MALLA REDDY INSTITUTE OF TECHNOLOGY &SCIENCE
(SPONSORED BY MALLA REDDY EDUCATIONAL SOCIETY)
Permanently Affiliated to JNTUH & Approved by AICTE, New Delhi
NBA Accredited Institution, An ISO 9001:2015 Certified, Approved by UK Accreditation Centre
Granted Status of 2(f) & 12(b) under UGC Act. 1956, Govt. of India.

DATABASE MANAGEMENT SYSTEMS

COURSE FILE

B.TECH CSE III-I

MALLA REDDY INSTITUTE OF TECHNOLOGY AND SCIENCE

(SPONSORED BY MALLA REDDY EDUCATIONAL SOCIETY)
Permanently Affiliated to JNTUH & Approved by AICTE, New Delhi
NAAC &NBA Accredited Institution, An ISO 9001:2015 Certified, Approved by UK Accreditation Centre
Granted Status of 2(f) & 12(b) under UGC Act. 1956, Govt. of India.

Department of Computer Science and Engineering

Programme Objectives

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and

design documentation, make effective presentations, and give and receive clear instructions.

11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Programme Educational Objectives

PEO1: The graduates of the program will understand the concepts and principles of Computer Science and Engineering inclusive of basic sciences.

PEO2: The program enables the learners to provide the technical skills necessary to design and implement computer systems and applications, to conduct open-ended problem solving, and apply critical thinking.

PEO3: The graduates of the program will practice the profession with work effectively on teams to communicate in written and oral form, ethics, integrity, leadership and social responsibility through safe engineering leading them to contribute their might for the good of the human race.

PEO4: The program encourages the students to become lifelong activity and as a means to the creative discovery, development, and implementation of technology as well as to keep up with the dynamic nature of the Computer Science and Engineering discipline.

Program Specific Outcomes

PSO1: Design and development of software applications by using data mining techniques.

PSO2: Enrichment of graduates with global certifications to produce reliable software solutions.

II.SYLLABUS

CS404PC: DATABASE MANAGEMENT SYSTEMS

B.Tech. II Year II Sem.

Course Objectives:

- To understand the basic concepts and the applications of database systems.
- To master the basics of SQL and construct queries using SQL.
- Topics include data models, database design, relational model, relational algebra, transaction Control, concurrency control, storage structures and access techniques.

Course Outcomes:

- Gain knowledge of fundamentals of DBMS, database design and normal forms.
- Master the basics of SQL for retrieval and management of data.
- Be acquainted with the basics of transaction processing and concurrency control.
- Familiarity with database storage structures and access techniques.

UNIT – I

Database System Applications: A Historical Perspective, File Systems versus a DBMS, the Data Model, Levels of Abstraction in a DBMS, Data Independence, Structure of a DBMS.

Introduction to Database Design: Database Design and ER Diagrams, Entities, Attributes, and Entity Sets, Relationships and Relationship Sets, Additional Features of the ER Model, Conceptual Design With the ER Model

UNIT – II

Introduction to the Relational Model: Integrity constraint over relations, enforcing integrity constraints, querying relational data, logical data base design, introduction to views, destroying/altering tables and views.

Relational Algebra, Tuple relational Calculus, Domain relational calculus.

UNIT – III

SQL: QUERIES, CONSTRAINTS, TRIGGERS: form of basic SQL query, UNION, INTERSECT, and EXCEPT, Nested Queries, aggregation operators, NULL values, complex integrity constraints in SQL, triggers and active data bases.

Schema Refinement: Problems caused by redundancy, decompositions, problems related to decomposition, reasoning about functional dependencies, FIRST, SECOND, THIRD normal forms, BCNF, lossless join decomposition, multi-valued dependencies, FOURTH normal form, FIFTH normal form.

UNIT – IV

Transaction Concept, Transaction State, Implementation of Atomicity and Durability, Concurrent Executions, Serializability, Recoverability, Implementation of Isolation, Testing for serializability, Lock Based Protocols, Timestamp Based Protocols, Validation- Based Protocols, Multiple Granularity, Recovery and Atomicity, Log–Based Recovery, Recovery with Concurrent Transactions.

UNIT – V

Data on External Storage, File Organization and Indexing, Cluster Indexes, Primary and Secondary Indexes, Index data Structures, Hash Based Indexing, Tree base Indexing, Comparison of File Organizations, Indexes and Performance Tuning, Intuitions for tree Indexes, Indexed Sequential Access Methods (ISAM), B+ Trees: A Dynamic Index Structure.

TEXT BOOKS:

1. Database Management Systems, Raghurama Krishnan, Johannes Gehrke, Tata Mc Graw Hill 3rd Edition
2. Database System Concepts, Silberschatz, Korth, Mc Graw hill, V edition.

REFERENCE BOOKS:

1. Database Systems design, Implementation, and Management, Peter Rob & Carlos Coronel 7th Edition.
2. Fundamentals of Database Systems, Elmasri Navrate, Pearson Education
3. Introduction to Database Systems, C. J. Date, Pearson Education
4. Oracle for Professionals, The X Team, S.Shah and V. Shah, SPD.
5. Database Systems Using Oracle: A Simplified guide to SQL and PL/SQL,Shah, PHI.
6. Fundamentals of Database Management Systems, M. L. Gillenson, Wiley Student Edition.

III .CLASS TIMETABLE AND INDIVIDUAL TIMETABLE

DAY/TIME	9:15-10:15	10:15-11:15	11:15-12:15pm	12:15 pm - 1:15pm	1:15 pm - 2:00 pm	2:00 pm - 3:00 pm	3:00 pm - 4:00 pm
MON	DS	DBMS LAB			LUNCH	FLAT	DW&DM
TUE	DBMS	CNS	DW&DM	DAA		DS	FLAT
WED	JP	DS	DAA	FLAT		DBMS	SEM /TUT
THU	DW&DM	CNS LAB				DAA	CNS
FRI	DS	DAA/IPR	ACS LAB			DW&DM	DBMS
SAT	DBMS	DBMS/IPR	CNS	DAA		FLAT	LIB \SPO

DAY/TIME	9:15-10:15	10:15 am - 11:15 am	11:15 am - 12:15pm	12:15 pm - 1:15 pm	1:15 pm - 2:00 pm	2:00 pm - 3:00 pm	3:00 pm - 4:00 pm
MON		DBMS LAB			LUNCH		
TUE	DBMS						
WED						DBMS	
THU							
FRI							DBMS
SAT	DBMS						

IV STUDENT ROLL LIST

Name	Student Roll Number	Section:
ABHISHEK TIWARI	20S11A6201	CS
AISHWARYA KATHI	20S11A6202	CS
AKSHITHA GOLI	20S11A6203	CS
ALEKYA KANNA	20S11A6204	CS
BHOVANESHWAR NAIK DHARAVATH	20S11A6205	CS
DEVI PRASAD GANDLA	20S11A6206	CS

GOPI KRISHNA BOJJA	20S11A6207	CS
GOPICHAND PATHIPAKA	20S11A6208	CS
HEMANTH MOULI CHERUPALLI	20S11A6209	CS
HEMANTH PAADI	20S11A6210	CS
HINDU RANI LETHAKULA	20S11A6211	CS
HUMERA BANU	20S11A6212	CS
ISHWARYA PITTALA	20S11A6213	CS
JITENDER REDDY JAMBULLA	20S11A6214	CS
KARTHIK MARTHA	20S11A6215	CS
LALITH GOUD KOSURI	20S11A6216	CS
LAXMI MEGHANA MOUDHGALYA GADE	20S11A6217	CS
MADHU CHARAN CHILIVERI	20S11A6218	CS
MOHAMMED KAIF	20S11A6219	CS
NAGENDER REDDY JEEDIPALLY	20S11A6220	CS
NEHA KOTLA	20S11A6221	CS
NILESH KUMAR METHRI	20S11A6222	CS
NISHITHA ALURI	20S11A6223	CS
NITHIN GADDAM	20S11A6224	CS
PRANAY KUNTA	20S11A6225	CS
RAKESH KADAMANDA	20S11A6226	CS
RANJITH BOYINI	20S11A6227	CS
REVANTH KUMAR BOMMARA	20S11A6228	CS
SAI SHASHANK RAGI	20S11A6229	CS
SAI TEJA ADLA	20S11A6230	CS
SAYETHIA M	20S11A6231	CS
SHAIK ASRAF	20S11A6232	CS
SHRAVANI PASHAPU	20S11A6233	CS
SIVA SAI L	20S11A6234	CS
SMITHA BISTA	20S11A6235	CS
SRI SATYA SAI PAVAN KONA	20S11A6236	CS
SRINIVAS KUNCHETTI	20S11A6237	CS
VAISHNAVI GOURISHETTY	20S11A6238	CS
VARSHITH REDDY BANGLA	20S11A6239	CS
VASANTH REDDY ANKUR	20S11A6240	CS
VIKAS GUNDETI	20S11A6241	CS
VINEELA MALLUGARI	20S11A6242	CS
VISHWAS GARIGE	20S11A6243	CS
YASHWANTH NAYAK LAKAVATH	20S11A6244	CS
BOPPA VAMSHI	20S11A6245	CS
AITHA MADHU	20S11A6246	CS
ANIL KUMAR THIRUPATHI	21S15A6201	CS
ANISH KANJARLA	21S15A6202	CS
DINESH KUMAR JENDRALA	21S15A6203	CS
MOHAMMAD SAMEERUDDIN	21S15A6204	CS
SEKHAR VK	21S15A6205	CS
ABHISHEK TIWARI	20S11A6201	CS
AISHWARYA KATHI	20S11A6202	CS
AKSHITHA GOLI	20S11A6203	CS

ALEKYA KANNA	20S11A6204	CS
BHOVANESHWAR NAIK DHARAVATH	20S11A6205	CS
		CS

V.LESSON PLAN

MALLA REDDY INSTITUTE OF TECHNOLOGY &SCIENCE
(SPONSORED BY MALLA REDDY EDUCATIONAL SOCIETY)
Permanently Affiliated to JNTUH & Approved by AICTE, New Delhi
NBA Accredited Institution, An ISO 9001:2015 Certified, Approved by UK Accreditation Centre
Granted Status of 2(f) & 12(b) under UGC Act. 1956, Govt. of India.

Name of the Faculty	: G.SANDHYA	Academic Year: 2022-2023
Course Code	: CS404PC	Course Name: Data Base Management System
Program	: B.Tech	Branch: Computer Science & Engineering
Year/Sem	: III-I	Section: CSE-CS

Course Objectives:

Cob1: To understand the basic concepts and the applications of database systems.

Cob2:To master the basics of SQL and construct queries using SQL.

Cob3:To understand the relational database design principles.

Cob4:To become familiar with the basic issues of transaction processing and concurrency control.

Cob5:To become familiar with database storage structures and access techniques.

Course Outcomes:

CO 1: Demonstrate the basic elements of a relational database management system.

CO 2: Ability to identify the data models for relevant problems.

CO 3: Ability to design entity relationship model and convert entity relationship diagrams into RDBMS and formulate SQL queries on the data.

CO 4: Apply normalization for the development of application software

R4: Database Development and Management, Lee Chao, Auerbach publications, Taylor & Francis Group. Introduction to Database Systems, C. J. Date, Pearson Education.

NBA ACCREDITED
ISO CERTIFIED
NAAC ACCREDITED
UGC 2(f) & 12(B)
MALLA REDDY INSTITUTE OF TECHNOLOGY & SCIENCE
MRITS
MRITS

VI. UNIT WISE LECTURE NOTES

DEFINITION OF DBMS

DBMS is software which is used to manage the collection of interrelated data.

File systems Vs DBMS:

The typical file processing system is supported by the operating systems. Files are created and manipulated by writing programs so the permanent records are stored in various files. Before the advent of DBMS, organizations typically stored the information using such systems.

Ex: Using COBOL we can maintain several files (collection of records) to access those files we have to go through the application programs which have written for creating files, updating file, inserting the records

The problems in file processing system are

- Data redundancy and consistency
- Difficulty in accessing data
- Data isolation
- Integrity problems
- Atomicity problems
- Security problems

To solve the above problems DBMS has been invented.

View of data:

The man purpose of DBMS is to provide users with an abstract view of the data.

The data abstraction is in three levels.

- Physical level: How the data are actually stored that means what data structures are used to store data on Hard disk
  Ex: Sequential , Tree structured

- Logical Level : What data are stored in database
- View level : It is the part of data base
  Ex: Required records in table.

Instance:

The collection of information stored in the database at a particular moment is called an instance of thedata base.

Data base schema:

- Logical schema
- Physical schema

Data independence:

The ability to modify schema definition in one level with affecting a schema definition in the next higher level is called data independence.

- Physical independence
- Logical independence

Data models:

Underlying the structure of a data base is the data model. The collection of conceptual tools for describing data, data relationships, data semantics.

Three types of data models are there:

- Object based logical model:
  These are used to describe the data at logical level, view level. It is divided into several types.
  Entity relationship model
- Object oriented model
- The semantic data model
- The function data model

Record based logical model :

In contrast to object based model they are used both to specify the overall logical structure of data base and to pride a higher level description of the implementation.

- Relation model
- Network model
- Hierarchical model

Database languages:

Database system provides two types of languages

- Data definition language :

  Database schema is specified by a set of definitions expressed by a special language called DDL.

- Data manipulation language

This enables the user to access or manipulates data organized by the appropriate data model.

Database administrator: the person who has the central control over the DBMS is called the data base administrator (DBA).

The functions of DBA:

- Schema definition
- Access methods definition
- Schema and physical organization modification
- Granting authorization for data
- Integrity constraint specification

Structure of a DBMS

Figure 1.3 shows the structure of a typical DBMS.

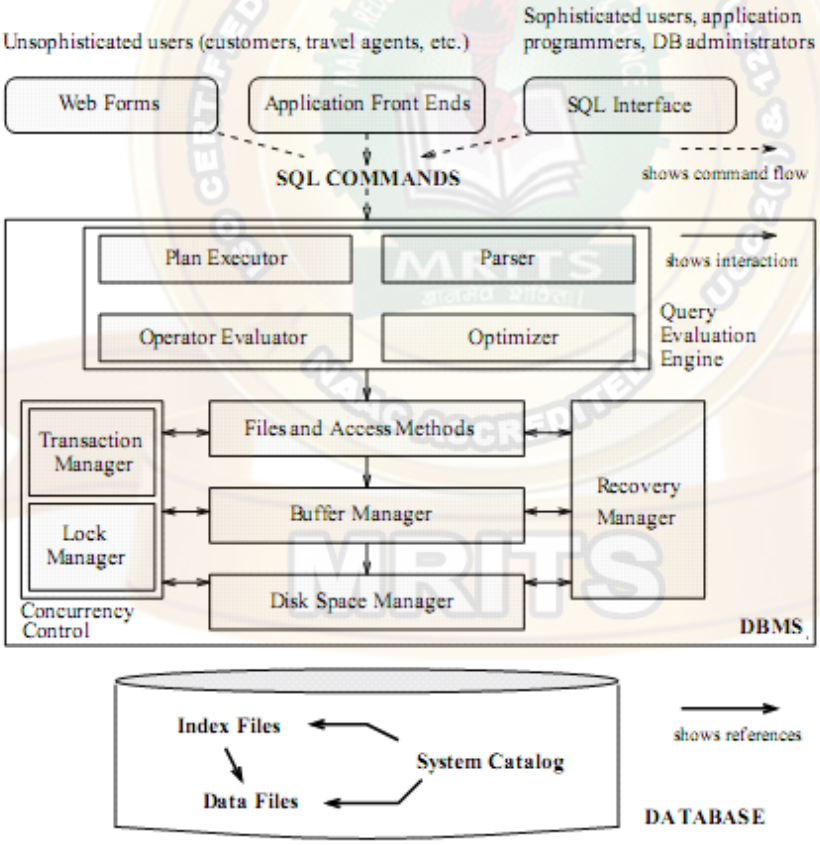


Figure 1.3 Architecture of a DBMS

This unit-2 presents details about the entity relationship model. This model provides a high level view of the issues. To design the data base we need to follow proper way that way is called data model. So we see how to use the E-R model to design the data base.

Contents :

- Over view of data base design
- ER model
- Features of ER model
- Conceptual design using ER model

Database Design:

The database design process can be divided into six steps.

The ER model is most relevant to the first three steps.

- Requirements Analysis :

  The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database.

- Conceptual Database Design :

The information gathered in the requirements analysis step is used to develop to high level description of the data to the stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high level data model.

- Logical Database Design:

  We must choose a DBMS to implement our database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS.

- Schema Refinement :

  The fourth step in database design is to analyse the collection of relations in our relational database schema to identify potential problems, and to refine it. In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory.

- Physical Database Design :

  In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This tep may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.

- Security Design:

  In this step, we identify different user groups and different roles played by various users (Eg : the development team for a product, the customer support representatives, the product manager).

For each role and user group, we must identify the parts of the database that they must be able to access and the parts of the database that they should not be allowed to access, and take steps to ensure that they can access.

Over view of ER (Entity – Relationship) model:

The entity relationship (E-R) data model is based on a perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects.

It was developed to facilitate database design by allowing the specificatin of an enterprise schema, which represents the overall logical structure of a database.

The database structure, employing the E-R model is usually shown pictorially using entity – relationship (E-R) diagrams. The entities and the relationships between them are shown in fig. Using the following conventions :

- An entity set is shown as a rectangle.
- A diamond represents the relationship among a number of entities, which are connected to the diamond by lines.
- The attributes, shown as ovals, are connected to the entities or relationship by lines.
- Diamonds, ovals, and rectangles are labeled. The type of relationship existing between the entities is represented by giving the cardinality of the relationship on the line joining the relationship to the entity.

Mapping Cardinalities :

Mapping Cardinalities, or cardinality rations, express the number of entities to which another enlity can be associated via a relationship set.

Mapping cardinalities are most useful in describing binary relationship sets, although occasionally they contribute to the description of relationship sets that involve more than two entity sets.

For, a binary relationship set R between entiry sets A and B, the mapping cardinality must be one of the following

One to One :

An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One to many .

An entity in A is associated with any number of entities in B. An entity in B. However, can be associated with at most one entity in A.

Many to one :

An entity in A is associated with at most one entity in B.

An entity in B however, can be associated with any number of entities in A.

Many to Many:

An entity in A is associated with any number of entities in B

And an entity in B is associated with any number of entities in A

The appropriate mapping cardinality for a particular relationship set is obviously dependent on the real world situation that is being modeled by the relationship set. The overall logical structure of a database can be expressed graphically by an E-R diagram, which is built up from the following components.

- Rectangles, which represent entity sets
- Ellipse, which represent attributes
- Diamonds, which represent relationship sets
- Lines, which link attributes to entity sets and entity sets to relationship sets
- Double ellipses, which represent multivalued attributes
- Double lines, which indicate total participation of an entity in a relationship set

Entities:

An entity as a thing which can be distinctly identified.

If then goes on to classify entities into regular entities and weak entities.

A weak entity is an entity that is existence dependent on some other entity, in the sence that it cannot exist if that other entity does not also exist.

For example in Fig. An employee's dependents might be weak entities – they cannot exist (so far as the database is concerned) if the relevant employee does not exist. In particular, if a given employee is delete, all dependents of that employee must be deleted too.

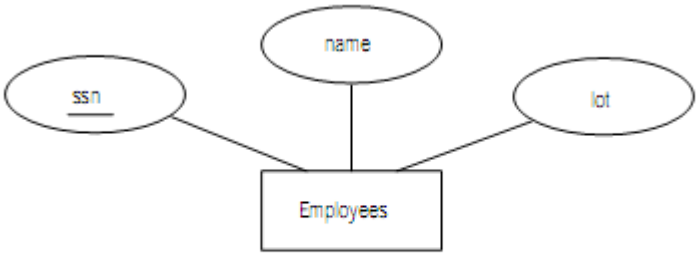


Figure 2.1 The Employees Entity Set

A regular entity, by contrast, is an entity that is not weak.

Eg : Employees might be regular entities

Note : some use the term “strong entity” instead of “regular entity”

Properties (Attributes)

Entities and also relationships, have properties.

All entities or relationships of a given type have certain kinds of properties in common, for example, all employees have an employee number, a name, a salary, and so on.

Each kind of property draw its values from a corresponding value set (i.e domain, in relational terms)

Furthermore, properties can be :

- simple or composite
  For Eg: the composite property employee name might be made up of the simple properties first name, middle initial and last name

- Key (i.e. unique, possibly within some context)
  For eg: a dependent's name might be unique within the context of a given employee

- Single or multi valued (in other words, repeating groups are permitted) all properties shown in fig. Are single valued, but if
  Eg : a given supplier could have several distinct supplier locations, then supplier city might be a multi valued property.

- Missing
  Ex: Unknown or not applicable

- Base or derived :
  For Ex: total quantity for a given part might be derived as the sum of the individual shipment quantities for that part.

Note : some use the term " attibute" instead of "property" in an E/R context.

Relationships:

Relationship as "an association among entities"

For Ex: there is a relationship called DEPT –EMP between departments and employees, representing the fact that certain departments employ certain employees.

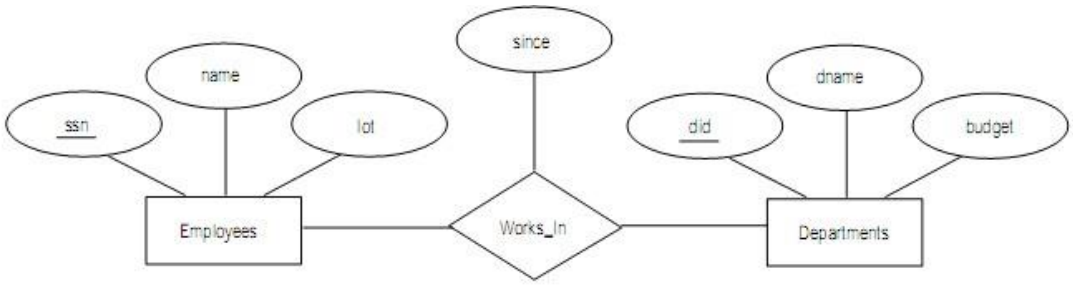


Figure 2.2 The Works_In Relationship Set

As with entities, it is necessary in principle to distinguish between relationship types and relationship instances.

The entries involved in a given relationship are said to be participants in that relationship. The number of participants in a given relationship is called the degree of that relationship.

Let R be a relationship type that involves entity type E as a participant. If every instance of E participates in atleast one instance of R, then the participation of E in R is said to be total, otherwise it is said to be partial.

For Ex: if every employee must belong to a department, then the participation of employees in DEPT-EMP is total; if it is possible for a given department to have no employees at all, then the participation of departments in DEPT – EMP is partial.

The database structure, employing the E-R model is usually shown pictorially using entity – relationship.

(E-R) diagrams: The entities and the relationships between them are shown in fig. Using the following conventions:

- an entity set is shown as a rectangle
- A diamond represents the relationship among a number of entities, which are connected to the diamond by lines
- The attributes, shown as ovals, are connected to the entities or relationship by lines.
- Diamonds, ovals, and rectangles are labeled. The type of relationship existing between the entities is represented by giving the cardinality of the relationship on the line joining the relationship to the entity.

In Unit – 2 we discuss about Data storage and retrieval. It deals with disk, file, and file system structure, and with the mapping of relational and object data to a file system. A variety of data access techniques are presented in this unit , including hashing, B+ - tree indices, and grid file indices. External sorting which will be done in secondary memory is discussed here.

Conceptual Database Design With The ER Model

Developing an ER diagram presents several choices, including the following:

- Should a concept be modeled as an entity or an attribute?
- Should a concept be modeled as an entity or a relationship?
- What are the relationship sets and their participating entity sets?
- Should we use binary or ternary relationships?
- Should we use aggregation?

UNIT – II

Overview :

The Relational Model defines two root languages for accessing a relational database -- Relational Algebra and Relational Calculus. Relational Algebra is a low-level, operator-oriented language. Creating a query in Relational Algebra involves combining relational operators using algebraic notation. Relational Calculus is a high-level, declarative language. Creating a query in Relational Calculus involves describing what results are desired.

The basic form of SQL,SQL (Structured Query Language) is a database sublanguage for querying and modifying relational databases. The basic structure in SQL is the statement how to write the queries and modify tables and columns.

Contents:

- Introduction to the Relational Model
- Integrity Constraint Over relations
- Enforcing Integrity constraints
- Querying relational data
- Introduction to Views, Destroying /altering Tables and Views.
- Relational Algebra
- Selection and projection set operations
- Renaming
- Joins
- Division
- Relational calculus
- Tuple relational Calculus
- Domain relational calculus
- Expressive Power of Algebra and calculus

- Form of Basic SQL Query,Examples
- Introduction to Nested Queries, Correlated Nested Queries Set
- Comparison Operators, Aggregative Operators
- NULL values

- Logical connectivity's
- Outer Joins

9. Triggers and Active Data bases

Introduction to the Relational Model

The main construct for representing data in the relational model is a relation. A relation consists of a relation schema and a relation instance. The relation instance is a table, and the relation schema describes the column heads for the table. We first describe the relation schema and then the relation instance. The schema specifies the relation's name, the name of each field (or column, or attribute), and the domain of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

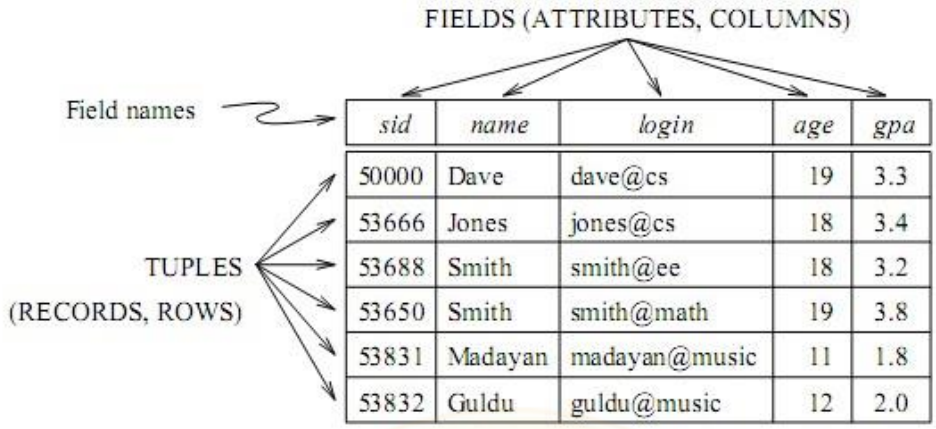
Eg:

Students(sid: string, name: string, login: string, age: integer, gpa: real)

This says, for instance, that the field named sid has a domain named string. The set of values associated with domain string is the set of all character strings.

An instance of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema. A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields.

An instance of the Students relation appears in Figure 3.1.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.1 An Instance S1 of the Students Relation

A relation schema specifies the domain of each field or column in the relation instance. These domain constraints in the schema specify an important condition that we want each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column. Thus, the domain of a field is essentially the type of that field, in programming language terms, and restricts the values that can appear in the field.

Domain constraints are so fundamental in the relational model that we will henceforth consider only relation instances that satisfy them; therefore, relation instance means relation instance that satisfies the domain constraints in the relation schema.

The degree, also called arity, of a relation is the number of fields. The cardinality of a relation instance is the number of tuples in it. In Figure 3.1, the degree of the relation (the number of columns) is five, and the cardinality of this instance is six.

A relational database is a collection of relations with distinct relation names. The relational database schema is the collection of schemas for the relations in the database.

Creating and Modifying Relations

The SQL-92 language standard uses the word table to denote relation, and we will often follow this convention when discussing SQL. The subset of SQL that supports the creation, deletion, and modification of tables is called the **Data Definition Language (DDL)**.

To create the Students relation, we can use the following statement:

The CREATE TABLE statement is used to define a new table.

CREATE TABLE Students (sid CHAR(20), name CHAR(30), login CHAR(20), age INTEGER, gpa REAL)

Tuples are inserted using the INSERT command. We can insert a single tuple into the Students table as follows:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, 'Smith', 'smith@ee', 18,3.2)

We can delete tuples using the DELETE command. We can delete all Students tuples with name equal to Smith using the command:

DELETE FROM Students S WHERE S.name = 'Smith'

We can modify the column values in an existing row using the UPDATE command. For example, we can increment the age and decrement the gpa of the student with sid 53688:

UPDATE Students S SET S.age = S.age + 1, S.gpa = S.gpa - 1 WHERE S.sid = 53688

Integrity Constraints Over Relations

An integrity constraint (IC) is a condition that is specified on a database schema, and restricts the data that can bstored in an instance of the database. If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

Key Constraints

Consider the Students relation and the constraint that no two students have the same student id. This IC is an example of a key constraint. A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple. A set of fields that uniquely identifies a tuple according to a key constraint is called a **candidate key** for the relation; we often abbreviate this to just key.In the case of the Students relation, the (set of fields containing just the) sid field is a candidate key.

There are two parts to the definition of (candidate) key:

1. Two distinct tuples in a legal instance (an instance that satisfies all ICs, including the key constraint) cannot have identical values in all the fields of a key.

2. No subset of the set of fields in a key is a unique identifier for a tuple.

The first part of the definition means that in any legal instance, the values in the key fields uniquely identify a tuple in the instance

The second part of the definition means, for example, that the set of fields {sid, name} is not a key for Students, because this set properly contains the key {sid}.Theset {sid, name} is an example of a **superkey**, which is a set of fields that contains a key.

Out of all the available candidate keys, a database designer can identify a **primary** key. Intuitively, a tuple can be referred to from elsewhere in the database by storing the values of its primary key fields. For example, we can refer to a Students tuple by storing its sid value.

Specifying Key Constraints in SQL

CREATE TABLE Students (sid CHAR(20), name CHAR(30), login CHAR(20), ageINTEGER, gpa REAL, UNIQUE (name, age), CONSTRAINT StudentsKey PRIMARY KEY

(sid))

This definition says that sid is the primary key and that the combination of name and age is also a key. The definition of the primary key also illustrates how we can name a constraint by preceding it with CONSTRAINT constraint-name. If the constraint is violated, the constraint name is returned and can be used to identify the error.

Foreign Key Constraints

Sometimes the information stored in a relation is linked to the information stored in another relation. If one of the relations is modified, the other must be checked, and perhaps modified, to keep the data consistent. An IC involving both relations must be specified if a DBMS is to make such checks. The most common IC involving two relations is a foreign key constraint.

Suppose that in addition to Students, we have a second relation:

Enrolled(sid: string, cid: string, grade: string)

To ensure that only bonafide students can enroll in courses, any value that appears in the sid field of an instance of the Enrolled relation should also appear in the sid field of some tuple in the

Students relation. The sid field of Enrolled is called a **foreign key** and **refers** to Students. The foreign key in the referencing relation (Enrolled, in our example) must match the primary key of the referenced relation (Students), i.e., it must have the same number of columns and compatible data types, although the column names can be different.

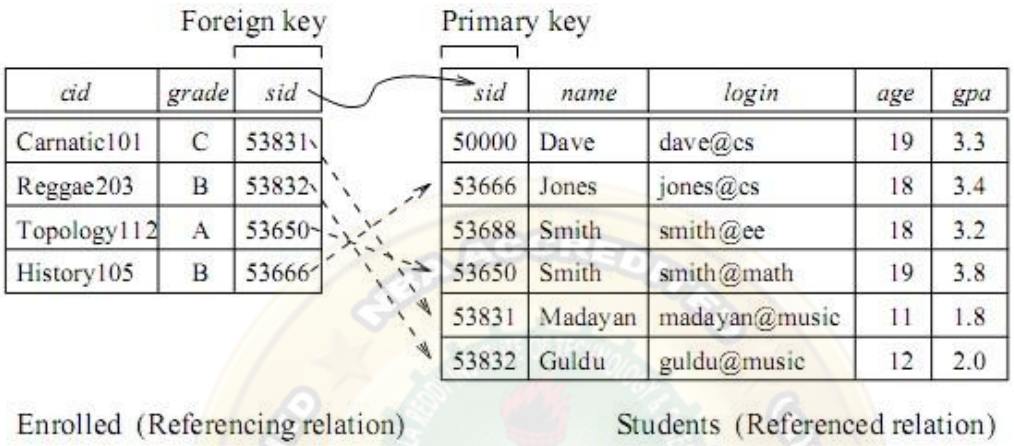


Figure 3.4 Referential Integrity

Specifying Foreign Key Constraints in SQL

CREATE TABLE Enrolled (sid CHAR(20), cid CHAR(20), grade CHAR(10), PRIMARY KEY

(sid, cid), FOREIGN KEY (sid) REFERENCES Students)

Enforcing Integrity Constraints

Consider the instance S1 of Students shown in Figure 3.1. The following insertion violates the primary key constraint because there is already a tuple with the sid 53688, and it will be rejected by the DBMS:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, ‘Mike’, ‘mike@ee’, 17, 3.4)

The following insertion violates the constraint that the primary key cannot contain null:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (null, ‘Mike’, ‘mike@ee’, 17,3.4)

Querying Relational Data

A **relational database query** is a question about the data, and the answer consists of a new relation containing the result. For example, we might want to find all students younger than 18 orall students enrolled in Reggae203.

A **query language** is a specialized language for writing queries.

SQL is the most popular commercial query language for a relational DBMS. Consider the instance of the Students relation shown in Figure 3.1. We can retrieve rows corresponding to students who are younger than 18 with the following SQL query:

SELECT * FROM Students S WHERE S.age < 18

The symbol * means that we retain all fields of selected tuples in the result. The condition S.age

< 18 in the WHERE clause specifies that we want to select only tuples in which the age field has a value less than 18. This query evaluates to the relation shown in Figure 3.6.

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.6 Students with *age* < 18 on Instance $S1$

Introduction To Views

A **view** is a table whose rows are not explicitly stored in the database but are computed as needed from a **view definition**. Consider the Students and Enrolled relations. Suppose that we are often interested in finding the names and student identifiers of students who got a grade of B in some course, together with the cid for the course. We can define a view for this purpose. Using SQL notation:

CREATE VIEW B-Students (name, sid, course) AS SELECT S.sname, S.sid, E.cid FROM Students S, Enrolled E WHERE S.sid = E.sid AND E.grade = 'B'

This view can be used just like a **base table**, or explicitly stored table, in defining new queries or views. Given the instances of Enrolled and Students shown in Figure 3.4, BStudents contains the tuples shown in Figure 3.18.

name	*sid*	*course*
Jones	53666	History105
Guldu	53832	Reggae203

Figure 3.18 An Instance of the B-Students View

Destroying/Altering Tables and Views

To destroy views, use the DROP TABLE command. For example, DROP TABLE Students RESTRICT destroys the Students table unless some view or integrity constraint refers to Students; if so, the command fails. If the keyword RESTRICT is replaced by CASCADE, Students is dropped and any referencing views or integrity constraints are (recursively) dropped as well; one of these two keywords must always be specified. A view can be dropped using the DROP VIEW command, which is just like DROP TABLE.

ALTER TABLE modifies the structure of an existing table. To add a column called maiden-name to Students, for example, we would use the following command:

ALTER TABLE Students ADD COLUMN maiden-name CHAR(10)

The definition of Students is modified to add this column, and all existing rows are padded with null values in this column. ALTER TABLE can also be used to delete columns and to add or drop integrity constraints on a table.

Relational Algebra

Relational algebra is one of the two formal query languages associated with the relational model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result. This property makes it easy to compose operators to form a complex query —a **relational algebra expression** is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions. We describe the basic operators of the algebra (selection, projection, union, cross-product, and difference).

Selection and Projection

Relational algebra includes operators to select rows from a relation (σ)and to project columns (π).

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

Figure 4.1 Instance $S1$ of Sailors

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

Figure 4.2 Instance $S2$ of Sailors

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

Figure 4.3 Instance $R1$ of Reserves

These operations allows to manipulate data in a single relation. Consider the instance of the Sailors relation shown in Figure 4.2, denoted as S2. We can retrieve rows corresponding to expert sailors by using the s operator. The expression (S2) evaluates to the relation shown in Figure 4.4. The subscript rating>8 specifies the selection criterion to be applied while retrieving tuples.

$$\sigma_{rating>8}(S2)$$

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
58	Rusty	10	35.0

Figure 4.4 $\sigma_{rating>8}(S2)$

sname	*rating*
yuppy	9
Lubber	8
guppy	5
Rusty	10

Figure 4.5 $\pi_{sname,rating}(S2)$

$$\pi_{sname,rating}(S2)$$

Set Operations

The following standard operations on sets are also available in relational algebra: union (U), intersection (n), set-difference (-), and cross-product (×).

- **Union:** RUS returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both). R and S must be unioncompatible, and the schema of the result is defined to be identical to the schema of R.
- **Intersection:** RnS returns a relation instance containing all tuples that occur in both R and S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.
- **Set-difference:** R-S returns a relation instance containing all tuples that occur in R but not in S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.
- **Cross-product:** R×S returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S). The result of R × S contains one tuple r, s (the concatenation of tuples r and s) for each pair of tuples r ∈ R, s ∈ S. The cross-product opertion is sometimes called **Cartesian product**.

Joins

The join operation is one of the most useful operations in relational algebra and is the most commonly used way to combine information from two or more relations. Although a join can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products.

Condition Joins

The most general version of the join operation accepts a join condition c and a pair of relation instances as arguments, and returns a relation instance. The join condition is identical to a selection condition in form. The operation is defined as follows:

$$R \bowtie_c S = \sigma_c(R \times S)$$

As an example, the result of $S1 \bowtie_{S1.sid<R1.sid} R1$.

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Figure 4.12 $S1 \bowtie_{S1.sid<R1.sid} R1$

Relational Calculus

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or declarative, in that it allows to describe the set of answers without being explicit about how they should be computed.

The variant of the calculus that we present in detail is called the tuple relational calculus (TRC). Variables in TRC take on tuples as values. In another variant, called the domain relational calculus (DRC), the variables range over field values.

Tuple Relational Calculus

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

As a simple example, consider the following query.

Find all sailors with a rating above 7

{S | S ∈ Sailors A S.rating > 7}

Syntax of TRC Queries

Let Rel be a relation name, R and S be tuple variables, a an attribute of R,and b an attribute of S. Let op denote an operator in the set {<, >, =, =, =, =}. An atomic formula is one of the following:

- R ∈ Rel
- R.a op S.b
- R.a op constant, or constant op R.a

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

- any atomic formula
- ¬p, p A q, p ∨ q,orp ⇒ q
- ∃R(p(R)), where R is a tuple variable
- ∀R(p(R)), where R is a tuple variable

Domain Relational Calculus

A **domain variable** is a variable that ranges over the values in the domain of some attribute (e.g., the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers). A DRC query has the form {x | p(x1,x2,...,xn)},where each x is either a domain variable or a constant and p(x1,x2,...,xn) denotes a **DRC formula** whose only free variables are the variables among the x_i, $1 \leq i \geq n$. The result of this query is the set of all tuples x1,x2,...,xi for which the formula evaluates to true.

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

An **atomic formula** in DRC is one of the following:

- <x1,x2,...,xn> ∈Rel,where Rel is a relation with n attributes; each x, 1 ≤i≥ n is either a variable or a constant.
- X op Y

- X op constant,or constant op X

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

- any atomic formula
- ¬p, p ∧ q, p ∨ q,orp ⇒ q
- ∃X(p(X)), where X is a domain variable
- ∀X(p(X)), where X is a domain variable

Eg:Find all sailors with a rating above 7.

{<I, N, T, A>|<I, N, T, A>∈Sailors ∧ T>7}

The Form of a Basic SQL Query:

SELECT [DISTINCT] select-list

FROM from-list

WHERE qualification

Every query must have a SELECT clause, which specifies columns to be retained in the result, and a FROM clause, which specifies a cross-product of tables. The optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

Eg: 1. Find the names and ages of all sailors.

SELECT DISTINCT S.sname, S.age FROM Sailors S

2.Find all sailors with a rating above 7.

SELECT S.sid, S.sname, S.rating, S.age FROM Sailors AS S WHERE S.rating > 7

We now consider the syntax of a basic SQL query in detail.

- The **from-list** in the FROM clause is a list of table names. A table name can be followed by a **range variable**; a range variable is particularly useful when the same table name appears more than once in the from-list.
- The **select-list** is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.
- The **qualification** in the WHERE clause is a boolean combination (i.e., an expression using the logical connectives AND, OR,andNOT) of conditions of the form expression op expression,whereop is one of the comparison operators {<, <=, = , <>, >=,>}.An expression is a column name, a constant, or an (arithmetic or string) expression.
- The DISTINCT keyword is optional. It indicates that the table computed as an answer to this query should not contain duplicates, that is, two copies of the same row. The default is that duplicates are not eliminated.

The following is the conceptual evaluation strategy of SQL query

1. Compute the cross-product of the tables in the **from-list**.

2. Delete those rows in the cross-product that fail the **qualification** conditions.

3. Delete all columns that do not appear in the **select-list**.

4. If DISTINCT is specified, eliminate duplicate rows.

ADDITIONAL TOPIC:

UNION, INTERSECT, AND EXCEPT

SQL provides three set-manipulation constructs that extend the basic query form. Since the answer to a query is a multiset of rows, it is natural to consider the use of operations such as union, intersection, and difference. SQL supports these operations under the names UNION, INTERSECT,andEXCEPT.

Union:

Eg: Find the names of sailors who have reserved a red or a green boat.

SELECT S.sname FROM Sailors S, Reserves R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’

UNION

SELECT S2.sname FROM Sailors S2, Boats B2, Reserves R2 WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’

This query says that we want the union of the set of sailors who have reserved red boats and the set of sailors who have reserved green boats.

Intersect:

Eg:Find the names of sailors who have reserved both a red and a green boat.

SELECT S.sname FROM Sailors S, Reserves R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’

INTERSECT

SELECT S2.sname FROM Sailors S2, Boats B2, Reserves R2 WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’

Except:

Eg:Find the sids of all sailors who have reserved red boats but not green boats.

SELECT S.sid FROM Sailors S, Reserves R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’

EXCEPT

SELECT S2.sid FROM Sailors S2, Reserves R2, Boats B2 WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’

SQL also provides other set operations: IN (to check if an element is in a given set), op ANY, op ALL (to compare a value with the elements in a given set, using comparison operator op), and EXISTS (to check if a set is empty). IN and EXISTS can be prefixed by NOT,withthe obvious modification to their meaning. We cover UNION, INTERSECT,andEXCEPT in this section, and the other operations in Section 5.4.

NESTED QUERIES

A **nested query** is a query that has another query embedded within it; the embedded query is called a **subquery**.

SQL provides other set operations: IN (to check if an element is in a given set),NOT IN(to check if an element is not in a given set).

Eg:1. Find the names of sailors who have reserved boat 103.

```
SELECT S.sname FROM Sailors S
 WHERE S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R.bid = 103 )
```

The nested subquery computes the (multi)set of sids for sailors who have reserved boat 103, and the top-level query retrieves the names of sailors whose sid is in this set. The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested.

2.Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname FROM Sailors S
WHERE S.sid NOT IN ( SELECT R.sid
                     FROM Reserves R
                     WHERE R.bid IN ( SELECT B.bid
                                      FROM Boats B
                                      WHERE B.color = 'red' )
```

Correlated Nested Queries

In the nested queries that we have seen, the inner subquery has been completely independent of the outer query. In general the inner subquery could depend on the row that is currently being examined in the outer query .

Eg: Find the names of sailors who have reserved boat number 103.

```
SELECT S.sname FROM Sailors S
WHERE EXISTS ( SELECT *
               FROM Reserves R
               WHERE R.bid = 103 AND R.sid = S.sid )
```

The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty.

Set-Comparison Operators

SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators {<, <=, =, <>, >=,>}.

Eg:1. Find sailors whose rating is better than some sailor called Horatio.

```
SELECT S.sid FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                       FROM Sailors S2
                       WHERE S2.sname = 'Horatio' )
```

If there are several sailors called Horatio, this query finds all sailors whose rating is better than that of some sailor called Horatio.

2.Find the sailors with the highest rating.

```
SELECT S.sid FROM Sailors S
WHERE S.rating >= ALL ( SELECT S2.rating
                        FROM Sailors S2 )
```

Aggregate Operators

SQL supports five aggregate operations, which can be applied on any column

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

Eg:1. Find the average age of all sailors.

```
SELECT AVG (S.age) FROM Sailors S
```

2.Count the number of sailors.

```
SELECT COUNT (*) FROM Sailors S
```

The GROUP BY and HAVING Clauses

Thus far, we have applied aggregate operations to all (qualifying) rows in a relation. Often we want to apply aggregate operations to each of a number of **groups** of rows in a relation, where the number of groups depends on the relation instance.

Syntax:

SELECT [DISTINCT] select-list

FROM from-list

WHERE qualification

GROUP BY grouping-list

HAVING group-qualification

Eg:Find the age of the youngest sailor for each rating level.

SELECT S.rating, MIN (S.age) FROM Sailors S GROUP BY S.rating

NULL VALUES

SQL provides a special column value called null to use where some column does not have a value to hold or the value is unknown. We use null when the column value is either unknown or inapplicable.

Logical Connectives AND, OR, and NOT

The logical operators AND, OR,andNOT using a three-valued logic in which expressions evaluate to true, false,or unknown.OR of two arguments evaluates to true if either argument evaluates to true, and to unknown if one argument evaluates to false and the other evaluates to unknown. (If both arguments are false, of course, it evaluates to false.) AND of two arguments evaluates to false if either argument evaluates to false, and to unknown if one argument evaluates to unknown and the other evaluates to true or unknown.

Outer Joins

The join operation that rely on null values, called **outer joins**, are supported in SQL. Consider the join of two tables, say Sailors & Reserves. Tuples of Sailors that do not match some row in Reserves according to the join condition c do not appear in the result. In an outer join, on the other hand, Sailor rows without a matching Reserves row appear exactly once in the result, with the result columns inherited from Reserves assigned null values.

In fact, there are several variants of the outer join idea. In a **left outer join**, Sailor rows without a matching Reserves row appear in the result, but not vice versa. In a **right outer join**, Reserves rows without a matching Sailors row appear in the result, but not vice versa. In a **full outer join**, both Sailors and Reserves rows without a match appear in the result.

Triggers and Active Databases

A **trigger** is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an **active database**.

A trigger description contains three parts:

- **Event**: A change to the database that **activates** the trigger.
- **Condition**: A query or test that is run when the trigger is activated.
- **Action**: A procedure that is executed when the trigger is activated and its condition is true.

Eg: The trigger called init count initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called incr count increments the counter for each inserted tuple that satisfies the condition age < 18.

```
CREATE TRIGGER init count BEFORE INSERT ON Students /* Event */
DECLARE
    count INTEGER;
BEGIN                                                    /* Action */
    count := 0;
END

CREATE TRIGGER incr count AFTER INSERT ON Students       /* Event */
WHEN (new.age < 18)                                      /* Condition*/
FOR EACH ROW
BEGIN                                                    /* Action */
    count:=count+1;
END
```

UNIT-III

Overview :

Only construction of the tables is not only the efficient data base design. Solving the redundant data problem is the efficient one. For this we use functional dependences. And normal forms those will be discussed in this chapter.

Contents :

- Schema refinement
- Use of Decompositions
- Functional dependencies
- Normal forms
- Multi valued dependencies

Introduction To Schema Refinement

We now present an overview of the problems that schema refinement is intended to address and a refinement approach based on decompositions. Redundant storage of information is the root cause of these problems. Although decomposition can eliminate redundancy, it can lead to problems of its own and should be used with caution.

Problems Caused by Redundancy

Storing the same information **redundantly**, that is, in more than one place within a database, can lead to several problems:

- **Redundant storage:** Some information is stored repeatedly.
- **Update anomalies:** If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.
- **Insertion anomalies:** It may not be possible to store some information unless some other information is stored as well.
- **Deletion anomalies:** It may not be possible to delete some information without losing some other information as well.

Use of Decompositions

Redundancy arises when a relational schema forces an association between attributes that is not natural. Functional dependencies can be used to identify such situations and to suggest refinements to the schema. The essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of 'smaller' relations. Each of the smaller relations contains a subset of the attributes of the original relation. We refer to this process as decomposition of the larger relation into the smaller relations.

Problems Related to Decomposition

Decomposing a relation schema can create more problems than it solves. Two important questions must be asked repeatedly:

1. Do we need to decompose a relation?

2. What problems (if any) does a given decomposition cause?

To help with the first question, several normal forms have been proposed for relations. If a relation schema is in one of these normal forms, we know that certain kinds of problems cannot arise. Considering the normal form of a given relation schema can help us to decide whether or not to decompose it further. If we decide that a relation schema must be decomposed further, we must choose a particular decomposition.

With respect to the second question, two properties of decompositions are of particular interest. The lossless-join property enables us to recover any instance of the decomposed relation from corresponding instances of the smaller relations. The dependency preservation property enables us to enforce any constraint on the original relation by simply enforcing some constraints on each of the smaller relations. That is, we need not perform joins of the smaller relations to check whether a constraint on the original relation is violated.

Normalization:

In general, database normalization involves splitting tables with columns that have different types of data (and perhaps even unrelated data) into multiple table, each with fewer columns that describe the attributes of a single concept of physical object or being.

The goal of normalization is to prevent the problems (called modification anomalie) that plague a poorly designed relation (table).

Suppose, for example, that you have a table with resort guest ID numbers, activities the guests have signed up to do, and the cost of each activity – all together in the following GUEST – ACTIVITY-COST table:

Each row in the table represents a guest that has signed up for the named activity and paid the specified cost. Assuming that the cost depends only on the activity that is, a specific activity costs the same for all guests if you delete the row for GUEST – ID 2587, you lose not only the fact that guest 2587 signed up for scuba diving, but also the fact that scuba diving costs $ 250.00 per outing. This is called a deletion anomaly – when you delete a row, you lose more information than you intended to remove.

In the current example, a single deletion resulted in the loss of information on two entities what activity a guest signed up to do and how much a particular activity costs.

Now, suppose the resort adds a new activity such as horseback riding. You cannot enter the activity name (horseback riding) or cost ($190.00) in to the table until a guest decides to sign up for it. The unnecessary restriction of having to wait until someone signs up for an activity before you can record its name and cost is called an insertion anomaly.

In the current example, each insertion adds facts about two entities. Therefore, you cannot INSERT a fact about one entity until you have an additional fact about the other entity. Conversely, each deletion removes facts about two entities. Thus, you cannot DELETE the information about one entity while leaving the information about the other in table.

You can eliminate modification anomalies through normalization – that is, splitting the single table with rows that have attributes about two entities into two tables, each of which has rows with attributes that describe a single entity.

You will be ablve to remove the aromatherapy appointment for guest 1269 without losing the fact that an aromatherapy session costs $75.00. Similarly, you can now add the fact that horseback riding costs $ 190.00 per day to the ACTIVITY – COST table without having to wait for a guest to sign up for the activity.

During the development of relational database systems in the 1970s, relational theorists kept discovering new modification anomalies. Some one would find an anomaly, classify it, and then figure out a way to prevent it by adding additional design criteria to the definition of a "well formed relation. These design criteria are known as normal forms. Not surprisingly E.F codd (of the 12 rule database definition fame), defined the first, second, and third normal forms, (INF, 2NF, and 3NF).

After Codd postulated 3 NF, relational theorists formulated Boyce-codd normal form (BCNF) and then fourth normal form (4NF) and fifth normal form (5NF)

First Normal form :

Normalization is a processes by which database designers attempt to eliminate modification anomalies such as the :

- Deletion anomaly
  The iniability to remove a single fact from a table without removing other (unrelated) facts you want to keep.
- Insertion anomaly
  The inability to insert one fact without inserting another (and some times, unrelated) fact.
- Update anomaly.

Changing a fact in one column creates a false fact in another set of columns. Modification anomalies are a result of functional dependencies among the columns in a row (or tuple, to use the precise relational database term.

A functional dependency means that if you know the value in one column or set of columns, you can always determine the value of another. To put the table in first normal form (INF) you could break up the student number list in the STUDENTS column of each row such that each row had only one of the student Ids in the STUDENTS column. Doing so would change the table's structure and rows to : The value given by the combination (CLASS, SECTION, STUDENT) is the composite key for the table because it makes each row unique and all columns atomic. Now that each the table in the current example is in INF, each column has a single, scalar value.

Unfortunately, the table still exhibits modification anomalies:

Deletion anomaly :

If professor SMITH goes to another school and you remove his rows from the table, you also lose the fact that STUDENTS 1005, 2110 and 3115 are enrolled in a history class.

Insertion anomaly :

If the school wants to add an English Class (EI00), it cannot do so until a student signs up for the course (Remember, no part of a primary key can have a NULL value).

Update anomaly :

If STUDENT 4587 decides to sign up for the SECTION 1, CS100 CLASS instead of his math class, updating the Class and section columns in the row for STUDENT 4587 to reflect the change will cause the table to show TEACHER RAWL INS as being in both the MATH and the COMP-SCI departments.

Thus, 'flattening' a table's columns to put it into first normal form (INF) does not solve any of the modification anomalies

All it does is guarantee that the table satisfies the requirements for a table defined as "relational" and that there are no multi valued dependencies between the columns in each row.

Second Normal Form :

The process of normalization involves removing functional depencies between columns n order to eliminate the modification anomalies caused by these dependencies.

Putting a table in first normal form (INF) requires removing all multi valued dependencies.

When a table is in second normal form, it must be in first normal form (no multi valued dependencies and have no partial key dependencies.

A partial key dependency is a situation in which the value in part of a key can be used to determine the value of another attribute (column)

Thus, a table is in 2NF when the value in all nonkey columns depends on the entire key. Or, said another way, you cannot determine the value of any of the columns by using part of the key.

With (CLASS, SECTION, STUDENT) as its primary key. If the university has two rules about taking classes no student can sign up for more than one section of the same class, and a student can have only one major then the table, while in 1 NF, is not in 2NF.

Given the value of (STUDENT, COURSE) you can determine the value of the SECTION, since no student can sign up for two sections of the same course. Similarly since students can sign up for only one major, knowing STUDENT determines the value of MAJOR. In both instances, the value of a third column can be deduced (or is determined) by the value in a portion of the key (CLASS, SECTION, STUDENT) that makes each row unique.

To put the table in the current example in 2NF will require that it be split in to three tables described by :

Courses (Class, Section, Teacher, Department)

PRIMARY KEY (Class, Section)

Enrollment (Student, Class, Section)

PRIMARY KEY (Student, class)

Students (student, major)

PRIMARY KEY (Student)

Unfortunately, putting a table in 2NF does not eliminate modification anomalies.

Suppose, for example, that professor Jones leaves the university. Removing his row from the COURSES table would eliminate the entire ENGINEERING department, since he is currently the only professor in the department.

Similarly, if the university wants to add a music department, it cannot do so until it hires a professor to teach in the department.

Understanding Third Normal Form :

To be a third normal form (3NF) a table must satisfy the requirements for INF (no multi valued dependencies) and 2NF (all nonkey attributes must depend on the entire key). In addition, a table in 3NF has no transitive dependencies between nonkey columns.

Given a table with columns, (A,B,C) a transitive dependency is one in which a determines B, and B determines C, therefore A determines C, or, expressed using relational theory notation :

If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$.

When a table is in 3NF the value in every non key column of the table can be determined by using the entire key and only the entire key,. Therefore, given a table in 3NF with columns (A,B,C) if A is the PRIMARY KEY, you could not use the value of B (a non key column) to determine the value of a C (another non key column). As such, A determines B($A \rightarrow B$), and A determines C($\rightarrow C$). However, knowing the value of column B does not tell you have value in column C that is, it is not the case that $B \rightarrow C$.

Suppose, for example, that you have a COURSES tables with columns and PRIMARY KEY described by

Courses (Class, section, teacher, department , department head)

PRIMARY KEY (Class, Section)

That contains the Data :

(←----------A----------→ (B) (C) (D)

Class	Section	Teacher	Dept.	Dept. Head
H1		Smith	History	Smith
00				
H1002		Riley	History	Smith
CS100	1	Bowls	Comp.Sci	Peroit
M2003		Rawlins	Math	Hastings
M2002		Brown	Math	Hastings
M2004		Riley	Math	Hastings
E1001		Jones	Engg.	Jones

Given that a TEACHER can be assigned to only one DEPARTMENT and that a DEPARTMENT can have only one department head, the table has multiple transitive dependencies.

For example, the value of TEACHER is dependant on the PRIMARY KEY (CLASS, SECTION), since a particular SECTION of a particular CLASS can have only one teacher that is A→ B. Moreover, since a TEACHER can be in only one DEPARTMENT, the value in DEPARTMENT is dependant on the value in TEACHER that is B→C. However, since the

PRIMARY KEY (CLASS, SECTION) determines the value of TEACHER, it also determines the value of DEPARTMENT that is A→ C. Thus, the table exhibits the transitive dependency in which A→ B and B→C, therefore A→ C.

The problem with a transitive dependency is that it makes the table subject to the deletion anomaly. When smith retires and we remove his row from the table, we lose not only the fact that smith taught SECTION 1 of H100 but also the fact that SECTION 1 of H100 was a class that belonged to the HISTORY department.

To put a table with transitive dependencies between non key columns into 3 NF requires that the table be split into multiple tables. To do so for the table in the current example, we would need split it into tables, described by :

Courses (Class, Section, Teacher)

PRIMARY KEY (class, section)

Teachers (Teacher, department)

PRIMARY KEY (teacher)

Departments (Department, Department head)

PRIMARY KEY (department)

UNIT-IV

Overview:

In this unit we introduce two topics first one is concurrency control. The stored data will be accessed by the users so if any two or users try to access same data at a time it may raise the problem of data inconsistency to solve that concurrency control methods are invented. Recovery is used to maintain the data without loss when the problem of power failure, software failure and hardware failure.

Contents:

Concepts of transactions and schedules

Lock based concurrency control

Crash Recovery

Introduction to crash recovery

Log recovery

Check pointing

ARIES

Transactions

Collections of operations that form a single logical unit of work are called Transactions. A database system must ensure proper execution of transactions despite failures – either the entire transaction executes, or none of it does.

A transaction is a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high level data manipulation language or programming language (for example SQL, COBOL, C, C++ or JAVA), where it is delimited by statements (or function calls) of the form Begin transaction and end transaction. The transaction consists of all operations executed between the begin transaction and end transaction.

To ensure integrity of the data, we require that the database system maintain the following properties of the transaction.

Atomicity:

Either all operations of the transaction are reflected properly in the database, or non are .

Consistency :

Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

Isolation :

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transaction Ti and Tj, ti appears to Ti that either Tj finished execution before Ti started, or Tj started execution after Ty finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

Durability :

After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called the ACID properties, the acronym is derived from the first letter of each of the four properties.

To gain a better understanding of ACID properties and the need for them, consider a simplified banking system consisting of several accounts and a set of transactions that access and update those accounts.

Transactions access data using two operations :

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

In a real database system, the write operation does not necessarily result in the immediate update of the data on the disk; the write operation may be temporarily stored in memory and executed on the disk later.

For now, however, we shall assume that the write operation updates the database immediately.

Let Ty be a transaction that transfers $50 from account A to account B. This transaction can be defined as

Ti : read (A);

A; = A-50;

Write (A);

Read (B);

B:=B+50;

Write (B).

Let us now consider each of the ACID requirements.

Consistency :

Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

The consistency requirement here is that the sum of A and B be unchanged by the execution of the transaction. Without the consistency requirement, money could be created or destroyed by the transaction. It can be verified easily that, if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

Ensuring consistency for an individual transaction is the responsibility of the application programmer who codes the transaction. This task may be facilitated by automatic testing of integrity constraints.

Atomicity :

Suppose that, just before the execution of transaction Ty the values of accounts A and B are $1000 and $2000, respectively.

Now suppose that, during the execution of transaction Ty, a failure occurs that prevents Ti from completing its execution successfully.

Examples of such failures include power failures, hardware failures, and software errors

Further, suppose that the failure happened after the write (A) operation but before the write (B) operation. In this case, the values of amounts A and B reflected in the database are $950 and $2000. The system destroyed $50 as a result of this failure.

In particular, we note that the sum A + B is no longer preserved. Thus, because of the failure, the state of the system no longer reflects a real state of the world that the database is supposed to capture. WE term such a state in inconsistent state. We must ensure that such inconsistencies are not visible in a database system.

Note, however, that the system must at some point be in an inconsistent state. Even if transaction Ty is executed to completion, there exists a point at which the value of account A is $ 950 and the value of account B is $2000 which is clearly an inconsistent state.

This state, however is eventually replaced by the consistent state where the value of account A is $ 950, and the value of account B is $ 2050.

Thus, if the transaction never started or was guaranteed to complete, such an inconsistent state would not be visible except during the execution of the transaction.

That is the reason for the atomicity requirement:

If the atomicity property is present, all actions of the transaction are reflected in the database or none are.

The basic idea behind ensuring atomicity is this:

The database system keeps track (or disk) of the old values of any data on which a transaction performs a write, and, if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed.

Ensuring atomicity is the responsibility of the database system itself; specifically, it is handed by a component called the transaction management component.]

Durability :

Once the execution of the transaction completes successfully, and the user who initiated the transaction has been notified that the transfer of funds has taken place, it must be the case that no system failure will result in a loss of data corresponding to this transfer of funds.

The durability property guarantees that, once a transaction completes successfully, all the updates that it carried out on the data base persist, even if there is a system failure after the transaction complete execution.

We assume for now that a failure of the computer system may result in loss of data in main memory, but data written to disk are never lost. We can guarantee durability by ensuring that either

- The updates carried out by the transaction have been written to disk before the transaction completes.
- Information about the updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after the failure.

Ensuring durability is the responsibility of a component of the database system called the recovery management component. The transaction management component and the recovery management component the closely related.

Isolation ;

Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way, resulting in an inconsistent state.

For example, as we saw earlier, the database is temporarily inconsistent while the transaction to transfer funds from A to B is executing, with the deducted total written to A and the increased total yet to be written to B.

If a second concurrently running transaction reads A and B at this intermediate point and computes A + B it will observe an inconsistent value. Furthermore, if this second transaction then performs updates on A and B based on the inconsistent values that it read, the database may be left in an inconsistent state even after both transactions have completed.

A way to avoid the problem of concurrently executing transactions is to execute transactions serially that is, one after the other. However, concurrent execution of transactions provides significant performance benefits.

Other solutions have therefore been developed; they allow multiple transactions to execute concurrently.

The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

Ensuring the isolation property is the responsibility of a component of the database system called the concurrency control component.

Transaction state :

In the absence of failures, all transactions complete successfully. A transaction may not always complete its execution successfully. Such a transaction is termed aborted. If we are to ensure the atomicity property, an aborted transction must have no effect on the state of the database.

Thus, any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, we say that the transaction has been rolled back. It is part of the responsibility of the recovery scheme to manage transaction aborts.

A transaction that completes its execution successfully is said to be committed. A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure.

Once a transction has committed, we cannot undo its effects by aborting it. The only way to undo the effects of committed transaction is to execute a compensating transaction. For instance, if a transaction added $20 to an account, the compensating transaction would subtract $20 from the account. However, it is not always possible to create such a compensating transaction. Therefore, the responsibility of writing and executing a compensating transaction is left to the user, and is not handled by the database system.

By successful completion of a transaction, A transaction must be in one of the following states :

Active :

The initial state ; the transaction stays in this state while it is executing

Partially committed :

After the final statement has been executed

Failed :

After the discovery that normal execution can no longer proceed

Aborted :

After the transaction has been rolled back and the database has been restrored to its state prior to the start of the transaction

Committed:

After successful completion

We say that a transaction has committed nly if it has entered the committed state. Similarly, we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have terminated if has either committed or aborted.

A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be termporarily residing in main momory, and thus a hardware failure may preclude its successful completion.

The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be recreated when the system restarts after the failure. When the last of this information is written out, the transaction enters the committed state.

A transaction enters the filed state after the system determines that the transaction can no longer proceed with its normal execution (for example, because of hard ware or logical errors) such a transaction must be rolled back. Then, it enters the aborted state. At this point, the system has two options.

- It can restart the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction.
- It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

We must be cautious when dealing with observable external writes, such as writes to a terminal or printer. Once such a write has occurred, it cannot be erased, since it may have been seen external to the database system. Most systems allows such writes to take place only after the transaction has entered the committed state.

Lock-Based Concurrency Control

ADBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions. A DBMS typically uses a locking protocol to achieve this. A locking protocol is a set of rules to be followed by each transaction, in order to ensure that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order.

Strict Two-Phase Locking(Strict2PL):

The most widely used locking protocol, called Strict Two-Phase Locking, or Strict2PL,

has two rules. The first rule is

1.If a transaction T wants to read an object, it first requests a shared lock on the object.

Of course, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a

transaction holds an exclusive lock on an object no other transaction holds a shared or exclusive lock on the same object.

The second rule in Strict2PL is:

(2)All locks held by a transaction are released when the transaction is completed.

Multiple-Granularity Locking

Another specialized locking strategy is called multiple-granularity locking, and it allows us to efficiently set locks on objects that contain other objects. For instance, a database contains several files , a file is a collection of pages , and a page is a collection of records . A transaction that expects to access most of the pages in a file should probably set a lock on the entire file, rather than locking individual pages as and when it needs them. Doing so reduces the locking overhead considerably. On the other hand, other transactions that require access to parts of the file — even parts that are not needed by this transaction are blocked. If a transaction accesses relatively few pages of the file, it is better to lock only those pages. Similarly, if a transaction accesses ever alrecords on a page, it should lock the entire page, and if it accesses just a few records, it should lock just those records.

The question to be addressed is how a lock manager can efficiently ensure that a page,

for example, is not locked by a transaction while an other transaction holds a conflicting

lock on the file containing the page.

The **recovery manager** of a DBMS is responsible for ensuring two important properties of transactions: atomicity and durability. It ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive **system crashes**, (e.g., a core dump caused by a bus error) and **media failures** (e.g., a disk is corrupted).

The Log

The log, sometimes called the **trail** or **journal**, is a history of actions executed by the DBMS. Physically, the log is a file of records stored in stable storage, which is assumed to survive crashes; this durability can be achieved by maintaining two or more copies of the log on different disks, so that the chance of all copies of the log being simultaneously lost is negligibly small.

The most recent portion of the log, called the **log tail**,is kept in main memory and is periodically forced to stable storage. This way, log records and data records are written to disk at the same granularity.

Every **log record** is given a unique id called the **log sequence number (LSN)**. As with any record id, we can fetch a log record with one disk access given the LSN. Further, LSNs should be assigned in monotonically increasing order; this property is required for the ARIES recovery algorithm. If the log is a sequential file, in principle growing indefinitely, the LSN can simply be the address of the first byte of the log record.

A log record is written for each of the following actions:

Updating a page: After modifying the page, an update type record is appended to the log tail. The page LSN of the page is then set to the LSN of the update log record.

Commit: When a transaction decides to commit, it **force-writes** a commit type log record containing the transaction id. That is, the log record is appended to the log, and the log tail is written to stable storage, up to and including the commit record.

The transaction is considered to have committed at the instant that its commit log record is written to stable storage

Abort: When a transaction is aborted, an abort type log record containing the transaction id is appended to the log, and Undo is initiated for this transaction

End: As noted above, when a transaction is aborted or committed, some additional actions must be taken beyond writing the abort or commit log record. After all these additional steps are completed, an end type log record containing the transaction id is appended to the log.

Undoing an update: When a transaction is rolled back (because the transaction is aborted, or during recovery from a crash), its updates are undone. When the action described by an update log record is undone, a compensation log record,or CLR, is written.

Other Recovery-Related Data Structures

In addition to the log, the following two tables contain important recovery-related information:

Transaction table: This table contains one entry for each active transaction. The entry contains the transaction id, the status, and a field called **lastLSN**, which is the LSN of the most recent log record for this transaction. The **status** of a transaction can be that it is in progress, is committed, or is aborted.

Dirty page table: This table contains one entry for each dirty page in the buffer pool, that is, each page with changes that are not yet reflected on disk. The entry contains a field **recLSN**, which is the LSN of the first log record that caused the page to become dirty. Note that this LSN identifies the earliest log record that might have to be redone for this page during restart from a crash.

Checkpoint

A **checkpoint** is like a snapshot of the DBMS state, and by taking checkpoints periodically, as we will see, the DBMS can reduce the amount of work to be done during restart in the event of a subsequent crash.

Introduction To ARIES

ARIES is a recovery algorithm that is designed to work with a steal, no-force approach. When the recovery manager is invoked after a crash, restart proceeds in three phases:

1. **Analysis:** Identifies dirty pages in the buffer pool and active transactions at the time of the crash.

2. **Redo:** Repeats all actions, starting from an appropriate point in the log, and restores the database state to what it was at the time of the crash.

3. Undo: Undoes the actions of transactions that did not commit, so that the database reflects only the actions of committed transactions.

There are three main principles behind the ARIES recovery algorithm:

Write-ahead logging: Any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.

Repeating history during Redo: Upon restart following a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of the crash. Then, it undoes the actions of transactions that were still active at the time of the crash.

Logging changes during Undo: Changes made to the database while undoing a transaction are logged in order to ensure that such an action is not repeated in the event of repeated restarts.

UNIT-V

Overview:

In this Unit we discuss about Data storage and retrieval. It deals with disk, file, and file system structure, and with the mapping of relational and object data to a file system. A variety of data access techniques are presented in this unit , including hashing, B+ - tree indices, and grid file indices. External sorting which will be done in secondary memory is discussed here.

Contents:

File Organisation :

- Storage Media

- Buffer Management
- Record and Page formats
- File organizations
- Various kinds of indexes and external storing
- ISAM
- B++ trees
- Extendible vs. Linear Hashing.

This chapter internals of an RDBMS

The lowest layer of the software deals with management of space on disk, where the data is to be stored. Higher layers allocate, deal locate, read and write pages through (routines provided by) this layer, called the disk space manager.

On top of the disk space manager, we have the buffer manager, which partitions the available main memory into a collection of pages of frames. The purpose of the buffer manager is to bring pages in from disk to main memory as needed in response to read requests from transactions.

The next layer includes a variety of software for supporting the concepts of a file, which, in DBMS, is a collection of pages or a collection of records. This layer typically supports a heap file, or file or unordered pages, as well as indexes. In addition to keeping track of the pages in a file, this layer organizes the information within a page.

The code that implements relational operators sits on top of the file and access methods layer. These operators serve as the building blocks for evaluating queries posed against the data.

When a user issues a query, the query is presented to a query optimizer, whish uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is usually represented as tree of relational operators (with annotations that contain additional detailed information about which access methods to use.

Data in a DBMS is stored on storage devices such as disks and tapes ; the disk space manager is responsible for keeping tract of available disk space. The file manager, which provides the abstraction of a file of records to higher levels of DBMS code, requests to the disk space manager to obtain and relinquish space on disk.

When a record is needed for processing, it must be fetched from disk to main memory. The page on which the record resides is determined by the file manager (the file manager determines the page on which the record resides)

Sometimes, the file manager uses auxiliary data structures to quickly identify the page that contains a desired record. After identifying the required page, the file manager issues a request for the page to a layer of DBMS code called the buffer manager. The buffer manager fetches requested pages from disk into a region of main memory called the buffer pool, and informs the file manager.

The Memory Hierarchy:

Memory in a computer system is arranged in a hierarchy. At the top, we have primary storage, which consists of cache and main memory, and provides very fast access to data.

Then comes secondary storage, which consists of slower devices such as magnetic disks.

Tertiary storage is the slowest class of storage devices

For Ex: Tapes

DISKS :

Buffer Manager :

In terms of the DBMS architecture the buffer manager is the software layer that is responsible for bringing pages from disk to main memory as needed. The buffer manager manages the available main memory by partitioning it into a collection of pages, which we collectively refer to as, the buffer pool. The main memory pages in the buffer pool are called frames. It is convenient to think of them as slots that can hold a page (that usually resides disk or other secondary storage media). Checks the buffer pool to see if it contains the requested page.

If the page is not in the pool, the buffer manager brings it in as follows:

- Chooses a frame for replacement, using the replacement policy
- If the dirty bit for the replacement frame is on, writes the page it contains to disk (that is, the disk copy of the page is overwritten with the concents of the frame).
- Reads the requested page into the replacement frame.

Buffer replacement policies :

- Least recently used
- Clock replacement

(i) Least Recently used :

The best known replacement policy is least recently used (LRU). This can be implemented in the buffer manager using a queue of pointers to frames with pin-count 0. A frame is added to the end of the queue when it becomes a candidate for replacem ent (that is, when the pin-count goes to 0). The page chosen for replacement is the one in the frame at the head of the queue.

(ii) Clock Policy :

A variant of LRU, called clock replacement, has similar behaviour but less over head.

Other replacement policies include first in first out (FIF)) and most recently used (MRU) which also entail overhead similar to LRU, and random, among others.

Record Formats :

Fixed – Length Records :

In a fixed length record, each field has a fixed length (that is, the value in this field is of the same length in all records) and the number of fields is also fixed. The fields of a such record can be stored consecutively, and given the address of the record, the address of a particular field can be calculated using information about the lengths of preceding fields, which is available in the system catalog.

Variable – Length Records :

In the relational model, every record in a relation contains the same number of fields. If the number of fields is fixed, a record is of variable length only because some of its fields are of variable length.

Page Formats :

How a collection of records can be arranged on a page?

A page as a collection of slots, each of which contains a record. A record is identified by using the pair.

[page id, slot number]

This identifier is called a record id(or rid), and serves as a ‘pointer’ to a record.

Fixed length records :

If all records on the page are guaranteed to be of the same length, record slots are uniform and can be arranged consecutively within a page. At any instant, some slots are occupied by records, and

others are unoccupied. When a record is inserted into the page, we must locate an empty slot and place the record there. The main issues are how we keep track of empty slots, and how we locate all records on a page. The alternatives hinge on how we handle the deletion of a record.

Les and Indexes :

We know how pages can be stored on disk and brought into main memory as needed, and how the space on a page can be organized to store a collection of records.

Page : A collection of records

A file of records is a collection of records that may reside on several pages.

How a collection of pages can be organized as a file.

The basic file structure that we consider stores records in random order, and supports retrieval of all records or retrieval of a particular record specified by its rid. Some times, we want to retrieve records by specifying some condition on the fields of desired records, for example, "Find all employee records with age 35'. To speed up such selections, we can build auxiliary data structure that allow us to …..

Heap files :

The simplest file structure is an unordered file or heap file. The data in the page of a heap file is not ordered in any way, and the only guarantee is that one can retrieve all records in the file by repeated requests for the next record. Every record in the file has a unique rid, and every page in a file is of the same size.

Supported operations on a heap file include create and destroy files, insert a record, delete a record with a given rid, get a record with a given rid, and scan all records in the file. To get or delete a record with a given rid, note that we must be able to find the id of the page containing the record, given the id of the record.

We must keep track of the pages in each help file in order to support scans, and we must keep track of pages that contain free space in order to implement insertion efficiently.

The two internal organizations of heap files are :

- Linked list of pages
- Directory of pages

Linked list of pages :

One possibility is to maintain a heap file as a doubly linked list of pages. The DBMS can remember where the first page is located by maintaining a table of (heap-file-name, page – 1 – addr) pairs in a known location on disk. We call the first page of the file the header page.

An improvement task is to maintain information about empty slots created by deleting a record from the heap file.

This task has two distinct parts

How to keep track of free space within a page and how to keep track of pages that have some free space.

Directory of pages :

An alternative to a linked list of pages is to maintain a directory of pages. The DBMS must remember where the first directory page of each heap file is located. The directory is itself a collection of pages and is shown as a linked list.

Each directory entry identifies a page (or a sequence of pages) in the heap file. As the heap file grows or shrinks, the number of entries in the directory and possibly the number of pages in the directory itself – grows or shrinks correspondingly. Note that since each directory entry is quite small in comparison to a typical page, the size of the directory is likely to be very small in comparision to the size of the heap file.

Free space can be managed by maintaining a bit per entry, indicating whether the corresponding page has any free space, or a count per entry, indicating the amount of free space on the page. If the file contains variable length records, we can examine the free space count for an entry to determine if the record will fit on the page pointed to by the entry. Since several entries fit on a directory page, we can efficiently search for a data page with enough space to hold a record that is to be inserted.

FILE ORGANIZATIONS AND INDEXING:

A file organization is a way of arranging the records in a file when the file is stored on disk. A file of records is likely to be accessed and modified in a variety of ways, and different of ways arranging the records enable different operations over the file to be carried out efficiently.

For example, if we want to retrieve employee records in alphabetical order, sorting the file by name is a good file organization. On the other hand, if we want to retrieve all employees whose salary is in a given range, sorting employee records by name is not a good file organization.

A DBMS supports several file organization techniques, and an important task of a DBA is to choose a good organization for each file, based on its expected pattern of use.

The three different file organizations are :

- Heap files : Files of randomly ordered are called Heap files.
- Sorted files : these are the files sorted on some field
- Hashed files : Files that are hashed on some fields are called hashed files.

4.1 COST MODEL

The cost model that allows us to estimate the cost (in terms of execution time) of different database operations.

The following notation and assumptions in our analysis.

There are B data pages with R records per page. The average time to read or write a disk page is D, and the average time to process a record (e.g. to compare a field value to a selection constant) is C. In the hashed file organization, we will use a function, called a hash function, to map a record into a range of numbers; the time required to apply the hash function to a record is H.

Typical values today are D = 25 milliseconds.

C and H = 1 to 10 microseconds ; we therefore expect the cost of I/O to dominate. This conclusion is supported by current hardware trends, in which CPU speeds are steadily rising, where as disk speeds are not increasing at a similar pace. On the other hand, one should keep in mind that as main memory sizes increase, a must larger fraction of the needed pages are likely to fit in memory, leading to fewer I/O requests.

The number of disk page I/O s can be used as the cost metric.

Two important considerations of cost model are :

The real systems must consider other aspects of cost, such as CPU costs (and transmission costs in a distributed database).

Since I/O is often (even typically) the dominant component of the cost of database operations, considering I/O costs gives us a good first approximation to he true cost.

A simplistic model in which user count the number of pages that are read from or written to disk as a measure of I/O. In blocked access, disk systems allow user to read a block of contiguous pages in a single I/O request. The cost is equal to the time required to seek the first page in the block and to transfer all pages in the block. Such blocked access can be much cheaper than issuing one I/O request per page in the block, especially if these requests do not follow consecutively.

COMPARISON OF THREE FILE ORGANISATIONS :

The costs of some simple operations for three basic file organizations;

Files of randomly ordered records, or heap files;

Files sorted on a sequence of fields or sorted files

Fiels that are hashed on a sequence of fields or hashed files

The choice of file organization can have a significant on performance. The choice of an appropriate file organization depends on the following operations.

Scan :

Fetch all records in the file. The pages in the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record for locating the record on the page (in the pool).

Search with equality selection:

Fetch all records that satisfy an equality selection, for example, " find the students record for the student with sid 23' Pages that contain qualifying records must be fetched from disk, and qualifying records must be located within retrieved pages.

Search with Range selection ;

Fetch all records that satisfy a range section, for example, "find all students records with name alphabetically after 'smith"

Insert :

Insert a given record into the file. We must identify the page in the file into which the new record must be inserted, fetch that page from disk, modify it to include the new record, and then write back the modified page. Depending on the file organization, we may have to fetch, modify and write back other pages as well.

Delete :

Delete a record that is specified using its record identity 9rid). We must identify the page that contains the record, fetch it from disk, modify it, and write it back. Depending on the file organization, we may have to fetch, modify and write back other pages as well.

Heap files :

Files of randomly ordered records are called heap files.

The various operations in heap files are :

Scan :

The cost is B(D+RC) because we must retrieve each of B pages taking time D per page, and for each page, process R records taking time C per record.

Search with Equality selection :

Suppose that user knows in advance that exactly one record matches the desired equality selection, that is, the selection is specified on a candidate key. On average, user must scan half the file, assuming that the record exists and the distribution of values in the search field is uniform.

For each retrieved data page, user must check all records on the page to see if it is the desired record. The cost is 0.5B(D+RC). If there is no record that satisfies the selection then user must scan the entire file to verify it.

Search with Range selection :

The entire file must be scanned because qualifying records could appear anywhere in the file, and does not know how many records exist. The cost is B(D+RC).

Insert : Assume that records are always inserted at the end of the file so fetch the last page in the file, add the record, and write the page back. The cost is 3D+C.

Delete :

First find the record, remove the record from the page, and write the modified page back. For simplicity, assumption is made that no attempt is made to compact the file to reclaim the free space created by deletions. The cost is the cost of searching plus C+D.

The record to be deleted is specified using the record id. Since the page id can easily be obtained from the record it, user can directly read in the page. The cost of searching is therefore D

Sorted files :

The files sorted on a sequence of field are known as sorted files.

The various operation of sorted files are

• Scan : The cost is B(D+RC) because all pages must be examined the order in which records are retrieved corresponds to the sort order.

(ii) Search with equality selection:

Here assumption is made that the equality selection is specified on the field by which the file is sorted; if not, the cost is identical to that for a heap file. To locate the first page containing the desired records or records, qualifying records must exists, with a binary search in log 2 B steps. Each step requires a disk I/O two comparisons. Once the page is known the first qualifying record can again be located by a binary search of the page at a cost of Clog2 R. The cost is Dlog2 B + Clog2B. This is significant improvement over searching heap files.

(iii) Search with Range selection :

Assume that the range selection is on the soft field, the first record that satisfies the selction is located as it is for search with equality. Subsequently, data pages are sequentially retrieved until a record is found that does not satisfy the range selection ; this is similar to an equality search with many qualifying records.

(iv) Insert :

To insert a record preserving the sort order, first find the correct position in the file, add the record, and then fetch and rewrite all subsequent pages. On average, assume that the inserted record belong in the middles of the file. Thus, read the latter half of the file and then write it back after adding the new record. The cost is therefore the cost of searching to find the position of the new record plus 2 * (0.5B(D+RC)), that is, search cost plus B(D+RC)

(v) Delete :

First search for the record, remove the record from the page, and write the modified page back. User must also read and write all subsequent pages because all records that follow the deleted record must be moved up to compact the free space. The cost is search cost plus B(D+RC) Given the record identify (rid) of the record to delete, user can fetch the page containing the record directly.

Hashed files :

A hashed file has an associated search key, which is a combination of one or more fields of the file. In enables us to locate records with a given search key value quickly, for example, “Find the students record for Joe” if the file is hashed on the name field we can retrieve the record quickly.

This organization is called a static hashed file; its main drawback is that long chains of overflow pages can develop. This can affect performance because all pages ina bucket have to be searched.

The various operations of hashed files are ;

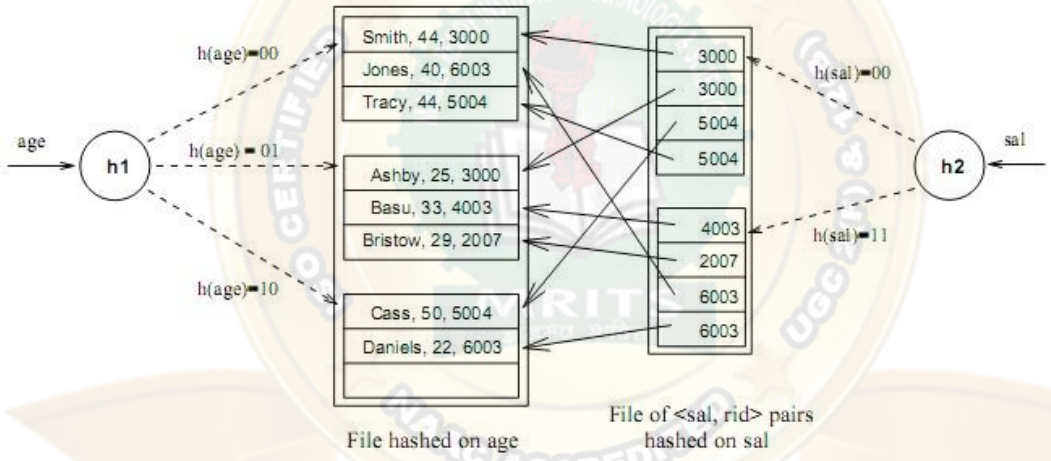


Fig: File Hashed on age,with Index on salary

Scan :

In a hashed file, pages are kept at about 80% occupancy (in order to leave some space for futue insertions and minimize over flow pages as the file expands). This is achieved by adding a new page to a bucket when each existing page is 80% full, when records are initially organized into a hashed file structure. Thus the number of pages, and therefore the cost of scanning all the data pages, is about 1.25 times the cost of scaning an unordered file, that is, 1.25B(D+RC)

Search with Equality selection :

The hash function associated with a hashed file maps a record to a bucket based on the values in all the search key fields; if the value for anyone of these fields is not specified, we cannot tell which bucket the record belongs to. Thus if the selection is not an equality condition on all the search key fields, we have to scan the entire file.

Search with Range selection :

The harsh structure offers no help at all; even if the range selection is on the search key, the entire file must be scanned. The cost is 1.25 B{D+RC}

Insert :

The appropriate page must be located, modified and then written back. The cost is thus the cost of search plus C+D.

Delete :

We must search for the record, remove it from the page, and write the modified page back. The cost is again the cost of search plus C+D (for writing the modified page).

Choosing a file organization :

The below table compares I/O costs for three file organizations

- A heap file has good storage efficiency, and supports fast scan, insertion, and deletion of records. However it is slow for searches.
- A stored file also offers good storage efficiency, but insertion and deletion of records is slow. It is quite for searches, and in particular, it is the best structure for range selections.

• A hashed file does not utilize space quite as well as sorted file, but insertions and deletions are fast, and equality selections are very fast. However, the structure offers no support for range selections, and full file scans are title slower; the lower space utilization means that files contain more pages.

INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

The potential large size of the index file motivates the ISAM idea. Building an auxiliary file on the index file and so on recursively until the final auxiliary file fits on one page? This repeated construction of a one-level index leads to a tree structure that is illustrated in Figure The data entries of the ISAM index are in the leaf pages of the tree and additional overflow pages that are chained to some leaf page. In addition, some systems carefully organize the layout of pages so that page boundaries correspond closely to the physical characteristics of the underlying storage device. The ISAM structure is completely static and facilitates such low-level optimizations.

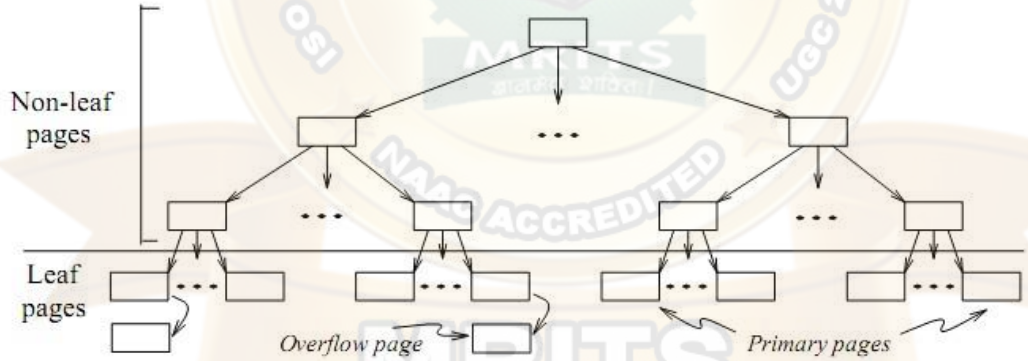


Fig ISAM Index Structure

Each tree node is a disk page, and all the data resides in the leaf pages. This corresponds to an index that uses Alternative (1) for data entries, we can create an index with Alternative (2) by storing the data records in a separate file and storing key, rid pairs in the leaf pages of the ISAM index. When the file is created, all leaf pages are allocated sequentially and sorted on the search key value.The non-leaf level pages are then allocated. If there are several inserts to the file subsequently, so that more entries are

inserted into a leaf than will fit onto a single page, additional pages are needed because the index structure is static. These additional pages are allocated from an overflow area. The allocation of pages is illustrated in below Figure.

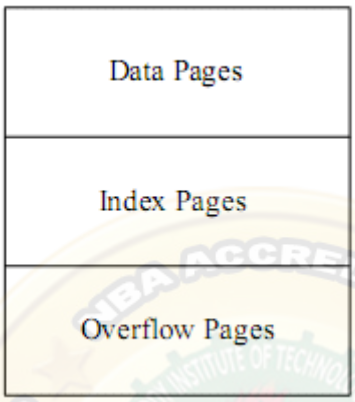


Fig: Page allocation in ISAM

B+ tree

A static structure such as the ISAM index suffers from the problem that long overflow chains can develop as the file grows, leading to poor performance. This problem motivated the development of more flexible, dynamic structures that adjust gracefully to inserts and deletes. The **B+ tree** search structure, which is widely used, is a balanced tree in which the internal nodes direct the search and the leaf nodes contain the data entries. Since the tree structure grows and shrinks dynamically, it is not feasible to allocate the leaf pages sequentially as in ISAM, where the set of primary leaf pages was static. In order to retrieve all leaf pages efficiently, we have to link them using page pointers. By organizing them into a doubly linked list, we can easily traverse the sequence of leaf pages in either direction. This structure is illustrated in Figure.

The following are some of the main characteristics of a B+ tree:

Operations (insert, delete) on the tree keep it balanced.However, deletion is often implemented by simply locating the data entry and removing it, without adjusting the tree as needed to guarantee the 50 percent occupancy, because files typically grow rather than shrink. Searching for a record requires just a traversal from the root to the appropriate leaf. We will refer to the length of a path from the root to a leaf—any leaf, because the tree is balanced—as the **height** of the tree.

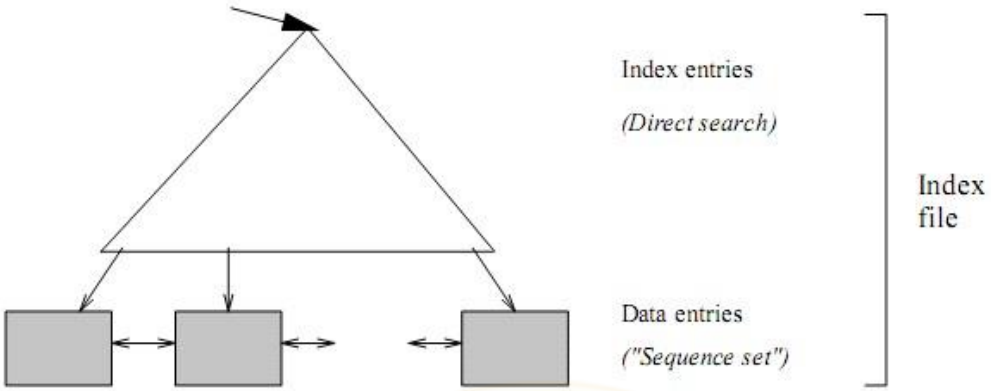


Fig: structure of B+ Tree

B) Assignment Questions:

1)Explain briefy about levels of abstraction?
2)Explain structure of DBMS?
3)Explain Relationship,attributes,entity sets,relationship sets?
4)Explain Relational algebra and relational calculus?
5)Explain Integrity constraints over relationships?
6)Define normal form and explain its types?
7)Explain ACID properties?
8)Compare and describe timestamp based protocol,validation based protocol?

- **Unit-1**

1. Explain DBMS? Explain Database system Applications.

2. Make a comparison between Database system and File system.

3. Explain storage manager component of Database System structure.

4. Explain the Database users and user interfaces.

5. Explain levels of data abstraction.

6. List and explain the functions of data base administrator

7. What is an ER diagram? Specify the notations used to indicate various components of ER-diagram

8. Explain the Transaction management in a database.

9. What are the types of languages a database system provides? Explain.

10.How to specify different constraints in ER diagram with examples.

11. What is an unsafe query? Give an example and explain why it is important to disallow such queries?

12. Explain the Participation Constraints.

13. List the six design goals for relational database and explain why they are desirable.

14. A company database needs to store data about employees, departments and children of employees. Draw an ER diagram that captures the above data.

15. Discuss aggregation versus ternary Relationships.

16. Explain conceptual design for large Databases.

17. Explain how to differentiate attributes in Entity set?

18. What is the composite Attribute? How to model it in the ER diagram? Explain with an example.

19. Compare candidate key , primary key and super key.

Unit-2

1. What is a relational database query? Explain with an example.

2. Relational Calculus is said to be a declarative language, in contrast to algebra, which is a procedural language. Explain the distinction.

3. Discuss about Tuple Relational Calculus in detail.

Write the following queries in Tuple Relational Calculus for following Schema.

Sailors (sid: integer, sname: string, rating: integer, age: real)

Boats (bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer, day: date)

i. Find the names of sailors who have reserved a red boat

ii. Find the names of sailors who have reserved at least one boat

iii. Find the names of sailors who have reserved at least two boats

iv. Find the names of sailors who have reserved all boats.

4. Explain various operations in relational algebra with example.

5. Compare procedural and non procedural DML's.

6. Explain about Relation Completeness

7.Consider the following schema:

Suppliers (sid : integer, sname: string, address: string)

Parts (pid : integer, pname: string, color: string)

Catalog (sid : integer; pid : integer, cost: real)

The key fields are underlined. The catalog relation lists the price changes for parts by supplies. Write the following queries in SQL.

i. Find the pnames of parts for which there is some supplier.

ii. Find the snames of suppliers who supply every part.

iii. Find the pnames of parts supplied by raghu supplier and no one else.

iv. Find the sids of suppliers who supply only red parts.

8. Explain in detail the following

i. join operation

ii. Nested-loop join

iii.Block Nested-Loop join.

9. Write the SQL expressions for the following relational database?

sailor schema(sailor id, Boat id, sailorname, rating, age)

Reserves(Sailor id, Boat id, Day)

Boat Schema(boat id, Boatname,color)

i. Find the age of the youngest sailor for each rating level?

ii. Find the age of the youngest sailor who is eligible to vote for each rating level with at lead two such sailors?

iii. Find the No. of reservations for each red boat?

iv. Find the average age of sailor for each rating level that atleast 2 sailors.

10. What is outer join? Explain different types of joins?

11. What is a trigger and what are its 3 parts. Explain in detail.

12. What is view? Explain the Views in SQL.

Unit-3

1. a. What is Normalization? give types of normalization

   b.What are the advantages of normalized relations over the unnormalized relations?

2. What is redundancy? What are the problems caused by redundancy?

3. What is dependency preserving decomposition?

4. Explain multivalued dependencies with example

5. Explain lossless join decomposition

6. Consider the relation R(A,B,C,D,E) and FD's

   A->BC

   C->A

   D->E

   F->A

   E->D

Is the Decomposition R into R1(A,C,D),R2(B,C,D) and R3(E,F,D) lossless?

7. Explain BCNF with example?

8. Explain 3NF and 4NF with examples.

9. Explain 5NF with examples.

Unit-4

1. What is transaction? Explain the states and properties of transaction?

2. Explain the time stamp based protocols.

3. Discuss how to handle deadlocks?

4. Explain about multiple granularity

5. Explain read-only ,write-only & read-before-write protocols in serialazability.

6. Describe each of the following locking protocols

i. Two phase lock

ii. Conservative two phase lock

7. Explain the implementation of atomicity and durability.

8. Explain ACID properties of Transaction?

9. Explain different types of failures?

10. a. Explain logical undo Logging?

b. Explain Transaction Rollback?

11. Explain Log-Record Buffering in detail.

12. a. What are the merits & demerits of using fuzzy dumps for media recovery.

b. Explain the phases of ARIES Algorithm.

c. Explain 3 main properties of ARIES Algorithm

13. What information does the dirty page table and transaction table contain.

14. Explain about Buffer Manager in detail.

15. describe the shadow paging recovery technique

16. Explain the difference between system crash and disaster?

Unit-5

1. Explain the following

   a. Cluster indexes

   b. Primary and secondary indexes

   c. Clustering file organization

2. Discuss various file organizations.

3. Write short notes on dense and spare indices

4. Explain about the B+ tree structure in detail with an example

5. Write a short notes on ISAM.

6. Compare the Ordered Indexing with Hashing.

7. Compare Linear Hashing with extendable Hashing

8. Explain about external storage media.

9. Differentiate between Extendible vs. Linear Hashing.

D) BEYOND THE SYLLABUS TOPICS AND NOTES

DBMS Implementation

A database management system handles the requests generated from the SQL interface, producing or modifying data in response to these requests. This involves a multilevel processingsystem.

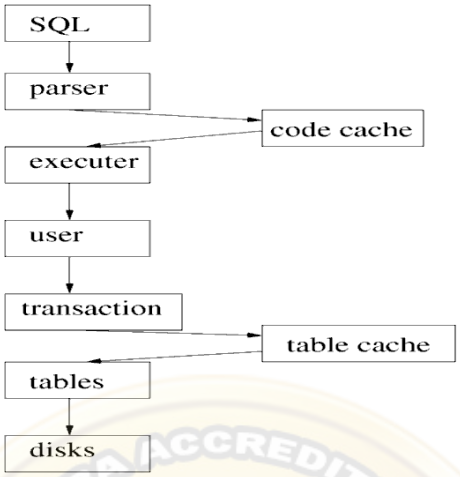


Figure : DBMS Execution and Parsing

This level structure processes the SQL submitted by the user or application.

- Parser: The SQL must be parsed and tokenised. Syntax errors are reported back to the user. Parsing can be time consuming, so good quality DBMS implementations cache queries after they have

been parsed so that if the same query is submitted again the cached copy can be used instead. To make the best use of this most systems use placeholders in queries, like:

- SELECT empno FROM employee where surname = ?

  The '?' character is prompted for when the query is executed, and can be supplied separately from the query by the API used to inject the SQL. The parameter is not part of the parsing process, and thus once this query is parsed once it need not be parsed again.

- Executer: This takes the SQL tokens and basically translates it into relational algebra. Each relational algebra fragment is optimised, and the passed down the levels to be acted on.

- User: The concept of the user is required at this stage. This gives the query context, and also allows security to be implemented on a per-user basis.

- Transactions: The queries are executed in the transaction model. The same query from the same user can be executing multiple times in different transactions. Each transaction is quite separate.

- Tables : The idea of the table structure is controlled at a low level. Much security is based on the concept of tables, and the schema itself is stored in tables, as well as being a set of tables itself.

- Table cache: Disks are slow, yet a disk is the best way of storing long-term data. Memory is much faster, so it makes sense to keep as much table information as possible in memory. The disk remains synchronised to memory as part of the transaction control system.

- Disks : Underlying almost all database systems is the disk storage system. This provides storage for the DBMS system tables, user information, schema definitions, and the user data itself. It also provides the means for transaction logging.

The 'user' context is handled in a number of different ways, depending on the database system being used. The following diagram gives you an idea of the approach followed by two differentsystems, Oracle and MySQL.

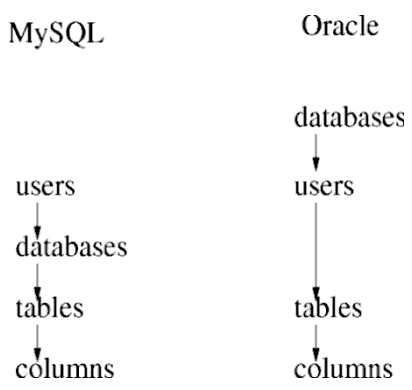


Figure : Users and Tablespaces

All users in a system have login names and passwords. In Oracle, during the connection phase, a database name must be provided. This allows one Oracle DBMS to run multiple databases, eachof which is effectively isolated from each other.

Once a user is connected using a username and password, MySQL places the user in a particulartablespace in the database. The name of the tablespace is independent of the same. In Oracle, tablespaces and usernames are synonymous, and thus you should really be thinking of different usernames for databases that serve different purposes. In MySQL the philosophy is more like a username is a person, and that person may want to do a variety of tasks.

Once in a tablespace, a number of tables are visible, and in each table columns are visible.

In both approaches, tables in other tablespaces can be accessed. MySQL effectively sees a tablespace and a database being the same concept, but in Oracle the two ideas are kept slightlymore separate. However, the syntax remains the same. Just as you can access column owner oftable CAR, if it is in your own tablespace, by saying

```
SELECT car.owner FROM car;
```

You can access table CAR in another tablespace (lets call it vehicles) by saying:

```
SELECT vehicles.car.owner FROM vehicles.car;
```

The appearance of this structure is similar in concept to the idea of file directories. In a databasethe directories are limited to "folder.table.column", although "folder" could be a username, a tablename, or a database, depending on the philosophy of the database management system.

Even then, the concept is largely similar.

Disk and Memory

The tradeoff between the DBMS using Disk or using main memory should be understood...

Issue	Main Memory VS Disk
Speed	Main memory is at least 1000 times faster than Disk
Storage Space	Disk can hold hundreds of times more information than memory for the same cost

Persistence	When the power is switched off, Disk keeps the data, main memory forgets everything
Access Time	Main memory starts sending data in nanoseconds, while disk takes milliseconds
Block Size	Main memory can be accessed 1 word at a time, Disk 1 block at a time

The DBMS runs in main memory, and the processor can only access data which is currently in main memory. The handling of the differences between disk and main memory effectively is atthe heart of a good quality DBMS.

Disk Arrangements

Efficient processing of the DBMS requests requires efficient handling of disk storage. The important aspects of this include:

- Index handling
- Transaction Log management
- Parallel Disk Requests
- Data prediction

With indexing, we are concerned with finding the data we actually want quickly and efficiently, without having to request and read more disk blocks than absolutely necessary. There are many approaches to this, but two of the more important ones are hash tables and binary trees.

When handling transaction logs, the discussion we have had so far has been on the theory of these techniques. In practice, the separation of data and log is much more blurred. We will lookat one technique for implementing transaction logging, known as shadow paging.

Finally, the underlying desire of a good DBMS is to never be in a position where no further work can be done until the disk gives us some data. Instead, by using prediction, prefetching, and parallel disk operations, it is hoped that CPU time becomes the limiting factor.

Hash tables

A Hash table is one of the simplest index structures which a database can implement. The majorcomponents of a hash index is the "hash function" and the "buckets". Effectively the DBMS constructs an index for every table you create that has a PRIMARY KEY attribute, like:

```
CREATE TABLE test (
  id  INTEGER PRIMARY KEY
 ,name varchar(100)
);
```

In table test, we have decided to store 4 rows...

```
insert into test values
(1,'Gordon');insert into test values
(2,'Jim'); insert into test values
(4,'Andrew');insert into test values
(3,'John');
```

The algorithm splits the places which the rows are to be stored into areas. These areas are calledbuckets. If a row's primary key matches the requirements to be stored in that bucket, then that iswhere it will be stored. The algorithm to decide which bucket to use is called the hash function.For our example we will have a nice simple hash function, where the bucket number equals the primary key. When the index is created we have to also decide how many buckets there are. In this example we have decided on 4.

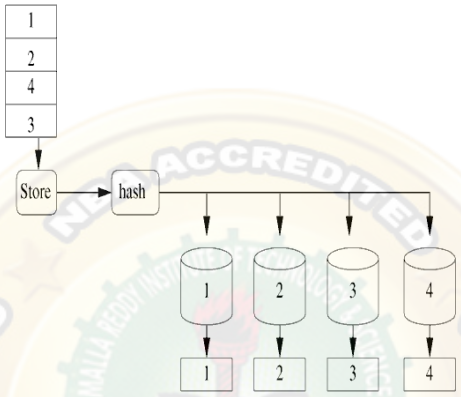


Figure : Hash Table with no collisions

Now we can find id 3 quickly and easily by visiting bucket 3 and looking into it. But now the buckets are full. To add more values we will have to reuse buckets. We need a better hash function based on mod 4. Now bucket 1 holds ids (1,5,9...), bucket 2 holds (2,6,10...), etc.

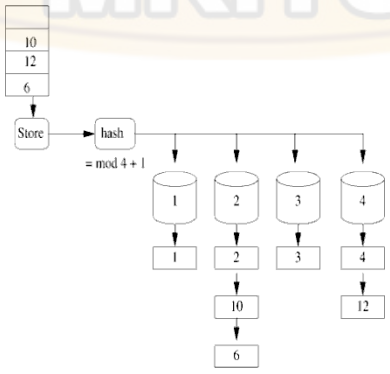


Figure : Hash Table with collisions

We have had to put more than 1 row in some of the buckets. This is called a hash collision. Themore collisions we have the longer the collision chain and the slower the system will get. For instance, finding id 6 means visiting bucket 2, and then finding id 2, then 10, and then finally 6.

In DBMS systems we can usually ask for a hash index for a table, and also say how many buckets we thing we will need. This approach is good when we know how many rows there is likely to be. Most systems will handle the hash table for you, adding more buckets over time ifyou have made a mistake. It remains a popular indexing technique.

Binary Tree

Binary trees is the latest approach to providing indexes. It is much cleverer than hash tables, and attempts to solve the problem of not knowing how many buckets you might need, and that some collision chains might be much longer than others. It attempts to create indexes such that all rowscan be found in a similar number of steps through the storage blocks.

The state of the art in binary tree technology is called B+ Trees. With B+ tree, the order of the original data is maintained in its creation order. This allows multiple B+ tree indices to be keptfor the same set of data records.

- the lowest level in the index has one entry for each data record.
- the index is created dynamically as data is added to the file.
- as data is added the index is expanded such that each record requires the same number of index levels to reach it (thus the tree stays `balanced').
- the records can be accessed via an index or sequentially.

Each index node in a B+ Tree can hold a certain number of keys. The number of keys is often referred to as the `order'. Unfortunately, `Order 2' and `Order 1' are frequently confused in the database literature. For the purposes of our coursework and exam, `Order 2' means that there canbe a maximum of 2 keys per index node. In this module, we only ever consider order 2 B+ trees.

B+ Tree Example

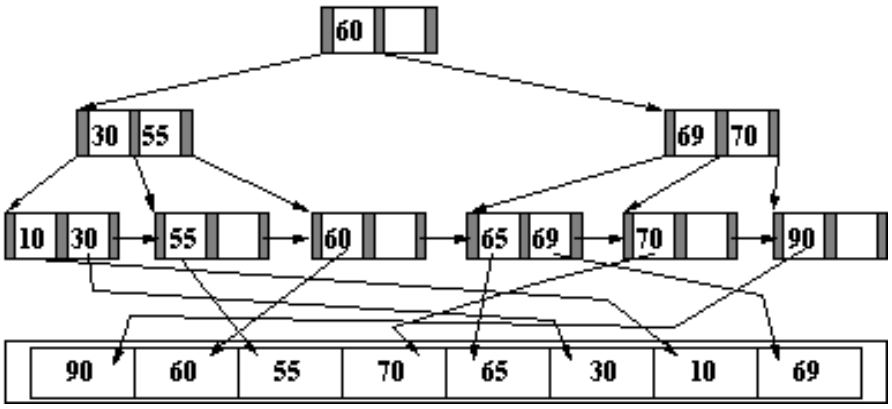


Figure : Completed B+ Tree

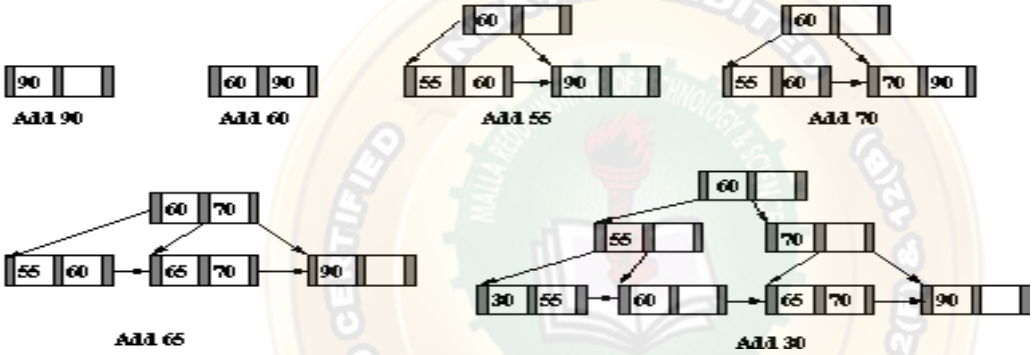


Figure : Initial Stages of B+ Tree

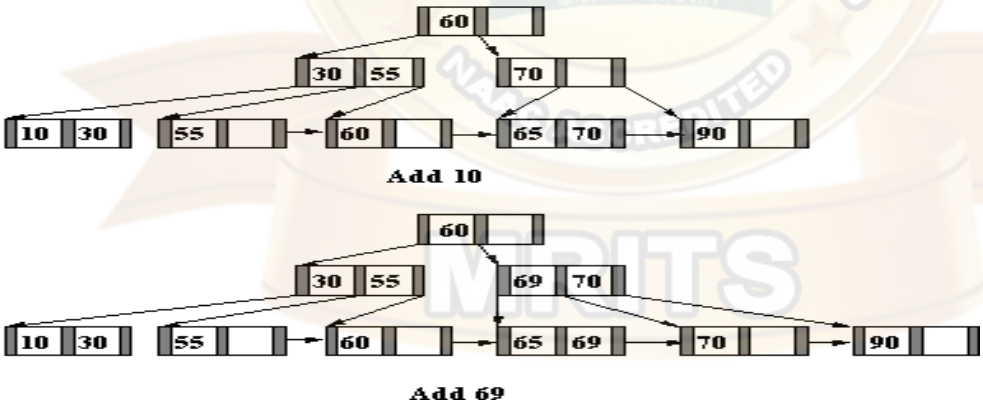


Figure : Final Stages of B+ Tree

Index Structure and Access

- The top level of an index is usually held in memory. It is read once from disk at the start of queries.

- Each index entry points to either another level of the index, a data record, or a block of data records.

- The top level of the index is searched to find the range within which the desired record lies.
- The appropriate part of the next level is read into memory from disc and searched.
- This continues until the required data is found.
- The use of indices reduce the amount of file which has to be searched.

Costing Index and File Access

- The major cost of accessing an index is associated with reading in each of the intermediate levels of the index from a disk (milliseconds).
- Searching the index once it is in memory is comparatively inexpensive (microseconds).
- The major cost of accessing data records involves waiting for the media to recover the required blocks (milliseconds).
- Some indexes mix the index blocks with the data blocks, which means that disk accesses can be saved because the final level of the index is read into memory with the associated data records.

Use of Indexes

- A DBMS may use different file organisations for its own purposes.
- A DBMS user is generally given little choice of file type.
- A B+ Tree is likely to be used wherever an index is needed.
- Indexes are generated:
- (Probably) for fields specified with `PRIMARY KEY' or `UNIQUE' constraints in a CREATE TABLE statement.
- For fields specified in SQL statements such as CREATE [UNIQUE] INDEX indexname ON tablename (col [,col]...);
- Primary Indexes have unique keys.
- Secondary Indexes may have duplicates.
- An index on a column which is used in an SQL `WHERE' predicate is likely to speed up an enquiry.
- this is particularly so when `=' is involved (equijoin)

- no improvement will occur with `IS [NOT] NULL' statements
- an index is best used on a column with widely varying data.
- indexing a column of Y/N values might slow down enquiries.
- an index on telephone numbers might be very good but an index on area code might be a poor performer.
- Multicolumn index can be used, and the column which has the biggest range of values or is the most frequently accessed should be listed first.
- Avoid indexing small relations, frequently updated columns, or those with long strings.
- There may be several indexes on each table. Note that partial indexing normally supports only one index per table.
- Reading or updating a particular record should be fast.
- Inserting records should be reasonably fast. However, each index has to be updated too, so increasing the indexes makes this slower.
- Deletion may be slow.
- particularly when indexes have to be updated.
- deletion may be fast if records are simply flagged as `deleted'.

Shadow Paging

The ideas proposed for implementing transactions are prefectly workable, but such an approach would not likely be implemented in a modern system. Instead a disk block transaction techniquewould more likely be used. This saves much messing around with little pieces of information, while maintaining disk order and disk clustering.

Disk clustering is when all the data which a query would want has been stored close together onthe disk. In this way when a query is executed the DBMS can simple "scoop" up a few tracks from the disk and have all the data it needs to complete the query. Without clustering, the disk may have to move over the whole disk surface looking for bits of the query data, and this could be hundreds of times slower than being able to get it all at once. Most DBMS systems perform clustering techniques, either user-directed or automatically.

With shadow paging, transaction logs do not hold the attributes being changed but a copy of the whole disk block holding the data being changed. This sounds expensive, but actually is highly efficient. When a transaction begins, any changes to disk follow the following procedure:

- If the disk block to be changed has been copied to the log already, jump to 3.

- Copy the disk block to the transaction log.
- Write the change to the original disk block.

On a commit the copy of the disk block in the log can be erased. On an abort all the blocks in thelog are copied back to their old locations. As disk access is based on disk blocks, this process is fast and simple. Most DBMS systems will use a transaction mechanism based on shadow paging.

Disk Parallelism

When you look at an Oracle database implementation, you do not see one file but several...

```
ls -sh

/u02/oradata/grussell/

2.8M control01.ctl

        M
        control02.c
        tl 2.8M
        control03.c
        tl 11M
        redo01.log
        11M
        redo02.log
        11M
        redo03.log
        351M
        sysaux01.d
        bf451M
        system01.d
        bf3.1M
        temp01.db
        f 61M
        undotbs01.
        dbf 38M
        users01.db
        f
```

Each of these files has a separate function in Oracle, and requests can be fired to each of them inparallel. The transaction logs are called redo logs. The activesql interface is stored completely inusers01. In my case all the files are in a single directory on a single disk, but each of the files could be on a different disk, meaning that the seek times for each file could be in parallel.

Caching of the files is also going on behind the scenes. For instance, the activesql tables only take up 38MB, and thus can live happly in memory. When queries come in the cache is accessedfirst, and if there is a need to go to disk then not only is the data requested read, but frequently data nearby that block is also read. This is called prefetching, and is based on the idea that if I need to go to disk for something, I might as well get more than I need. If it turns

out that the other stuff is not needed, then not much time or resource was wasted, but if the other stuff is needed in the near future, the DBMS gains a huge performance hit. Algorithms help to steer the preloading strategy to its best possible probability of loading useful data.

Lastly, the maximum performance of a database is achieved only when there are many querieswhich can run in parallel. In this case data loaded for one query may satisfy a different query. The speed of running 1 query on an empty machine may not be significantly different from running 100 similar queries on the machine.

DISTRIBUTED DATABASE

A **distributed database** is a database in which storage devices are not all attached to a common processing unit such as the CPU,[1] controlled by a **distributed database management system** (together sometimes called a **distributed database system**). It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely-coupled sites that share no physical components.

System administrators can distribute collections of data (e.g. in a database) across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Because they store data across multiple computers, distributed databases can improve performance at end-user worksites by allowing transactions to be processed on many machines, instead of being limited to one.[2]

Two processes ensure that the distributed databases remain up-to-date and current: replication and duplication.

- Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be complex and time-consuming depending on the size and number of the distributed databases. This process can also require a lot of time and computer resources.

- Duplication, on the other hand, has less complexity. It basically identifies one database as a master and then duplicates that database. The duplication process is normally done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, users may change only the master database. This ensures that local data will not be overwritten.

Both replication and duplication can keep the data current in all distributive locations.[2]

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

When discussing access to distributed databases, Microsoft favors the term **distributed query**, which it defines in protocol-specific manner as "[a]ny SELECT, INSERT, UPDATE, or DELETE statement that references tables and rowsets from one or more external OLE DB data sources".[3] Oracle provides a more language-centric view in which distributed queries and distributed transactions form part of **distributed SQL**.[4]

Architecture

A database user accesses the distributed database through:

Local applications

applications which do not require data from other sites.

Global applications

applications which do require data from other sites.

A **homogeneous distributed database** has identical software and hardware running all databases instances, and may appear through a single interface as if it were a single database. A **heterogeneous distributed database** may have different hardware, operating systems, database management systems, and even data models for different databases.

Homogeneous DDBMS

In a homogeneous distributed database all sites have identical software and are aware of each other and agree to cooperate in processing user requests. Each site surrenders part of its autonomy in terms of right to change schema or software. A homogeneous DDBMS appears to the user as a single system. The homogeneous system is much easier to design and manage. The following conditions must be satisfied for homogeneous database:

- The operating system used, at each location must be same or compatible.[*according to whom?*][*further explanation needed*]
- The data structures used at each location must be same or compatible.
- The database application (or DBMS) used at each location must be same or compatible.

Heterogeneous DDBMS

In a heterogeneous distributed database, different sites may use different schema and software. Difference in schema is a major problem for query processing and transaction processing. Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing. In heterogeneous systems, different nodes may have different hardware & software and data structures at various nodes or locations are also incompatible.

Different computers and operating systems, database applications or data models may be used at each of the locations. For example, one location may have the latest relational database management technology, while another location may store data using conventional files or old version of database management system. Similarly, one location may have the Windows NT operating system, while another may have UNIX. Heterogeneous systems are usually used when individual sites use their own hardware and software. On heterogeneous system, translations are required to allow communication between different sites (or DBMS). In this system, the users must be able to make requests in a database language at their local sites. Usually the SQL database language is used for this purpose. If the hardware is different, then

the translation is straightforward, in which computer codes and word-length is changed. The heterogeneous system is often not technically or economically feasible. In this system, a user at one location may be able to read but not update the data at another location.

Important considerations

Care with a distributed database must be taken to ensure the following:

• The distribution is transparent — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access among other things.

• Transactions are transparent — each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub-transactions, each sub-transaction affecting one database system.

There are two principal approaches to store a relation r in a distributed database system:

A) Replication

B) Fragmentation/Partitioning

A) Replication: In replication, the system maintains several identical replicas of the same relation r in different sites.

- Data is more available in this scheme.
- Parallelism is increased when read request is served.
- Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.
- Multi-datacenter replication provides geographical diversity: http://basho.com/tag/multi-datacenter-replication/

B) Fragmentation: The relation r is fragmented into several relations r_1, r_2, r_3 ..... r_n in such a way that the actual relation could be reconstructed from the fragments and then the fragments are scattered to different locations. There are basically two schemes of fragmentation:

- Horizontal fragmentation - splits the relation by assigning each tuple of r to one or more fragments.
- Vertical fragmentation - splits the relation by decomposing the schema R of relation r.

Advantages

- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.
- Increase reliability and availability
- Easier expansion
- Reflects organizational structure — database fragments potentially stored within the departments they relate to
- Local autonomy or site autonomy — a department can control the data about them (as they are the ones familiar with it)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database)
- Economics — it may cost less to create a network of smaller computers with the power of a single large computer
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems)
- Reliable transactions - due to replication of the database
- Hardware, operating-system, network, fragmentation, DBMS, replication and location independence
- Continuous operation, even if some nodes go offline (depending on design)
- Distributed query processing can improve performance
- Distributed transaction management
- Single-site failure does not affect performance of system.
- All transactions follow A.C.I.D. property:
- A-atomicity, the transaction takes place as a whole or not at all
- C-consistency, maps one consistent DB state to another
- I-isolation, each transaction sees a consistent DB
- D-durability, the results of a transaction must survive system failures

The Merge Replication Method is popularly used to consolidate the data between databases.[citation needed]

Disadvantages

- Complexity — DBAs may have to do extra work to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.

- Economics — increased complexity and a more extensive infrastructure means extra labour costs

- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (for example, by encryptingthe network links between remote sites).

- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a networkmay require too much of the network's resources to be feasible

- Inexperience — distributed databases are difficult to work with, and in such a young field there isnot much readily available experience in "proper" practice

- Lack of standards — there are no tools or methodologies yet to help users convert a centralizedDBMS into a distributed DBMS[citation needed]

- Database design more complex — besides of the normal difficulties, the design of a distributeddatabase has to consider fragmentation of data, allocation of fragments to specific sites and data replication

- Additional software is required

- Operating system should support distributed environment

- Concurrency control poses a major issue. It can be solved by locking and timestamping.

- Distributed access to data

- Analysis of distributed data.

E)OBJECTIVE QUESTIONS:

1. The component of the system assumes a state that is not desirable is…..
   [c]
   A) Fault B) Failures C) errors D) None
2. The assignment and management of memory blocks is?
   [b]
   A)Virtual memory B) Buffer management C) Memory management D) None
3. The coordination of the simultaneous execution of transactions in multi user database is?

   [b]
   A) ACID property B) Concurrency control C)Transaction control D) None of the above
4. Shared lock is used for……?
   [a]
   A) Read data B) Write data C) both a and b D) None
5. The two transaction wait indefinitely for each other to unlock data…….
   [b]
   A) Shared lock B) Deadlock C) Exclusive lock D) Both a and b
6. The correction from the failures is known as?

   [a]
   A)Recovery B)abort C) Commit D) None

7.Redundancy is dangerous as it is a potential threat to data
[b]

a)Integrity b)consistency c)sufficiency d)both a and b

8.Once the changes caused by an aborted transaction have been undone,we say that transaction
[c]
has been…..
a)Commited b)Partially commited c)Rolled back D)Aborted

9.Validation protocols have the----phase?
a)Read b)Write C)Validation D)All of the above
[d]
10.The page allocation is ISAM contains…….
a)Data pages b)Index pages c)overflow pages d)All of the above
[d]
11.Clustering indices are also called as
a)Primary indices b)Secondary indices c)dense indices d)sparse indices
[a]

12.Where performance and reliability are both important in RAID level [
d]

a)0 b)1 c)2 d)0+1

13.Which one of the following is not a secondary storage

[c]

a)magnetic disks b)magnetic tapes c)ram d)none

14.Which of the following is a physical storage media?

[d]

a)Tape storage b)optical storage c)flash memory d)All of the mentioned

15.The ------is the fastest and most costly form of storage ,which is relatively small,its use is managed by the computer system hardware

a)Cache b)Disk c)Main memory d)Flash memory

[a]

16.The information about data in a database is called

[a]

a)Meta data b)Hyper data c)Tera data d)None

17.Which of the following belongs to transaction failure

[c]

a)Read error b)Boot error c)Logical error d)All of the mentioned

18.The database is partitioned into fixed length storage units called

[b]

a)Parts b)Blocks c)Reads d)Build

19.Which of the following causes system to crash

a)Bug in software b)loss of volatile data c)hardware malfunction d)All of the mentioned

[d]

20.Which of the following are introduced to reduce the overheads caused by the log based recovery

[a]

a)checkpoints b)indices c)deadlocks d)locks

21.Which of the following protocols ensures conflict serializability and safety from deadlocks?

a)Two phase locking protocol b)Time stamp ordering protocol c)Graph based protocol d)none [b]

22.Identify the characteristics of transactions [d]

a)Atomicity b)Durability c)Isolation D)All of the mentioned

23.The property of a transaction that persists all the crashes is

[b]

a)Atomocity b)Durability c)Isolation D)All of the mentioned

24.Storage devices like tertiary storage,magnetic disk comes under

[b]

a)Volatile storage b)Non-volatile storage c)Stable storage d)Dynamic storage

25.The highest level in the hierarchy of data organization is called [b]

a)data bank b)data base c)data file d)data record

26.Space of disk is managed by the…….?

Ans)disk space manager

27……..can help when we have to access a collection of records in multiple ways?

Ans)indexing

28.Storing the same information several times in a database is known as?

Ans)Redundancy

29……is the property of the transactions that describes either all or none of transactions are reflected properly in the database

Ans)atomicity

30…….manager fetches the pages from disk if it is not already in memory

Ans)buffer

31.The schedule for the concurrent execution of the transaction is……

Ans)Serializability

32.The system does not function according to its specifications and fails is known as……

Ans)Failure

33.A…….is a program unit whose execution may change the contain of a database

Ans)Transaction

34)ISAM stands for……

Ans)Indexed sequential access method

35.Which level of RAID refers to disk mirroring with block striping?

Ans)RAID level 1

36.A dbms uses a transaction …..to keep track of all transactions that update the database

Ans)Log

37.A……is organized logically as a sequence of records

Ans)File

38.The bucket to which a value is assigned is determined by a function called…..

Ans)Hash function

39.A----------is a unit of exchange between disk and main memory

Ans)Page

40.----------hashing technique allows the hash function to be modified dynamically to accommodate the growth of the database

Ans)Dynamic hashing

41.In the ----------normal form composite attribute is converted to individual attributes.

Ans)First

42.Tables in second normal form----------

Ans)Eliminate all hidden dependencies

43.Functional dependencies are the types of constraints that are based on--------

Ans)key

44.What is the full form of RAID----------
Ans)Redundant array of independent disks
45.The process of duplicating every disk is called----------
Ans)Mirroring
46.How many levels of RAID exixt---------
Ans)7 (0 to 6)
47.RAID level 5 refers to-------
Ans)Block interleaved distributed parity
48.---------- is used for manufacturing chips?
ans)semiconductor
49.The remote backup site is sometimes also called the----------
Ans)Secondary site
50.Remote backup system must be ----------with the primary site
a)Synchronised b)separated c)Connected d)Detached but related

51. DBMS manages the interaction between _____ and database. [c]
A) Users B) Clients C) End users D) Stake holders
52. Which of the following is not involved in DBMS? [d]

A)Data B) End users C) Application Request D) HTML
53. Relational Algebra is? [c]

A) Data Definition Language B) Meta language C) Procedural Query Language D) None of the above
54. Key to represent relationship between tables is called? [b]

A) primary key B) foreign key C) super key D) All of the above

55. Which one produces the relation that has attributes of R1 and R2 [a]

A) Cartesian product B) Difference C) Intersection D) Product
56. DBMS helps to achieve? [d]

A) Data independence B) Centralized control of data C) Neither A Nor B D) Both A and B
57. Which of the following are the properties of entity [c]

A) Groups B) Table C) Attributes D) View
58. What is a relationship called when it is maintained between two two entities [b]

A) Unary B) Binary C) Ternary D) Quaternary
59. Which of the following operation is used if you are interested in only certain columns of a table [a]
A) Projection B) Selection C) Union D) Join

60. Which of the following is a valid MySQL type? [d]

A) CHARACTER B) NUMERIC C) FLOAT D) ALL OF THE ABOVE
61. The full form of DDL is. [c]

A) Dynamic data language B) Detailed data language C) Data definition language D) Data derivation language
62. Which of the following is an advantage of view? [d]

A)Data security B) Derived columns C) Hiding of complex queries D) All of the above
63. Which database level is closest to the users? [a]

A) External B) Physical C) Conceptual D) Internal
64. The result of the UNION operation between R1 and R2 is a relation that includes? [d]

A) all the tuples of R1 B) all the tuples of R2 C) all the tuples of R1 and R2 D) all the tuples of R1 and R2 which have common values
65. Which of the following is other name for weak entity? [a]

A) Child B) Owner C) Dominant D) All of the above
66. Null is ? [c]

A)the same as 0 for integer B) the same as blank for character C) Not a value D) Both A and B
67. In ER diagram attributes are represented by [a]

A) Ellipse B) Rectangle C) Diamond D) Triangle
68. DBMS stands for [c]

A) Database marginal system B) Directory based memory standard C) Database management systems D) Dual bus mask storage
69. Relational calculus is a [b]

A) Procedural language B) Non procedural language
C) Data definition language D) High level language
70. Architecture of the database can be viewed as? [c]

A) one level B) two level C)three level D) four level
71. In a relational model relations are termed as [a]

A) Tables B) Tuples C) Attributes D) Rows

72. DML is provided for [c]

A) Description of logical structure of database B) Addition of new structures in the database
C) Manipulation and processing of database D) Definition of physical structure of database

73. Incase of entity integrity the primary key may be? [b]

A) null B) not null C) both null and not null D) any value

74. An entity set that doesn't have sufficient attributes to form a primary key is a [b]

A) Strong entity set B) Weak entity set C) simple entity set D) Primary entity set

75. Count function in Mysql returns the number of [a]

A) Values B) Distinct values
C) Groups D) Columns

76.Which mysql statement is used to update data in a database?
Ans)update

77.Which mysql statement is used to return only different values?
Ans)select distinct

78.Which keyword is used to sort the result set?
Ans)orderby

79.How can you return the number of records in the persons table
Ans)select count(*)from persons

80.Which operator is used to select values with in a given range
Ans)between

81.Which operator is used to search for a specified pattern in a column
Ans)like

82.Which are of the following refers to the data about data
Ans)meta data

83.To which of the following term DBA refers
Ans)database administrator

84)TCL stands for
Ans)Transaction control language

85.Which of the following command is used to save any transaction permanently into the database.
Ans)commit

86.Which of the following command is used to restore the database to the last committed state?
Ans)rollback

87.a functional dependency is a relationship between of
Ans)attributes

88.The variables in the triggers are declared using
Ans)@

89.Which of the following is not an aggregate function
Ans)with

90.In the .....normal form,a composite attribute is converted to individual attributes
Ans)first

91.Functional dependencies are the types of constraints that are based on
Ans)key

92.Which forms are based on the concept of functional dependencies

Ans)3nf
93. The term "SQL" stands for
Ans)structured query language

94.Which one of the following is commonly used to define the overall design of the database?
Ans)schema
95.In general, a file is basically a collection of all related_____.
Ans)records
96.In one-to-many relationship the table in 'one' side is called _____
Ans)parent
97.It is used to establish an association between related tables.
Ans)relationship
98.Two tables can be linked with relationship to _____
Ans)ensure data integrity
99.The overall description of a database is called_____.
Ans)database schema
100.Which of the following is not a database model
Ans)none (that means all the other options are database models)

F) NPTEL VIDEOS

https://nptel.ac.in/courses/106106220

VII.STUDENT SEMINAR TOPICS

1)Normal forms
2)ACID Properties
3)Triggers
4)Indexing techniques

VIII .PREVIOUS PAPERS

Code No: 114CQ **R13**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B.Tech II Year II Semester Examinations, May-2015
DATABASE MANAGEMENT SYSTEMS
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10

marks and may have a, b, c as sub questions.

PART- A **(25 Marks)**

Differentiate between schema and data model. [2M]

b) Give an example for total participation and partial participation. [3M]

c) List the primitive operators in Relational Algebra. [2M]

d) What is an active database? [3M]

e) Define SECOND Normal form. [2M]

f) Write about join dependencies. [3M]

g) What methods are used to assign timestamps to transactions? [2M]

h) What is the significance remote backup system? [3M]

i) What is meant by secondary index? [2M]

j) How to compute the disk access time? [3M]

PART - B **(50 Marks)**

2.a) List various categories of database users and discuss their interfaces to DBMS.

b) Discuss the functionality of query evaluation engine. [5+5]

OR

3. Construct an Entity-Relationship diagram for a online shopping systems such as Jabong/Flipcart. Quote your assumptions and list the requirements considered by you for conceptual database design for the above system. [10]

4.a) With a suitable example explain division operation in relational algebra.

b) What is the usage of 'group by' and 'having' clauses in SQL? [5+5]

OR

5. Consider the following schema to write queries in Domain relational calculus:
Sailor(sid, sname, age, rating)
Boats(bid, bname, bcolor)
Reserves(sid,bid,day)

a) Find the boats reserved by sailor with id 567.
b) Find the names of the sailors who reserved 'red' boats.
c) Find the boats which have at least two reservations by different sailors. [10]

6. What is meant by closure of F? Where F is the set of functional dependencies. Explain computing F+ with suitable examples. [10]

OR

7.a) Differentiate bet een FD and MVD.

b) Explain the problems related to decomposition. [5+5]

8.a) Explain transaction states and desirable properties.

b) How to test serializability of a schedule? Explain with an example. [5+5]

OR

9.a) Explain Failure classification.

b) What is log? What is log tail? Explain the concept of checkpoint log record. [5+5]

10. Explain extendable hashing techniques for indexing data records. Consider yourclass students data records and roll number as index attribute and show the hash directory. [10]

OR

11.a) Is disk cylinder a logical concept? Justify your answer.

b) Compare heap file organization with hash file organization. [5+5]

Code No: 114CQ R13

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B.Tech II Year II Semester Examinations, May - 2016

DATABASE MANAGEMENT SYSTEMS

(Common to CSE, IT)

Time: 3 Hours Max. Marks: 75

Note: This question paper contains two parts A and B.

Part A is compulsory which carries 25 marks. Answer all questions in Part A.

Part B consists of 5 Units. Answer any one full question from each unit.

Each question carries 10 marks and may have a, b, c as sub questions.

PART - A (25 Marks)

1.a) Discuss about DDL. [2]

b) Write brief notes on altering tables and views. [3]

c) Describe about outer join. [2]

d) What is meant by nested queries? [3]

e) What is second normal form? [2]

f) Describe the inclusion dependencies. [3]

g) What is meant by buffer management? [2]

h) What is meant by remote backup system? [3]

i) Discuss about primary indexes. [2]

j) What is meant by linear hashing? [3]

PART - B (50 Marks)

2. Explain the relational database architecture. [10]

OR

3. State and explain various features of E-R Models. [10]

4. Explain Tuple relational calculus. [10]

OR

5. Discuss about domain relational calculus. [10]

6. What is meant by functional dependencies? Discuss about second normal form. [10]

OR

7. Explain fourth normal form and BCNF. [10]
8. What is meant by concurrency control? [10]

OR

9. Discuss about failure with loss of nonvolatile storage. [10]
10. What is meant by extendable hashing? How it is different from linear hashing? [10]

OR

11. What are the indexed data structures? Explain any one of them. [10]

R18

Code No: 154AM

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B. Tech II Year II Semester Examinations, November/December - 2020

DATABASE MANAGEMENT SYSTEMS

(Common to CSE, IT)

Time: 2 Hours **Max. Marks: 75**

Answer any Five Questions

All Questions Carry Equal Marks

\- - -

1.a) What is DBMS? List four significant difference between file processing system and a DBMS?
b) What is E-R model? Draw an E-R Diagram for any Banking enterprise system. [6+9]

2.a) Explain about outer join operation in relational algebra.
b) Explain about domain relational calculus with example. [7+8]

3) a)Explain the following Operators in SQL with examples
i) SOME ii) IN iii) EXCEPT iv) EXISTS
b) Explain various DML functions in SQL with examples. [8+7]

4.Explain shadow paging recovery scheme for recovering the data base? [15]

5.Explain deletion and insertion operations in linear hashing with examples.

[15]

6.a) Explain about various database users and administrators in DBMS.
b) Draw an ER-Diagram for Week entity set and Strong entity set with example. [7+8]

7.a) Explain modification of the database operations in relational algebra with example.

b) Explain about domain relational calculus with example. [8+7]

8. What is normalization? Explain 4NF and 5NF Normal forms with example. [15]

---ooOoo---