, finding the record in the block and just marking it deleted, and then writing the modified block to the disk.

Sorted Sequential File: In a sorted file, first the record is inserted at the end of the file and then moved to its correct location (ascending or descending). Records are stored in order of the values of the key field.



A sequential file organization

A sequential file usually has an overflow area. This area is to avoid sorting the file after every deletion, insertion and/or modification. All records, which were added after the file was first populated, go to this overflow area. The overflow area is not itself sorted it is a pile file with fixed size record. At some convenient time the entire file is sorted where records from the overflow area go to their correct position in the main file.

Retrieval: Records are retrieved in a consecutive order. The order of record storage determines order of retrieval. During retrieval several required operations (partial result output etc.) can be performed simultaneously.

Insert, delete and modify (Update): Sequential files are usually not updated in place. Each operation regenerates a new version of the file. The new version contains the up-to-date information and the old version is usually saved for recovery. The new version becomes the old version and the next update to the file uses this old version to generate next new version. The intensity or frequency of use of a sequential file is defined by a parameter called ―*Hit ratio*‖, which defines is defined as follows:

$$Hit\ ratio = \frac{No. of\ records\ accessed\ for\ responding\ to\ a\ query}{Total\ number\ of\ records\ in\ the\ file}.$$

Desirable: high hit ratio value. This means a larger number of records are accessed to respond to a query. Interactive transactions have very low hit ratio.

Advantages of sequential file

- Good for batch transactions.
- Simple to implement
- Good for report generation, statistical computation and inventory control.

Disadvantages

- Not good for interactive transactions
- High overheads in file processing for simple queries.

Index File Organization

Index organization tries to reduce access time. It may not reduce the storage requirement of a file. Some important terms of this organization are.

Index: An index is similar to a pointer with an additional property. This additional property allows an index to identify the record it is pointing to. For example, an index to a record of employees points and identifies an employee record. This means that an index has some semantics associated with it. Formally, an index maps the *key space* to the *record space*. Index for a file may be created on the *primary* or *secondary* keys.

Index types: Primary, Non dense, and Dense.
Primary Index: An ordered file of index record. This file contains index records which are of fixed length. Each index record has two fields:

- one field holds the primary key of the data file record.
- the other holds pointer to disk block where the data file is stored.

Nondense index: No. of entries in the index file << no. of records in the data file.
Dense index: No. of entries in the index file = no. of records in the data file.
Example: How record search time is reduced by indexing.

Index Sequential File (ISAM - Index Sequential Access Method)

Introduced by IBM in early 1960s. This file organization closely related to the physical characteristics of the storage media. It has two parts: (a) Index Part and (b) Data Part. Index Part: This part stores pointers to the actual record location on the disk. It may have several levels and each level represents some physical layout such as sector, cylinder, of the storage media. Data Part: It holds actual data records and is made up of two distinct areas: Prime area and Overflow area. The prime area holds records of the file. The overflow area holds records when the prime area overflows.

Direct File Indexing provides the most powerful and flexible tools for organizing files. However, (a) indexes take up space and time to keep them organized and (b) the amount of I/O increases with the size of index. This problem can be minimized by direct file organization where the address of the desired record can be found directly (no need of indexing or sequential search). Such file are created using some *hashing* function so they are called *hashing organization* or *hashed files*.

Hashing Two types of hashing: (a) Static and (b) Dynamic.

- **Static**: The address space size is predefined and does not grow or shrink with file.
- **Dynamic**: The address space size can grow and shrink with file.

<u>2. Purpose of Database System:</u>

Database management systems were developed to handle the following difficulties of typical file-processing systems supported by conventional operating systems.

- Data redundancy and Consistency
- Self Describing Nature (of Database System)
- Data isolation or Abstraction
- Integrity
- Atomicity
- Concurrent Access or sharing of Data
- Security
- Support for multiple views of Data

(i) Data redundancy and Consistency:

In file System, each user maintains separate files and programs to manipulate these files because each requires some data not available from other user‘s files. This redundancy in defining and storage of data results in

- wasted storage space,
- redundant efforts to maintain common update,
- higher storage and access cost and
- Leads to inconsistency of data (ie.,) various copies of same data may not agree.

In Database approach, a single repository of data is maintained that is maintained that is defined once and then accessed by various users. Thus redundancy is controlled and it is consistent.

(ii) Self Describing Nature of Database System:

In File System, the structure of the data file is embedded in the access programs. A database system contains not only the database itself but also a complete definition or description of database structure and constraints. This definition is stored in System catalog which contains information such as structure of each file, type and storage format of each data item and various constraints on the data. Information stored in the catalog is called Meta-Data. DBMS is not written for specific applications, hence it must refer to catalog to know structure of file etc., and hence it can work equally well with any number of database applications.

(iii) Data Isolation or Abstraction:

Conventional File processing systems do not allow data to be retrieved in convenient and efficient manner. More responsive data retrieval systems are required for general use. The structure of the data file is embedded in the access programs. So any changes to the structure of a file may require changing all programs that access this file. Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve appropriate data is difficult. But the DBMS access programs do not require such changes in most cases. The structure of data files is stored in DBMS catalog separately from the access programs. This property is known as program data independence. Operations are separately specified and can be changed without affecting the interface. User application programs can operate on data by invoking these operations regardless of how they are implemented. This property is known as program operation independence. Program data independence and program operation independence are together known as data independence.

(iv) Enforcing Integrity Constraints:

The data values must satisfy certain types of consistency constraints. In File System, Developers enforce constraints by adding appropriate code in application program. When new constraints are added, it is difficult to change the programs to enforce them.

In data base system, DBMS provide capabilities for defining and enforcing constraints. The constraints are maintained in system catalog. Therefore application programs work independently with addition or modification of constraints. Hence integrity problems are avoided.

(v) Atomicity:

A Computer system is subjected to failure. If failure occurs, the data has to be restored to the consistent state that existed prior to failure. The transactions must be atomic – it must happen in entirety or not at all. It is difficult to ensure atomicity in File processing System. In DB approach, the DBMS ensures atomicity using the Transaction manager inbuilt in it. DBMS supports online transaction processing and recovery techniques to maintain atomicity.

(vi) Concurrent Access or sharing of Data:

When multiple users update the data simultaneously, it may result in inconsistent data. The system must maintain supervision which is difficult because data may be accessed by many different application programs that may have not been coordinated previously. The database (DBMS) include concurrency control software to ensure that several programs /users trying to update the same data do so in controlled manner, so that the result of update is correct. (vii) Security: Every user of the database system should not be able to access the data. But since the application programs are added to the system in an adhoc manner, enforcing such security constraints is difficult in file system. DBMS provide security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.

(viii) Support for multiple views of Data: Database approach support multiple views of data. A database has many users each of whom may require a different view of the database. View may be a subset of database or virtual data retrieved from database which is not explicitly stored. DBMS provide multiple views of the data or DB.Different application programs are to be written for different views of data.

3. Database System Terminologies:

a) File system:

A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. It is controlled by Operating system.

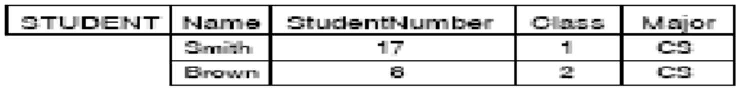
b) Data:

Data are Known facts that can be recorded and that have implicit meaning

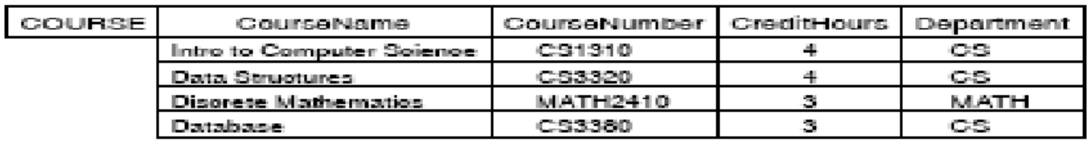
c) Database: Database is a collection of related data with some inherent meaning. It is designed built and populated with data for specific purpose. It represents some aspects of real world.

d) Database Management System: It is a collection of programs that enables to create and maintain database. It is a general purpose Software system that facilitates the process of defining, constructing and manipulating data bases for various applications.

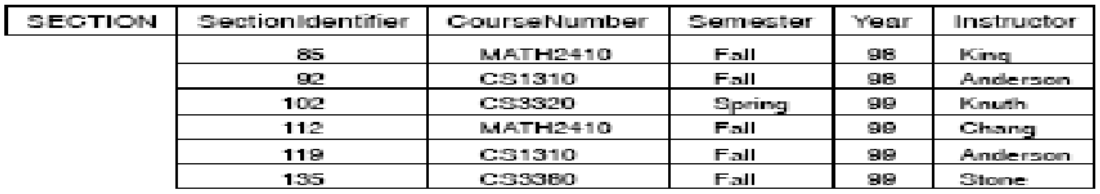
Defining involves specifying data types, structures and constraints for the data to be stored in the database.
Constructing is the process of storing the database itself on some storage medium that is controlled by DBMS.
Manipulating includes functions such as querying the database to retrieve specific data, updating database to reflect changes and generating reports from the data. **Eg.** Oracle, Ms-access, Sybase, Informix, Foxpro.



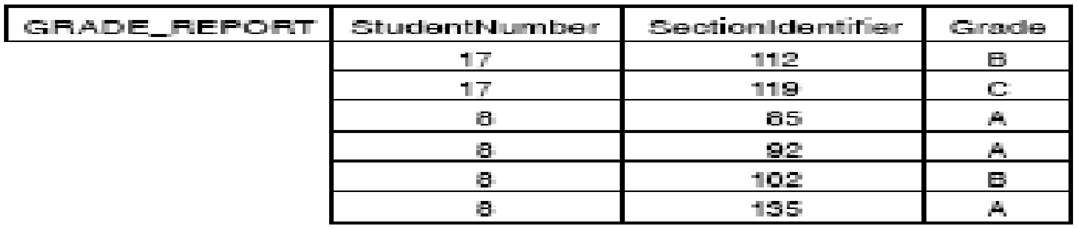
STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS



COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

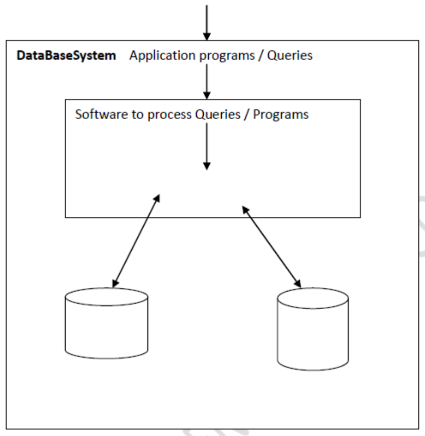


SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone



GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

Example for a database - A University Database
e) Database System: Database and DBMS together known as Database system.
User / programmers


DataBaseSystem
Application programs / Queries
Software to process Queries / Programs

Database System Environment

f) Applications of Database System:

- Banking – Customer information, accounts, loans and transactions
- Airlines – Reservation and schedule information

- Universities – Student, course, registration and grades
- Credit and Transactions – purchases on cards and report
- Telecommunications – Records of calls, report, balances etc.,
- Finance – Holdings, sales, stocks and bond purchase.
- Sales – customer, product, purchase.
- Manufacturing – Inventory items, orders, purchase.
- Human Resources-Employee details, salary, tax etc.,

4. Database Characteristics:

- Self-Describing Nature of a Database System
- Insulation between Programs and Data, and Data Abstraction
- Support of Multiple Views of the Data
- Sharing of Data and Multiuser Transaction Processing

Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the system catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data, and it describes the structure of the primary database. The catalog is used by the DBMS software and also by database users who need information about the database structure. A general purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with *any number of database applications*—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog. DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions. Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record.

Insulation between Programs and Data, and Data Abstraction

The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property *program-data independence*. In a DBMS environment, we just need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birthdate; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used. An operation (also called a *function*) is specified in two parts. The *interface* (or *signature*) of an operation includes the operation name and the data types of its arguments (or parameters). The *implementation* (or *method*) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed *program-operation* independence. The characteristic that allows program-data independence and program-operation independence is called *data abstraction*. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model *hides* storage and implementation details that are not of interest to most database users.

Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A *view* may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of applications must provide facilities for defining multiple views. For example, one user of the database may be interested only in the transcript of each student; the view for this user is displayed. A second user, who is interested only in checking that students have taken all the prerequisites of each course they register for, may require the different view.

Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger. These types of applications are generally called *on-line transaction processing* (OLTP) applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly.

5. Data Models

5.1 Definitions:

a) Data Model: Data model is a collection of concepts that can be used to describe the structure of a database (ie., Data, Relationships, Data types and constraints).
b) Schema: Complete definition and description of the database is known as database schema. Each object in the schema is known as schema construct. It is known as Intension.
c) Data Base State: The data in the database at a particular moment in time is called a database state or snapshot. It is known as the extension of database schema.
DBMS restores the description of the schema constructs and constraints (meta data) in DBMS Catalog.

5.2 Types of Data model:

- High level or Conceptual: Close to users
- Low level or Physical: how data is stored in computer
- Representational or Implementational: Concepts understood by users and not too far from the way data is organized Eg. Network, Hierarchical Model.

5.3 DBMS Architecture:

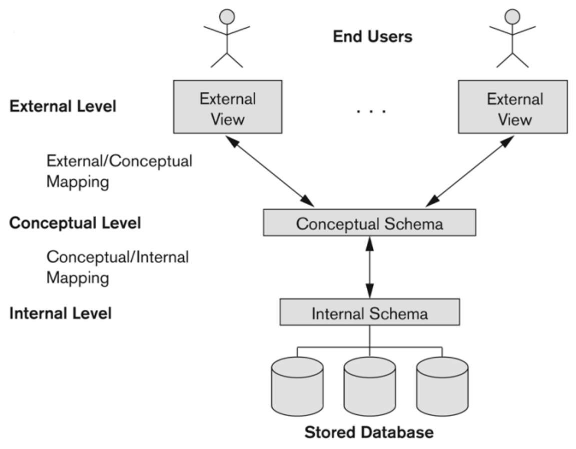
- Three schema architecture
- Achieve the database characteristics

Three Schema Architecture: Separates the user applications and physical database. Schemas can be defined in three levels:

(i) Internal Level:
- It has an internal schema which describes physical storage structure of the database.
- How the data are actually stored
- Uses physical model
- Describes the complete details of data storage and access paths for the database.

(ii) Conceptual Level:
- It has an conceptual schema which describes the structure of whole database
- What data are stored and what relationships exist among data.
- Uses high level or implementational data model.
- Hides the details of physical storage structures and describes datatypes, relationships, operations and constraints.

(iii) External or View Level:
- Includes a number of external schemas or views.
- Each external schema describes the part of the database and hides the rest.
- Uses high level or implementational data model.



DBMS architecture
- **Data Independence:** The capacity to change the schema at one level of database system without having to change the schema at the next higher level.
- **Logical Data independence:** The capacity to change the conceptual schema without having to change the External schema.
- **Physical Data Independence:** The capacity to change the physical schema without having to change the Conceptual schema.

5.4 Data base Application Architecture:
- Client : Machine in which remote database users work
- Server: machine in which database system runs.

Two-tier Architecture:
- Application is partitioned into component that resides at the client machine, which invokes database system functionality at server machine through query language statements.

- Eg. API – ODBC & JDBC used for Interaction.

Three tier Architecture:
- The client merely acts as a front end and does not contain any direct database calls.
- Client end communicates with application server usually through form interface.
- Application server (has Business logic) communicates with database system.

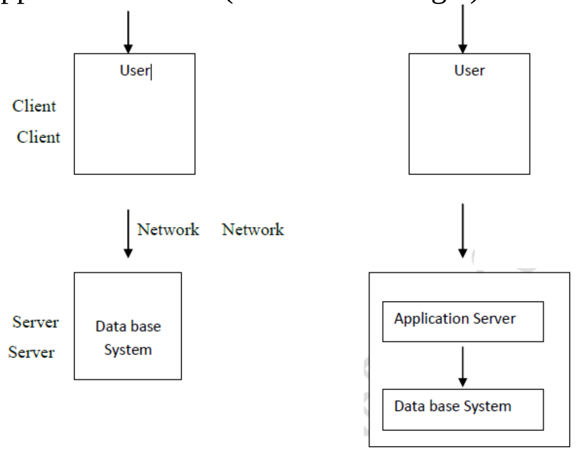
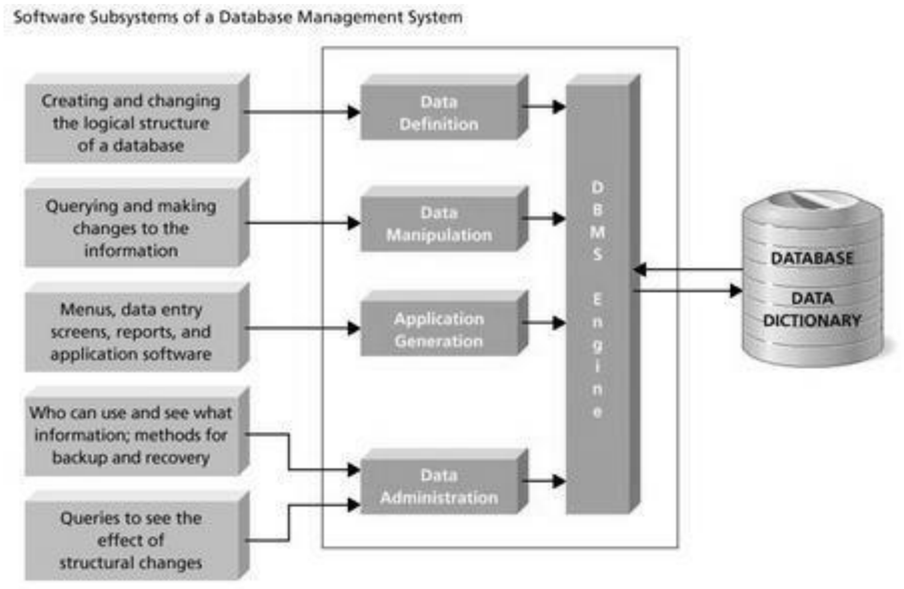


Fig 1.3 Two – Tier and Three Tier Architecture

<u>Components of DBMS</u>

DBMSs are the technology tools that directly support managing organizational data. With a DBMS you can create a database including its logical structure and constraints, you can manipulate the data and information it contains, or you can directly create a simple database application or reporting tool. Business applications, which perform the higher level tasks of managing business processes, interact with end users and other applications and, to store and manage data, rely on and directly operate their own underlying database through a standard programming interface like ODBC. The following diagram illustrates the five components of a DBMS.


Software Subsystems of a Database Management System
Creating and changing the logical structure of a database
Querying and making changes to the information
Menus, data entry screens, reports, and application software
Who can use and see what information; methods for backup and recovery
Queries to see the effect of structural changes
Data Definition
Data Manipulation
Application Generation
Data Administration
DBMS Engine
DATABASE
DATA DICTIONARY

Database Engine:

The Database Engine is the core service for storing, processing, and securing data. The Database Engine provides controlled access and rapid transaction processing to meet the requirements of the most demanding data consuming applications within your enterprise. Use the Database Engine to create relational databases for online transaction processing or online analytical processing data. This includes creating tables for storing data, and database objects such as indexes, views, and stored procedures for viewing, managing, and securing data. You can use SQL Server Management Studio to manage the database objects, and SQL Server Profiler for capturing server events.

Data dictionary:

A data dictionary is a reserved space within a database which is used to store information about the database itself. A data dictionary is a set of table and views which can only be read and never altered. Most data dictionaries contain different information about the data used in the enterprise. In terms of the database representation of the data, the data table defines all schema objects including views, tables, clusters, indexes, sequences, synonyms, procedures, packages, functions, triggers and many more. This will ensure that all these things follow one standard defined in the dictionary.

The data dictionary also defines how much space has been allocated for and / or currently in used by all the schema objects. A data dictionary is used when finding information about users, objects, and schema and storage structures. Every time a data definition language (DDL) statement is issued, the data dictionary becomes modified. A data dictionary may contain information such as:

- Database design information
- Stored SQL procedures
- User permissions
- User statistics
- Database process information
- Database growth statistics
- Database performance statistics

Query Processor:

A relational database consists of many parts, but at its heart are two major components: the storage engine and the query processor. The storage engine writes data to and reads data from the disk. It manages records, controls concurrency, and maintains log files.The query processor accepts SQL syntax, selects a plan for executing the syntax, and then executes the chosen plan. The user or program interacts with the query processor, and the query processor in turn interacts with the storage engine. The query processor isolates the user from the details of execution: The user specifies the result, and the query processor determines how this result is obtained. The query processor components include

- DDL interpreter
- DML compiler
- Query evaluation engine

Report writer/ generator:

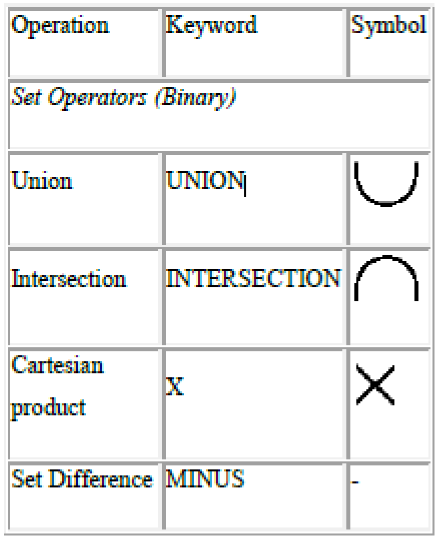
Report writer is a program, usually part of a database management system that extracts information from one or more files and presents the information in a specified format. Most report writers allow you to select records that meet certain conditions and to display selected fields in rows and columns. You can also format data into pie charts, bar charts, and other diagrams. Once you have created a format for a report, you can save the format specifications in a file and continue reusing it for new data.

7. Relational Algebra:

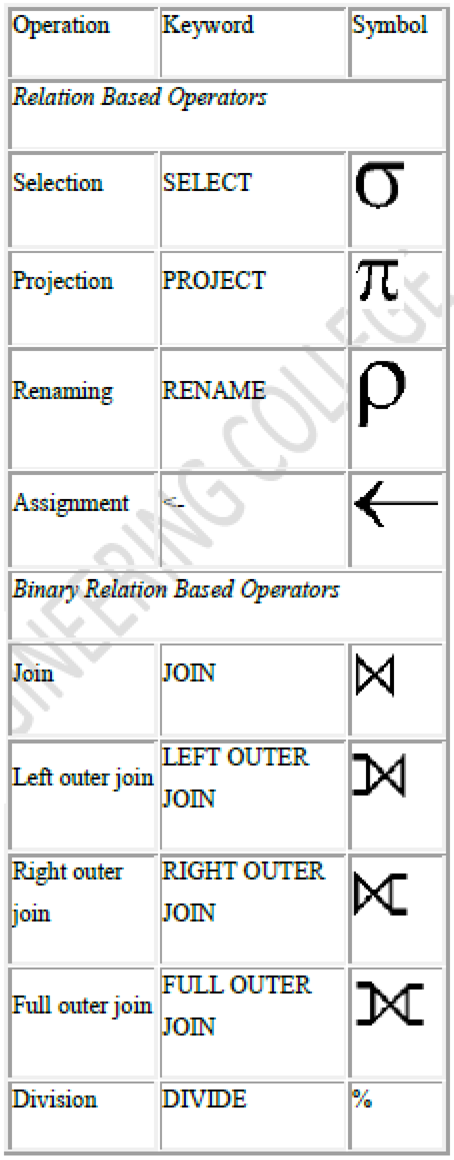
Relational algebra is a procedural query language, which consists of basic set of relational model operations. These operations enable the user to specify basic retrieval requests. It takes one or more relations as input and result of retrieval is a relation. Relational Algebra operations are divided into two groups

(i) Set operations
(ii) Fundamental operations (Relation based operations)

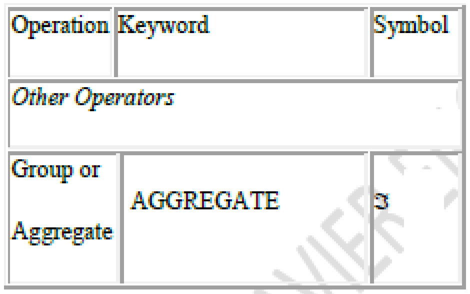
Relational Algebra Notations:



Operation	Keyword	Symbol
Set Operators (Binary)		
Union	UNION	∪
Intersection	INTERSECTION	∩
Cartesian product	X	×
Set Difference	MINUS	-



Operation	Keyword	Symbol
Relation Based Operators		
Selection	SELECT	σ
Projection	PROJECT	π
Renaming	RENAME	ρ
Assignment	<-	←
Binary Relation Based Operators		
Join	JOIN	⋈
Left outer join	LEFT OUTER JOIN	⟕
Right outer join	RIGHT OUTER JOIN	⟖
Full outer join	FULL OUTER JOIN	⟗
Division	DIVIDE	%



Operation	Keyword	Symbol
Other Operators		
Group or Aggregate	AGGREGATE	ℑ

Relational Algebra

A **query language** is a language in which user requests information from the database. it can be categorized as either **procedural** or **nonprocedural.** In a procedural language the user instructs the system to do a sequence of operations on database to compute the desired result. In nonprocedural language the user describes the desired information without giving a specific procedure for obtaining that information.

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produces a new relation as output.

Fundamental Operations

- SELECT
- PROJECT
- UNION
- SET DIFFERENCE
- CARTESIAN PRODUCT
- RENAME

Select and project operations are unary operation as they operate on a single relation. Union, set difference, Cartesian product and rename operations are binary operations as they operate on pairs of relations.

Other Operations

- SET INTERSECTION
- NATURAL JOIN
- DIVISION
- ASSIGNMENT

The select operation: - to identify a set of tuples which is a part of a relation and to extract only these tuples out. The select operation selects tuples that satisfy a given predicate or condition. It is a unary operation defined on a single relation.

- It is denoted as **σ**.

Consider the following table "Book" :-
Code:

```
+--------+--------+------------------+
| Acc-no | Yr-pub | title            |
+--------+--------+------------------+
| 734216 | 1982   | Algorithm design |
| 237235 | 1995   | Database systems |
| 631523 | 1992   | Compiler design  |
| 543211 | 1991   | Programming      |
| 376112 | 1992   | Machine design   |
+--------+--------+------------------+
```

Example1:- Select from the relation “Book” all the books whose year of publication is 1992.
Code:

```
σ Yr-pub=1992(Book)
```

Example2:- Select from the relation “Book” all the books whose Acc-no is greater than equal to 56782.
Code:

```
σ  Acc-no>=56782(Book)
```

The project operation: - returns its argument relation with certain attributes left out. It is a unary operation defined on a single relation
It is denoted as **Π**.
Example:- List all the Title and Acc-no of the “Book” relation.
Code:

```
Π Acc-no, Title (Book)
```

The union operation: - is used when we need some attributes that appear in either or both of the two relations.

- It is denoted as **U**.

Example:
Borrower (customer-name, loan-number)
Depositor (customer-name, account-number)
Customer (customer-name, street-number, customer-city)

List all the customers who have either an account or a loan or both
Code:

```
Π customer-name (Borrower) U Π customer-name (Depositor)
```

The set difference operation: - finds tuples in one relation but not in other.

- It is denoted as –

Example:
Find the names of all customers who have an account but not a loan.
Code:

```
Π customer-name (Depositor) - Π customer-name (Borrower)
```

The Cartesian product operation: - allows combining information from two relations.

- It is denoted as **r X s** where r and s are relations.

Consider the following relation or table "r" :-
Code:

```
+--------+-------+
| A     | B     |
+--------+-------+
|  a   |  1  |
|  b   |  2  |
|  a   |  2  |
+--------+-------+
```

Consider another relation or table "s" :-

Code:

```
+--------+-------+
| B     | C     |
+--------+-------+
|  3   |  1a |
|  2   |  2b |
+--------+-------+
```

Therefore, rXs gives:-
Code:

```
+-----------+---------+---------+----------+
| r.A     |r.B     | s.B    | s.C      |
+-----------+---------+---------+----------+
|  a      |  1     |   3    | 1a      |
|  a      |  1     |   2    | 2b      |
|  b      |  2     |   3    | 1a      |
|  b      |  2     |   2    | 2b      |
|  a      |  2     |   3    | 1a      |
|  a      |  2     |   2    | 2b      |
+-----------+---------+---------+----------+
```

If relation r has n1 tuples and relation s has n2 tuples then **r X s** has n1*n2 tuples.

Example:
Borrower (customer-name, loan-number)
Loan (loan-number, branch-name, city, amount)

List the names of all customers who have a loan in "Perryridge" branch
Code:

```
Π customer-name (σ Borrower.loan-number=Loan.loan-number (σ branch-name="Perryridge" (Borrower X Loan)))
```

The rename operation: - used to rename.
It is denoted as **ρ**.

E **:** relational algebra expression

ρ x (E): returns the result of expression E under the name x.
ρ x (A1, A2, A3… An) (E): returns the result of expression E under the name x with attributes renamed to A1, A2, A3… An.

The set intersection operation: - finds tuples in both the relations.

- It is denoted as ∩.

Example:
Borrower (customer-name, loan-number)
Depositor (customer-name, account-number)
Customer (customer-name, street-number, customer-city)

List all the customers who have both a loan and an account.
Code:

```
Π customer-name (Borrower) ∩ Π customer-name (Depositor)
```

The natural join operation: -
it is a binary operation and a combination of certain selections and a Cartesian product into one operation.

- It is denoted as |X| .
- It is associative.

It forms a Cartesian product of its two arguments.
Then performs a selection forcing equality on those attributes those appear in both the relations.
And finally removes duplicates attributes.

r(R): r is a relation with attributes R.
s(S): s is a relation with attributes S.

If R ∩ S = Φ i.e. they have no attributes in common then **r |X| s = r X s**

The division / quotient operation: -

- It is denoted as ÷.

Letr(R) and s(S) be relations

r ÷ s: - the result consists of the restrictions of tuples in r to the attribute names unique to R, i.e. in the Header of r but not in the Header of s, for which it holds that all their combinations with tuples in s are present in r.
Example:
Relation or table "r":-
Code:

```
+--------+-------+
| A    | B    |
+--------+-------+
|  a   |  1  |
|  b   |  2  |
|  a   |  2  |
|  p   |  3  |
|  p   |  4  |
+--------+-------+
```

Relation or table "s":-
Code:

```
+------+
| B  |
+------+
| 2  |
| 3  |
+------+
```

Therefore, r ÷ s

Code:

```
+------+
| A   |
+------+
| b   |
| a   |
| p   |
+------+
```

<u>7. Logical Database Design: Relational DBMS - Codd's Rule</u>

A **relational database management system (RDBMS)** is a database management system (DBMS) that is based on the relational model as introduced by *E. F. Codd*. Most popular commercial and open source databases currently in use are based on the relational model.

A short definition of an RDBMS may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.

E.F. Codd, the famous mathematician has introduced 12 rules for the relational model for databases commonly known as **Codd's rules**. The rules mainly define what is required for a DBMS for it to be considered *relational*, i.e., an RDBMS. There is also one more rule i.e. Rule00 which specifies the relational model should use the relational way to manage the database. The rules and their description are as follows:-

Rule 0: Foundation Rule A *relational database management system* should be capable of using its *relational* facilities (exclusively) to *manage* the *database*.

Rule 1: Information Rule All information in the database is to be represented in one and only one way. This is achieved by values in column positions within rows of tables.

Rule 2: Guaranteed Access Rule All data must be accessible with no ambiguity, that is, Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

Rule 3: Systematic treatment of null values Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

Rule 4: Dynamic On-line Catalog Based on the Relational Model The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data. The authorized users can access the database structure by using common language i.e. SQL.

Rule 5: Comprehensive Data Sublanguage Rule A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

- data definition
- view definition
- data manipulation (interactive and by program)
- integrity constraints
- authorization
- Transaction boundaries (begin, commit, and rollback).

Rule 6: View Updating Rule All views that are theoretically updateable are also updateable by the system.

Rule 7: High-level Insert, Update, and Delete The system is able to insert, update and delete operations fully. It can also perform the operations on multiple rows simultaneously.

Rule 8: Physical Data Independence Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

Rule 9: Logical Data Independence Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

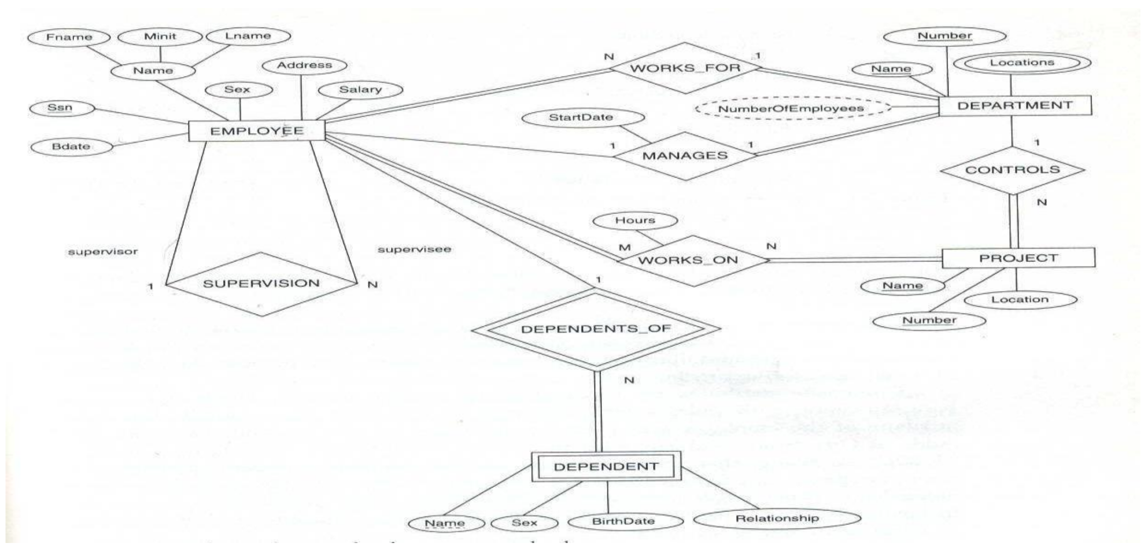
Rule 10: Integrity Independence Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

Rule 11: Distribution Independence The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

Rule 12: Nonsubversion Rule If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

ENTITY-RELATIONSHIP MODEL

- The basic object that the ER model represents is an entity, which is a "thing" in the real world with an independent existence.
- An entity may be an object with a physical existence—a particular person, car, house, or employee—or it may be an object with a conceptual existence—a company, a job, or a university course.
- Each entity has attributes—the particular properties that describe it.
- For example, an employee entity may be described by the employee‘s name, age, address, salary, and job.
- A particular entity will have a value for each of its attributes.
- The attribute values that describe each entity become a major part of the data stored in the database.



Types of attributes:

- Simple versus Composite - Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

- Single-Valued versus Multivalued - Most attributes have a single value for a particular entity; such attributes are called single-valued. A multivalued attribute may have lower and upper bounds on the number of values allowed for each individual entity.

- Stored versus Derived - In some cases two (or more) attribute values are related.

Entity Types, Entity Sets, Keys, and Value Sets

Entity Types and Entity Sets

- A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.
- An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.
- Figure shows two entity types, named EMPLOYEE and COMPANY, and a list of attributes for each. A few individual entities of each type are also illustrated, along with the values of their attributes.
- The collection of all entities of a particular entity type in the database at any point in time is called an entity set; the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.
- An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name.
- Attribute names are enclosed in ovals and are attached to their entity type by straight lines.
- Composite attributes are attached to their component attributes by straight lines.
- Multivalued attributes are displayed in double ovals.

- An entity type describes the schema or intension for a set of entities that share the same structure.
- The collection of entities of a particular entity type are grouped into an entity set, which is also called the extension of the entity type.

Key Attributes of an Entity Type

- An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes.
- An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely.
- For example, the Name attribute is a key of the COMPANY entity type in Figure, because no two companies are allowed to have the same name. For the PERSON entity type, a typical key attribute is SocialSecurityNumber.
- Sometimes, several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity. If a set of attributes possesses this property, we can define a composite attribute that becomes a key attribute of the entity type.
- In ER diagrammatic notation, each key attribute has its name underlined inside the oval,

Value Sets (Domains) of Attributes

- Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. If the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Extended Entity-Relationship (EER) Model

The extended entity-relationship (EER) model is a language for definition of structuring (and functionality) of database or information systems. It uses inductive development of structuring.

Basic attributes are assigned to base data types. *Complex attributes* can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed.

Entity types conceptualize structuring of things of reality through attributes.

Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined on the corresponding type. Typical integrity constraint of the extended entity-relationship model is participation, lookup and general cardinality constraints.

Entity, cluster and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

The extended entity-relationship model is mainly used as a language for conceptualization of the structure of information systems applications. Conceptualization of database or information systems aims in a representation of the logical and physical structure of an information system in a given database management system (or for a database paradigm), so that it contains all the information required by the user and required for the efficient behavior of the whole information system for all users. Furthermore, conceptualization may target to specify the database application processes and the user interaction. Description of structuring is currently the main use of the extended ER model. The diagram representation of EER schemata uses rectangles and diamonds for the entity and relationship types.

Functional Dependencies, Anomaly-1NF to 5NF

Functional Dependencies

A functional dependency is a constraint between two sets of attributes from the database.
Let R be the relational schema R={A1,A2,…An}. A functional dependency denoted by X→ Y Between two sets of attributes X and Y that are subset of R specifies a constraint on the possible tuples that can forma a relation state r of R.
X→ Y --- X functionally determines Y (or) there is a functional dependency from X to Y (or) Y is functionally dependent on X.
X → L.H.S of F.D.
Y → R.H.S of F.D.

Definition 1: X functionally determines Y in a relational schema R iff whenever two tuples of r(R) agree on their X value , they must necessarily agree on their Y value. F.D. is a property of semantics or meaning odf attributes.

Definition 2 :

For any two tuples t1 and t2 in r if t1[x]=t2[x], we must have t1[y]=t2[y], ie., values of Y component of a tuple in r depend on and determined by the values of X component.

Normal Forms:

Normalization was proposed by Codd (1972)

- It takes the relational schema through a series tests whether it satisfies a certain normal form.
- It proceeds in Top down fashion Design by analysis.
- Codd has defined I, II., III NF and a stronger definition of 3NF known as BCNF.(Boyce Codd Normal Form)
- All these normal forms are based on f.ds among attributes of a relation.

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let’s discuss about anomalies first then we will discuss normal forms with examples.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee’s id, emp_name for storing employee’s name, emp_address for storing employee’s address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn’t allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.
To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization
Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)
As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says "each attribute of a table must have atomic (single) values", the emp_mobile values for employees Jon & Lester violates that rule.
To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)
A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age
The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".
To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- **Transitive functional dependency** of non-prime attribute on any super key should be removed.

An attribute that is not part of any **candidate key** is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a **super key** of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.
Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)
It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every **functional dependency**X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}
The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}
This is now in BCNF as in both the functional dependencies left side part is a key.

Domain-Key Normal Form (DKNF)

There is no hard and fast rule about defining normal forms only up to 5NF. Historically, the process of normalization and the process of discovering undesirable dependencies was carried through 5NF as a meaningful design activity, but it has been possible to define stricter normal forms that take into account additional types of dependencies and constraints. The idea behind domain-key normal form (DKNF) is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints. A relation is said to be in DKNF if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and key constraints on the relation. For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.

Denormalization:

The ultimate goal during normalization is to separate the logically related attributes into tables to minimize redundancy, and thereby avoid the update anomalies that lead to an extra processing overhead to maintain consistency in the database.

Denormalization doesn‘t mean not normalizing. It‘s a step in the process.

First we normalize, then we realize that we now have hundreds or thousands of small tables and that the performance cost of all those joins will be prohibitive, and then carefully apply some denormalization techniques so that the application will return results before the werewolves have overrun the city.

Since normalization is about reducing redundancy, denormalizing is about deliberately adding redundancy to improve performance. Before beginning, consider whether or not it‘s necessary.Data Normalization, Denormalization, and the Forces of Darkness / Hollingsworth / p18

● Is the system‘s performance unacceptable with fully normalized data? Mock up a client and do some testing.

● If the performance is unacceptable, will denormalizing make it acceptable? Figure out where your bottlenecks are.

● If you denormalize to clear those bottlenecks, will the system and its data still be reliable? Unless the answers to all three are ―yes,‖ denormalization should be avoided. Unfortunately for me, the Council of Light has some pretty old hardware and can‘t or won‘t upgrade to newer and more efficient SQL software. I‘d better get to work.

UNIT II SQL & QUERY OPTIMIZATION 8

SQL Standards - Data types - Database Objects- DDL-DML-DCL-TCL-Embedded SQL-Static Vs Dynamic SQL - QUERY OPTIMIZATION: Query Processing and Optimization - Heuristics and Cost Estimates in Query Optimization.

SQL Standards

The SQL language may be considered one of the major reasons for the commercial success of relational databases. Because it became a standard for relational databases, users were less concerned about migrating their database applications from other types of database systems. The advantage of having such a standard is that users may write statements in a database application program that can access data stored in two or more relational DBMSs without having to change the database sublanguage (SQL) if both relational DBMSs support standard SQL. The name **SQL** is presently expanded as Structured Query Language. Originally, SQL was called SEQUEL (Structured English Query Language).

SQL is now the standard language for commercial relational DBMSs. A joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) has led to a standard version of SQL (ANSI 1986), called SQL-86 or SQL1. A revised and much expanded standard called SQL-92 (also referred to as SQL2) was subsequently developed. The next standard that is well-recognized is SQL:1999, which started out as SQL3. Two later updates to the standard are SQL:2003 and SQL:2006, which added XML features among other updates to the language. Another update in 2008 incorporated more object database features in SQL. We will try to cover the latest version of SQL as much as possible. SQL is a comprehensive database language: It has statements for data definitions, queries, and updates. Hence, it is both a DDL *and* a DML.

Data types

The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.

- **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION).
- **Character-string** data types are either fixed length—CHAR(*n*) or CHARACTER(*n*), where *n* is the number of characters—or varying length— VARCHAR(*n*) or CHAR VARYING(*n*) or CHARACTER VARYING(*n*), where *n* is the maximum number of characters.
- **Bit-string** data types are either of fixed length *n*—BIT(*n*)—or varying length—BIT VARYING(*n*), where *n* is the maximum number of bits.
- A **Boolean** data type has the traditional values of TRUE or FALSE.
- The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.
- A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.
- Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type.

Database Objects

A database object in a relational database is a data structure used to either store or reference data. The most common object that people interact with is the table. Other objects are indexes, stored procedures, sequences, views and many more.

When a database object is created, a new object type cannot be created because all the various object types created are restricted by the very nature, or source code, of the relational database model being used, such as Oracle, SQL Server or Access. What is being created is instances of the objects, such as a new table, an index on that table or a view on the same table.

Most of the major database engines offer the same set of major database object types:

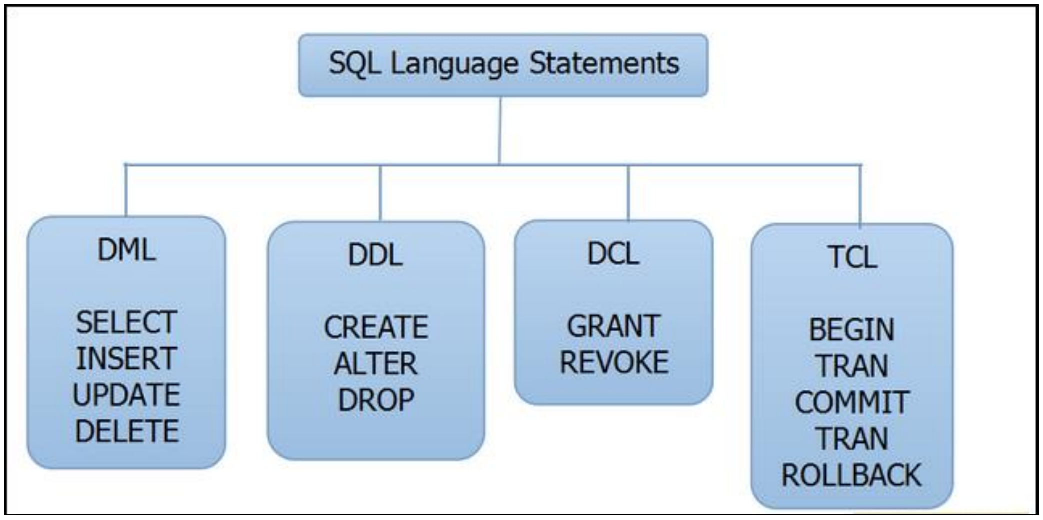
- Tables
- Indexes
- Sequences
- Views
- Synonyms

DDL-DML-DCL-TCL

SQL language is divided into four types of primary language statements: DML, DDL, DCL and TCL. Using these statements, we can define the structure of a database by creating and altering database objects, and we can manipulate data in a table through updates or deletions. We also can control which user can read/write data or manage transactions to create a single unit of work.

The four main categories of SQL statements are as follows:

1. **DML (Data Manipulation Language)**
2. **DDL (Data Definition Language)**
3. **DCL (Data Control Language)**
4. **TCL (Transaction Control Language)**



DML (Data Manipulation Language)

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:

SELECT – select records from a table

INSERT – insert new records

UPDATE – update/Modify existing records
DELETE – delete existing records

DDL (Data Definition Language)

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.
CREATE – create a new Table, database, schema
ALTER – alter existing table, column description
DROP – delete existing objects from database

DCL (Data Control Language)

DCL statements control the level of access that users have on database objects.
GRANT – allows users to read/write on certain database objects
REVOKE – keeps users from read/write permission on database objects

TCL (Transaction Control Language)
TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.
BEGIN Transaction – opens a transaction
COMMIT Transaction – commits a transaction
ROLLBACK Transaction – ROLLBACK a transaction in case of any error

<u>**Embedded SQL**</u>

The technique is for the SQL statements where that can be embedded in a general-purpose programming language. The two languages:C and Java. The examples used with the C language, known as **embedded SQL.** In this embedded approach, the programming language is called the **host language**. Most SQL statements—including data or constraint definitions, queries, updates, or view definitions—can be embedded in a host language program.

SQL provides a powerful declarative query language. However, access to a database from a general-purpose programming language is required because,

- SQL is not as powerful as a general-purpose programming language. There are queries that cannot be expressed in SQL, but can be programmed in C, Fortran, Pascal, Cobol, etc.
- Nondeclarative actions -- such as printing a report, interacting with a user, or sending the result to a GUI -- cannot be done from within SQL.

The SQL standard defines embedding of SQL as *embedded SQL* and the language in which SQL queries are embedded is referred as *host language*.
The result of the query is made available to the program one tuple (record) at a time.
To identify embedded SQL requests to the preprocessor, we use EXEC SQL statement:
EXEC SQL embedded SQL statement END-EXEC.

Note: A semi-colon is used instead of END-EXEC when SQL is embedded in C or Pascal.
Embedded SQL statements: **declare cursor**, **open**, and **fetch** statements.

```
EXEC SQL
        declare c cursor for
        select cname, ccity
         from deposit, customer
```

where *deposit.cname* = *customer.cname* **and** *deposit.balance* > *:amount*
END-EXEC
where *amount* is a host-language variable.

EXEC SQL **open** *c* END-EXEC
This statement causes the DB system to execute the query and to save the results within a temporary relation.

A series of **fetch** statement are executed to make tuples of the results available to the program.
EXEC SQL **fetch** *c* **into** *:cn, :cc* END-EXEC

The program can then manipulate the variable *cn* and *cc* using the features of the host programming language.
A single **fetch** request returns only one tuple. We need to use a **while** loop (or equivalent) to process each tuple of the result until no further tuples (when a variable in the SQLCA is set).
We need to use **close** statement to tell the DB system to delete the temporary relation that held the result of the query.
EXEC SQL **close** *c* END-EXEC

Embedded SQL can execute any valid **update**, **insert**, or **delete** statements.

Examples:

```
0) int loop ;
1) EXEC SQL BEGIN DECLARE SECTION ;
2) varchar dname [16], fname [16], lname [16], address [31] ;
3) char ssn [10], bdate [11], sex [2], minit [2] ;
4) float salary, raise ;
5) int dno, dnumber ;
6) int SQLCODE ; char SQLSTATE [6] ;
7) EXEC SQL END DECLARE SECTION ;
```

```
//Program Segment E1: Program segment E1, a C program segment with embedded SQL
0) loop = 1 ;
1) while (loop) {
2) prompt("Enter a Social Security Number: ", ssn) ;
3) EXEC SQL
4) select Fname, Minit, Lname, Address, Salary
5) into :fname, :minit, :lname, :address, :salary
6) from EMPLOYEE where Ssn = :ssn ;
7) if (SQLCODE == 0) printf(fname, minit, lname, address, salary)
8) else printf("Social Security Number does not exist: ", ssn) ;
9) prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
10) }
```

Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

```
//Program Segment E2:
0) prompt("Enter the Department Name: ", dname) ;
1) EXEC SQL
2) select Dnumber into :dnumber
3) from DEPARTMENT where Dname = :dname ;
4) EXEC SQL DECLARE EMP CURSOR FOR
5) select Ssn, Fname, Minit, Lname, Salary
6) from EMPLOYEE where Dno = :dnumber
```

```
FOR UPDATE OF Salary ;
EXEC SQL OPEN EMP ;
EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
while (SQLCODE == 0) {
printf("Employee name is:", Fname, Minit, Lname) ;
prompt("Enter the raise amount: ", raise) ;
EXEC SQL
update EMPLOYEE
set Salary = Salary + :raise
where CURRENT OF EMP ;
EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
}
EXEC SQL CLOSE EMP ;
```

Static Vs Dynamic SQL

Static SQL is SQL statements in an application that do not change at runtime and, therefore, can be hard-coded into the application. **Dynamic SQL** is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own queries. Thus, the SQL statements cannot be hard-coded into the application.

Static SQL provides performance advantages over dynamic SQL because static SQL is pre-processed, which means the statements are parsed, validated, and optimized only once.

If you are using a standards-based API, such as ODBC, to develop your application, static SQL is probably not an option for you. However, you can achieve a similar level of performance by using either statement pooling or stored procedures.

S.No.	**Static (embedded) SQL**	**Dynamic (interactive) SQL**
1.	In static SQL how database will be accessed is predetermined in the embedded SQL statement.	In dynamic SQL, how database will be accessed is determined at run time.
2.	It is more swift and efficient.	It is less swift and efficient.
3.	SQL statements are compiled at compile time.	SQL statements are compiled at run time.
4.	Parsing, validation, optimization, and generation of application plan are done at compile time.	Parsing, validation, optimization, and generation of application plan are done at run time.
5.	It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non-uniformly.
6.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.
7.	It is less flexible.	It is more flexible.

QUERY OPTIMIZATION: Query Processing and Optimization

A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated. The **scanner** identifies the query tokens—such as SQL keywords, attribute names, and relation names—that appear in the text of the query, whereas the **parser** checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language. The query must also be **validated** by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created, usually as a tree data structure called a **query tree**. It is also possible to represent the query using a graph data structure called a **query graph**. The DBMS must then devise an **execution strategy** or **query plan** for retrieving the results of the query from the database files. A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as **query optimization**.

Introduction to Query Processing

- In databases that provide low-level access routines such as IMS or flat file databases, the programmer must write code to perform the queries.
- With higher level database query languages such as SQL and QUEL, a special component of the DBMS called the Query Processor takes care of arranging the underlying access routines to satisfy a given query.
- Thus queries can be specified in terms of the required results rather than in terms of how to achieve those results.

A query is processed in four general steps:

1. Scanning and Parsing
2. Query Optimization or planning the execution strategy
3. Query Code Generator (interpreted or compiled)
4. Execution in the runtime database processor

1. Scanning and Parsing

- When a query is first submitted (via an applications program), it must be scanned and parsed to determine if the query consists of appropriate syntax.
- **Scanning** is the process of converting the query text into a tokenized representation.
- The tokenized representation is more compact and is suitable for processing by the parser.
- This representation may be in a tree form.
- The **Parser** checks the tokenized representation for correct syntax.
- In this stage, checks are made to determine if columns and tables identified in the query exist in the database and if the query has been formed correctly with the appropriate keywords and structure.
- If the query passes the parsing checks, then it is passed on to the Query Optimizer.

2. Query Optimization or Planning the Execution Strategy

- For any given query, there may be a number of different ways to execute it.
- Each operation in the query (SELECT, JOIN, etc.) can be implemented using one or more different *Access Routines*.
- For example, an access routine that employs an index to retrieve some rows would be more efficient that an access routine that performs a full table scan.
- The goal of the **query optimizer** is to find a *reasonably efficient* strategy for executing the query (not quite what the name implies) using the access routines.
- Optimization typically takes one of two forms: *Heuristic Optimization* or *Cost Based Optimization*
- In **Heuristic Optimization**, the query execution is refined based on *heuristic rules* for reordering the individual operations.
- With **Cost Based Optimization**, the overall cost of executing the query is systematically reduced by estimating the costs of executing several different execution plans.

3. Query Code Generator (interpreted or compiled)

- Once the query optimizer has determined the execution plan (the specific ordering of access routines), the code generator writes out the actual access routines to be executed.
- With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution.
- It is also possible to *compile* the access routines and store them for later execution.

4. Execution in the runtime database processor

- At this point, the query has been scanned, parsed, planned and (possibly) compiled.

- The runtime database processor then executes the access routines against the database.
- The results are returned to the application that made the query in the first place.
- Any runtime errors are also returned.

Query optimization is a function of many relational database management systems. The **query optimizer** attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs. However, some database engines allow guiding the query optimizer with hints.

A query is a request for information from a database. Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information.

Heuristic Query Optimization

Oracle calls this Rule Based optimization.

A query can be represented as a tree data structure. Operations are at the interior nodes and data items (tables, columns) are at the leaves.

The query is evaluated in a *depth-first* pattern.

Consider this query from the Elmasri/Navathe textbook:

```
SELECT PNUMBER, DNUM, LNAME
FROM   PROJECT, DEPARTMENT, EMPLOYEE
WHERE  DNUM=DNUMBER and MGREMPID=EMPID and
       PLOCATION = 'Stafford';
```

Or, in relational algebra:

$$\pi_{\text{pnumber, dnum, lname}} \; (\sigma_{\text{plocation = 'Stafford'}} \; (\sigma_{\text{mgrssn=ssn}} \; (\sigma_{\text{dnum=dnumber}}$$

on the following schema:

```
EMPLOYEE TABLE
FNAME    MI LNAME       EMPID BDATE     ADDRESS                  S SALARY SUPERMPID DNO
-------- -- ------- --------- --------- ------------------------ - ------ --------- --
JOHN     B  SMITH   123456789 09-JAN-55 731 FONDREN, HOUSTON, TX M  30000 333445555 5
FRANKLIN T  WONG    333445555 08-DEC-45 638 VOSS,HOUSTON TX      M  40000 888665555 5
ALICIA   J  ZELAYA  999887777 19-JUL-58 3321 CASTLE, SPRING, TX  F  25000 987654321 4
JENNIFER S  WALLACE 987654321 20-JUN-31 291 BERRY, BELLAIRE, TX  F  43000 888665555 4
RAMESH   K  NARAYAN 666884444 15-SEP-52 975 FIRE OAK, HUMBLE, TX M  38000 333445555 5
JOYCE    A  ENGLISH 453453453 31-JUL-62 5631 RICE, HOUSTON, TX   F  25000 333445555 5
AHMAD    V  JABBAR  987987987 29-MAR-59 980 DALLAS, HOUSTON, TX  M  25000 987654321 4
JAMES    E  BORG    888665555 10-NOV-27 450 STONE, HOUSTON, TX   M  55000           1
```

```
DEPARTMENT TABLE:

DNAME            DNUMBER  MGREMPID MGRSTARTD
```

```
--------------- --------- --------- ---------
HEADQUARTERS            1 888665555 19-JUN-71
ADMINISTRATION          4 987654321 01-JAN-85
RESEARCH                5 333445555 22-MAY-78
```

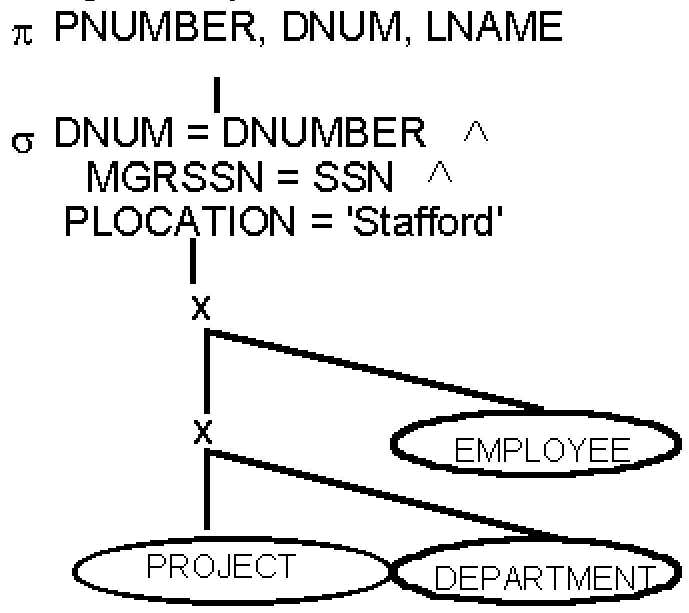
```
PROJECT TABLE
PNAME            PNUMBER  PLOCATION   DNUM
---------------- -------  ----------  ----
ProductX               1  Bellaire     5
ProductY               2  Sugarland    5
ProductZ               3  Houston      5
Computerizatn.        10  Stafford     4
Reorganization        20  Houston      1
NewBenefits           30  Stafford     4
```

- An overall rule for heuristic query optimization is to perform as many select and project operations as possible before doing any joins.
- There are a number of transformation rules that can be used to transform a query:
    1. Cascading selections. A list of conjunctive conditions can be broken up into separate individual conditions.
    2. Commutativity of the selection operation.
    3. Cascading projections. All but the last projection can be ignored.
    4. Commuting selection and projection. If a selection condition only involves attributes contained in a projection clause, the two can be commuted.
    5. Commutativity of Join and Cross Product.
    6. Commuting selection with Join.
    7. Commuting projection with Join.
    8. Commutativity of set operations. Union and Intersection are commutative.
    9. Associativity of Union, Intersection, Join and Cross Product.
    10. Commuting selection with set operations.
    11. Commuting projection with set operations.
    12. Logical transformation of selection conditions. For example, using DeMorgan's law, etc.
    13. Combine Selection and Cartesian product to form Joins.

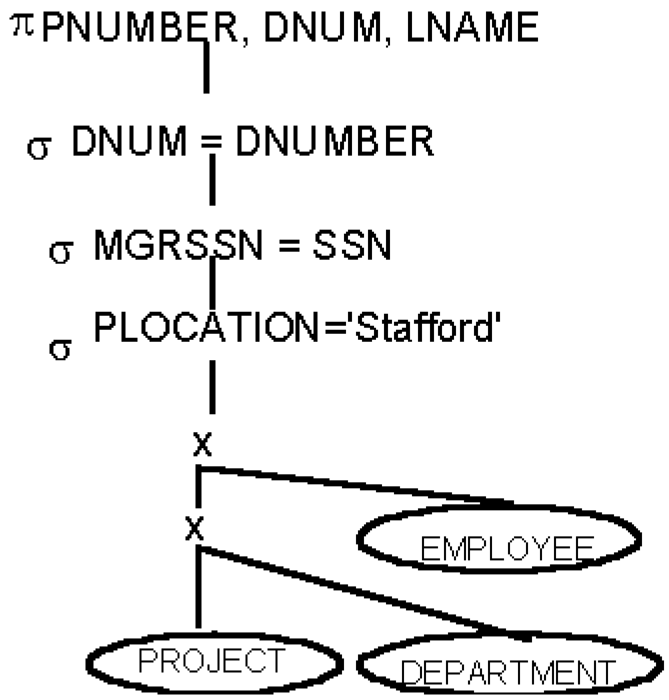
- These transformations can be used in various combinations to optimize queries. Some general steps follow:
    1. Using rule 1, break up conjunctive selection conditions and chain them together.
    2. Using the commutativity rules, move the selection operations as far down the tree as possible.
    3. Using the associativity rules, rearrange the leaf nodes so that the most restrictive selection conditions are executed first. For example, an equality condition is likely more restrictive than an inequality condition (range query).
    4. Combine cartesian product operations with associated selection conditions to form a single Join operation.
    5. Using the commutativity of Projection rules, move the projection operations down the tree to reduce the sizes of intermediate result sets.
    6. Finally, identify subtrees that can be executed using a single efficient access method.

Example of Heuristic Query Optimization

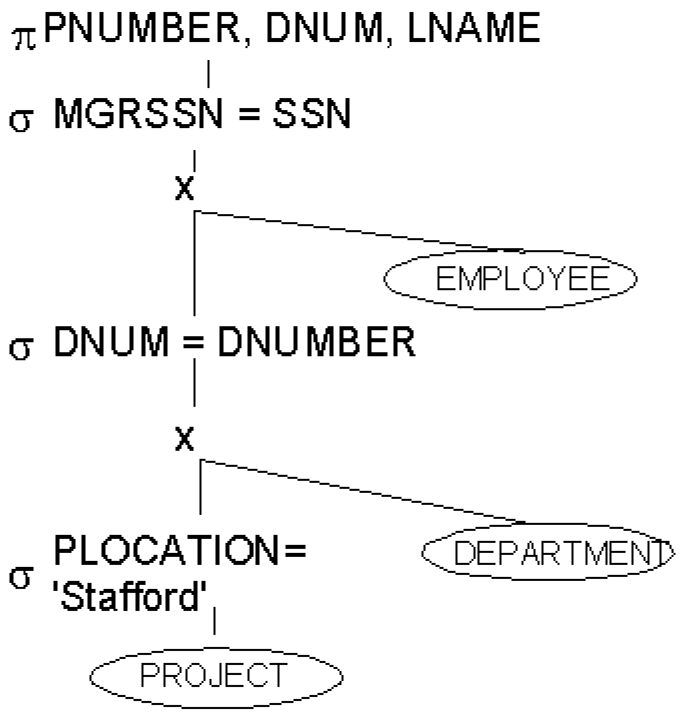
1. Original Query Tree



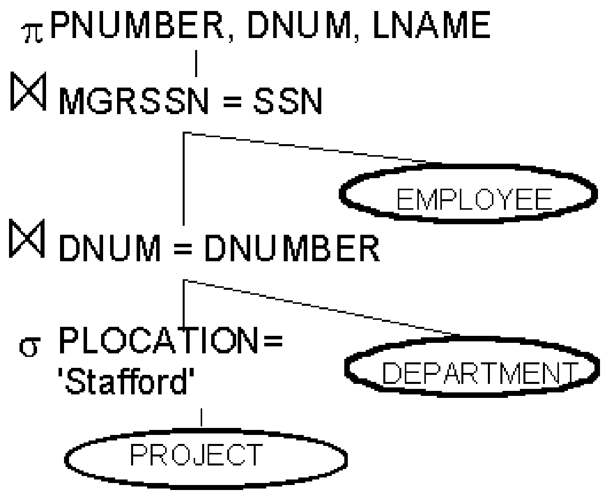
2. Use Rule 1 to Break up Cascading Selections



3. Commute Selection with Cross Product



4. Combine Cross Product and Selection to form Joins



<u>**Cost Estimates in Query Optimization**</u>

A query optimizer does not depend solely on heuristic rules; it also estimates and compares the costs of executing a query using different execution strategies and algorithms, and it then chooses the strategy with the *lowest cost estimate*. For this approach to work, accurate *cost estimates* are required so that different strategies can be compared fairly and realistically.
The cost of executing a query includes the following components:

- Access cost to secondary storage.
- Disk storage cost.
- Computation cost.
- Memory usage cost.

- Communication cost.

Example to Illustrate Cost-Based Query Optimization

We will consider query Q2 and its query tree to illustrate cost-based query optimization:

```
Q2:     SELECT Pnumber, Dnum, Lname, Address, Bdate
        FROM PROJECT, DEPARTMENT, EMPLOYEE
        WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';
```

The first cost-based optimization to consider is join ordering. As previously mentioned, we assume the optimizer considers only left-deep trees, so the potential join orders—without CARTESIAN PRODUCT—are:

1. PROJECT $\bowtie$ DEPARTMENT $\bowtie$ EMPLOYEE
2. DEPARTMENT $\bowtie$ PROJECT $\bowtie$ EMPLOYEE
3. DEPARTMENT $\bowtie$ EMPLOYEE $\bowtie$ PROJECT
4. EMPLOYEE $\bowtie$ DEPARTMENT $\bowtie$ PROJECT

UNIT III TRANSACTION PROCESSING AND CONCURRENCY CONTROL 8
Introduction-Properties of Transaction- Serializability- Concurrency Control – Locking Mechanisms- Two Phase Commit Protocol-Dead lock.

Introduction

The concept of transaction provides a mechanism for describing logical units of database processing. Transaction processing systems are systems with large databases and hundreds of concurrent users executing database transactions. Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications. These systems require high availability and fast response time for hundreds of concurrent users. A transaction is typically implemented by a computer program, which includes database commands such as retrievals, insertions, deletions, and updates.

Single-User versus Multiuser Systems

One criterion for classifying a database system is according to the number of users who can use the system concurrently. A DBMS is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system—and hence access the database—concurrently. Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser. For example, an airline reservations system. Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems. Multiple users can access databases—and use computer systems—simultaneously because of the concept of multiprogramming, which allows the operating system of the computer to execute multiple programs—or processes—at the same time. A single central processing unit (CPU) can only execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on.

Transactions, Database Items, Read and Write Operations, and DBMS Buffers

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations—these can include insertion, deletion, modification, or retrieval operations. One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction; otherwise it is known as a read-write transaction. A database is basically represented as a collection of *named data items.* The size of a data item is called its granularity.

A data item can be a *database record*, but it can also be a larger unit such as a whole *disk block*, or even a smaller unit such as an individual *field (attribute) value* of some record in the database. The DBMS will maintain in the database cache a number of data buffers in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed. **Concurrency control and recovery mechanisms** are mainly concerned with the database commands in a transaction. Several problems can occur when concurrent transactions execute in an uncontrolled manner.

- The Lost Update Problem.

- The Temporary Update (or Dirty Read) Problem
- The Incorrect Summary Problem.
- The Unrepeatable Read Problem

If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

- A computer failure (system crash).
- A transaction or system error/
- Local errors or exception conditions detected by the transaction.
- Concurrency control enforcement
- Disk failure
- Physical problems and catastrophes

Properties of Transaction

Transactions should possess several properties, often called the **ACID** properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

■ **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

■ **Consistency preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

■ **Isolation.** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

■ **Durability or permanency.** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

- **Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case

these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.
To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Equivalence Schedules
An equivalence schedule can be of the following types −

Result Equivalence
If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

View Equivalence
Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.
For example −

- If T reads the initial data in S1, then it also reads the initial data in S2.
- If T reads the value written by J in S1, then it also reads the value written by J in S2.
- If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Conflict Equivalence
Two schedules would be conflicting if they have the following properties −

- Both belong to separate transactions.
- Both accesses the same data item.
- At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

Concurrency Control

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

A **lock** is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items.
Types of Locks and System Lock Tables
Several types of locks are used in concurrency control.

- Binary locks, which are simple, but are also too restrictive for database concurrency control purposes
- Shared/exclusive locks—also known as read/write locks—which provide more general locking capabilities and are used in practical database locking schemes
- Certify lock, - show how it can be used to improve performance of locking protocols.

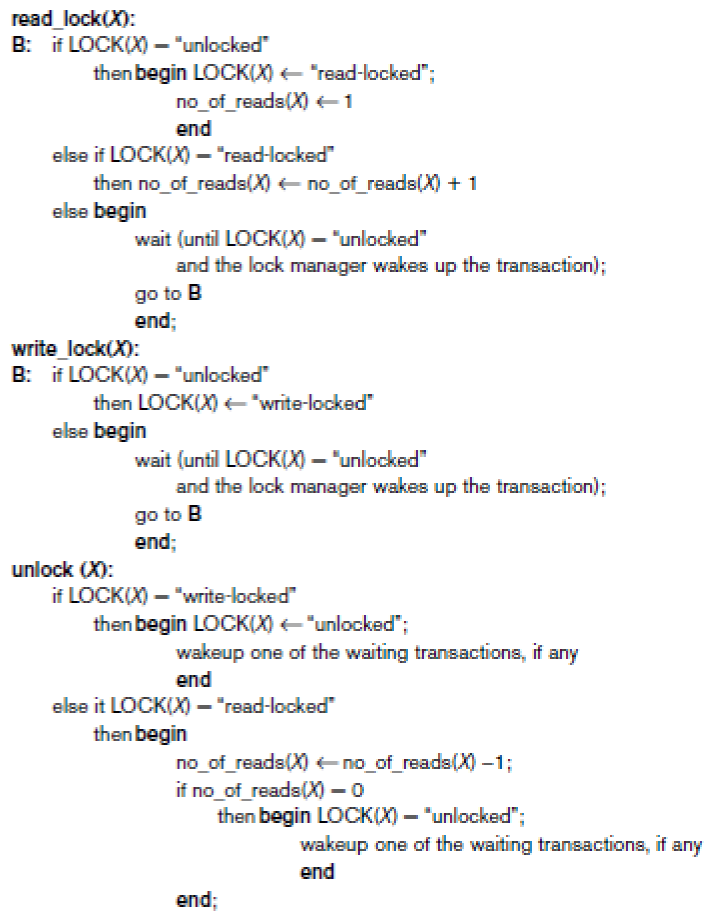
Binary Locks. A **binary lock** can have two **states** or **values:** locked and unlocked (or 1 and 0, for simplicity). Two operations, lock_item and unlock_item, are used with binary locking.

```
lock_item(X):
B:  if LOCK(X) = 0                (* item is unlocked *)
        then LOCK(X) ←1           (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                      (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

If the simple binary locking scheme described here is used, every transaction must obey the following rules:
1. A transaction *T* must issue the operation lock_item(*X*) before any read_item(*X*) or write_item(*X*) operations are performed in *T*.
2. A transaction *T*must issue the operation unlock_item(*X*) after all read_item(*X*) and write_item(*X*) operations are completed in *T*.
3. A transaction *T* will not issue a lock_item(*X*) operation if it already holds the lock on item *X*.1
4. A transaction *T* will not issue an unlock_item(*X*) operation unless it already holds the lock on item *X*.

Shared/Exclusive (or Read/Write) Locks. The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for reading purposes only. a different type of lock called a multiple-mode lock is used.
In this scheme—called shared/exclusive or read/write locks—there are three locking operations: _lock(X), write_lock(X), and unlock(X).
A read-locked item is also called share-locked because other transactions are allowed to read the item, whereas a write- read locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item. Each record in the lock table will have four fields: <Data_item_name, LOCK, No_of_reads, Locking_transaction(s)>. The value (state) of LOCK is either read-locked or write-locked, suitably coded (if we assume no records are kept in the lock table for unlocked items).
When we use the shared/exclusive locking scheme, the system must enforce the following rules:
1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.

2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.



```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else if LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) −1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.3
4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X. This rule may be relaxed, as we discuss shortly.
5. A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X. This rule may also be relaxed, as we discuss shortly.
6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.

Conversion of Locks. Sometimes it is desirable to relax conditions 4 and 5 in the preceding list in order to allow lock conversion; that is, a transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another. For example, it is possible for a transaction T to issue a read_lock(X) and then later to upgrade the lock by issuing a write_lock(X) operation. If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded; otherwise, the transaction must wait. It is also possible for a transaction T to issue a write_lock(X) and then later to downgrade the lock by issuing a read_lock(X) operation.

Two Phase Commit Protocol

A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction. Such a transaction can be divided into two phases: an expanding or growing (first) phase, during which new locks on items can be acquired but none can be released; and a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired. If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase. Hence, a read_lock(X) operation that downgrades an already held write lock on X can appear only in the shrinking phase.

It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules. The locking protocol, by enforcing two-phase locking rules, also enforces serializability. Although the two-phase locking protocol guarantees serializability (that is, every schedule that is permitted is serializable), it does not permit all possible serializable schedules (that is, some serializable schedules will be prohibited by the protocol). Basic, Conservative, Strict, and Rigorous Two-Phase Locking. There are a number of variations of two-phase locking (2PL). The technique just described is known as basic 2PL. A variation known as conservative 2PL (or static 2PL) requires a transaction to lock all the items it accesses before the transaction begins execution, by predeclaring its read-set and write-set.

Deadlock

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T_ in the set. Hence, each transaction in the set is in a waiting queue, waiting for one of the other transactions in the set to release the lock on an item. But because the other transaction is also waiting, it will never release the lock.

Deadlock Prevention Protocols. One way to prevent deadlock is to use a deadlock prevention protocol. One deadlock prevention protocol, which is used in conservative two-phase locking, requires that every transaction lock all the items it needs in advance (which is generally not a practical assumption). A second protocol, which also limits concurrency, involves ordering all the items in the database and making sure that a transaction that needs several items will lock them according to that order.

A number of other deadlock prevention schemes have been proposed that make a decision about what to do with a transaction involved in a possible deadlock situation. Some of these techniques use the concept of **transaction timestamp** TS(*T*), which is a unique identifier assigned to each transaction. The rules followed by these schemes are:

■ **Wait-die.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) *Ti* is allowed to wait; otherwise (*Ti* younger than *Tj*) abort *Ti* (*Ti dies*) and restart it later *with the same timestamp.*

■ **Wound-wait.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) abort *Tj* (*Ti wounds Tj*) and restart it later *with the same timestamp;* otherwise (*Ti* younger than *Tj*) *Ti* is allowed to wait.

Deadlock Detection. A second, more practical approach to dealing with deadlock is **deadlock detection**, where the system checks if a state of deadlock actually exists. This solution is attractive if we know there will be little interference among the transactions—that is, if different transactions will rarely access the same items at the same time. A simple way to detect a state of deadlock is for the system to construct and maintain a **wait-for graph**. One node is created in the wait-for graph for each transaction that is currently executing.

Timeouts. Another simple scheme to deal with deadlock is the use of **timeouts**. This method is practical because of its low overhead and simplicity. In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it—regardless of whether a deadlock actually exists or not.

Starvation. Another problem that may occur when we use locking is **starvation**, which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally. This may occur if the waiting scheme for locked items is unfair, giving priority to some transactions over others. One solution for starvation is to have a fair waiting scheme, such as using a **first-come-first-served** queue.

--

UNIT IV TRENDS IN DATABASE TECHNOLOGY 10

Overview of Physical Storage Media – Magnetic Disks – RAID – Tertiary storage – File Organization – Organization of Records in Files – Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing - Introduction to Distributed Databases- Client server technology- Multidimensional and Parallel databases- Spatial and multimedia databases- Mobile and web databases- Data Warehouse-Mining- Data marts.

UNIT V ADVANCED TOPICS 9

DATABASE SECURITY: Data Classification-Threats and risks – Database access Control – Types of Privileges –Cryptography- Statistical Databases.- Distributed Databases-Architecture-Transaction Processing-Data Warehousing and Mining-Classification-Association rules-Clustering-Information Retrieval- Relevance ranking-Crawling and Indexing the Web- Object Oriented Databases-XML Databases.

Overview of Physical Storage Media

Storage media are classified by speed of access, cost per unit of data to buy the media, and by the medium's reliability. Unfortunately, as speed and cost go up, the reliability does down.

1. **Cache** is the fastest and the most costly for of storage. The type of cache referred to here is the type that is typically built into the CPU chip and is 256KB, 512KB, or 1MB. Thus, cache is used by the operating system and has no application to database, per se.
2. **Main memory** is the volatile memory in the computer system that is used to hold programs and data. While prices have been dropping at a staggering rate, the increases in the demand for memory have been increasing faster. Today's 32-bit computers have a limitation of 4GB of memory. This may not be sufficient to hold the entire database and all the associated programs, but the more memory available will increase the response time of the DBMS. There are attempts underway to create a system with the most memory that is cost effective, and to reduce the functionality of the operating system so that only the DBMS is supported, so that system response can be increased. However, the contents of main memory are lost if a power failure or system crash occurs.
3. **Flash memory** is also referred to as *electrically erasable programmable read-only memory (EEPROM)*. Since it is small (5 to 10MB) and expensive, it has little or no application to the DBMS.
4. **Magnetic-disk storage** is the primary medium for long-term on-line storage today. Prices have been dropping significantly with a corresponding increase in capacity. New disks today are in excess of 20GB. Unfortunately, the demands have been increasing and the volume of data has been increasing faster. The organizations using a DBMS are always trying to keep up with the demand for storage. This media is the most cost-effective for on-line storage for large databases.
5. **Optical storage** is very popular, especially CD-ROM systems. This is limited to data that is read-only. It can be reproduced at a very low-cost and it is expected to grow in popularity, especially for replacing written manuals.
6. **Tape storage** is used for backup and archival data. It is cheaper and slower than all of the other forms, but it does have the feature that there is no limit on the amount of data that can be stored, since more tapes can be purchased. As the tapes get increased capacity, however, restoration of data takes longer and longer, especially when only a small

amount of data is to be restored. This is because the retrieval is sequential, the slowest possible method.

Magnetic Disks

A typical large commercial database may require hundreds of disks!

Physical Characteristics of Disks

Disks are actually relatively simple. There is normally a collection of platters on a spindle. Each platter is coated with a magnetic material on both sides and the data is stored on the surfaces. There is a read-write head for each surface that is on an arm assembly that moves back and forth. A motor spins the platters at a high constant speed, (60, 90, or 120 revolutions per seconds.) The surface is divided into a set of tracks (circles). These tracks are divided into a set of sectors, which is the smallest unit of data that can be written or read at one time. Sectors can range in size from 31 bytes to 4096 bytes, with 512 bytes being the most common. A collection of a specific track from both surfaces and from all of the platters is called a cylinder.

Platters can range in size from 1.8 inches to 14 inches. Today, 5 1/4 inches and 3 1/2 inches are the most common, because they have the highest seek times and lowest cost. A disk controller interfaces the computer system and the actual hardware of the disk drive. The controller accepts high-level command to read or write sectors. The controller then converts the commands in the necessary specific low-level commands. The controller will also attempt to protect the integrity of the data by computing and using checksums for each sector. When attempting to read the data back, the controller recalculates the checksum and makes several attempts to correctly read the data and get matching checksums. If the controller is unsuccessful, it will notify the operating system of the failure.

The controller can also handle the problem of eliminating bad sectors. Should a sector go bad, the controller logically remaps the sector to one of the extra unused sectors that disk vendors provide, so that the reliability of the disk system is higher. It is cheaper to produce disks with a greater amount of sectors than advertised and then map out bad sectors than it is to produce disks with no bad sectors or with extremely limited possibility of sectors going bad.
There are many different types of disk controllers, but the most common ones today are SCSI, IDE, and EIDE.

One other characteristic of disks that provides an interesting performance is the distance from the read-write head to the surface of the platter. The smaller this gap is means that data can be written in a smaller area on the disk, so that the tracks can be closer together and the disk has a greater capacity. Often the distance is measured in microns. However, this means that the possibility of the head touching the surface is increased. When the head touches the surface while the surface is spinning at a high speed, the result is called a "head crash", which scratches the surface and defaces the head. The bottom line to this is that someone must replace the disk.

Performance Measures of Disks

1. ***Seek time*** is the time to reposition the head and increases with the distance that the head must move. Seek times can range from 2 to 30 milliseconds. ***Average seek time*** is the average of all seek times and is normally one-third of the worst-case seek time.
2. ***Rotational latency time*** is the time from when the head is over the correct track until the data rotates around and is under the head and can be read. When the rotation is 120 rotations per second, the rotation time is 8.35 milliseconds. Normally, the ***average rotational latency time*** is one-half of the rotation time.
3. ***Access time*** is the time from when a read or write request is issued to when the data transfer begins. It is the sum of the seek time and latency time.

4. ***Data-transfer rate*** is the rate at which data can be retrieved from the disk and sent to the controller. This will be measured as megabytes per second.
5. ***Mean time to failure*** is the number of hours (on average) until a disk fails. Typical times today range from 30,000 to 800,000 hours (or 3.4 to 91 years).

Optimization of Disk-Block Access

Requests for disk I/O are generated by both the file system and by the virtual memory manager found in most systems. Each request specifies the address on the disk to be referenced; that address specifies is in the form of a block number. Each block is a contiguous sequence of sectors from a single track of one platter and ranges from 512 bytes to several kilobytes of data. The lower level file manager must convert block addresses into the hardware-level cylinder, surface, and sector number.

Since access to data on disk is several orders of magnitude slower is access to data in main memory; much attention has been paid to improving the speed of access to blocks on the disk. This is also where more main memory can speed up the response time, by making sure that the data needed is in memory when it is needed.

This is the same problem that is addressed in designing operating systems, to insure the best response time from the file system manager and the virtual memory manager.

- **Scheduling**. Disk-arm scheduling algorithms attempt to order accesses in an attempt to increase the number of accesses that can be processed in a given amount of time. The might include First-Come/First-Serve, Shortest Seek First, and elevator.
- **File organization**. To reduce block-access time, data could be arranged on the disk in the same order that it is expected to be retrieved. (This would be storing the data on the disk in order based on the primary key.) At best, this starts to produce less and less of a benefit, as there are more inserts and deletes. Also we have little control of where on the disk things get stored. The more the data gets fragmented on the disk, the more time it takes to locate it.
- **Nonvolatile write buffer**. Using non-volatile memory (flash memory) can be used to protect the data in memory from crashes, but it does increase the cost. It is possible that the use of an UPS would be more effective and cheaper.
- **Log disk**. You can use a disk for writing a sequential log.
- **Buffering**. The more information you have in buffers in main memory, the more likely you are to not have to get the information from the disk. However it is more likely that more of the memory will be wasted with information not necessary.

RAID

RAIDs are Redundant Arrays of Inexpensive Disks. There are six levels of organizing these disks:

- 0 -- Non-redundant Striping
- 1 -- Mirrored Disks
- 2 -- Memory Style Error Correcting Codes
- 3 -- Bit Interleaved Parity
- 4 -- Block Interleaved Parity
- 5 -- Block Interleaved Distributed Parity
- 6 -- P + Q Redundancy

Tertiary Storage

This is commonly optical disks and magnetic tapes.

Storage Access

A database is mapped into a number of different files, which are maintained by the underlying operating system. Files are organized into block and a block may contain one or more data item. A major goal of the DBMS is to minimize the number of block transfers between the disk and memory. Since it is not possible to keep all blocks in main memory, we need to manage the allocation of the space available for the storage of blocks. This is also similar to the problems encountered by the operating system, and can be in conflict with the operating system, since the OS is concerned with processes and the DBMS is concerned with only one family of processes.

Buffer Manager

Programs in a DBMS make requests (that is, calls) on the buffer manager when they need a block from a disk. If the block is already in the buffer, the requester is passed the address of the block in main memory. If the block in not in the buffer, the buffer manager first allocates space in the buffer for the block, through out some other block, if required, to make space for the new block. If the block that is to be thrown out has been modified, it must first be written back to the disk. The internal actions of the buffer manager are transparent to the programs that issue disk-block requests.

- ***Replacement strategy***. When there is no room left in the buffer, a block must be removed from the buffer before a new one can be read in. Typically, operating systems use a least recently use (LRU) scheme. There is also a Most Recent Used (MRU) that can be more optimal for DBMSs.
- ***Pinned blocks***. A block that is not allowed to be written back to disk is said to be pinned. This could be used to store data that has not been committed yet.
- ***Forced output of blocks***. There are situations in which it is necessary to write back to the block to the disk, even though the buffer space is not currently needed. This might be done during system lulls, so that when activity picks up, a write of a modified block can be avoided in peak periods.

<u>File Organization</u>

Fixed-Length Records

Suppose we have a table that has the following organization:

```
type deposit = record
   branch-name : char(22);
   account-number : char(10);
   balance : real;
end
```

- If each character occupies 1 byte and a real occupies 8 bytes, then this record occupies 40 bytes. If the first record occupies the first 40 bytes and the second record occupies the second 40 bytes, etc. we have some problems.
- It is difficult to delete a record, because there is no way to indicate that the record is deleted. (At least one system automatically adds one byte to each record as a flag to show if the record is deleted.) Unless the block size happens to be a multiple of 40 (which is extremely unlikely), some records will cross block boundaries. It would require two block access to read or write such a record.

One solution might be to compress the file after each deletion. This will incur a major amount of overhead processing, especially on larger files. Additionally, there is the same problem on inserts!

Another solution would be to have two sets of pointers. One that would link the current record to the next logical record (linked list) plus a free list (a list of free slots.) This increases the size the file.

Variable-Length Records
We can use variable length records:

- Storage of multiple record types in one file.
- Record types that allow variable lengths for one or more fields
- Record types that allow repeating fields.

A simple method for implementing variable-length records is to attach a special *end-of-record* symbol at the end of each record. But this has problems:

- To easy to reuse space occupied formerly by a deleted record.
- There is no space in general for records to grow. If a variable-length record is updated and needs more space, it must be moved. This can be very costly.

It could be solved:

- By making a variable-length into a fixed length.
- By using pointers to point to fixed length records, chained together by pointers.

As you can see, there is not an easy answer.

<u>Organization of Records in Files</u>

Heap File Organization
Any record can be placed anywhere in the file. There is no ordering of records and there is a single file for each relation.
Sequential File Organization
Records are stored in sequential order based on the primary key.
Hashing File Organization
Any record can be placed anywhere in the file. A hashing function is computed on some attribute of each record. The function specifies in which block the record should be placed.
Clustering File Organization
Several different relations can be stored in the same file. Related records of the different relations can be stored in the same block.
Data Dictionary Storage
A RDBMS needs to maintain data about the relations, such as the schema. This is stored in a data dictionary (sometimes called a system catalog):

- Names of the relations
- Names of the attributes of each relation
- Domains and lengths of attributes
- Names of views, defined on the database, and definitions of those views
- Integrity constraints
- Names of authorized users
- Accounting information about users
- Number of tuples in each relation
- Method of storage for each relation (clustered/non-clustered)
- Name of the index
- Name of the relation being indexed
- Attributes on which the index in defined
- Type of index formed