NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIAL

CS 208 PRINCIPLES OF DATABASE DESIGN

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Courses offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Team workand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

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.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

SUBJECT CODE: C213	
COURSE OUTCOMES	
C208.1	Define, explain and illustrate the fundamental concepts of databases.
C208.2	Construct an E-R model from specifications to perform the transformation of the conceptual model into corresponding logical data structures.
C208.3	Model and design a relational database following the design principles.
C208.4	Develop queries for relational database in the context of practical applications.
C208.5	Define, explain and illustrate fundamental principles of data organization, query optimization and concurrent transaction processing.
C208.6	Acquire knowledge about the latest trends in databases.

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C208.1		3	3	3	3							
C208.2	3	3	3									
C208.3	3	3	3	3	3							
C208.4	3	3	3	3	2							
C208.5	3	2	2	2	2				2			
C208.6	3	2	2	2	3							
C208	3	2.67	2.67	2.16	2.16				2			

CO'S	PSO1	PSO2	PSO3
C208.1	3	3	3
C208.2		3	3
C208.3	3		
C208.4		3	3
C208.5	3		
C208.6			2
C208	3	3	2.75

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

S:NO;	TOPIC	PAGE NO:
1	Information Retrieval Query languages And their brief description	155
2	Latest tools used for ER diagram	156

Course code	Course Name	L-T-P - Credits	Year of Introduction
CS208	**Principles of Database Design**	**2-1-0-3**	**2016**

Pre-requisite: CS205 Data structures

Course Objectives

- To impart the basic understanding of the theory and applications of database management systems.
- To give basic level understanding of internals of database systems.
- To expose to some of the recent trends in databases.

Syllabus:

Types of data, database and DBMS, Languages and users. Software Architecture, E-R and Extended E-R Modelling, Relational Model – concepts and languages, relational algebra and tuple relational calculus, SQL, views, assertions and triggers, relational db design, FDs and normal forms, Secondary storage organization, indexing and hashing, query optimization, concurrent transaction processing and recovery principles, recent topics.

Expected outcome.

Students will be able to:

1. define, explain and illustrate the fundamental concepts of databases.
2. construct an Entity-Relationship (E-R) model from specifications and to perform the transformation of the conceptual model into corresponding logical data structures.
3. model and design a relational database following the design principles.
4. develop queries for relational database in the context of practical applications
5. define, explain and illustrate fundamental principles of data organization, query optimization and concurrent transaction processing.
6. appreciate the latest trends in databases.

Text Books:

1. Elmasri R. and S. Navathe, *Database Systems: Models, Languages, Design andApplication Programming*, Pearson Education, 2013.
2. Sliberschatz A., H. F. Korth and S. Sudarshan, *Database System Concepts*, 6/e, McGraw Hill, 2011.

References:

1. Powers S., *Practical RDF*, O'Reilly Media, 2003.
2. Plunkett T., B. Macdonald, *et al.*, *Oracle Big Data Hand Book*, Oracle Press, 2013.

Course Plan

Module	Contents	Hours (42)	Sem. Exam Marks
I	**Introduction:** Data: structured, semi-structured and unstructured data**,** Concept & Overview of DBMS, Data Models, Database Languages, Database Administrator, Database Users, Three Schema architecture of DBMS. Database architectures and classification. (Reading: Elmasri Navathe, Ch. 1 and 2. Additional Reading: Silbershatz, Korth, Ch. 1) **Entity-Relationship Model:** Basic concepts, Design Issues, Mapping Constraints, Keys, Entity-	**06**	**15%**

	Relationship Diagram, Weak Entity Sets, Relationships of degree greater than 2 (Reading: Elmasri Navathe, Ch. 7.1-7.8)		
II	**Relational Model:** Structure of relational Databases, Integrity Constraints, synthesizing ER diagram to relational schema (Reading: Elmasri Navathe, Ch. 3 and 8.1, Additional Reading: Silbershatz, Korth, Ch. 2.1-2.4) **Database Languages:** Concept of DDL and DML relational algebra (Reading: Silbershatz, Korth, Ch 2.5-2.6 and 6.1-6.2, Elmasri Navathe, Ch. 6.1-6.5)	**06**	**15%**
FIRST INTERNAL EXAM			
III	**Structured Query Language (SQL)**: Basic SQL Structure, examples, Set operations, Aggregate Functions, nested sub-queries (Reading: Elmasri Navathe, Ch. 4 and 5.1) **Views, assertions and triggers** (Reading: Elmasri Navathe, Ch. 5.2-5.3, Optional reading: Silbershatz, Korth Ch. 5.3).	**07**	**15%**
IV	**Relational Database Design:** Different anomalies in designing a database, normalization, functional dependency (FD), Armstrong's Axioms, closures, Equivalence of FDs, minimal Cover (proofs not required). Normalization using functional dependencies, INF, 2NF, 3NF and BCNF, lossless and dependency preserving decompositions (Reading: Elmasri and Navathe, Ch. 14.1-14.5, 15.1-15.2. Additional Reading: Silbershatz, Korth Ch. 8.1-8.5)	**07**	**15%**
SECOND INTERNAL EXAM			
V	**Physical Data Organization**: index structures, primary, secondary and clustering indices, Single level and Multi-level indexing, B+-Trees (basic structure only, algorithms not needed), (Reading Elmasri and Navathe, Ch. 17.1-17.4) **Query Optimization**: heuristics-based query optimization, (Reading Elmasri and Navathe, Ch. 18.1, 18.7)	**07**	**20%**
VI	**Transaction Processing Concepts:** overview of concurrency control and recovery acid properties, serial and concurrent schedules, conflict serializability. Two-phase locking, failure classification, storage structure, stable storage, log based recovery, deferred database modification, check-pointing, (Reading Elmasri and Navathe, Ch. 20.1-20.5 (except 20.5.4-20.5.5) , Silbershatz, Korth Ch. 15.1 (except 15.1.4-15.1.5), Ch. 16.1 – 16.5) **Recent topics (preliminary ideas only):** Semantic Web and RDF(Reading: Powers Ch.1, 2), GIS, biological databases (Reading: Elmasri and Navathe Ch. 23.3-23.4) Big Data (Reading: Plunkett and Macdonald, Ch. 1, 2)	**09**	**20%**
END SEMESTER EXAM			

Question Paper Pattern:

1. There will be *five* parts in the question paper – A, B, C, D, E
2. Part A
   a. Total marks : 12
   b. *Four* questions each having *3* marks, uniformly covering module I and II; All *four* questions have to be answered.
3. Part B
   a. Total marks : 18
   b. *Three* questions each having *9* marks, uniformly covering module I and II; T*wo* questions have to be answered. Each question can have a maximum of three subparts
4. Part C
   a. Total marks : 12
   b. *Four* questions each having *3* marks, uniformly covering module III and IV; All *four* questions have to be answered.
5. Part D
   a. Total marks : 18
   b. *Three* questions each having *9* marks, uniformly covering module III and IV; T*wo* questions have to be answered. Each question can have a maximum of three subparts
6. Part E
   a. Total Marks: 40
   b. *Six* questions each carrying 10 marks, uniformly covering modules V and VI; *four* questions have to be answered.
   c. A question can have a maximum of three sub-parts.
7. There should be at least 60% analytical/numerical/design questions.

MODULE NOTES & QUESTION BANK

MODULE I				
Q:NO:	**QUESTIONS**	**CO**	**KL**	**PAGE NO:**
1	Define Database Management System.	CO1	K5	5
2	Describe Data Independence?	CO1	K2	14
3	Discuss the main characteristics of the database approach and how it differs from traditional file systems?	CO1	K2	19
4	Explain different Database users.	CO1	K4	22
5	List the responsibilities of DBA.	CO1	K2	25
6	Explain different types of attributes.	CO1	K2	58
7	Define Entity Relationship design Issues.	CO1	K5	18
8	Describe Weak Entity set with Example.	CO1	K5	39
MODULE II				
1	Write a sample statement in DML and in DDL.	CO2	K2	46
2	Explain different steps in mapping E R Diagram into Relational Schema with Example.	CO2	K4	42
3	Define the following terms: a) Super key :	CO2	K2	44
4	Define Candidate Key	CO2	K2	44
5	Define Primary Key	CO2	K2	44
6	Explain Integrity Constraints.	CO2	K3	56
7	What are different Relational Algebra Operations	CO2	K5	62
MODULE III				
1	Define Assertion in SQL with example.	CO3	K2	82
2	Describe View Materialization?	CO3	K3	85
3	Illustrate group by clause with the help of example	CO3	K4	71
4	Differentiate having and where in SQL with example.	CO3	K5	76

5	Consider the following database with primary keys underlined Suppliers(sid, sname, address) Parts(pid, pname, color) Catalog(sid,pid, cost) Write SQL queries for the following: i. Find the names of suppliers who supply red part. ii. Find the names of suppliers who supply red part or green part. iii. Find the names of suppliers who supply red part and green part. iv. Find the names of supplier who supplies parts of maximum cost with part name.	CO3	K5	71
6	Consider the following database with primary keys underlined Course(Courseid,CName,Credits) Student(RollNo,Name,Gender,Address,Advisor) Professor(ProfId, PName,Phone) Enrollment(RollNo,CourseId,Grade) Write SQL statements for the following: i. Names of female students ii. Names of male students with advisor name RollNo & Name of students who have not enrolled for any course.	CO3	K5	71
MODULE IV				
1	Define Functional Dependancy.	CO4	K2	90
2	Discuss different anomalies in database design with examples.	CO4	K1	87

3	Describe Armstrong's Axioms.	CO4	K3	88
4	Define Minimal Cover	CO4	K2	92
5	Explain Normalization	CO4	K5	88
6	Define Lossless decomposition	CO4	K2	98
7	Differentiate between 3NF and BCNF with proper examples.	CO4	K3	92
8	Discuss Functional Dependancy.	CO4	K3	85
9	Discuss different anomalies in database design with examples.	CO4	K2	86
10	Explain the Algorithm for lossless decomposition.	CO4	K4	102
11	Compare different Normal Forms	CO4	K4	90
12	Justify the role of Normalization	CO4	K5	86
MODULE V				
1	Compare different indexing techniques	CO5	K4	104
2	With neat diagram , explain single level indexing.	CO5	K2	107
3	Write short notes on primary indexing.	CO5	K3	104
4	Briefly explain clustering indexing.	CO5	K2	111
5	With neat sketch explain the multilevel indexing	CO5	K2	112
6	Investigate in detail about basic structure of B+ trees	CO5	K5	116
7	Explain in detail about Query Optimization.	CO5	K2	121
8	Explain heuristics based Query optimization.	CO5	K2	124
MODULE VI				
1	With neat diagram explain Transaction states.	CO5	K4	130
2	Write short notes on ACID properties.	CO5	K2	132
3	Briefly explain System Logs.	CO5	K3	137
4	Investigate in detail about transaction schedule.	CO5	K2	128
5	Explain in detail about two phase locking.	CO5	K2	139
6	Explain Log based Recovery	CO5	K5	140
7	Describe GIS, Semantic Web	CO5	K2	144
8	Define Conflict serializability	CO5	K2	128

MODULE I

→ INTRODUCTION

* Data ✓
* Concept & overview of DBMS ✓ } 1
* Data Models ✓ 2
* Database Languages ✓ 2
* Database Adminstrator ✓ 2
* Database Users ✓ 2
* Three Schema Architecture ✓ } 3
* Db architecture & classification

→ ENTITY RELATIONSHIP MODEL

* Basic Concepts
* Design Issues ✓ 4
* Mapping Constraints } 5
* Keys ✓
* E-R Diagram ✓ 6
* Weak Entity Sets ✓ 6
* Relationships of degree > 2 ✓ 7

Questions: Bank

1. Define diff. terms.
2. Discuss the main categories of data models.
3. What is the diff. b/w db schema & db state.
4. Describe three-schema architecture. Why do we need mappings b/w schema levels? How diff. schema defn lang. support this architecture?
5. What is the diff. b/w logical data Independence & phy data Independence.
6. What is the diff. b/w procedural & non procedural lang?
7. Discuss the role of a high-level data model in the db design process.
8. List the various cases where use of a null value would be appropriate.

9. What is an entity type? What is an entity set? Explain the diff. among an entity, an entity type & an entity set?
10. Explain the diff. b/w an attribute & value set?
11. Explain the distinction b/w total & partial Constraints.
12. Explain diff. b/w a weak & a strong entity set.
13. Explain the distinctions among the terms primary key, candidate key & super key.
14. Explain Entity Integrity & Referential Integrity with suitable Examples.
15. Explain the different types of keys with suitable eg:
16. Explain diff. constraints on binary rel^nships.

Previous Questions: June 2017

1. List out Salient features of Database S/ms. Ref: Pg 4-5
2. How is DML different from ~~DML~~ DDL?. Write a sample stmt in DML & one in DDL. Ref. Page: 6

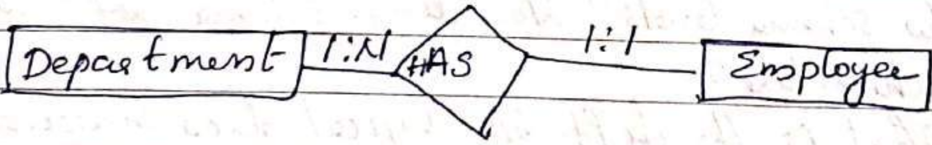
   Eg: DDL in SQL

   create table student (Name char(20), Reg No int(10), Mark int (10));

   DML in SQL:

   insert into student values ('XYZ', 101, 90);

3. Can we represent the situation modelled by the following ER diagram without relationship 'HAS'? If so, draw the new diagram. If not, give the reasons. (Entities are Department & Employee)



4. A company has the following scenario. There are a set of salespersons. Some of them manage other salespersons. However a salesperson cannot have more than one manager. A salesperson can be an agent for many customers. A customer is mnged by exactly

* Data

Data means known facts that can be recorded & have implicit meaning. For eg: names, telephone no.s & addresses of the people.

Data can be classified into
- Structured data
- Semistructured data
- Unstructured data.

→ Structured Data: refers to any data that resides in a ~~w~~ fixed field within a record or file. This includes data contained in relational databases & spreadsheets. Data has been organised into a formatted repository, a database, so that its elmts can be made addressable for more effective processing & analysis. A data structure is a kind of repository that organizes inf^n for that purpose. In a database, each field is discrete & its inf^n can be retrieved either separately or along with data from other fields, in a variety of combinations.

→ Semi_structured Data: is a form of structured data that doesn't conform with the formal structure of data models associated with relational dbs or other forms of data tables. It is self describing structure. It has not been organized into a specialised repository, such as db,

→ Unstructured Data: is information that either doesn't have a pre-defined data model or is not organized in a pre-defined manner. Unstructured inf^n is text heavy, but may contain data such as dates, no.s & facts as well. This results in irregularities & ambiguities that make it difficult to understand.

Example: A word document is generally considered to be unstructured data. We can add metadata tags in the form of keywords & other metadata that represent the document cont. & make it easier for that document to be found when people search for those terms - the data is now semi structured. The ~~doct~~ document is converted into complex orgⁿ of db, it is fully structured data.

Comparison:

Structured s/ms are those where the activity of processing & o/p is predetermined & highly organized. ATM transactions, airline reservations etc are all forms of structured s/ms.

Unstructured data are that have no predeterm form or structure. Eg: Email, reports, contracts etc.

Concept & Overview of DBMS:

* Database: is a collection of related data.

Properties of Database:

1. represents some aspects of real world. It is called the universe of discourse (UoD). Changes to this are reflected in db.

2. a logically coherent collⁿ of data with some inherent meaning.

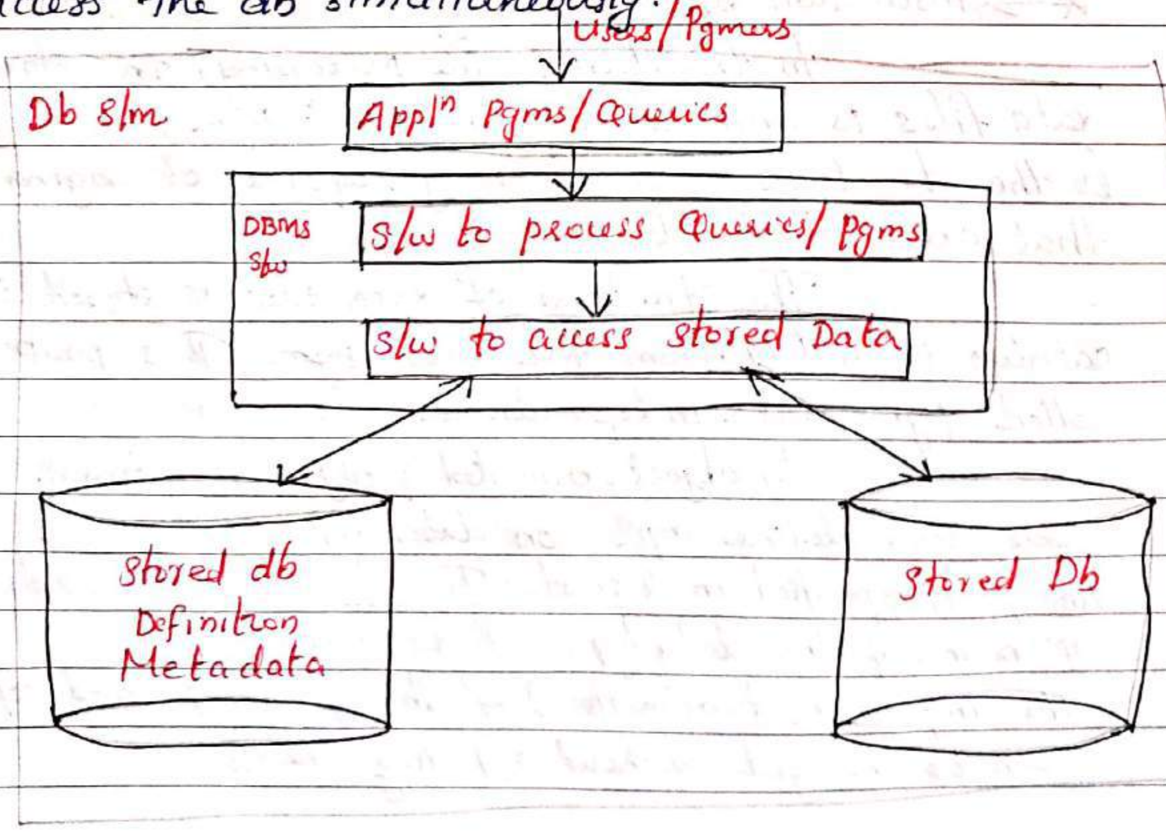
3. It is designed, built & populated with data for a specific purpose. It has intended users.

Eg: Amazon, large ~~co~~ commercial db.

* Database Management Systems:

is a collection of pgms that enables users to create & maintain a database. It facilitates the processes of defining, constructing, manipulating & sharing db among various users & appl^ns.

- Defining a db involves specifying the data types, structures & constraints of the data to be stored in the database.
- Meta data: The db definition or descriptive inf^n is also stored in the db in the form of a db catalog or dictionary.
- Constructing the db is the process of storing the data on some storage medium that is ctalled by the DBMS.
- Manipulating a db includes functions such as querying the db to retrieve specific data, updating the db to reflect changes & generating reports.
- Sharing a db allows multiple users & pgms to access the db simultaneously.



Characteristics of Database Approach:
* traditional file Processing
* → Self-Describing Nature of a Database S/m.
* → Insulation b/w pgms & data, & data abstraction.
* → Support of multiple views of Data.
* → Sharing of data & multiuser transⁿ processing.

* → Self Describing Nature of a Database S/m:

The Database S/m contains not only the db itself, but also a complete defⁿ or description of the db structure & constraints. The infⁿ stored in catalog is called meta-data, & it describes the structure of the primary db.

In traditional file processing, data defⁿ is part of the applⁿ pgms themselves. These pgms are constrained to work with only one specific db, whose structure is declared in the applⁿ pgms. Eg: an applⁿ pgm written in C++ may have struct or class declaration

* → Insulation b/w Pgms & Data and Data Abstraction

In traditional file processing, the structure of data files is embedded in the applⁿ pgms, so any changes to the structure of a file may require changing all pgms that access that file.

The structure of data files is stored in the DBMS catalog separately from the access pgms. This property is called pgm-data independance.

In object-oriented & object-relational s/ms, users can define opⁿs on data as part of the db defⁿs. An opⁿ is specified in 2 parts: The i/f of an opⁿ includes the opⁿ name & the data types of its args.
The implementation (method) of the opⁿ is specified separately & can be changed without affecting the i/f.

User appl^n pgms can operate on the data by invoking these op^ns through their names & args, regardless how the op^ns are implemented. This is called pgm-op^n independance.

The characteristic that allows pgm-data independance & pgm-op^n independance is called data abstraction.

*→ Support of Multiple Views of the Data:

— Users may require a diff. perspective or view of the db. A view may be a subset of the db or it may contain virtual data that is derived from the db files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored/derived.

* Sharing of Data & Multiuser Transaction Processing:

— A multiuser DBMS, must allow multiple users to access the db at the same time. The DBMS must include concurrency ctrl s/w to ensure that several users trying to update the same data so that the results of updates is correct.

Eg: Airticket reservation s/m: updation of seat availability. Such applications are called Online Trans^n Processing (OLTP).

Advantages of using a DBMS

* Controlling Redundancy:

~~Reda~~ Redundancy means storing same data in different ~~to~~ places. Problems due to Redundancy are

1. Consider student data record; Entering data on a new student; there is a need to perform a single logical update multiple times. once for each file where student data is recorded.

2. ~~Stud~~ Storage space is wasted when same data is stored repeatedly.

3. Inconsistent data - when an update is applied to some of the files but not to others.

* Restricting Unauthorized Access:
DBMS provides a security & authorization subs/m which DBA uses to create accounts & to specify account restri

* Providing Persistent Storage for Program Objects & Data Structu
Databases can be used to provide persistent storage for pgm objs & data structures. The values of pgm variables are discarded once a pgm terminates. The pgmmer explicitly stores them in permanent files. An obj is said to be persisten since it survives the termination of pgm exc^n & can later be directly retrieved by another pgm.

* Permitting Inferencing and Actions using Rules:
— Defines deduction rules for inferencing new inf^n from the stored db facts. Such s/ms are called deductive db.

* Providing Multiple User Interfaces:
DBMS provide a variety of user interfaces. Query languages for casual users; pgmming lang. i/f for appl^n pgmmers; forms & cmd codes for parametric users; & Menu driven i/f & natural lang. i/fs for stand alone users.

* Representing Complex Relationships Among Data:
A db may include numerous varaties of data that are interrelated in many ways. DBMS must have the capability to represent variety of complex rel^nships among the data as well as to retrieve & update related data easily & efficiently.

* Enforcing Integrity Constraints:
Most db applications have integrity constraints that must hold for the data. DBMS should provide capabilities for defining & enforcing these constraints. Eg: specifying a data type for each data item. Name - char :

* Providing Back up & Recovery:
DBMS provide facilities for recovering from h/w or s/w failures. The Back up or Recovery s/m is responsible for recovery.

* Database Administrators:

In any Orgⁿ, where many persons use the same resources, there is a need for a chief administrator to oversee & manage these resources. Here the primary resource is the db itself & secondary resource is the DBMS & related s/w. The DBA is responsible for authorizing access to the db, for coordinating & monitoring its use, & for acquiring s/w & h/w resources as needed. And also DBA is accountable for pblms such as breach of security or poor s/m response time.

* Database Users:

→ Database Designers: are responsible for identifying the data to be stored in the db & for choosing appropriate structures to represent & store this data. They communicate with all ~~pr~~ database users, in order to understand their rqmnts & to come up with a design that meets these rqmnts.

→ End Users: are the people use jobs require access to the db for querying, updating & generating reports.

Several categories of End users:

* Casual end users: occasionally access the db; but they may need diff. infⁿ each time. They use db query lang. to specify their rqsts.

* Naive or parametric end users: revolves around constantly querying & updating the db using std types of queries & updates called canned transⁿs. The tasks are:

Bank tellers check account balances & post withdrawals & deposits.

Reservation clerks for hotels, airlines etc check availability for a given rqst & make reservations.

* Sophisticated end users: engrs. scientists, business analysts & others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applⁿs to meet their complex rqmnts.

* Stand-alone users: maintain personal dbs by using ready-made pgm packages that provide easy to use menu or graphics based i/fs.
→ System Analysts: determine rqmnts of end users & develop specific"s for canned trans"s that meet these rqmnts.
Appl" pgmmers: implement these specific"s as pgms; then they test, debug, document & maintain canned trans"s.

* Data Models:

A data model is a collection of concepts that can be used to describe the structure of db. The concepts are data types, relationships, constraints & basic op"s for specifying retrievals & updates on the db.
- insert, delete, modify or retrieve are basic data model op"s.

Categories of Data Models:
High level/Conceptual data models: concepts that are close to the way many users perceive data.
Low level/physical data models: concepts that describe the details of how data is stored in a computer.
Representational/Implementation data models: concepts that may be understood by end users & the way data is organised within the computer.

Conceptual data models use concepts such as entity, attributes & rel"ships.
An entity represents real world objt or concept such as employee or a project. An attribute represents some property of entity eg: ID. Relationship represents interaction among entities.

Schemas: The description of db. which is specified during db design..

Student.

Name	S No.	Class	Major

Instances: The data in the db at a particular moment in time is called database state or snapshot.

Database Languages:

Data Definition Language (DDL): is used by DBA & db designers to define schema. The DBMS will have a DDL compiler whose fn is to process DDL stmts in order to identify descriptions of the schema constructs & to store schema description in the DBMS catalog.

Storage Definition Language: is used to specify internal schema.

View Definition Language (VDL): to specify user views & their mappings to the conceptual schema.

Data Manipulation Language (DML): to manipulate Db. ie; retrieval, insertion, deletion & modification.

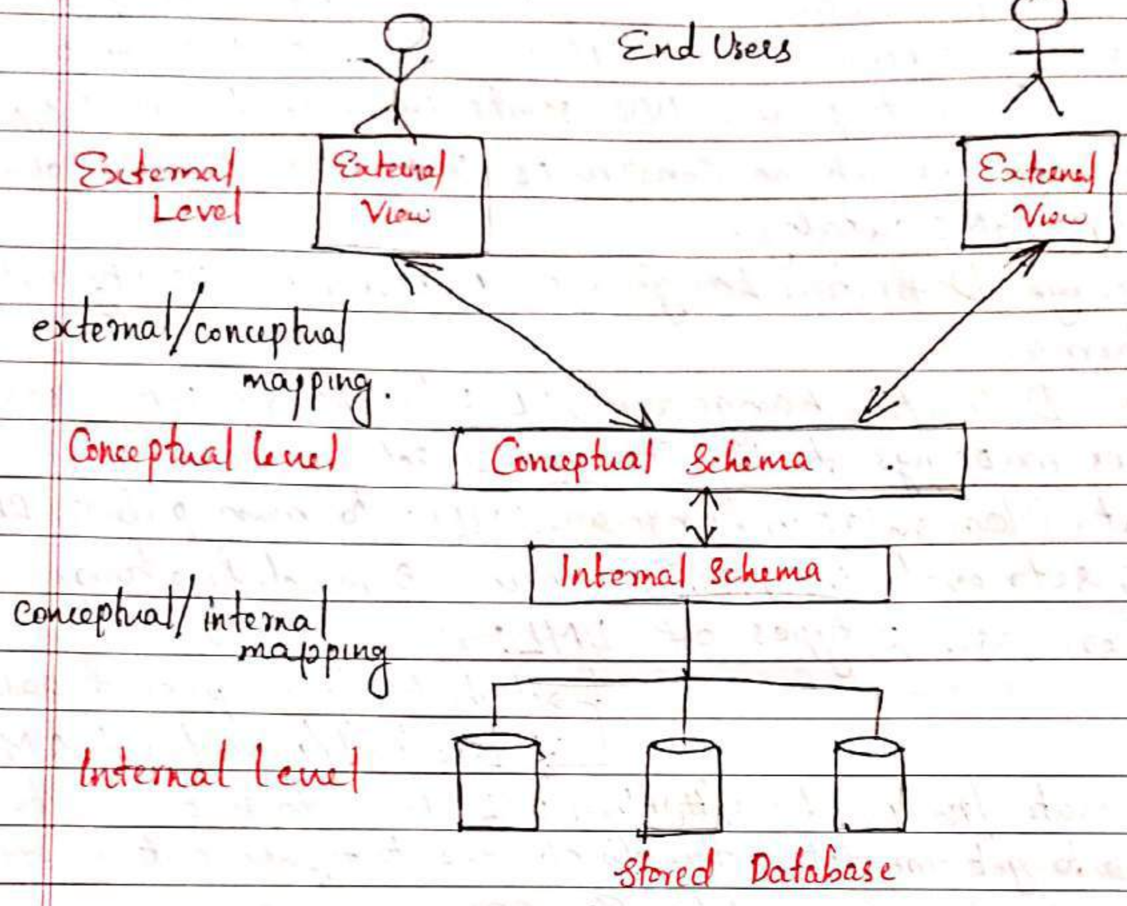
There are 2 types of DML:
- → High level/non procedural DML
- → Low level/procedural DML.

Declarative/Non procedural/High level DML: User doesn't have to specify how to get the data, the db s/m has to figure out an efficient means of accessing data. Eg: SQL.

Non declarative/Procedural/Low level lang: User have to specify what data are needed & how to get those data.

The data values stored in the db must satisfy certain consistency constraints: → Domain Constraints → Referential integrity → Assertions → Authorization ← read, insert, update, delete.

* Three Schema Architecture:



The goal of this architecture is to separate the user appl^n's & the phy. db. Schemas can be defined at the 3 levels:

Internal level: has an internal schema, which describes the phy. storage structure of the db. The internal schema describes complete storage details & access paths for the db.

Conceptual level: has a conceptual schema, which describes the structure of the whole db. The conceptual schema hides the details of phy. storage structures & concentrates on describ entities, data types, rel^nships user op^ns & constraints.

External or view level: includes a no. of ext. schemas or user views. describes the part of the db that a particular group is interested in & hides the rest of the db from that user group.

Data Independance: can be defined as the capacity to change the schema at one level of a db s/m without having to change the schema at the next higher level.

Two types of data Independance:

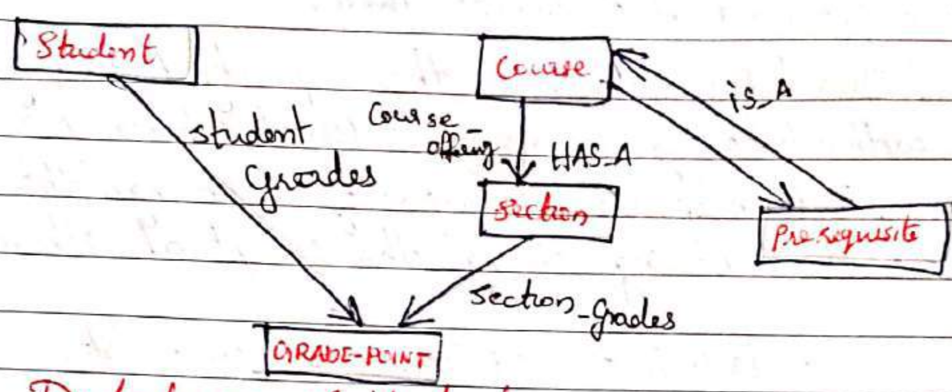
1. Logical Data Independance: is the capacity to change the conceptual schema without having to change external schema or appl^n pgms.
2. Physical Data Independance: capacity to change the internal schema without having change the conceptual schemas.

* Classification of Database Mgmt S/ms:

- Several criteria are used to classify DBMSs.

1. Data model / on which DBMS based
   - → relational model
   - → Object data model – db in terms of objts, propd & their op's.
   - → Objt relational model – RDBMS with objt db & capabilities.
   - → Hierarchical Model – tree structures, each hierarchy a no. of related records.
   - → N/w model – data as record types, 1:N rel^{n}ship
2. No. of users supported by the s/m
   - → Single user S/m
   - → Multi user S/m
3. No. of sites over which Db is distributed
   - → Centralised – if the data is stored at a single comp. site.
   - → Distributed – can have actual db & DBMS S/w distributed over many sites connected by a comp. n/w.
   - → Homogeneous – same DBMS S/w at multiple sites.
   - → Heterogeneous – several autonomous pre existing db.
   - → Federated DBMS – uses client server architecture.

4. Cost of the DBMS. — cost b/w $10000 & $100000
— cost b/w $100 & $3000.
→ more than $100,000.



Database Architecture

The architecture of a db s/m is greatly influen by the underlying computer s/m on which the db s/m runs. Db s/ms can be centralized or client-server, where one server m/c executes work on behalf of multiple client m/cs. Fig 1.6 S/m structure Sulbeeschatz.

1. Centralized DBMSs Architecture:

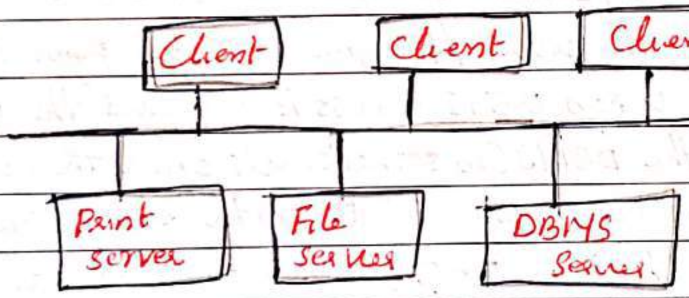
Architectures used mainframe computers to provide the main processing for all fns of the s/m, includ user appln pgms & user i/f pgms & DBMS fnality.
~~All~~ Most users accessed such s/ms via comp. terminals that didn't have processing power & only provided display capabilities. All processing was performed on the comp. s/m & only display infn & ctrls were sent from the computer to the display terminals, which were connected to the central comp. via various types of commn n/ws.

Most users replaced their terminals with PCs & workstations. They used centralised DBMS.

2. Client/Server Architectures:

– deal with computing envts in which a large no. of PCs, workstations, file servers, printers, db servers, web servers & other equipment are connected via a n/w. The idea is to define specialized servers with specific fnalities. For eg: it is possible to connect a no. of PCs or small workstations as clients to a file server that maintains the files of the client m/cs. Another m/c could be designated as a printer server by being connected to various printers; thereafter, all print rqsts by the clients are forwarded to this m/c. The client m/cs provide the user with the appropriate i/fs to utilize these servers, and with local processing power to run local appl"s.

Navathe. fig 2.4



Logical Two tier client/server Architecture.

The client/server framework consists of many PCs & workstations & smaller no. of ~~mf~~ mainframe m/cs connected via LAN or other comp. n/ws.

A client is a user m/c that provides user i/f capabilities & local processing. When a client requires access to additional fnality – such as db access – that doesn't exist at that m/c, it connects to a server that provides the needed fnality.

A server is a m/c that can provide services to the client m/cs such as file access, printing, archiving or db access. Some m/cs install only client s/w others only server s/w; & others may include both client & server s/w. Two main types of DBMS architectures:
- two-tier
- three-tier.

* Entity Relational Model

→ Database Application
→ ~~Conceptual~~ Application programs.
→ Phases of db design.
Fig : 3.1

→ Basic Concepts

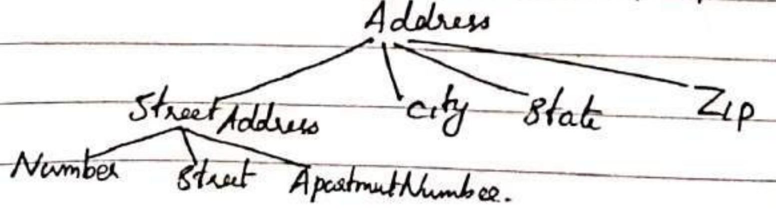
A db application can be displayed by means of the graphical notation known as E-R diagrams.
Example Company Database Application.
Fig 3.2.

* Entities & Attributes:

Entity is a basic objt which is a thing in the real world with an independent existence. Its an objt with a physical existence eg: person, car, house or employee. Each entity has attributes, particular properties that describe. Eg: employee entity - described by name, age, addr. & salary. A particular entity will have a value for each of its attributes. Several types of attributes:

1. Simple versus Composite Attributes:

Attributes that are not divisible are called Simple or atomic attributes. Composite attributes can be divided into smaller subparts, which represents more basic attributes with independent meanings. For eg: the Address attribute can be subdivided into StreetAddr., City, State & Zip. with diff. values. Street Address can be subdivided into 3 simple attributes, Number, street & Apartment No.



10

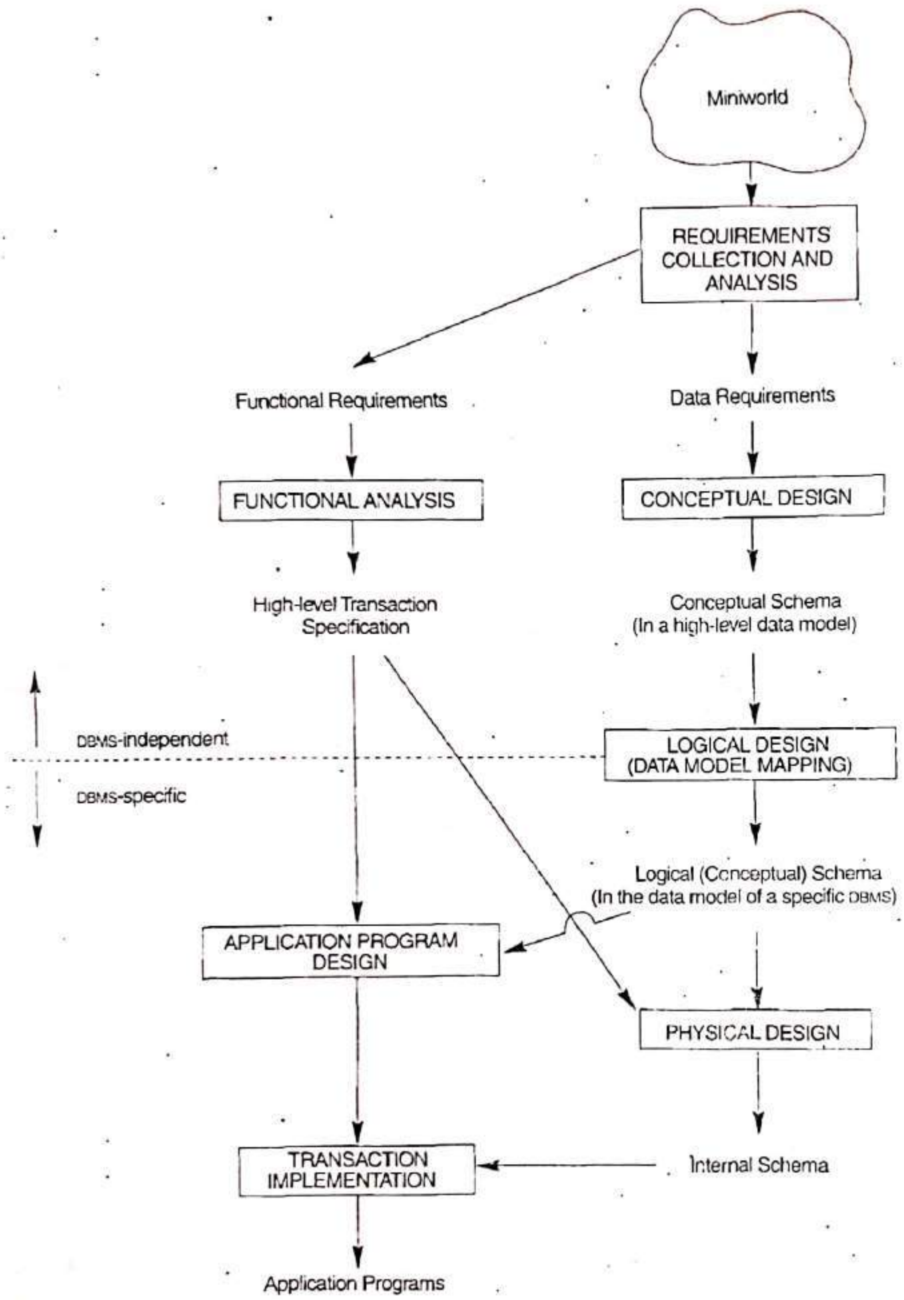


FIGURE 3.1 A simplified diagram to illustrate the main phases of database design.

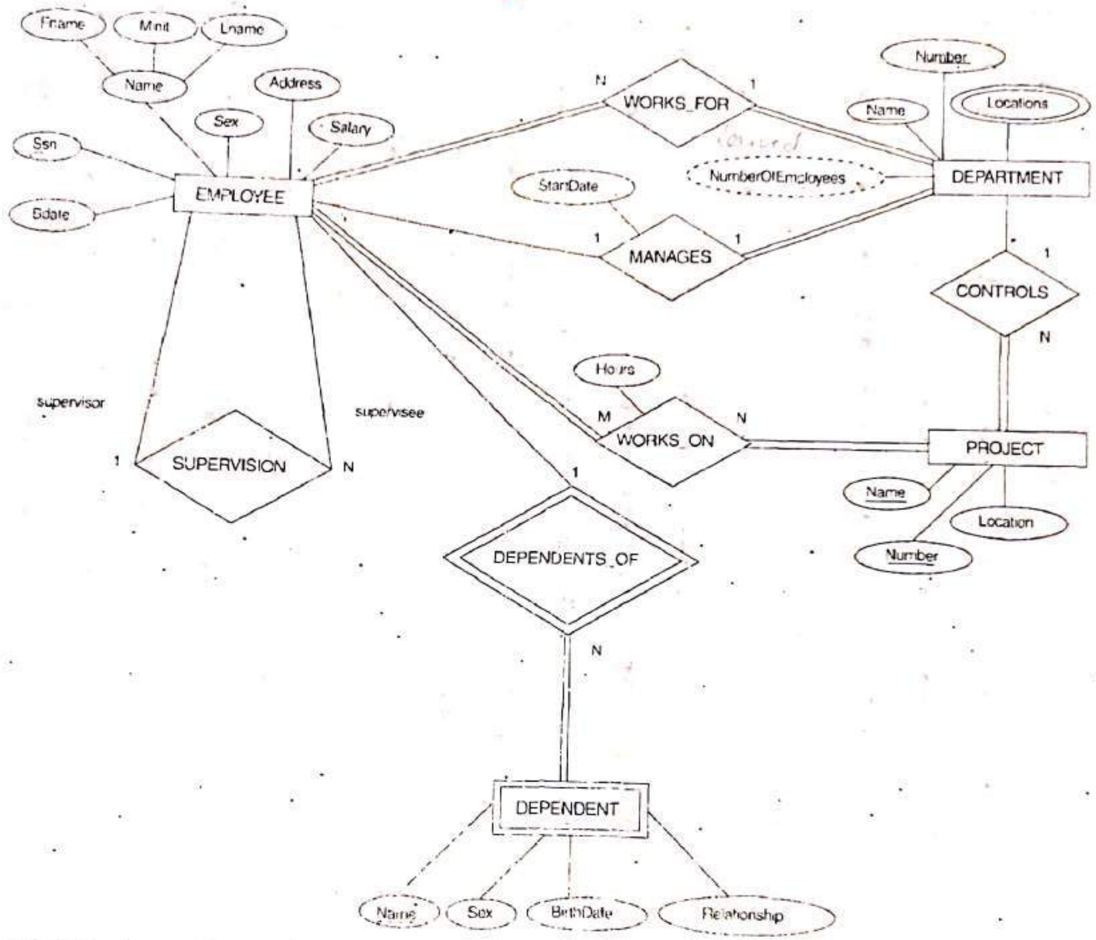


FIGURE 3.2 An ER schema diagram for the COMPANY database.

2. Single-valued Versus Multivalued Attributes:

Attributes that have a single value for a particular entity called single-valued attributes. Eg: Age.

Attributes that have a set of values for the same entity called multivalued attributes. Eg: College Degrees, Phone

3. Stored versus Derived Attributes:

The attribute that can be derived from ~~so~~ other attributes, called derived attribute. The attribute from which it is derived is called stored Attribute.

Eg: Age can be derived from Date of Birth.
↓ derived attribute   ↓ stored Attribute.

4. Null Values:

A particular entity may not have an applicable value for an attribute. eg: ApartmentNo of an address applies only to addresses that are in apartment buildings. (Not Applicable)
Null can also be used if we do not know the value of an attribute for a particular entity - eg: home phone of "Smith".

5. Complex Attributes:

Composite & multivalued attributes can be nested in an arbitrary way. Eg: {AddressPhone({Phone(AreaCode, PhNo)},
Address(StreetAddress(Number, Street, Apart.No) City, State, Zip))}.

Entity Types, Entity Sets, Keys & Value Sets:

~~An entities~~ An entity type defines a collection of entities that have the same attributes. Each entity type in the db is described by its name & attributes. Entities share the same attributes, but each entity has its own value for each attribute

Entity Type Name:	Employee Name, Age, Salary	Company Name, Headquarters, President
Entity set: Extension	e_1. J Smith, 55, 80k e_2. (Fred, 40, 30k) e_3. (Judy, 25, 20k) ⋮	c_1 (Sunco Oil, John Smith) c_2. (Fast, Dallas, Bob) ⋮

An entity type is represented in ER diagrams as a ~~see~~ rectangular box enclosing the entity type name. Attributes names are enclosed in ovals & are attached to their entity type by straight lines. Composite attributes attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

The coll^n of all entities of a particular entity type in the db at any pt. in time is called an entity set. Also called extension of the entity type.

Key Attributes of an Entity Type:

An entity type usually has an attribute whose values are distinct for each individual entity in the coll^n. Such ~~as~~ an attribute is called a key attribute & its values can be used to identify each entity uniquely. For eg: the Name attribute is a key of the Company entity type b/c no two companies are allowed to have the same name. For the Person entity type key attribute is SSN.

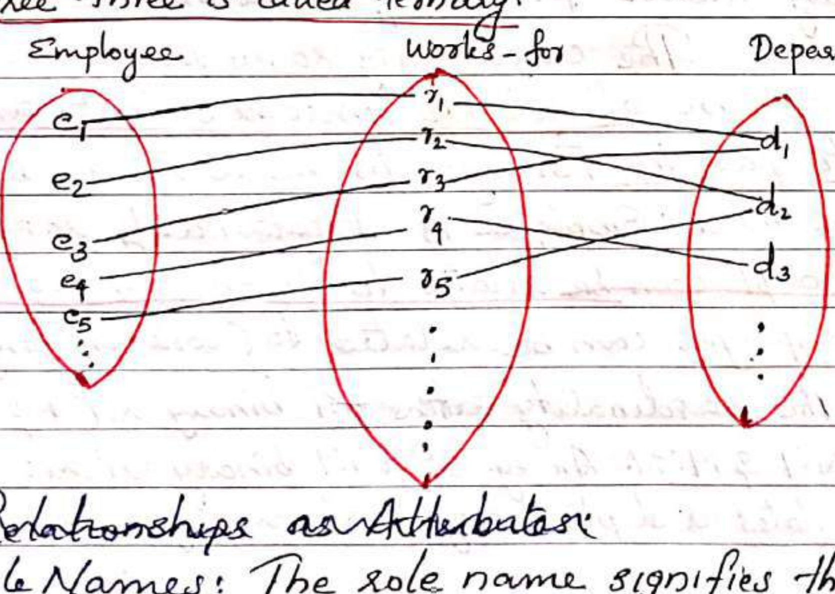
Several attributes together form a key, meaning that the comb^n of the attribute values must be distinct for each entity. If a set of attributes possesses this property its composite attribute that becomes a key attribute of the entity type. In ER diagram, each key attribute has its name underlined inside the oval.

Value Sets (Domains) of Attributes:

Each simple attribute of an entity type is associated with a value set (domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. Eg: The value set for Name attribute as being set of strings of alphabetic characters separated by blank characters & so on. Value sets are not displayed in ER diagrams.

Relationships, Relationship Types, Roles & Structural Constraints.

Relationships: an attribute of one entity type refers to another entity type; Eg: The attribute Manager of Department refers to an employee who manages the dept.
Relationship Types: A relationship type R among 'n' entity types $E_1, E_2 \cdots E_n$ defines a set of associations—or a rel'ship set—among entities from these types.
Relationship instance r_i in R is an association of entities, where association includes exactly one entity from each participating entity type. For eg: a relationship type works-for b/w the 2 entity types Employee & Department entity.
Relationship Degree; is the no. of participating entity types.
Eg: the works-for relationship is of degree two.
A relationship type of degree two is called binary & one of degree three is called ternary.



~~Relationships as Attributes:~~
Role Names: The role name signifies the role that a participating entity from the entity type plays in each rel'ship instance, & helps to explain what the rel'ship means.
Eg: Employee plays the role of employee & Department plays the role of dept. or employer.
Recursive Relationships: In some cases the same entity type participates more than once in a rel'ship type in

diff. roles. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such rel^nship types are called recursive relationships.
Eg: The supervision rel'ship type relates an employee to a supervisor, where both employee & supervisor entities are members of same Employee entity type.

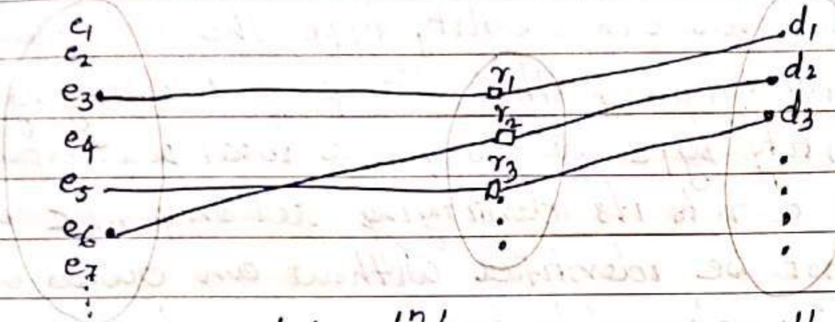
Constraints on Relationship Types:

~~Types of~~ Relationship types have certain constraints that limit the possible comb^n's of entities that may participate in the corres. rel^nship set. ~~For eg: in the Works for rel'ship type, Employee plays the role of employee or worker & Department plays the role of department or employer.~~
There are two main types of relationship constraints:
→ cardinality ratio.
→ participation.

Cardinality Ratios for Binary Relationships:
The cardinality ratio for a binary rel'ship specifies the no. of rel'ship instances that an entity can participate in. For eg:, in the works-for binary rel'ship type, Department : Employee is of cardinality ratio 1:N, ie; each dept can be related to (ie; employs) no. of employees but an employee can be related to (work for) only one dept.
The cardinality ratios for binary rel^nship types are 1:1, 1:N, N:1 & M:N. An eg. of a 1:1 binary rel'ship is MANAGE which relates a dept entity to the employee who manages that dept.

Participation Constraint:
— specifies whether the existence of an entity depends on its being related to another entity via the rel^nship type. There are 2 types of participation constraints — total & partial.

Eg: If a company policy states that every employee must work for a dept, then an employee entity can exist only if it participates in a works-for rel^nship instance. Thus, the participation of Employee in works-for is called total participation ⇒ every entity in "the total set" of employee entities must be related to a dept. entity via works for. Total participation is also called existence dependency.



1:1 rel^nship manages with partial participation of employee & total participation of Department.

Eg: we do not expect every employee to manage a dept, so the participation of employee in the manages rel^nship type is partial, ~~mead~~ meaning that some or part of the set of employee entities are related to a dept. entity via manages, but not necessarily all.

Cardinality ratio & participation constraints together is called structural constraints.

In ER diagrams, total participation as a double line connecting participating entity type to the rel^nship whereas partial is by single line.

Attributes of Relationship Types:

Relationship types can have attributes. For eg: to record no. hours/week that an employee works on a particular project, we can include an attribute Hours for the works-on relationship type.

classmate
Date
Page

* Weak Entity Types:

Entity types that do not have key attributes of their own are called weak entity types. Regular entity types that do have a key attribute are called strong entity types.

Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. We call this other entity type the identifying or owner entity type. & the rel^n^ship is identifying rel^n^ship. A weak entity type always has a total participation constraint w.r.t. its identifying rel^n^ship b/c weak entity cannot be identified without an owner entity.

A weak entity type normally has a partial key which is the set of attributes that can uniquely identify weak entities that are related the same owner entity.

In ER diagrams, both a weak entity type & its identifying rel^n^ship are represented by ~~sou~~ surrounding their boxes & diamonds with double lines. The partial key is underlined with dashed or dotted line.

fig:

* Relationship types of Degree greater than Two:

Relationship type of degree 2 binary & a rel^n^ship type of degree 3 ternary.

→ Choosing b/w Binary & Ternary (or Higher-Degree) Relationships:

A rel^n^ship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type. Fig(a) displays the schema for the supply rel^n^ship type.

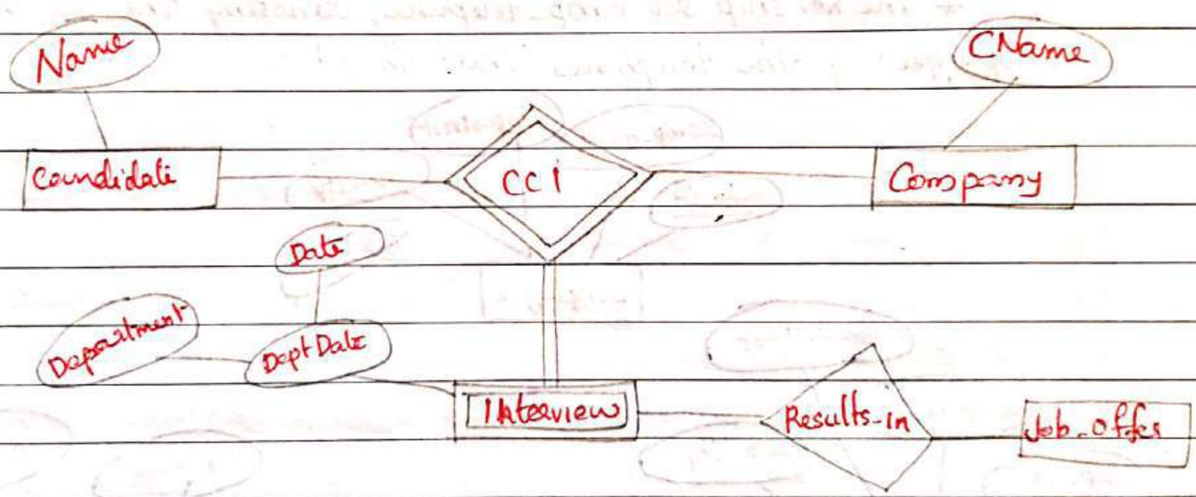
Fig(b) shows an ER diagram for the three binary rel^n^ship types CAN_SUPPLY, USES, and SUPPLIES.

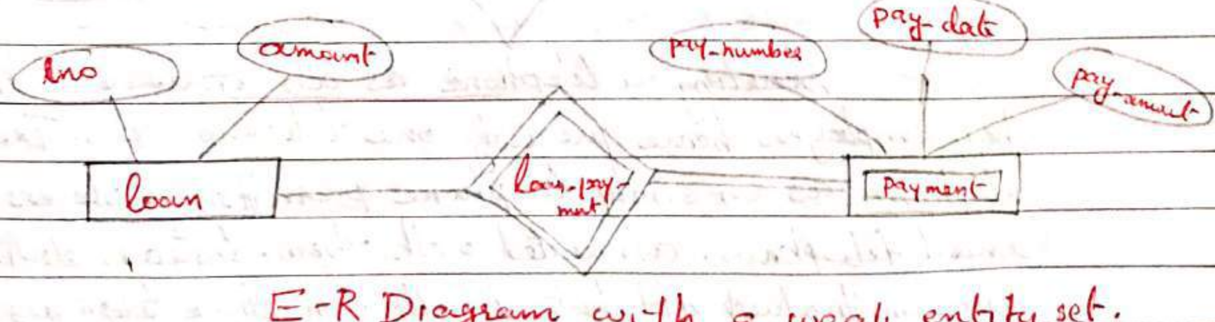
A ternary rel^n ship type represents more inf^n than do three binary rel^n ship types.

Fig(c) The 3 participating entity types SUPPLIER, PART & PROJECT are together the owner entity types. An entity in the weak entity type SUPPLY of fig is identified by the comb^n of its three owner entities from SUPPLIER, PART & PROJECT.

→ Constraints on Ternary (or Higher-Degree) Relationships:

There are two notations for specifying structural constraints on n-ary relationships, & they specify diff. constraints. They should thus both be used if it is important to fully specify the structural constraints on a ternary or higher degree rel^n ship.





E-R Diagram with a weak entity set.

classmate
Date
Page

Entity-Relationship Design Issues:

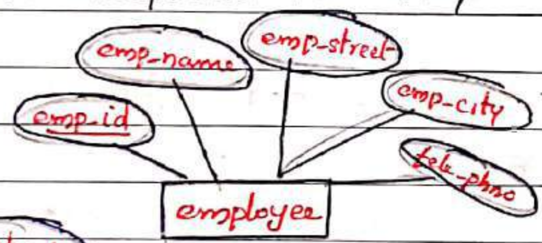
The notions of an entity set & a rel'ship set are not precise, & it is possible to define a set of etities & the rel'ships among them in a no. of diff. ways.

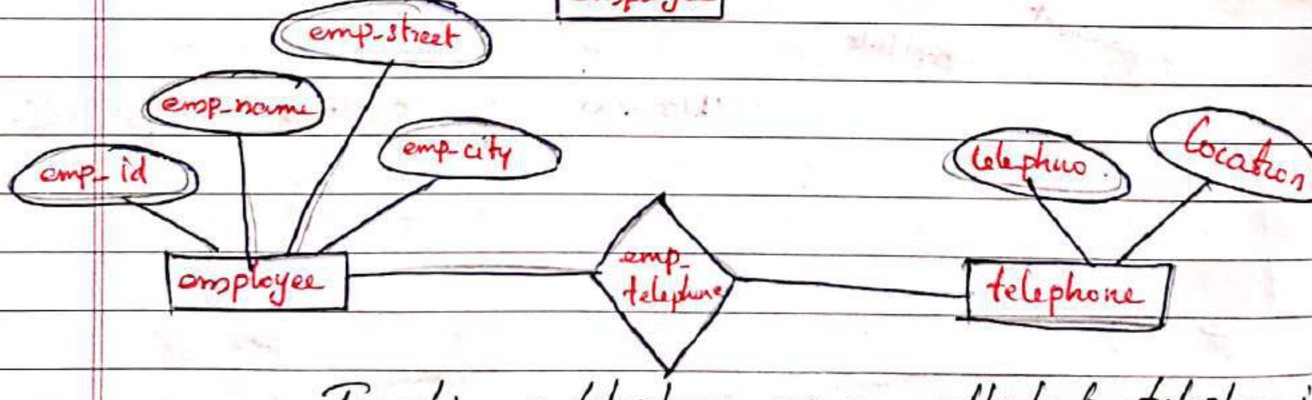
Basic Issues are:

1. Use of Entity Sets versus Attributes:

Consider the entity set employee with attributes emp-id, emp-name & tele-phno. Telephone can be an entity with attributes tele-phno & location (office, home, mobile ...). Redefine the employee entity set as

* The employee entity set with attributes emp-id & emp-name
* The telephone entity set with attributes tele-phno & locn.
* The rel'ship set emp_telephone, denoting the association b/w employees & the telephones that they have.





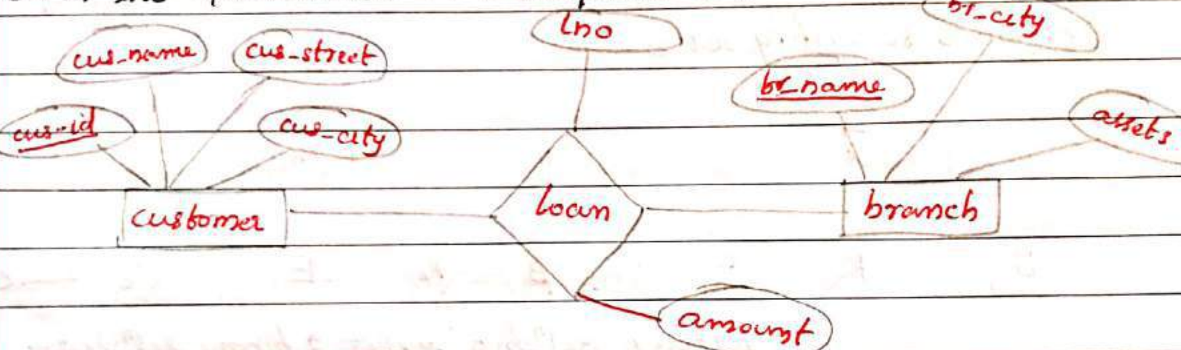
Treating a telephone as an attribute telephno implies that employees have precisely one telephno. each. Treating a telephone as an entity telephone permits employees to have several tele-phnos. associated with them. Instead, define telephno as a multivalued attribute to allow multiple telephones per employee.

The main diff. is that treating a telephone as an entity better models a situation where one may want to keep extra infn about a telephone such as its locn or its type or all who share the telephone

The distinctions b/w attribute & entity set depend on the structure of the real world enterprise being modeled & on the semantics associated with the attribute.

2. Use of Entity Sets versus Relationship Sets:

→ It is not always clear whether an objt is ~~bet~~ expressed by an entity set or a relationship set. For eg: a bank loan is modeled as an entity. An alternative is to model a loan not as an entity, but rather as a rel'ship b/w customers & branches with lno & amount as descriptive attributes.



We cannot represent a situation in which several customers hold jointly. To handle such a situation, we must define a separate rel'ship for each holder of the joint loan. Then we must replicate the values for the descriptive attributes lno & amount in each such rel'ship. Each such rel'ship must have same value for the attributes l-no & amount. (replication)

Two pblms arise as a result of the replication: (1) the data are stored multiple times, wasting storage space (2). updates potentially leave the data in an inconsistent state, where the values differ in two rel'ships for attributes that are supposed to have the same value.

3. Binary versus n-ary Relationship Sets:

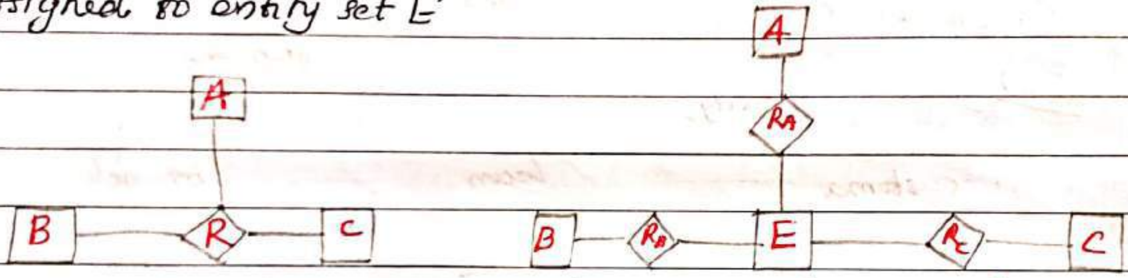
Rel'ships in dbs are often binary. Some rel'ships that appear to be non binary could actually be represented by several binary rel'ships.

For eg: one could create a ternary rel^nship parent, relating a child to his/her mother & father. Such a rel^nship could be represented by two binary rel^nships, mother & father, relating a child to his/her mother & father separately.

Consider the ternary rel^nship set R, relating entity sets A, B, & C. Replace the relat^nship set R by an entity set E, & create 3 rel^nship sets

* R_A, relating E & A * R_B, relating E & B
* R_C relating E & C.

If the rel^nship set R had any attributes, these are assigned to entity set E



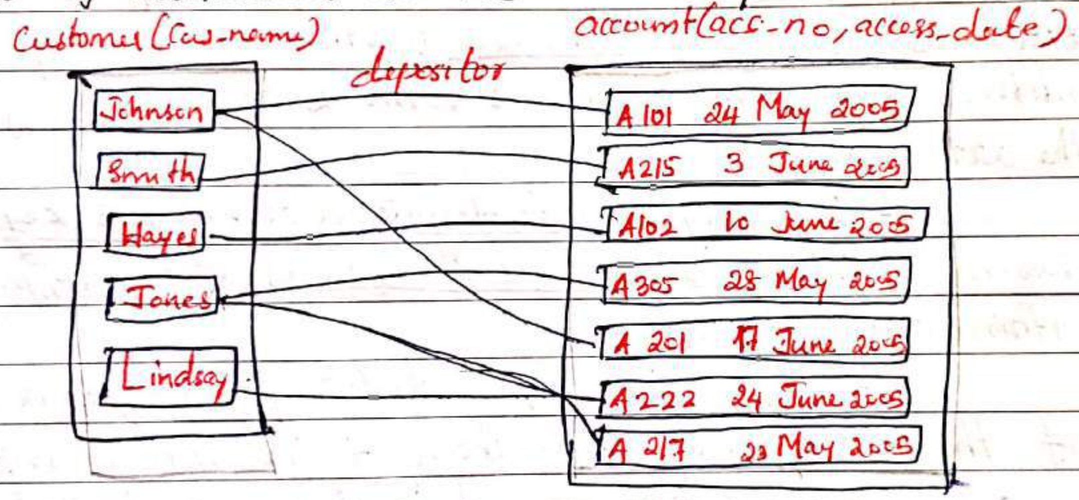
Ternary rel^nship versus 3 binary rel^nships.

The restriction to include only binary relationship set is not always desirable.

4) Placement of Relationship Attributes:

The cardinality ratio of a rel^nship can affect the placement of rel^nship attributes. Attributes of one-to-one or one-to-many rel^nship sets can be associated with one of the participating entity sets, rather than with the rel^nship set. For eg: Depositor is a one to many rel^nship set such that one customer may have several accounts, but each account is held by only one customer. In this case, the attribute access-date, which specifies when the customer last accessed that account, could be associated with the account entity set. Since each account entity participates in a rel^nship with at most one instance of customer, making this attribute designation could have the same meaning as would placing access-date with the depositor rel^nship set. Attributes of one to many rel^nship set can be repositioned to only the entity set on the "many" side of the rel^nship. For one-to-one rel^nship sets, the rel^nship attribute can be associated with either one of the participating entities.

The design decision of where to place descriptive attributes in such cases - as a rel^nship or entity attribute - should reflect the characteristics of the enterprise being modeled. The designer may choose to retain access-date as an attribute of depositor to express explicitly that an access occurs at the pt. of interaction b/w the customer & account entity sets.



Keys - Constraints.

Entities within a given entity set ~~are~~ be distinguished. The values of the attribute values of an entity must be such that they can uniquely identify the entity. A key allows ~~us~~ to identify a set of attributes to distinguish entities from each other. Keys also help uniquely identify relationships, & distinguish rel^nships from each other.

1. Entity Sets:

A superkey is a set of one or more attributes allow to identify uniquely an entity in the entity set. For eg: the cus-id attribute is sufficient to identify uniquely an entity in the entity set.

For eg: the cus-id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus cus-id is a superkey. The combination of cus-name & cus-id is a superkey for the entity set customer. The cus-name attribute of customer is not a superkey, b/c several people might have same name.

If K is a superkey, then so is any superset of K. Minimal superkeys are called candidate keys.

A combⁿ of cus-name and cus-street is sufficient to distinguish among members of the customer entity set. Both {cus-id} & {cus-name, cus-street} are candidate keys. Although the attributes cus-id & cus-name together can distinguish customer entities, their combⁿ doesn't form a candidate key, since the attribute cus-id alone is a candidate key.

— Primary key to denote a candidate key that is chosen by the db designer as the means of identifying entities within an entity set.

— A key (primary, candidate & super) is a property of the entity set, rather than of individual entities.

- Primary key should be chosen such that its attributes are never, or very rarely changed.

2. Relationship Sets:

— mechanism to distinguish among the various relⁿships of a relⁿship set. The composition of the primary key for a relⁿship set depends on the set of attributes associated with the relⁿship set R. If the relⁿship set depends on the set of attributes associated with it, then the set of attributes

primary_key(E_1) ∪ primary_key(E_2) ∪ --- primary

describes an individual relⁿship in set R.

If the relⁿship set R has attributes $a_1, a_2 --- a_m$ associated with it, then the set of attributes

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \dots \cup primarykey(E_n) \cup \{a_1, a_2, \dots, a_m\}$$

describes an individual rel^{n}ship in set R.

In both of the abone cases, the set of attributes

$$primary\_key(E_1) \cup primary\_key(E_2) \cup \dots \cup primary\_key(E_n).$$

forms a superkey for the rel^{n}ship set.

The structure of the primary key for the rel^{n}ship set depends on the mapping cardinality of the rel^{n}ship set. Consider the entity sets customer and account, and the rel^{n}ship set depositor, with attribute access-date. Suppose that the rel^{n}ship set is many to many. Then the primary key of depositor consists of the union of the primary keys of customer & account. If a customer can have only one account - ie; if the depositor relationship is many to one from customer to account - then the primary key of depositor is primary key of customer. If the rel^{n}ship is many to one from account to customer - ie; each account is owned by at most one customer then the primary key of depositor is primary key of account. For one to one relationships either primary key can be used.

24/1/18

salesperson. A customer can place any no. of orders. An order can be placed by exactly one customer. Each order lists one or more items. An item may be listed in many orders. An item is assembled from diff. parts & parts can be common for many items. One or more employees assemble an item from parts. A supplier can supply diff. parts in certain quantities. A part may be supplied by diff. suppliers.

1) Identify and list entities, attributes, primary keys & relⁿships to represent the scenario.

2.) Draw an ER diagram to model the scenario. min-max notation.

5. Justify the importance of weak entity sets with the help of an eg:

6. Consider a Relation R(A,B,C,D) where A is a key: R. Write any 3 relⁿal algebra expressions equivalent to $\pi_{A,B}(\sigma_{A=2 \text{ and } B=3}(R))$

~~7. Study~~ Supplementary Questions: July 2017

1. What are the responsibilities of the DBA?
2. Define the following terms:
   i) Data model ii) Database schema iii) Meta-data.
3. What are diff. ways of classifying a DBMS?
4. Explain the follo. terms :
   i) Participation Constraint ii) Overlap constraint iii) Covering Constraint.
5. Explain Three Schema Architecture with diagram.

classmate
Date
Page

MODULE II

> Relational Model
  * Structure of relational dbs.
  * Integrity Constraints
  * Synthesizing ER diagram to relational schema

> Database Languages:
  * Concept of DDL & DML
  * Relational Algebra.

Questions:

1. Why are tuples in a relation not ordered?
2. Why are duplicate tuples are not allowed in a relation?
3. What is the diff. b/w a key & a superkey?
4. Discuss the entity integrity & referential integrity constraints?
5. Define foreign key. How does it play role in the join Operation?
6. List the opns of relational algebra & purpose of each.

Previous Questions June 2017

1. How is DML diff. from DML? Write a sample stmt in DML & one in DDL?
2. Consider the relation R(A,B,C,D) where A is a key of R. Write any 3 relnal algebra expressions equivalent to $\pi_{A,B}(\sigma_{A=2 \text{ and } B=3}(R))$
3. Study the tables given below & write relational algebra expressions for the queries.

Student(RollNo, Name, Age, Gender, Address, Advisor)
Course(CourseId, CName, credits)
Professor(Profid, PName, Phone)
~~Enro~~ Enrollment(RollNo, Courseid, Grade)
Primary keys are underlined, Advisor is a foreign key refering to professor table. RollNo & Courseid in Enrollment

Scanned with CamScanner

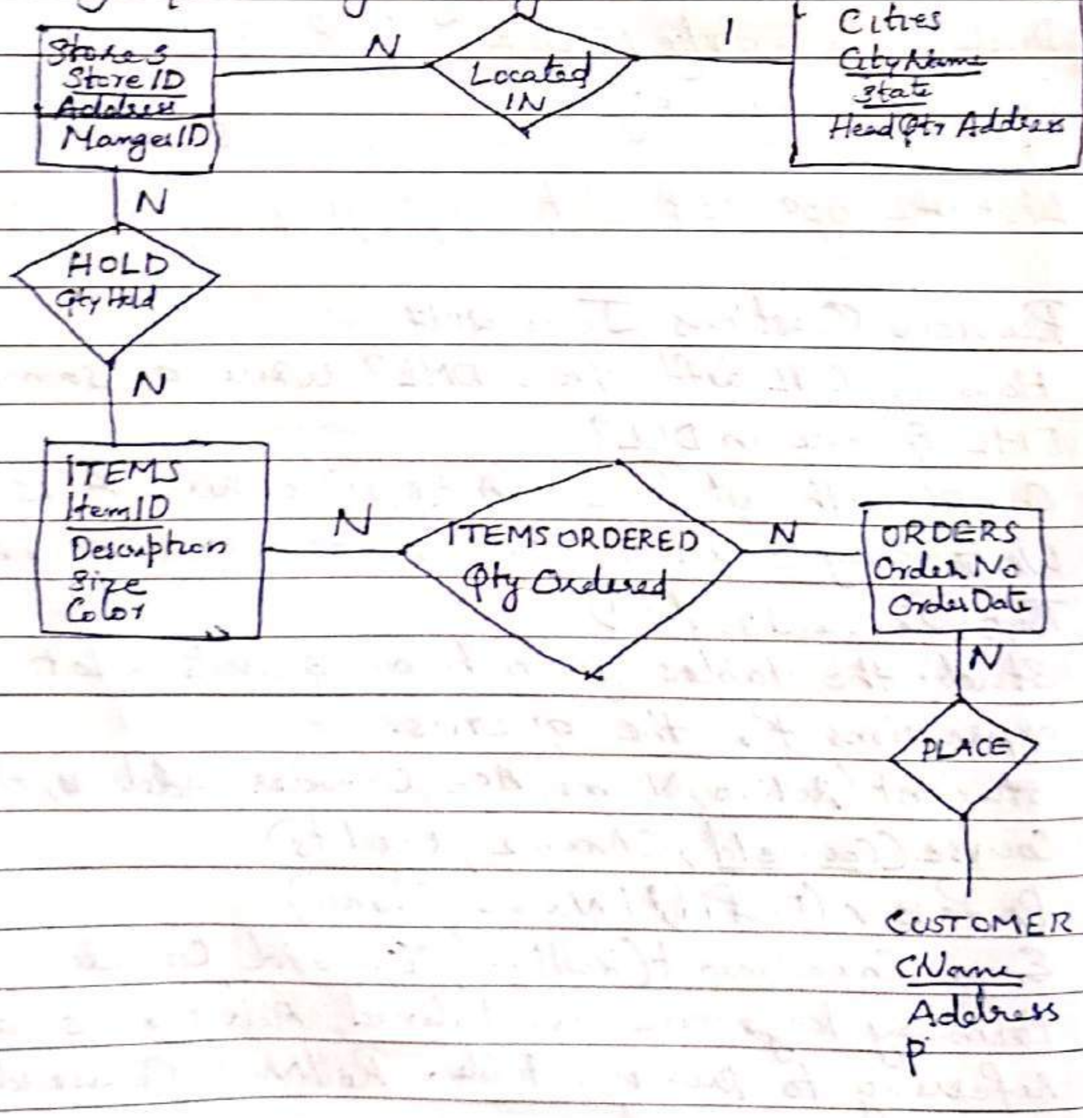
are the foreign keys referring to the primary key with the same name.

1) Names of female students.

2) Names of female students with advisor name

3) Roll number & name of students who have not enrolled for any course.

4: In the ER diagram, names of entity sets & relationships are shown in capital & corresponding attributes are listed under each such name. Key attributes are underlined. All participations are total. Use the std synthesis procedure to convert the ER diagram into the corresponding relational schema. Show primary & foreign keys.



> Relational Model

Relational model is the primary data model for commercial data-processing appl^ns.

* Structure of Relational Databases:

A relational db ~~is~~ consists of a coll^n of tables, each of which is assigned a unique name. A row in a table represents a rel^nship among a set of values. Informally, a table is an entity set, & a row is an entity.

→ Basic Structure:

Attributes: column headers eg: AccNo, B name, balance

Domain of Attribute: set of permitted values. eg: AccNo={A101, A-102, A-201...}

Eg:

AccNo	Bname	balance
A-101	DTown	500
A-102	Pridge	400
A-201	Bton	900
A-215	Mianus	700

Account Relation

Let D_1 denote the set of all AccNo, D_2 the set of all Bnames & D_3 the set of all balances. Any row of account must consist of a 3-tuple (v_1, v_2, v_3) where v_1 - AccNo, v_2 - Bname, v_3 - balance

Account will contain only a subset of the set of all possible rows.

$$D_1 \times D_2 \times D_3$$

ie; a table of n attributes must be a subset of

$$D_1 \times D_2 \times D_3 \cdots \times D_n.$$

Consider the Account relation, let the tuple variable t refer to the first tuple of the relation; t[AccNo] = "A-101", & t[Bname] = Dtown

t[1] denote the value of tuple t on the first attribut (AccNo), t[2] to denote Bname & so on.

A rel^n is a set of tuples, we use notation of $t \in r$ to denote that tuple t is in rel^n r.

→ The order in which tuples appear in relation is irrelevant, since a rel^n is a set of tuples.

→ For all rel^ns r, the domains of all attributes of r be atomic. A domain is atomic if elmts of the domain are considered to be indivisible units. For eg: the set of integers is an atomic domain, but the set of all sets of integers is a nonatomic domain.

→ Several attributes have the same domain. For eg: a relation customer that has 3 attributes Cus-name, cus_street & cus_city & a rel^n employee that includes the attribute Ename. The attributes Cus-name & Ename will ~~th~~ have the same domain.

→ One domain value that is a member of any possible domain is the null value, which signifies that the value is unknown or doesn't exist. For eg: the attribute telephno in the customer rel^n, a customer doesn't have a telephno or the telephno is unlisted.

*Database Schema:

- logical design of the db where db instance is a snapshot of the data in the db at a given instant in time.

Eg: Account_schema = (AccNo, Bname, Balance)

We denote the fact that account is a rel^n on Account Schema by account(Account_schema)

A rel^n instance corres. to ~~the~~ value of a variable. The value of a given variable may change with time; the conts. of a rel^n instance may change with time as the rel^n is updated.

Branch.schema = (Bname, Bcity, assets)

* Keys:

~~H~~ how tuples within a given relⁿ are distinguished. The values of the attribute values of a tuple must be such that they can uniquely identify the tuple.

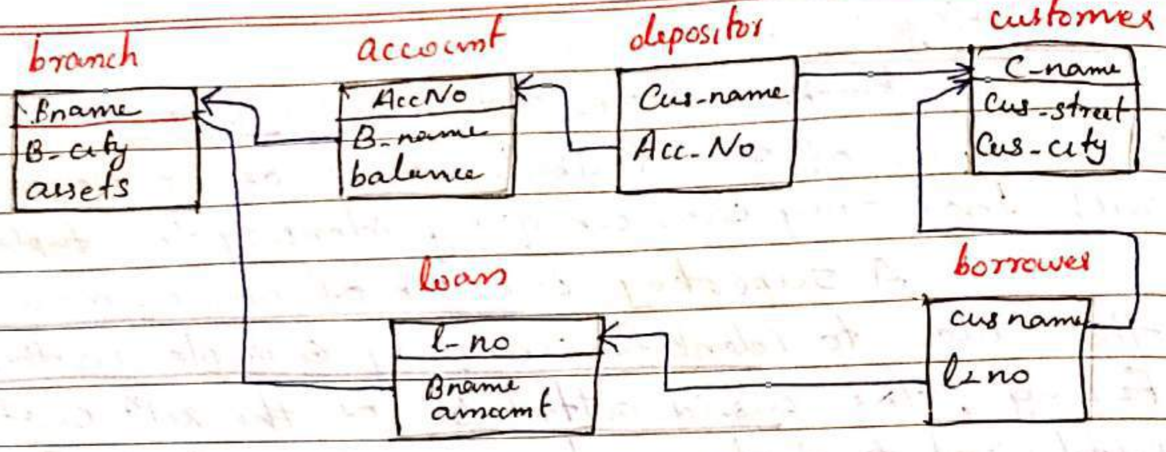
A superkey is a set of one or more attributes that allow to identify uniquely a tuple in the relation. For eg:, the cus-id attribute of the relⁿ customer is sufficient to distinguish one customer tuple from another. Illy, Combⁿ of cus-name & cus-id is a superkey for the relⁿ customer. The cus-name is not a superkey b/c several people might have the same name.

Superkey may contain extraneous attributes. If K is a superkey, then so is any superset of K. Minimal superkeys are called candidate keys.

It is possible that several distinct set of attributes could serve as a candidate key. A combⁿ of cus-name & cus-street is sufficient to distinguish among members of the relⁿ. Both ~~cust~~ {cus-id} and {cus-name, cus-street} are candidate keys. The attributes cus-id & cus-name together can distinguish customer tuples, their combⁿ doesn't form a candidate key, since the attribute cus-id alone is a candidate key.

Primary key denote a candidate key that is chosen by the db designer as the principal means of identifying tuples within a ~~tt~~ relⁿ.

A relⁿ schema r_1, may include among its attributes the primary key of another relⁿ schema r_2. This attribute is called a foriegn key from r_1 referencing r_2. The relⁿ r_1 is also called referencing relⁿ of the foriegn key dependency & r_2 is called the referenced relⁿ of the foreign key.



A db schema, along with primary key & foreign key dependencies can be depicted pictorially by schema diagrams. If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line in gray. Foreign key dependancies appear as arrows from the foreign key attributes of the referencing relⁿ to the primary key of the referenced relⁿ.

* Query Languages:

– is a lang. in which a user rqsts infⁿ from the db. Query lang. can be categorized as either procedural or non procedural. In a procedural lang., the user instructs the s/m to perform sequence of op's on the db to compute the desired result. In a non procedural lang, the user describes the desired infⁿ without giving a specific procedure for obtaining that infⁿ.

Eg: Relational Algebra : procedural

Domain Relational Calculus } non procedural.
Tuple Relational Calculus }

* Integrity Constraints:

- ensures that changes made to the db by authorized users do not result in a loss of data consistency.

Eg:- An account balance cannot be null.

- no 2 accounts can have the same acc.no.

- Every account no in the depositor relⁿ must have a matching account no in the account relⁿ.

- The hourly salary of a bank employee must be atleast $6.00 an hour.

⟹ Constraints on Single Relation:

- Primary key constraint

- Key constraints & entity integrity constraints are specified on individual relⁿs.

* not null
* UNIQUE
* check(<predicate>)

NOT NULL Constraint:

— null value is a member of all domains, & is a legal value for every attribute. For certain attributes, null values may be inappropriate. Consider a tuple in the account relⁿ where Acc-no is null. Such a tuple gives account infⁿ for an unknown account.

- The not null specificⁿ prohibits the insertⁿ of a null value for this attribute.

UNIQUE Constraint:

The unique specifⁿ says that attributes $A_{j1}, A_{j2}, \ldots A_{jm}$ form a candidate key; ie; & no two tuples in the relⁿ can be on all the primary key attributes.

The CHECK Clause:

can be applied to relⁿ declarations as well as to domain declarations. Clause check(P) specifies a predicate P that must be satisfied by every tuple in a relⁿ.

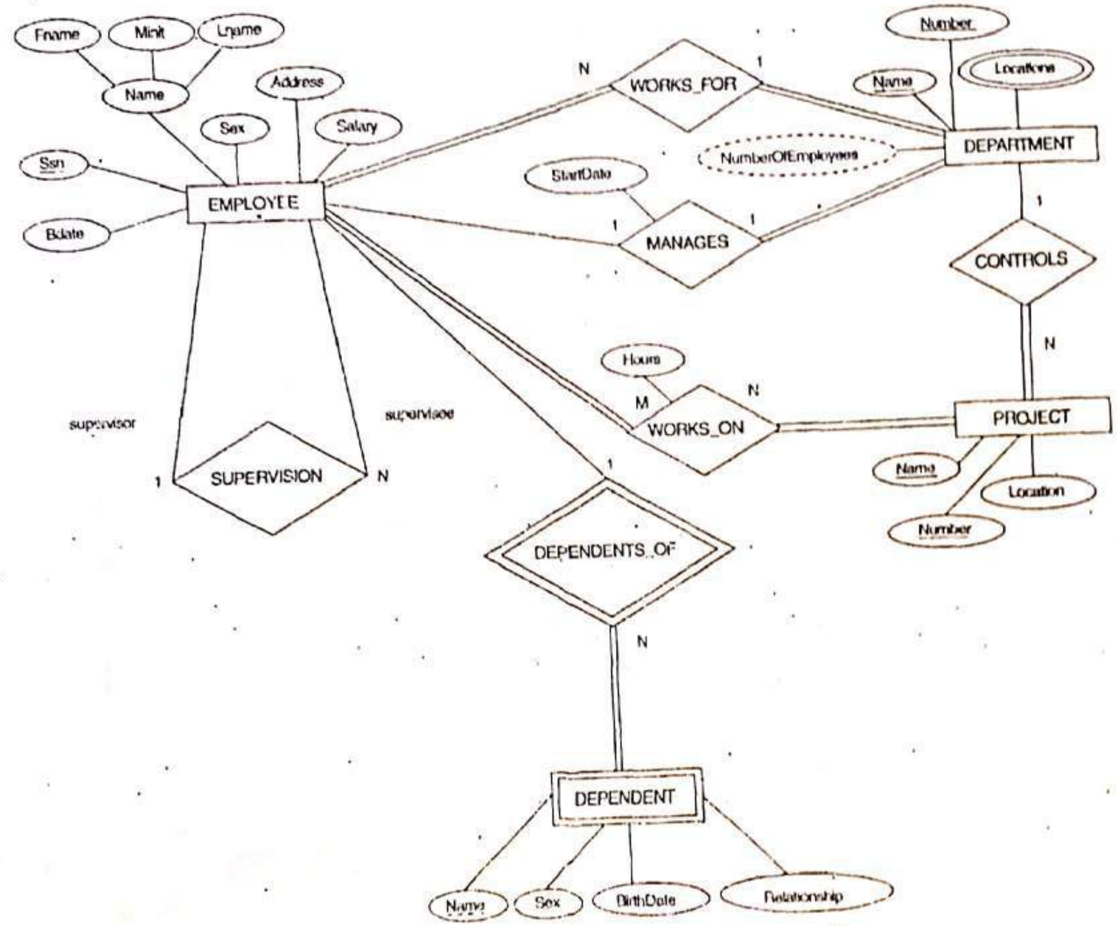
For eg:, a clause check (assets>=0) in the create table cmd for relⁿ branch would ensure that the value assets is nonnegative.

Key constraints & entity integrity constraints are specified on individual relⁿs. The ref integrity constraint is specified b/w 2 rel's & is used to maintain the consistency among tuples in the 2 relⁿs. The referential integrity constraint states that a tuple in one relⁿ that refers to another relⁿ must refer to an existing tuple in that relⁿ.

* → referencing relⁿ * → referenced relⁿ.

Eg: fig 5.7

Entity Integrity constraint states that no primary key value can be null. B/c primary key value is used to identify individual tuples.



IGURE 7.1 The ER conceptual schema diagram for the COMPANY database.

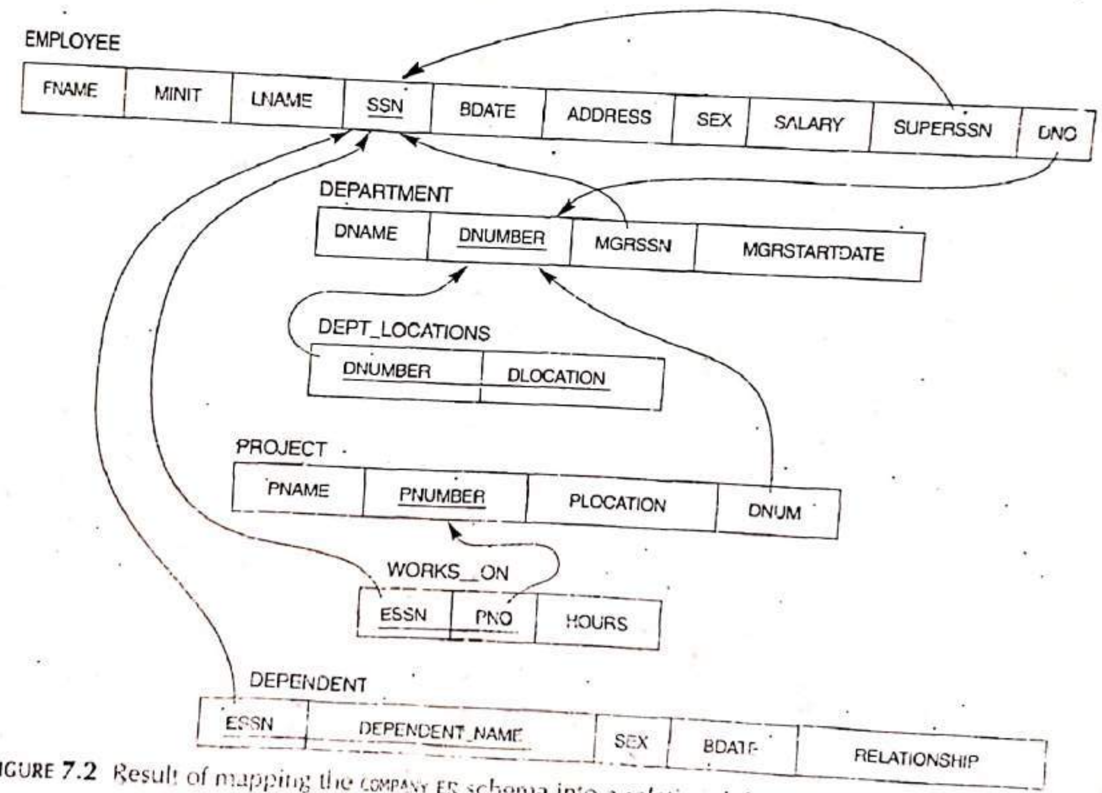


FIGURE 7.2 Result of mapping the COMPANY ER schema into a relational database schema.

* Synthesizing E-R diagram to relational Schema.

Fig 7.1 & 7.2

⇒ Mapping steps:

Step 1: Mapping of Regular Entity Types: For each regular (strong) entity type E in ER schema, create a rel^n R that includes all the simple attributes of E. Include only simple component attributes of a composite attribute. Choose one of the key attributes of E as primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

In eg: create the relations Employee, Department & Project to correspond to the regular entity types Employee, Department & Project. We choose SSN, DNo & PNo as primary keys for the relations.

Step 2: Mapping of Weak Entity Types: For each weak entity type W in the ER schema with owner entity type E, create a rel^n R & include all simple attributes (or all simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R the primary key attributes of the rel^n(s) that correspond to the owner entity type(s).

In eg:, we create the rel^n dependent, we include primary key SSN of the Employee relation - which corresponds to the owner entity type - as a foreign key attribute of dependent, renamed it ESSN. The primary key of the dependent rel^n is the combination {ESSN, dependent_name} b/c dependent_name is the partial key of dependent.

Step 3: Mapping of ~~1:1~~ 1:1 relationship: For each binary 1:1 rel^nship type R in the ER schema, identify the rel^ns S & T that correspond to the entity types participating in R.

Choose one of the rels - S ~~& T that correspond to the entity types participating in R~~ & include as foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes of the 1:1 relship type R as attributes of S.

For eg: map 1:1 relship type MANAGES by choosing the participating entity type Department to serve in the role of S, b/c its participation in the MANAGES relship type is total (every dept has a manager). Include the primary key of the employee rel as foreign key in the Department rel & rename it MGRSSN. Also include the simple attribute StartDate of the MANAGES relship type in the Department rel & rename it MgrStartDate.

Step 4: Mapping of 1:N relationship types: Identify the rels S that represents the participating entity type at the N-side of the relship type. Include primary key of the rel T as foreign key in S. Include any simple attributes of the 1:N relship type as attributes of S.

For eg: map the 1:N relship types works for, controls & supervision. For works for include the primary key DNo of Department relation as foreign key in the Employee relation & call it DNumber.

Step 5: Mapping of M:N Relationship type:

~~For each bi~~ create a new relation S to represent R. Include primary keys of the relations that represent the participating entity types as foreign key attributes in S. Include any simple attributes of M:N relship type as attributes of S. We cannot represent an M:N relship type by a single foreign key attribute in one of the participating rels b/c of M:N cardinality ratio.

Eg: Map the M:N rel^n^ship type works-on by creating the rel^n works-on. Include the primary keys of Project & Employee rel^ns as foreign keys in works-on & rename PNO & ESSN. Also include an attribute Hours in works-on to represent hours attribute of the rel^n^ship type. The primary key of the works-on rel^n is the comb^n of the foreign key attributes {ESSN, PNO}.

Step 6: Mapping of Multivalued Attributes: For each multivalued attribute A, create a new relation R. This rel^n R will include an attribute corresponding to A, plus primary key attribute K – as foreign key in R – of the rel^n that represents the entity type that has A as an attribute. The primary key of R is the comb^n of A & K. If multivalued attribute is composite, include its simple components.

- For eg: create a rel^n Dept-Locations. The attribute DLocation represents the multivalued attribute Locations of Department, while DNo – as foreign key – represents the primary key of the Department relation. The primary key of Dept-Locations is the combination of {DNo, DLocation}. A separate tuple will exist in Dept-Locations for each loc^n that a dept has.

Step 7: Mapping of n-ary Relationship: Create a new rel^n S to represent R. Include as foreign key attributes in S the primary keys of the rel^n that represent participating entity types. Also include ~~as foreign key~~ any simple attributes ~~in S~~ of the n-ary rel^n^ship as attributes of S.

> Database Languages.

A db s/m provides a data defⁿ lang. to specify the db schema & a data-manipulation lang. to express db queries & updates.

* Data Manipulation Language:

— enables users to access or manipulate data as organized by the appropriate data model. The types of access are

- Retrieval of infⁿ stored in the db.
- Insertion of new infⁿ into the db.
- Deletion of infⁿ from the db.
- Modificⁿ of infⁿ stored in the db.

There are basically 2 types:

* Procedural DMLs: require a user to specify what data are needed & how to get those data.

* Declarative DMLs: (nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

A query is a stmt requesting the retrieval of infⁿ. The portion of a DML that involves infⁿ retrieval is called a query lang.

Query lang, eg: SQL.

* Data Definition Language:

— Specify a db schema by a set of defⁿ's expressed by a special lang. called a data defⁿ lang. (DDL)

We specify the storage structure & access methods used by the db s/m by a set of stmts in a special type of DDL called a data storage & defⁿ lang.

The data values stored in the db must satisfy certain consistency constraints. For eg: suppose the bal. on an account should not fall below $100. The DDL provides facilities to specify such constraints.

The db s/m concentrate on integrity constraints that can be tested with minimal overhead:

* Domain Constraints: A domain of possible values must be associated with every attribute (eg: integer types, char types date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of the integrity constraint.

* Referential Integrity: A value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another rel^n. Database modific^ns can cause violations of referential integrity.

* Assertions: An assertion is any cond^n that the db must always satisfy. Domain constraints & referential integrity constraints are special forms of assertions. For eg: "Every loan has at least one customer who maintains an account with a min. bal of $100!" must be expressed as an assertion. When an assertion is created, the s/m tests it for validity. If the assertion is valid, then any future modific^n to the db is allowed only if it doesn't cause that assertion to be violated.

* Authorization: To differentiate among the users as far as the type of access they are permitted on various data values in the db. - read authorization, update authorization, delete authorization

The o/p ~~ow~~ of the DDL is placed in the data dictionary, which contains metadata ie; data about data.

* Relational Algebra

> Relational Algebra Operations:
- → unary operations – select, project & rename
- → binary operations – Union, Difference, Cartesian Pdt

* The Select Operation:

– selects tuples that satisfy a given predicate. 'σ' is used to denote selection. The predicate appears as a subscript to σ. Eg: select those tuples of the loan relation where the branch is "Pridge"

$\sigma_{bname = \text{"Pridge"}}(loan)$

$\sigma_{amount > 1200}(loan)$

– Allow comparisons using $=, \neq, <, \leq, >, \geq$ in the selection predicate.

– combine several predicates into a larger predicate by using the connectives and ($\wedge$), or ($\vee$) & not ($\neg$).

$\sigma_{bname = \text{"Pridge"} \wedge amount > 1200}(loan)$.

– include comparisons b/w 2 attributes.

Eg: $\sigma_{cusname = bname}(loan\text{-}officer)$

* Project Operation:

– list values of selected attributes.

– The project opn allows us to produce ~~this~~ reln

– This opn returns its arg. reln, with certain attributes left out. Since a reln is a set, any duplicate rows are eliminated.

– denoted by π. ⇒ list those attributes that we wish to appear in the result as a subscript to π. The arg. relation follows in paranthesis.

$\pi_{lno,\, amount}(loan)$

lno	amount
L-11	900
L-14	1500
L-15	1500
L-17	1000

* Composition of Relational Operations:

Eg: Find those customers who live in Harrison".

$\pi_{cname}(\sigma_{cus.city = \text{"Harrison"}}(customer))$

instead of giving the name of a relⁿ as the arg. of the projection opⁿ, we give an expression that evaluates to a relⁿ. Since the result of a relⁿal algebra opⁿ is the same type as its inputs, relational algebra opⁿs can be composed together into a relational algebra expression. Composing relⁿal algebra opⁿs into relational algebra expressions is just like composing arithmetic opⁿs (+, -, *, ÷) into arithmetic expressions.

* The Union Operation:

Eg: ~~to~~ Find the names of all bank customers who have either an account or a loan or both.

~~$\pi_{customer}$~~ $\pi_{cusname}(borrower)$

$\pi_{cus\text{-}name}(depositor)$

To answer the query, we need the union of these 2 sets:

$\pi_{cusname}(borrower) \cup \pi_{cus\text{-}name}(depositor)$

For a union opⁿ $r \cup s$ to be valid, we require that 2 condⁿs hold:

1. The relⁿs r & s must be of the same arity. ie; they must have the same no. of attributes.
2. The domains of the i^{th} attribute of s must be the same, for all i.

Cus-name
Adams
Curry
Hayes

* The Set-Difference Operation:

'−' allows us to find tuples that are in one relⁿ but are not in another. The exp. $r - s$ produces a relⁿ containing those tuples in r but not in s.

Eg: Find all customers of the bank who have an account but not a loan.

$\Pi_{cusname}(depositor) - \Pi_{cusname}(borrower)$.

Cus name
John
Lindsay
Turner

The set-difference opⁿ $r-s$ to be valid, we require that the relⁿs r & s be of the same arity, & that the domains of the i^{th} attribute of r & the i^{th} attribute of s be the same.

* The Cartesian-Product Operation:
 - denoted by $(\times)$, allows us to combine infⁿ from any two relⁿs, $r_1 \times r_2$.

 For eg: the relⁿ schema for r = borrower x loan is (borrower.cus_name, borrower.l-no, loan.l_no, loan.b-name, loan.amount)

* Set Intersection Operation: The result of this opⁿ denoted by $R \cap S$ is a relⁿ that includes all tuples that are in both R & S. Consider relⁿs students & instructors. The result of the intersection opⁿ includes only those who are both students & instructors.

→ UNION & INTERSECTION are commutative Opⁿs.

$R \cup S = S \cup R$ & $R \cap S = S \cap R$.

→ Associative: $R \cup (S \cup T) = (R \cup S) \cup T$

: $(R \cap S) \cap T = R \cap (S \cap T)$.

Scanned with CamScanner

The JOIN Operation:

— denoted by $\bowtie$, used to combine related tuples from 2 rel'^ns into single tuples. It allows to process rel'^nships among relations. Eg: To get the mngr's name, need to combine each dept tuple with the employee tuple whose SSN value matches the MgrSSN value in the dept tuple.

$$Dept\_MGR \leftarrow Department \bowtie_{MgrSSN = SSN} Employee$$
$$Result \leftarrow \pi_{Dname, Lname, Fname}(Dept\_Mgr)$$

— A general join condition is of the form:

<condition> and <condition> and <condition> --- and <condition>.

where each condⁿ is of the form $A_i \theta B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i & B_j have the same domain and θ is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$.
A join operation with such a general condⁿ is called a Theta JOIN.

Equijoin involves join cond'^ns with equality comparisons only. (Only = operator is used).

Natural Join: requires that the 2 join attributes have the same name in both relations. denoted by $*$.
If this is not the case, a renaming opⁿ is applied first.
Eg: first rename the DNumber attribute of Department to DNUM — so that it has the same name as the DNUM attribute in PROJECT — then apply Natural Join.

$$PROJ\_DEPT \leftarrow PROJECT * \rho_{(DNAME, DNUM, MGRSSN, MGRSTARTDATE)}(Department)$$

If the attributes on which the natural join is specified have the same names in both rel'^ns, renaming is unnecessary. For eg: to apply a natural join on the DNumber attributes of Department & Dept_Locations

$$Dept\_Locs \leftarrow Department * Dept\_Locations.$$

* The DIVISION Operation:

Eg: "Retrieve the names of employees who work on all the projects that John Smith works on".

Step 1: Retrieve the list of project no-s that 'John Smith' works on in the intermediate relⁿ Smith_PNOs.

$SMITH \leftarrow \sigma_{FNAME='John' \text{ AND } LNAME='Smith'}(Employee)$

$SMITH\text{-}PNOS \leftarrow \pi_{PNO}(WORKS\_ON \bowtie_{ESSN=SSN} SMITH)$

Step 2: Create a relation that includes a tuple <PNO, ESSN> whenever the employee whose social Security Number is ESSN works on the project whose no. is PNO in the intermediate relⁿ SSN-PNOs:

$SSN\_PNOs \leftarrow \pi_{ESSN, PNO}(Works\text{-}on)$

Step 3: Apply division Operation to the 2 relⁿs which gives the desired employees' social security numbers

$SSNS(SSN) \leftarrow SSN\text{-}PNOS \div SMITH\text{-}PNOS$

$Result \leftarrow \pi_{FName, LName}(SSNS * Employee)$

SSNPNOS

ESSN	PNO
789	1
789	2
444	3
453	1
453	2
555	2
455	3

Smith PNOS

PNO
1
2

SSNS

SSN
789
453

$SSNS \leftarrow SSNPNOS \div SmithPNOS$

R

A	B
a_1	b_1
a_2	b_1
a_3	b_1
a_4	b_2
a_1	b_2
a_3	b_3
a_1	b_4
a_2	b_4
a_3	b_4

S

A
a_1
a_2
a_3

$T \leftarrow R \div S$

T

B
b_1
b_4

Aggregate Functions & Grouping.

→ aggregate functions on collections of values from the db. Eg: SUM, AVERAGE, MAXIMUM & COUNT & MINIMUM.

→ Grouping the tuples in a rel^n by the value of some of their attributes & then applying an aggregate fn to each group.

Aggregate function can be defined as the symbol $\mathcal{F}$ to specify these rqsts as follows.

<grouping attributes> $\mathcal{F}$<function list> (R). where <grouping attributes> is a list of attributes of the relation specified in R & <fn list> is a list of (<function> <attribute>) pairs.

Eg: Retrieve each department number, the no. of employees in the dept, & their avg salary, while renaming the resulting attributes as indicated below.

$\rho_{R(DNo,\ No.of employees,\ Average\ sal)}$ (DNo $\mathcal{F}$ Count SSN, AVERAGE salary(Employee))

Result:

R DNo	No. of Employees	Average Sal
5	4	33250
4	3	31000
1	1	55000

Eg: DNo $\mathcal{F}$ Count SSN AVERAGE salary (Employee).

DNo	Count_SSN	Average Salary
5	4	33250
4	3	31000
1	1	55000

Eg: $\mathcal{F}$ Count SSN AVERAGE salary (Employee)

Count SSN	Average Salary
8	36125

Recursive Closure Operations:

This opⁿ is applied to a recursive relⁿship b/w tuples of the same type such as relationship b/w an employee & supervisor.

Eg: Retrieve all supervisees of an employee e at all levels – ie; all employees e' directly supervised by e, all employees e'' directly supervised by each employee e', all employees e''' directly supervised by each employee e'', so on.

Eg: Specify the SSns of all employees e' directly supervised at level one by the employee e whose name is 'James B'

$B\_SSn \leftarrow \pi_{Ssn}(\sigma_{Fname='James' \text{ AND } Lname='B'}(Employee))$

$Supervision(Ssn1, Ssn2) \leftarrow \pi_{Ssn, Super\_ssn}(Employee)$

$Result1(SSN) \leftarrow \pi_{Ssn1}(Supervision \bowtie_{Ssn2=Ssn} BSSN)$

To retrieve all employees supervised by B at level 2- ie; all employees e'' supervised by some employee e' who is directly supervised by B

$Result2(SSN) \leftarrow \pi_{SSN1}(Supervision \bowtie_{SSN2=SSN} Result1)$

To get both sets of employees supervised at levels 1 & 2 by 'James B' we can apply the UNION opⁿ to the 2 results as follows

$Result \leftarrow Result2 \cup Result1$

Outer Join & Outer Union Operations:

Eg: ~~Temp~~ List of all employee names & ~~are~~ also the name of the depts they manage if they happen to manage a dept;

$Temp \leftarrow (Employee ⟕_{SSN=MgrSSN} Department)$

$Result \leftarrow \pi_{Fname, Minit, Lname, Dname}(Temp)$

Examples of Queries in Relational Algebra:

Ref. Fig: 7.6

Query 1: Retrieve the name & address of all employees who work for the 'Research' dept.

$Research\_Dept \leftarrow \sigma_{name='Research'}(Department)$

$Research\_EMPS \leftarrow (Research\_Dept \bowtie_{DNumber=DNo} Employee)$

$Result \leftarrow \pi_{Fname, Lname, Address}(Research\_Emps)$

Query 2: For every project located in 'Stafford', list the project number, the ctrlling dept no & dept mgr's last name, address, & birth date.

$Stafford\_PROJS \leftarrow \sigma_{PLocation='stafford'}(Project)$

$ctr\_Dept \leftarrow (Stafford\_PROJS \bowtie_{Dnum=Dnumber} Department)$

$PROJ\_Dept\_mgr \leftarrow ctr\_Dept \bowtie_{mgrssn=ssn} Employee)$

$Result \leftarrow \pi_{number, Dnum, Lname, Address, Bdate}(PROJ\_DEPT\_MGR)$

Query 3: Find the names of employees who work on all the projects ctrlled by dept no 5.

$Dept5\_PROJS(PNO) \leftarrow \pi_{number}(\sigma_{Dnum=5}(Project))$

$Emp\_PROJ(SSN, PNO) \leftarrow \pi_{ESSN, PNo}(works\_on)$

$Result\_Emp\_ssn5 \leftarrow Emp\_PROJ \div Dept5\_PROJS$

$Result \leftarrow \pi_{Lname, Fname}(Result\_Emp\_SSN5 * Employee)$

Query 4: List the names of all Employees with 2 or more dependents.

$T_1(SSN, no.\ of\ Depts) \leftarrow {}_{ESSN}\mathcal{F}_{count\ Dependent\ name}(Dependent)$

$T_2 \leftarrow \sigma_{no.\ of\ Depts \geq 2}(T_1)$

** $Result \leftarrow \pi_{Lname, Fname}(T_2 * Employee)$

Query 5: Retrieve the names of employees who have no dependants.

All_Emps $\leftarrow \pi_{SSN}$ (Employee)

Emps-with-Deps(SSN) $\leftarrow \pi_{ESSN}$ (Dependant)

Emps-without-Deps $\leftarrow$ (All-Emps $-$ Emps-with-Deps)

Result $\leftarrow \pi_{Lname, Fname}$ (Emps-without-Deps * Employee)

Query 6: List the names of mngrs who have at least one dependant.

MGRS(SSN) $\leftarrow \pi_{mgrssn}$ (Department)

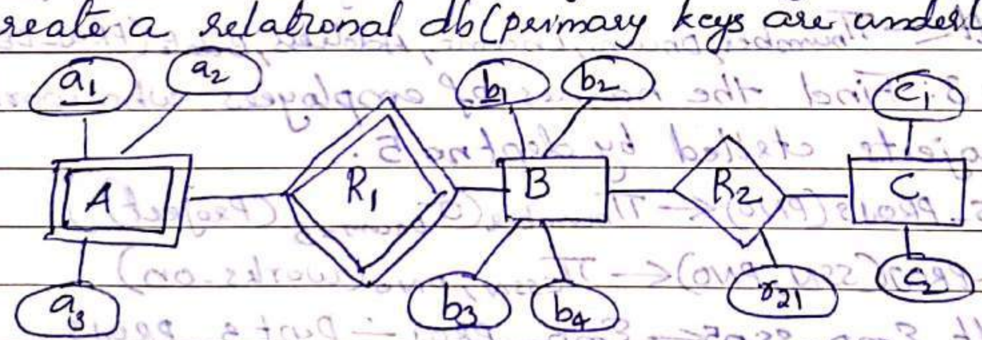
Emps-with-deps(SSN) $\leftarrow \pi_{ssn}$ (Dependant)

MGRS-with-deps $\leftarrow$ (MGRS $\cap$ Emps-with-deps)

Result $\leftarrow \pi_{Lame, Fname}$ (MGRS-with-deps)

Supplementary Previous Questions.

1. Consider the follo. E R diagram. Using this ER diagram create a relational db (primary keys are underlined).



2. Consider the follo. db with primary keys underlined

Suppliers(<u>sid</u>, sname, address)

Parts(<u>pid</u>, pname, color)

Catalog(<u>sid, pid</u>, cost)

Write relational algebra for the follo.

i) Find the names of suppliers who supply some red part

ii) Find the sids of suppliers who supply some red or green part

iii) Find the sids of suppliers who supply some red part and some green part.

Name	
Reg. No.	

NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
KERALA

SERIES TEST – I

CS 208 : Principles of Database Design

Year / Semester & Branch : II / S4 B.Tech. Computer Science & Engineering

Time : 90 Mins **Max. Marks: 20**

PART-A

Answer ALL Questions (3 x 2 = 6 Marks)

1. What are the responsibilities of DBA?
2. What is Meta data?
3. List out different types of Database languages.

PART- B

Answer ALL Questions (2 x 7 = 14 Marks)

4. Discuss the main characteristics of the database approach and how it differs from traditional file systems? (7 Marks)

OR

5. With the help of diagram explain three schema architecture of DBMS? (7 marks)

6. Explain different steps in mapping E R Diagram into Relational Schema (7 Marks)

OR

7 a. Define the following terms: a) Super key
b) Candidate Key
C) Primary Key (3 Marks)

b. Explain Integrity Constraints. (4 Marks)

MODULE III
STRUCTURED QUERY LANGUAGE (SQL)

→ Basic SQL Structure
  → Examples
→ Set Operations
→ Aggregate Functions
→ Nested Sub-queries.
→ Views, assertions & triggers.

S June 2017
1. Illustrate the group by clause with the help of eg.
2. Consider the query Select Name, Age from Student where Gender = 'Male' on the table student (RollNo, Name, Age, Gender, Address). Give a relational algebra expression corres. to the query. Is result produced by the query & your exprⁿ always the same? why?
3. In the follo. tables Advisor & taught by are foreign keys refering to the table Professor. RollNo & CourseID in Enrollment refer to tables with primary keys of the same name.
   Student (RollNo, Name, Age, Gender, Address, Advisor)
   Course (Courseid, CName, Taughtby, credits)
   Professor (Profid, PName, phone)
   Enrollment (RollNo, Courseid, Grade)
   Write SQL expressions for the follo. queries.
   1. Names of courses taught by 'Prof. Raju'.
   2. Names of students who have not enrolled for any course taught by 'Prof Ganapathy'.
   3. For each course, name of the ~~course~~ course & students enrolled for the course.

4. Differentiate b/w having & where clause in SQL.
5. Define assertion in SQL.
6. Explain View with View Materialization.
7. What are basic data types available for attributes in SQL
8. List the aggregate functions in SQL.?
9. Consider the following relations:

Customer(C_name, c-street, c-city)
Branch(bname, b-city, assets)
Account(AccNo, bname, balance)
Depositor(Cname, accNo)
Loan(lno, bname, amount)

Answer the follo. in SQL.

i) Create tables with primary keys & foreign keys.
ii) Create an assertion for the sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

Ans ii) create assertion sum_constraint check
(not exists(select * from Branch where(select sum(amount) from Loan where loan.bname = Branch.bname)>= (select sum(balance) from Account where Account.bname == Branch.bname)));

SQL DDL:

→ Schema for each relation
→ Domain values with each attribute.
→ Integrity Constraints
→ Security & authorization

Domain Types:

→ char(n), varchar(n), int, numeric, float etc.

Schema Definition in SQL:

* Create table command.

create table r(A_1 D_1, A_2 D_2, ... A_n D_n, <integrity Constraint$_1$> ... <integrity Constraint$_n$>);

Eg: create table customer (cname char(20), cstreet char(30), primary key (cname));

* Insert command: to load data into the relation.

Eg: insert into account values ('A-101', 'Pridge', 1200)

* delete command: delete tuples from a relation.

Eg: delete from account;

* drop command: To remove a reln from an SQL db.

Eg: drop table r; // remove relation r.

* alter table: to add attributes to an existing relation.

Eg: alter table r add A D

where r is the name of an existing relation, A is the name of the attribute to be added, and D is the domain of the added attribute.

⇒ ~~att~~ alter table r drop A; where r is the name of an existing relation, & A is the name of an attribute of the relation. ~~other~~

Basic Structure of SQL Queries:
→ consists of 3 clauses: select, from and where.

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

* The from clause corresponds to the Cartesian product opⁿ of the relational algebra. It lists the relⁿs to be scanned in the evaluation of the expr.

* The where clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relⁿs that appear in the from clause.

A typical SQL query has the form

select $A_1, A_2 \ldots A_n$
from $r_1, r_2 \ldots r_m$
where P

Relational Algebra Expr:

$$\Pi_{A_1, A_2 \ldots A_n}(\sigma_P(r_1 \times r_2 \times \ldots r_m))$$

⇒ The select clause:

Eg: Find the names of all branches in the loan relation-

select branch_name
from loan

*Elimination of duplicates:

Eg: select distinct branch.name
from loan

*all: to specify explicitly that duplicates are not removed.

Eg: select all branch.name
from loan

→ '*' can be used to denote "all attributes".
select * indicates that all attributes of all relation appearing in the from clause are selected.

The select clause may also contain arithmetic expressions involving the operators +, -, * & / operating on constants or attributes of tuples.

For eg: select lno, bname, amount * 100
from loan

will return a relⁿ that is the same as the loan relation, except that the attribute amount is multiplied by 100.

⇒ The where Clause:

Eg: Find all lno.s for loans made at the Pridge branch ~~wht~~ with loan amnts > \$1200.

select lno
from Loan
where b_name = 'Pridge' and amount > \$1200

Logical Connectives: and, or, & not. The operands of the logical connectives can be expressions involving the comparison operators ~~to co~~ <, <=, >, >=, = & <>.

between ~~co~~ comparison operator:

Eg: select lno from loan where amount between 90000 and 100000.

instead of

select lno from loan where amount <= 100000 and amount >= 90000

⇒ The from Clause:

Relational_algebra expression

$\Pi_{cname, lno, amount}(borrower \bowtie loan)$ for the query "For all customers who have a loan from the bank, find their names, loan numbers, and loan amount."

select c.name, borrower.lno, amount
from borrower, loan
where borrower.lno = loan.lno

classmate
Date
Page

⇒ The Rename Operation:

as clause: used for renaming both relations & attributes.

old name as new-name

For eg: if we want the attribute name lno to be replaced with the name lid, we can rewrite the preceding query as

select cname, borrower.lno as lid, amount
from borrower, loan
where borrower.l-no = loan.l-no

⇒ Tuple Variables:

Eg: For all customers who have a loan from the bank, find their names, loan numbers, and loan amnt as

select cname, T.lno, S.amount
from borrower as T, loan as S
where T.lno = S.lno

⇒ String Operations:

* like operator for pattern matching.
* percent (%): The % character matches any substring
* Underscore(_): The _ character matches any character.

Egs: 'Perry%' matches any string begining with "Perry".
'%idge%' matches any string containing "idge" as a substring, for eg: 'Perryridge', 'Rock Ridge', & 'Ridgeway'.
* '___' matches any string of exactly 3 characters.
* '___%' matches any string of atleast 3 characters.

Eg: Find the names of all ~~cu~~ customers whose street addr. includes the substring 'Main'.

Query: select cname
from customer
where c-street like '%Main%'

escape character:

SQL allows us to search for mispatches instead of by using the not like comparison operator.

→ upper() & lower()

⇒ Ordering the Display of Tuples:

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

Eg: To list in alphabetic order all customers who have a loan at the Pridge branch

select distinct c_name from borrower, loan
where borrower.lno = loan.lno and b_name = Pridge
order by c_name;

By default, the order by clause lists items in ascending order. specify desc for descending order & asc for ascending order.

Set Operations

* Union (Relational Algebra ∪)
* intersect (" ∩)
* except (" —)

1. The Union Operation:

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

(select c_name from depositor)
union
(select c_name from borrower)

The union opn eliminates duplicates. If we want to retain all duplicates, union all in place of union.

(select c-name from depositor)

union all

(select c-name from borrower)

2. The Intersect Operation:

Eg: To find all customers who have both a loan and an account at the bank.

(select distinct c-name from depositor)

intersect

(select distinct c-name from borrower)

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

(select c-name from depositor)

intersect all

(select c-name from borrower)

3. The Except Operation:

Eg: To find all customers who have an account but no loan at the bank.

(select distinct c-name from depositor)

except

(select c-name from borrower)

If we want to retain all duplicates, write except all in place of except

(select c-name from depositor)

except all

(select c-name from borrower)

Aggregate Functions:

SQL offers 5 built-in aggregate functions:

Average: avg, Minimum: min, Maximum: max, Total: sum, Count: count

The input to sum & avg must be a colln of nos.

Date
Page

Eg: Find the average account balance at the Pridge branch
select avg(balance) from account where b-name='Pridge'
If we want to apply the aggregate fn not only to a single set of tuples, but also to a group of sets of tuples use group by clause. The attribute/attributes given in the group by clause are pt used to form groups.
Eg: "Find the average account balance at each branch"
select b-name, avg (balance) from account
group by b-name
To eliminate duplicates
select b-name, count(distinct c-name)
from depositor, account
where depositor. acno=account. acno
group by b-name.
having clause:
Eg: branches where the average account balance is more than $1200. This condition does not apply to a single tuple, it applies to each group constructed by the group by clause. To express, we use the having clause. SQL applies predicates in the having clause after groups have been formed.
select b-name, avg (balance)
from account group by b-name
having avg(balance)>1200.

If a where clause & having clause appear in the same query, SQL applies the predicate in the where clause first. Tuples satisfying the where predicate are then placed into groups by the group by clause. SQL then applies the having clause, if it is present, to each group; it removes the groups to generate tuples of the result of the query.

Eg: Find the average balance for each customer who lives in Harrison & has at least 3 accounts.

select depositor.c-name, avg (balance)
from depositor, account, customer
where depositor.accno = account.accno and
depositor.c-name = customer.c-name and
c-city = 'Harrison'
group by depositor.c-name
having count(distinct depositor.accno) >3

Nested Subqueries:

A subquery is a select-from where expr that is nested within another query. A common use of subqueries is to perform tests for set membership, make set comparisons & determine set cardinality.

* Set Membership:

SQL allows testing tuples for membership in a relation. The in connective tests for set membership, where the set is a colln of values produced by a select clause. The not in connective tests for the absence of set membership.

Eg: Find all the customers who have both a loan & an account at the bank

select distinct c-name
from borrower
where c-name in (select c-name from depositor)

Eg: Find all customers who do have a loan at the bank but do not have an account at the bank

select distinct c-name from borrower
where c-name not in (select c-name from depositor)

Scanned with CamScanner

Eg. The names of customers who have a loan at the bank and whose names are neither Smith nor Jones.

select distinct c-name
from borrower where c-name not in ('Smith', 'Jones')

* Set Comparison:

→ to compare sets,

Eg: Find the names of all branches that have assets greater than those of ~~at le~~ atleast one branch located in Brooklyn.

select distinct T. b-name from branch as T, branch as S
where T. assets > S. assets and S. branch city = 'Brooklyn'.

* Test for Empty Relations:

→ testing whether a subquery has any tuples in it's result. The exists construct returns the value true if the arg. subquery is nonempty.

Eg: Find all customers who have both an account & a loan at the bank

select c-name from borrower where exists (select *
from depositor where depositor. c-name = borrower. c-name)

* Test for the Absence of Duplicate Tuples:

— testing whether a subquery has any duplicate tuples in its result.

Eg: Find all customers who have atmost one account at the ~~Perr~~ Ridge branch" ~~as~~

select T.c-name from depositor as T
where unique (select R.c-name from account, depositor
R where T.c-name = R.c-name and R.acc-no =
account. acc-no and account. b-name = 'Ridge')

Views:

Security considerations may require that certain data be hidden from users. Eg: A person who needs to know a customer's l-no & b-name, but has no need to see the loan amount.

Any relation that is not part of the logical model, but is made visible to a user as a virtual relation is called a view.

View Definition:

— view in SQL by ~~the~~ using the create view command.

create view V as <query expression> where <query expression> is any ~~logical~~ legal query expr.

Eg: create view all-customers as
(select b-name, c-name
from depositor, account
where depositor.acc-no = account.acc-no)
union
(select b-name, c-name
from borrower, loan
where borrower.l-no = loan.l-no)

Once defined a view, use view name to refer to the virtual relation that the view generates.

Eg: Find all customer of the Pudge branch
select c-name
from all-customers
where b-name = "Pudge".

The attribute names of a view can be specified explicitly as follows:

Eg: create view branch total_loan (b-name, total_loan)
as select b_name, sum(amount)
from loan group by b_name.

Certain db s/ms allow view relations to be stored. If the actual relations used in the view defⁿ change, the view is kept upto' date. Such views are called materialised views. The process of keeping the view upto date is called view maintenance.

Views Defined by using Other Views:

One view may be used in the expr. defining another view. Eg: create view Pridge_customer as
select c-name
from all_customer
where b-name = 'Pridge' where all-customer is itself a view relation.

View ~~Definition~~ expansion is a way to define the meaning of views defined in terms of other views. The view definitions are not recursive. i.e; no view is used in its own defⁿ. For eg: v_1 is used in the defⁿ of v_2, v_2 is used in defⁿ of v_3 & v_3 is used in the the defⁿ of v_1 then each of v_1, v_2 & v_3 is recursive.

Eg: for View expansion:

select * from
Pridge_customer where C name = 'John'

View expansion:

select * from (select C-name
from all-customer
where b-name = 'Pridge')
where cus name = 'John'.

It then generates

select * from (select c-name
from ((select b-name, c-name
~~from~~ from depositor, account
where depositor.accno = account.acc.no)
union

(select b_name, c_name
from borrower, loan
where borrower.l-no = loan.l-no))
where b_name = 'Pudge')
where c_name = 'John';

Assertions:

An assertion is a predicate expressing a condition that the db always to satisfy. Domain constraints & referential Integrity constraints are special forms of assertions. There are many constraint that we cannot express by using only these special forms.

Eg: The sum of all loan amnts for each branch must be less than the sum of account balances at the branch.

An assertion in SQL takes the form

create assertion <assertion_name> check <predicate>

When an assertion is created, the s/m tests it for validity. If the assertion is valid, then any future modificn to the db is allowed only if it doesn't cause assertion to be violated.

For eg: to specify the constraint that "the salary of an employee must not be greater than the salary of the mngr of the dept that the employee works for".

```
create assertion salary_constraint
check (not exists (select * from Employee E, Employee M,
                                  Department D
  where E.salary > M.salary and E.DNo = D.DNumber and
                          D.MGRSSN = M.SSN));
```

The constraint name salary_constraint is followed by a condn in paranthesis that must hold true on every ~~db~~ database state for the assertion to be satisfied.

For eg: to restrict the values of dept numbers to an integer no. b/w 1 & 20, we can write the follo.

create Domain DNum as integer
check (D-Num > 0 and D-NUM < 21);

Views Continuation

View Implementation: & View Update:

Query modification: involves modifying the view query into a query on the underlying base tables.

Disadv: time Consuming to execute; if multiple queries are applied to the view within a short period of time.

View materialization: physically creating a temporary view table when the view is first queried & keeping that table on the assumption that other queries on the view will follow.

```
Create View works-on1
  as select Fname, LName, PName, Hours
     from Employee, Project, works-on
     where SSN = ESSN and PNO = PNumber;
```

Works on1

FName	LName	PName	Hours

```
Update works on1
set PName = 'Product Y'
where LName = 'Smith' and FName = 'John' and
      PName = 'Product X';
```

This query can be mapped into several updates on the base rel^ns to give the effect on the View.

```
Update Works-on
Set PNo = (Select PNumber from Project
           where PName = 'Product Y')
where ESSN IN (select SSN from Employee where
               LName = 'Smith' and FName = 'John')
  and PNO IN (select PNumber from Project where PName = 'Product X').
```

(b) Update Project set PName = 'Product Y'
where PName = 'Product X';

Triggers:

Triggers are stored pgms, which are automatically executed when some event occurs. Triggers are written to be executed in response to any of the follo. events. A db manipulation stmt (delete, insert or update); A DDL stmt (create, alter or Drop). Triggers could be defined on the table, view, or db with which event is associated.

For updates, the trigger can specify columns whose update causes the trigger to execute.

Eg: create trigger overdraft_trigger after update of balance on account.

then the trigger would be executed only on updates to balance; updates to other attributes would not cause it to be executed.

The referencing old row as clause can be used to create a variable storing the old value of an updated or deleted row. The referencing new row as clause can be used with inserts in addition to updates.

As another eg:, suppose the value in a phno field of an inserted tuple is blank, which indicates absence of a phone no. We can define a trigger that replaces the value by the null value. The set stmt can be used to carry out such modific^ns.

```
create trigger setnull_trigger before update on r
referencing new row as nrow
for each row
when nrow.phno = ' '
set nrow.phno = null;
```

MODULE IV
Relational Database Design

- → Different anomalies in designing a db.
- → Normalization
- → Functional Dependency
- → Armstrong's Axioms
- → Closures
- → Equivalence of FDs
- → minimal Cover.
- → Normalization using FDs
- → 1NF, 2NF, 3NF & BCNF
- → lossless & dependency preserving decomposition

July 2017

(10) 1: Let $E = \{B \to A, D \to A, AB \to D\}$ is a set of Functional Dependencies. Find a minimal cover for E

The minimal cover for E is $\{B \to D, D \to A\}$

(11) 2 Define BCNF. Give eg: of a relation that is in 3NF but not in BCNF.

Eg: (Student Course Instructor)

FD_1 : {Student, Course} → Instructor // student course is a super key.

FD_2 : Instructor → Course   Course is a prime attribute

B/c of FD_2 its not in BCNF and also its in 3NF.

(13) 3. Given R(A, B, C, D, E) with the set of FDs, $F = \{AB \to CD, ABC \to E, C \to A\}$ i) Find any 2 candidate keys of R
ii) What is the normal form of R? Justify ur answer.

(i) $(AB)^+ = ABCDE$, // $AB \to CD$, $ABC \to E$

$(BC)^+ = BCAED$, // $C \to A$, $ABC \to E$, $AB \to CD$

Candidate keys: AB & BC.

classmate
Date
Page

ii) Prime attributes : A, B, C
non prime attributes: D & E ; are fully fnally dependent on the prime attributes ∴ rel^n in 2NF.
∴ Nonprime attributes are not transitively dependent on the primary key. So its in 3NF.
$C \to A$ its a non trivial FD
C is not a superkey. Hence relation is not in BCNF. Given relation is in 3NF.

14) 4 a) What are Armstrong's axioms.
b) Write an algm to compute the attribute closure of a set of attributes (α) under a set of FDs (F)
c) Explain uses of attribute closure algms.
i) To test if X is a super key, we compute X^+ & check if X^+ contains all attributes of R.
ii) We can check if a FD $X \to Y$ holds by checking if $Y \subseteq X^+$. ie; we compute X^+ by using attribute closure & then check if it contains Y.
iii) It gives us a alternate way to compute F^+ : For each $S \subseteq X^+$, we out FD $X \to S$.

→ 1) Define Functional Dependency.
2) What are different anomalies in database design?
3) Explain Armstrong's Axioms.
4) Define Minimal Cover.
5) Differentiate b/w 3NF & BCNF.

5. Determine any 2 candidate keys of the relation R(A...
with FDs $AB \to C$, $C \to AD$, $D \to EF$, $F \to B$

$AB^+ = ABCDEF$ ($AB \to C$, $C \to AD$ ∴ $AB \to AD$, $D \to EF$)

$C^+ = CADEFB$ ($C \to AD$, $D \to EF$, $F \to B$)

Candidate keys : AB & C.

Different Anomalies ~~&~~ in designing Database.

In the case of RDBMS, design, to minimize the redundancy, the data has to be organized by decomposing relations to produce smaller well structured relations. This process is called Normalization.

The main pblms in db design are
- Redundant Infn in tuples
- Update anomalies.

Redundant Infn in tuples.

Eg; Consider the table. EMP_DEPT (Ename, SSN, Bdate, Address, Dno, DName, DMgrSSN)

EName	SSN	BDate	Address	DNo	DName	DMgrSSN
Smith	123	1965-1-9	Houston	5	CSE	455
John	134	1955-12-8	Pridge	5	CSE	455
Joyce	145	1968-7-19	Castle	1	ECE	888.

The main goal of db design is to reduce the storage space. By grouping the attributes, storage space can be reduced.

Eg: Employee (EName, ~~B1~~ SSN, ~~Do~~ Bdate, Address, DNo, DMgrSSN)
Department (DName, DNumber, DMgrNo)

Anomalies:

Insertion Anomaly: It is difficult to insert some data in the db. For eg: Consider the above table EMP_DEPT ~~Student_Courses (Sid, Sname, phone,~~ Its not possible to insert a new department details that has no employees as yet in the relation. The only way to do this is to place null values in the attributes for employee. This causes a pblm b/c SSN is the primary key of EMP_DEPT & each tuple is supposed to represent an employee entity - not a department entity.

Deletion Anomaly: Consider the table EMP_DEPT as eg: If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the infn concerning that dept is lost from the database.

Modification/Update Anomaly: In EMP_DEPT, if we change the value of one of the attribute of a particular dept, — the mngr of dept 5 — we must update the tuples of all employees who work in that department; otherwise db will become inconsistent.

Functional Dependency

Functional Dependency is a constraint b/w 2 sets of attributes from the db. A functional dependency denoted by $X \rightarrow Y$, b/w 2 sets of attributes X & Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The cond is that, for any 2 tuples t_1 & t_2 in r that have $t_1[X] = t_2[X]$, we must also $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on or are determined by the values of X component. Or the values of X component of a tuple uniquely determine the values of Y component.

$X \rightarrow Y$ (X determines Y or Y is finally dependent on X)

For eg: EMP. SSN → Bdate
123 = 1965-1-9 when SSN changes to 134, BDate → 1955-…
X Y

Armstrong's Axioms
(Inference Rules for FDs.)

F = {SSN → {ENAME, Bdate, Address, Dno},
DNumber → {DName, DMgrSSN}}

We can infer the follo. addnal FDs from F:

SSN → {DName, DMgrSSN},

SSN → SSN, ~~DNumb~~ DNumber → DName

Inference Rules (IR)

IR_1 (reflexive rule): If $X \supseteq Y$, then $X \to Y$.

ie; can be stated as $X \to X$; ie; any set of attributes finally determines itself.

Proof: Suppose that $X \supseteq Y$ & that 2 tuples t_1 & t_2 exist in same relation instance r of R such that $t_1[X] = t_2[X]$. Then $t_1[Y] = t_2[Y]$ b/c $X \supseteq Y$; hence $X \to Y$ must hold in r.

IR_2 (augmentation rule): $\{X \to Y\} \models XZ \to YZ$ (By contradiction)

IR_3 (transitive rule): $\{X \to Y, Y \to Z\} \models X \to Z$.

Proof: same.

IR_4: (decomposition, or projective rule): $\{X \to YZ\} \models X \to Y$

Proof: $X \to YZ$ (given)

$YZ \to Y$ (using IR_1 & $YZ \supseteq Y$)

$X \to Y$ (using IR_3)

IR_5 (union or additive rule): $\{X \to Y, X \to Z\} \models X \to YZ$

Proof: (using IR_1 through IR_3)

Given $X \to Y$ & $X \to Z$

$XX \to XY$ (using IR_2)

③ $X \to XY$ by augmenting with X.

④ $XY \to YZ$ (using IR_2 by augmenting with Y)

$X \to YZ$ (using IR_3 on 3 & 4)

classmate
Date
Page

IR_6 (~~pseuo~~ pseudotransitive rule): $\{X \to Y, WY \to Z\} \models WX \to Z$

Proof: ① $X \to Y$, ② $WY \to Z$ given.

③ $WX \to WY$ (IR_2 augmenting with W)

$WX \to Z$ (IR_3 on 2 & 3)

Closures (F^+ closure)

Definition: set of all dependencies that include F as well as all dependencies that can be infered from F is called the closure of F. denoted by F^+.

Algorithm Determining X^+, the closure of X under F.

$X^+ := X$;

repeat

old $X^+ := X^+$;

for each FD $Y \to Z$ in F do

If $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;

~~until (X^+ = old X^+);~~

Consider the follo. from table EMP-PROJ

$F = \{SSN \to EName, PNumber \to \{PName, PLocation\}, \{SSN, Pnumber\} \to Hours$

$\{SSN\}^+ = \{SSN, EName\}$

$\{PNumber\}^+ = \{PNumber, PName, PLocation\}$

$\{SSN, PNumber\} \to \{SSN, EName, PNumber, PName, PLocation, Hours\}$

Equivalence of FDs.

Cover: A set of FDs F is said to cover another FD E if every FD in E is also in F^+. i.e; if every dependency in E can be inferred from F, then E is covered by F.

Equivalent: Two sets of FDs E & F are equivalent if $E^+ = F^+$. i.e; every FD in E can be inferred from F, & every FD in F can be inferred from E.

Scanned with CamScanner

Minimal Cover (Minimal Sets of FDs).

A set of functional dependancies F is minimal if it satisfies the follo. conditions:

1. Every dependancy in F has a single attribute for its right hand side.
2. We cannot replace any dependancy $X \rightarrow A$ in F with a dependancy $Y \rightarrow A$ where Y is a proper subset of X, & still have a set of dependancies that is eqt. to F.
3. We cannot remove any dependancy from F and still have a set of dependancies that is eqt. to F.

Normalization using FDs.

→ a process of analysing the given relation schema based on their FDs and primary keys to achieve the desirable properties of 1. minimizing redundancy and 2. minimizing the insertion, deletion & update anomalies.

Additional properties: lossless join or non additive join property, which guarantees that the spurious tuple generation pblm does not occur.

Dependancy preservation: ensures that each FD is represented in some individual rel's resulting after decomposit.

prime attribute & non prime attribute:

Prime attribute: it is member of some candidate key of R.

non prime attribute: if it is not member of candidate key of R.

Eg: works on

SSN	PNo	Hours

First Normal Form (1NF)

It states that the domain of an attribute must include only atomic (simple, indivisible) values & that the value of any attribute in a tuple must be a single value from the domain of that attribute. The only attribute values permitted by 1NF are single atomic (indivisible values).

Eg: Department (DName, DNumber, DMgrNo, DLocations)

Each dept can have a no. of locations.

DName	DNumber	DMgrNo	DLocations
Research	5	33344	{Banglore, Delhi, Hyderabad}
Administration	4	9875	{chennai}
HQuarters	1	8865	{Hyderabad}

There are 3 main techniques to achieve 1NF for such a relation.

(1) 1NF with redundancy:

DName	DNumber	DMgrNo	DLocations
Research	5	33344	Banglore
Research	5	33344	NewDelhi
Research	5	33344	Hyderabad
Administration	4	9875	Chennai
HQuarters	1	88865	H.bad

(2) Remove the attribute DLocations that violates 1NF and place it in a separate rel^n Dept-Locations along with primary key DNumber. The primary key of this relation is the combination of {DNumber, DLocation}

Department

DName	DNumber	DMgrNo
Research	5	33344
Admin	4	9875
HQuarters	1	88865

Dept Locations

DNumber	DLocation
1	Hyderabad
4	Chennai
5	Banglore
5	NewDelhi
5	Hyderabad

(3) If a max. no. of values is known for the attribute replace the DLocations attribute by 3 atomic attributes.
DLocation1, DLocation2 & DLocation3

DName	DNumber	DMGRNo	DLocation1	DLocation2	DLocation3
Research	5	33344	Banglore	NewDelhi	Hyderabad
Admin	4	9875	Chennai	—	—
HQuarters	1	88865	Hyderabad	—	—

Disadvantage: introduces null ~~vott~~ values if most depts have fewer than 3 locations.

Second Normal Form (2NF)

— in 1NF.

* Full Functional Dependancy:

A functional dependency $X \rightarrow Y$ is a full functional dependancy if removal of any attribute A from X means that the dependency doesn't hold any more.
ie; for any attribute $A \in (X - \{A\})$ doesn't functionally determine Y.

* Partial Functional Dependancy:

A functional dependency $X \rightarrow Y$ is a partial dependancy if some attribute $A \in X$ can be removed from X & the dependency still holds.
ie; for some $A \in X, (X - \{A\}) \rightarrow Y$.

For eg: {ENo, PNumber} → Hours is a full dependancy, neither ENo → Hours nor PNumber → Hours.

Eg: {ENo, PNumber} → EName is a partial b/c ENo → EName holds.

A relation schema R is in 2NF, if every non prime attribute A in R is fully fnally dependent on the primary key of R. In 2NF, ~~LS~~ LHS attributes are ~~primary key~~ part of the primary key.

ENo PNumber Hours EName PName PLocation

FD1 FD2 FD3

primary key - ENo, PNumber

$FD_1 = \{ENo, PNumber\} \rightarrow Hours$

$FD_2 = \{ENo\} \rightarrow EName$

$FD_3 = PNumber \rightarrow \{PName, PLocation\}$

FD_2 & FD_3 - not fully functionaly dependent on primary key

(i.e; primary key is ENo, PNumber

EName is partially dependent on primary key

Decompose into 3 relations

⇓

EP_1

ENo PNumber . Hours

EP_2

ENo EName

EP_3

PNumber PName PLocation

Third Normal Form

→ Concept of Transitive Dependancy.

A relation schema R is in 3NF, if it satisfies 2NF & no non prime attribute of R is transitively dependent on the primary key.

EMP-DEPT

EName ENo DOB Address DNumber DName DMgrSSN

↓

ED_1

EName ENo DOB Address DNumber

ED_2

DNumber DName MgrSSN

EMP.DEPT not in 3NF b/c DMGRSSN is transitively dependent on ENo through DNumber.

Boyce-Codd Normal Form

It eliminates all redundancy that can be based on functional dependencies.

Definition: A relation schema R is in BCNF w.r.to a set F of FDs if all FDs in F^+ of the form $X \to Y$, where $X \subseteq R$ & $Y \subseteq R$ at least one of the follo. holds.

i) $X \to Y$ is a trivial FD (i.e; $X \subseteq Y$)

ii) X is a superkey for schema R.

i) {ENo, EName} → {EName}

ii) {ENo, EName} → DNo

{ENo, EName} → Salary

{ENo, EName} → Hours.

Trivial Functional Dependency: A trivial FD is a FD of an attribute on a superset of itself.

{EID, EAaddress} → {EAddress} is trivial b/c {EAddress} → {EAddress}

Lossless & Dependancy preserving decomposition.

If each FD $X \to Y$ specified in F either appears directly in one of the relation schemas R_i in the decomposition D or could be inferred from the dependencies that appear in some R_i - This is the dependancy preservation condition. We want to preserve the dependancies b/c each dependency in F represents a constraint on the db.

A decomposition $D = \{R_1, R_2, \cdots R_m\}$ of R is dependancy-preserving w.r.to F if the union of projections of F on each R_i in D is equivalent to F

ie; $(\Pi_{R_1}(F)) \cup \cdots \cup (\Pi_{R_m}(F)))^+ = F^+$

Algorithm: Relational Synthesis algm with dependency preservation.

Input: A universal relation R and a set of FDs F on the atrributes of R.

1. Find minimal cover G for F (use Algm)
2. For each LHS X of a FD that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \cdots \{A_k\}\}$ where $X \to A_1, X \to A_2 \cdots X \to A_k$ are the only dependancies in G with X as LHS (X is the key of the relation).
3. Place any remaining attributes in a single relation schema to ensure the attribute preservation property. (that have not placed in any relation).

Lossless Joins (Non additive) & Decomposition

A decomposition $D = \{R_1, R_2 \cdots, R_m\}$ of R has the lossless (nonadditive) join property w.r.to the set of dependancies F on R if for every relation state r of R that satisfies F, the following holds, where $*$ is the natural join of all the relations in D.

classmate

Date

Page

$* \pi_{R_1}(r), \ldots \pi_{R_m}(r)) = r.$

If a decomposition doesn't have the lossless join property we may get additional spurious tuples after the Project (π) and Natural join ($*$) operations are applied; these additional tuples represent erroneous informⁿ.

Algorithm: Testing for the Lossless (nonadditive) join property.

Input: A universal Relation R, a decomposition $D = \{R_1, R_2, \ldots R_m\}$ of R & a set of FDs.

1. Create an initial matrix S one row i for each relation R_i in D & one column j for each attribute A_j in R.
2. Set $S(i,j) := b_{ij}$ for all matrix entries.
   (* each b_{ij} is a distinct symbol associated with indices (i,j) *)
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
      {if (Relation R_i includes attribute A_j) then set
         $S(i,j) := a_j$;};};
4. Repeat the follo. loop until a complete loop execution results in no changes to S.
   {for each FD $X \rightarrow Y$ in F
      {for all rows in S which have the same symbols in the columns corres. to attributes in X.
         {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:

         If any of the rows has an "a" symbol for the column, set other rows to the same "a" symbol in that column. If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute & set the other rows to that same "b" symbol in the column ;};};}

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it doesnot.

Eg: R = {SSN, EName, PNo, PName, PLocation, Hours}

R_1 = EMP-LOCS = {EName, PLocation}

R_2 = EMP-PROJ$_1$ = {SSN, PNo, Hours, PName, PLocation}

F = {SSN → EName; PNo → {PName, PLocation}, {SSN, PNo} → Hours}

	SSN	EName	PNo	PName	PLocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

After applying FDs. No change.

Eg:

classmate

Date ______

Page ______

5. Consider the Relation (A, B, C, D, E, F) with A as the only key. Assume that the dependancies $E \to F$ and $C \to DEF$ hold on R.

i) Is R is in 2NF? If not decompose into 2NF

ii) Is R is in 3NF? If not decompose into 3NF.

Ans: Given $E \to F$, $C \to DEF$.

R is in 2NF. // non prime attributes are fully finally dependent on primary key A.

3NF $\Rightarrow$ not in 3NF

A B C — in 3NF

C D E F — not in 3NF $\Rightarrow$

$\Downarrow$

C D E   E F

6. Calculate $\{SSN\}^+$ & $\{DNo\}^+$, the FDs are $G = \{SSN \to \{EName, BDate, Address, DNo\}, DNo \to \{DName, DMgrSSN\}\}$

$\{SSN\}^+ = \{SSN, EName, BDate, Address, DNo, DName, DMgrSSN\}$

$\{DNo\}^+ = \{DNo, DName, DMgrSSN\}$.

Is the set of FDs minimal.

7. Prove that any relation schema with 2 attributes is in BCNF.

A B

Continuation of 3NF & BCNF.

General Definition of 3NF:

A relation schema R is in 3NF if, whenever a nontrivial FD $X \rightarrow A$ holds in R, either

(a) X is a superkey of R or

(b) A is a prime attribute of R.

Qn: Check whether the given relation is in 3NF or not.

LOTS1

Property id #	County Name	LOT #	Area	Price

FD1

FD2

FD4

FD4 violates the condition for 3NF. Area is not a Superkey & Price is not a prime attribute.

Otherwise Price is transitively dependent on primary key property id #.

~~Depo~~ Decompose into 3NF.

LOTS1A

Property id #	Countyname	Lot #	Area

FD1

FD2

LOTS1B

Area	Price

FD2 – {Countyname Lot #} $\longrightarrow$ Super key.

General Definition of BCNF

The only diff. b/w the def^n of BCNF & 3NF is that condition (b) of 3NF, which allows A to be prime, is absent from BCNF.

Most relation schemas that are in 3NF are also in BCNF. Only if $X \rightarrow A$ holds in a relation schema R with X not being a superkey & A being prime attribute will R be in 3NF but not in BCNF.

1. R = {SSN, EName, PNo, PName, PLocation, Hours} D = {R_1, R_2, R_3}
R_1 = EMP = {SSN, EName}
R_2 = PROJ = {PNo, PName, PLocation}, R_3 = Works on = {SSN, PNo, Hours}
F = {SSN → EName ; PNo → {PName, PLocation} ; {SSN, PNo} → Hours.
Test lossless join property with Algorithm.

1. ~~Create~~
1. Create Matrix S & set $S(i,j) = b_{ij}$

	SSN	EName	PNo	PName	PLocation	Hours
R_1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}
R_3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}	b_{36}

2) Set $S(i,j) = a_j$ for each A_j in R_i

	SSN	EName	PNo	PName	PLocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	$\cancel{b_{32}}$ a_2	a_3	$\cancel{b_{34}}$ a_4	$\cancel{b_{35}}$ a_5	a_6

3) Apply Functional Dependencies.
X → Y check X & Y in same Row
Else Y in any row has a symbol for that column, set other rows to a. Else b.

* SSN → EName // for ~~SN~~ First row contains a for these columns ∴ $b_{32} \rightarrow a_2$
* PNo → {PName, PLocation} // 2nd row contains a for these columns. ∴ b_{34} & b_{35} ⇒ a_4 & a_5
* {SSN, PNo} → Hours // a for all 3 columns.

Third row contains all 'a's So its ~~lossless~~ decomposition has lossless join property.

classmate
Date
Page

June 2017

1. Assume that relation $R(P, Q, S, T, U)$ with FDs $P \to S$, $Q \to S$, $S \to T$, $TU \to S$, $SU \to P$ is decomposed into 5 relations. $R_1(P, T)$ $R_2(P, Q)$. $R_3(Q, U)$, $R_4(S, T, U)$ and $R_5(P, U)$. Apply the std algm to test if the decomposition is loss less join decomposition.

1. create matrix S & set $S(i,j) = b_{ij}$

	P	Q	S	T	U
R_1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
R_3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
R_4	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
R_5	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

2. Set $S(i,j) = a_j$ for each A_j in R_i

	P	Q	S	T	U
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	$\cancel{b_{23}}$	$\cancel{b_{24}}$ a_4	b_{25}
R_3	$\cancel{b_{31}}$ a_1	a_2	$\cancel{b_{33}}$ a_3	$\cancel{b_{34}}$ a_4	a_5
R_4	$\cancel{b_{41}}$ a_1	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	$\cancel{b_{53}}$ a_3	$\cancel{b_{54}}$ a_4	a_5

3. Apply FDs

* $P \to S$ ✗ * $S \to T$ ✓ * $SU \to P$ ✓
* $Q \to S$ ✗ * $TU \to S$ ✓

~~This decomposition doesn't have lossless join decomposition.~~ Lossless Decomposition.

MODULE – V

Physical Data Organization:

→ Index Structures
- * Primary Indices
- * Secondary Indices
- * Clustering Indices

} Single level Indices.

- * Multilevel Indices
- * B^+ Trees (basic Structure Only, Algms not needed)

→ Query Optimization
- * Heuristics based Query Optimization.

1. Consider a file with 200,000 records stored in disk with fixed length blk size 256 bytes. Each record of size 50 bytes. The primary key is 4 bytes & blk ptr is 6 bytes. Compute the follo., assuming multilevel primary index is used as access path.

i) Blking factor for data records ii) Bfr for index records
iii) No. of datablks iv) No. of first level index blks.
v) No. of levels of multilevel Index.

Given $r = 200{,}000$, $B = 256$ bytes, Record size $R = 50$ bytes
Index Entry $R_i = 6+4 = 10$ bytes.

ii) Blking factor for Index records $bfr_i = \lfloor B/R_i \rfloor = \lceil 256/10 \rceil = 25$

i) Blking factor for Data records $bfr = \lfloor B/R \rfloor = \lceil 256/50 \rceil = 5$

iii) No. of data blks $b = \lceil r/bfr \rceil = \lceil 200{,}000/5 \rceil = 40{,}000$ blks
(r = records)

iv) No. of 1st level index blks. $= b_i = \lceil \cancel{r_i/bfr_i} \rceil = \lceil 40000/25 \rceil$

No. of 1st level index blks $= \lceil 40000/bfr_{index} \rceil$ = No. of datablks = 40000

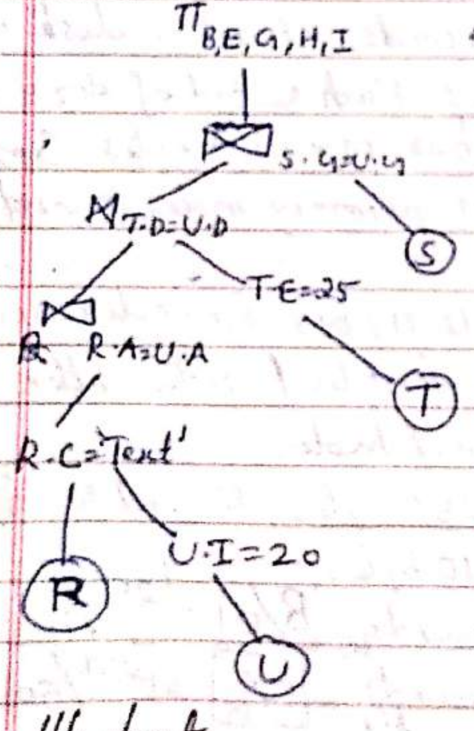
$= \lceil 40000/25 \rceil = 1600$

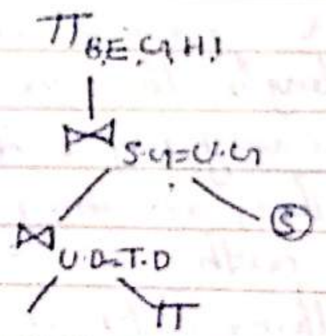
v) No of levels $= \log_{f_o}(r_1)$

$= 4$

2. What are diff. types of single level ordered indices? Explain
3. What is a B^+ tree?
4. Describe the structure of internal & leaf nodes of B^+ tree of order p.
5. Diff. b/w static hashing & dynamic hashing?
6. Consider the tables R(A, B, C), T(D, E, F), S(G, H) & U(A, D, G, I) where A, D & G in U are foreign keys referring to the primary keys with the same names. Show an initial query tree for the follo. query & optimize it using the rules of heuristics:

select B, E, G, H, I
from R, T, S, U
where T.D = U.D and S.G = U.G and R.C = 'Text' and U.I > 20 and T.E = 25





Illustrate

Index Structures

→ A file already exists with some primary org^n such as the unordered, ordered or a hashed organizations. Access structures called indexes are used to speed up the retrieval of records in response to certain search conditions. The index structures provide secondary access paths which provide alternative ways of accessing the records without affecting the physical placement of records on disk. They enable efficient access to records based on the indexing fields that are used to construct the index.

Single level Ordered Indexes

For a file with a given record structure consisting of several fields (or attributes), an index access structure is usually defined on a single field or a file, called an indexing field (attribute). The index stores each value of the index field along with a list of pointers to all disk blks that contain records with that field value. The values in the index are ordered so that we can do binary search on the index.

Types of Ordered Indexes:

- → Primary Index
- → Clustering Index
- → Secondary Index

Primary Index:

Primary index is specified on the ordering key field of an ordered file or records. Ordering key field is used to physically order the file records on disk & every record has a unique value for that field.

– is an ordered file whose records are fixed length with 2 fields. The first field is of same data type as the ordering key field – called primary key – of the data file.

and the second field is a pointer to a disk blk (block address)
There is one index entry (or index record) in the index file for each block in the data file.

Each index entry has the value of the primary key field for the first record in a blk and a ptr to that blk as its 2 field values.

2 field values of index entry i as ⟨K(i), P(i)⟩

The first 3 index entries are:

⟨K(1) = (Aaron, Ed), P(1) = address of blk 1⟩

⟨K(2) = (Adams, John), P(2) = " " 2⟩

⟨K(3) = (Alex, Ed), P(3) = " " 3⟩

Fig 6.1

The total no. of entries in the index is the same as the no. of disk blks in the ordered file.

~~Ach~~ Anchor record/Block anchor: The first record in each blk of the data file.

Indexes can also be characterised as dense or sparse. A dense index has an index entry for every search key value in the data file.

A sparse index has index entries for only some of the search values.

A primary index is hence nondense (sparse) index since it includes an entry for each disk blk of the data file rather than for every search value.

The index file for a primary index needs substantially fewer blks than the data file for 2 reasons.

1. there are fewer index entries than there are records in the data file.

2. Each index entry is smaller in size than a data record b/c it has only 2 fields.

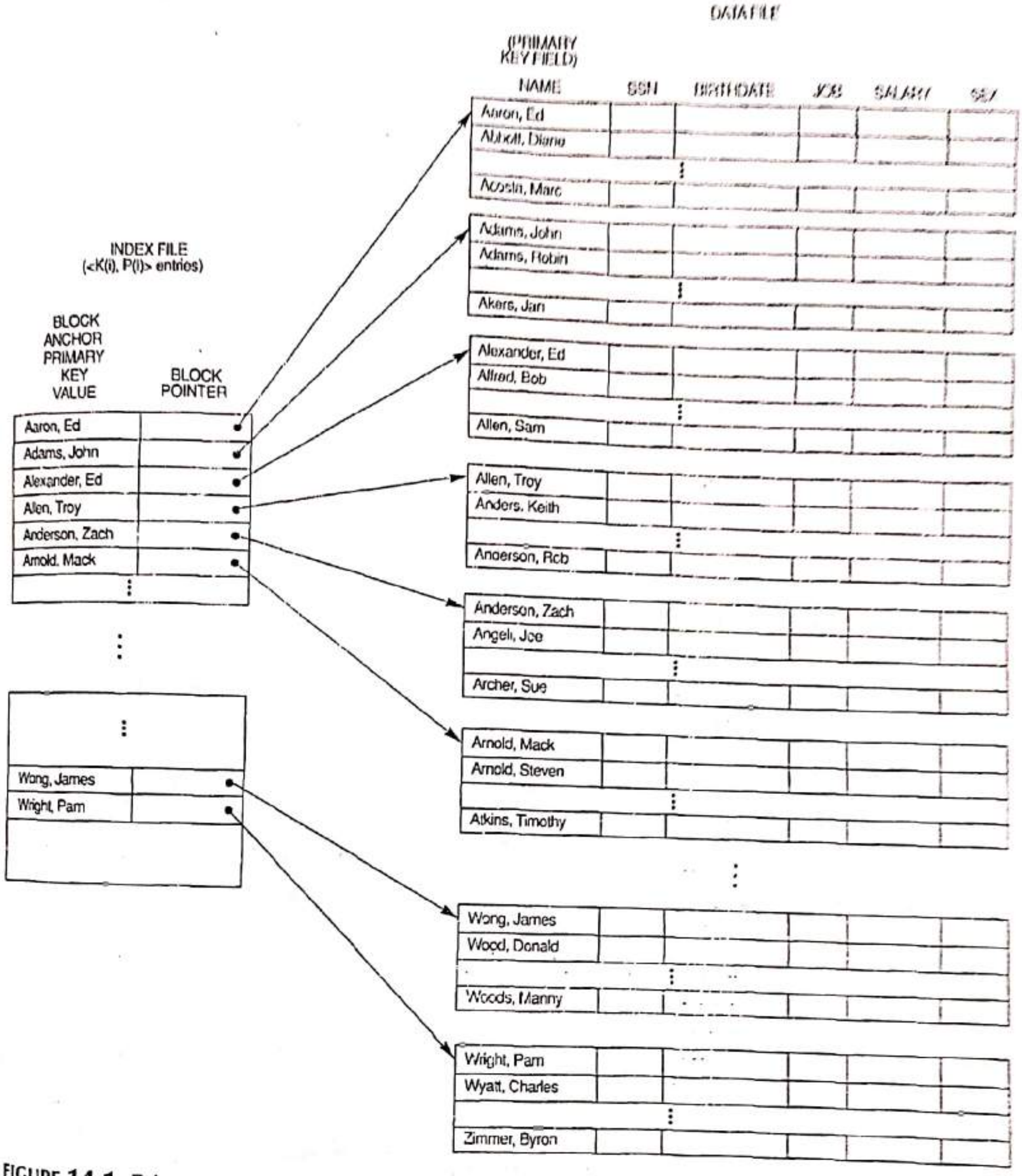


FIGURE 14.1 Primary index on the ordering key field of the file shown in Figure 13.7.

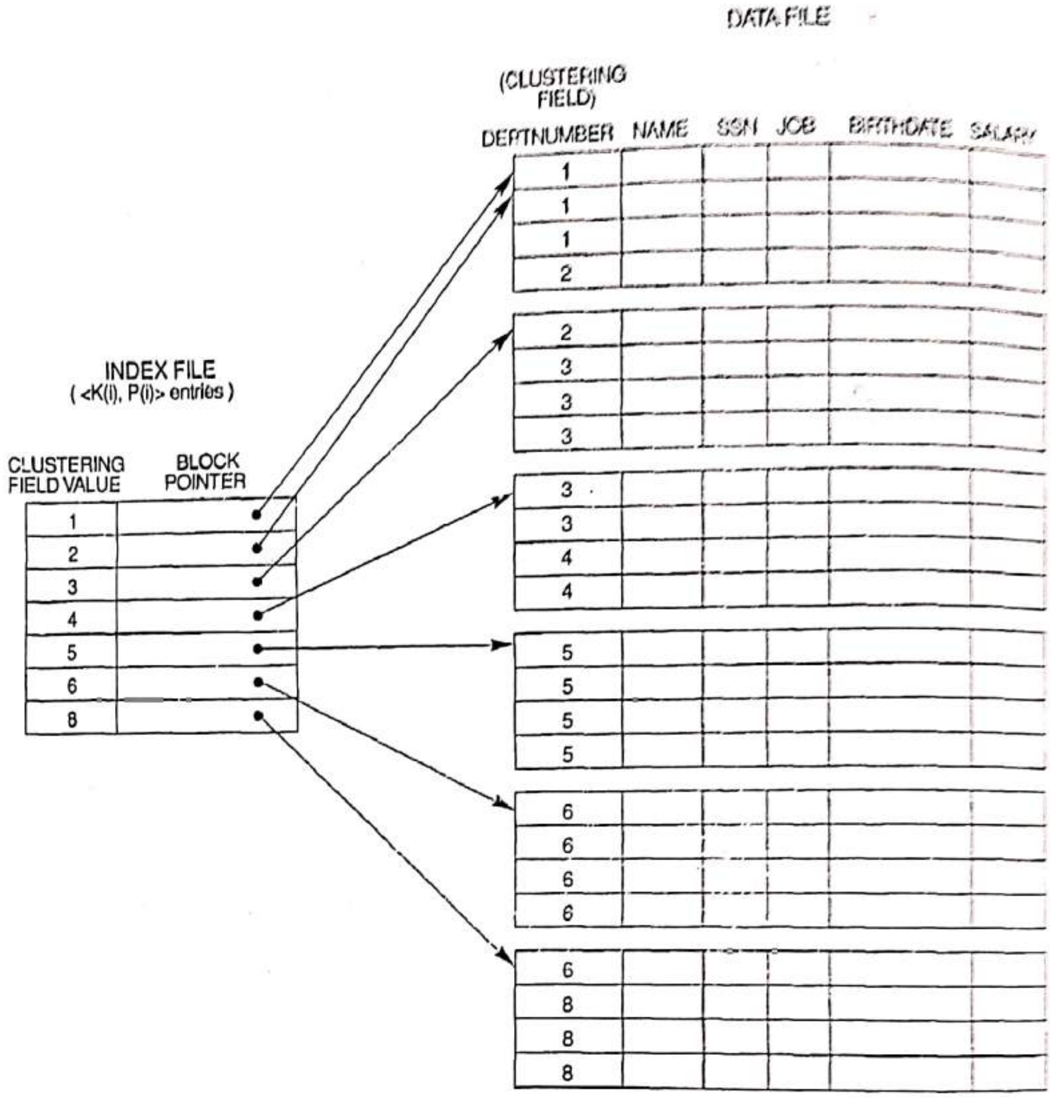


FIGURE 14.2 A clustering index on the DEPTNUMBER ordering nonkey field of an EMPLOYEE file.

block (or block cluster). This makes insertion and deletion relatively straightforward. Figure 14.3 shows this scheme.

A clustering index is another example of a *nondense* index, because it has an entry for every *distinct value* of the indexing field which is a nonkey by definition and hence has duplicate values rather than for every record in the file. There is some similarity between Figures 14.1 to 14.3, on the one hand, and Figure 13.11, on the other. An index is somewhat similar to the directory structures used for extendible hashing, described in Section 13.8.3. Both are searched to find a pointer to the data block containing the

classmate
Date
Page

Eg: We have an ordered file with $r = 30,000$ records stored on a disk with blk size $B = 1024$ bytes. File records are fixed size with record length $R = 100$ bytes.

The blking factor $bfr = \lfloor (B/R) \rfloor$
(for the file) $= \lfloor 1024/100 \rfloor = 10$ records/blk.

The no. of blks needed for the file $b = \lceil r/bfr \rceil = \lceil 30000/10 \rceil$
$= 3000$ blks.

Block accesses (Binary search) $= \log_2 b$
$= \lceil \log_2 3000 \rceil = 12$ blk accesses.

Clustering Indexes

If records of a file are physically ordered on a nonkey field, that field is called the ~~clus~~ clustering field.

A clustering index is also an ordered file with 2 fields; the first field is of the same type as the clustering field of the data file and the second field is a blk ptr. There is one entry in the clustering index for each distinct value of the clustering field, ~~of the~~ containing the value & a ptr to the 1st blk in the data file that has a record with that value for its clustering field.

Disadvantage / Drawback:

→ Record Insertion & deletion

B/c data records are physically ordered. To alleviate the pblm of insertion, it is common to reserve a whol blk for each value of clustering field. All records with that value are placed in the blk.

A clustering index is another eg. of a nondense index, b/c it has an entry for every distinct value of the indexing field than for every record in the file.

Fig: 6.2 & 6.3

Secondary Indexes

A secondary index is also an ordered file with 2 fields. The first field of the same data type as some nonordering field of the data file that is an indexing field. The second field is blk ptr or record ptr.

There is one index entry for each record in the data file, which contains the value of the secondary key for the record & a ptr either to the block in which record is stored or to record itself. Hence, such an index is dense.

Fig: 6.4.

A secondary index needs more storage space & longer search time.

Eg: $r = 30{,}000$, fixed length records of size $R = 100$ bytes stored on a disk with blk size $B = 1024$ bytes. The file has $b = 3000$ blks. non ordering key field length $V = 9$ bytes. blk ptr $P = 6$ bytes long. Calculate blocking factor.

Index entry $R_i = V + P = 9 + 6 = 15$ bytes.

Blocking factor (for index) $bfr_i = \lfloor B/R_i \rfloor$

$$= \left\lfloor \frac{1024}{15} \right\rfloor = 68 \text{ entries/blk.}$$

The no. of blk needed for the index

$$b_i = \left\lceil r_i / bfr_i \right\rceil = \left\lceil \frac{30000}{68} \right\rceil = 442 \text{ blks}$$

No. of blk accesses (binary search) $= \lceil \log_2 b_i \rceil$

$$= \lceil \log_2 442 \rceil = 9 \text{ block accesses.}$$

One additional blk access to data file

$9 + 1 = 10$ blk access

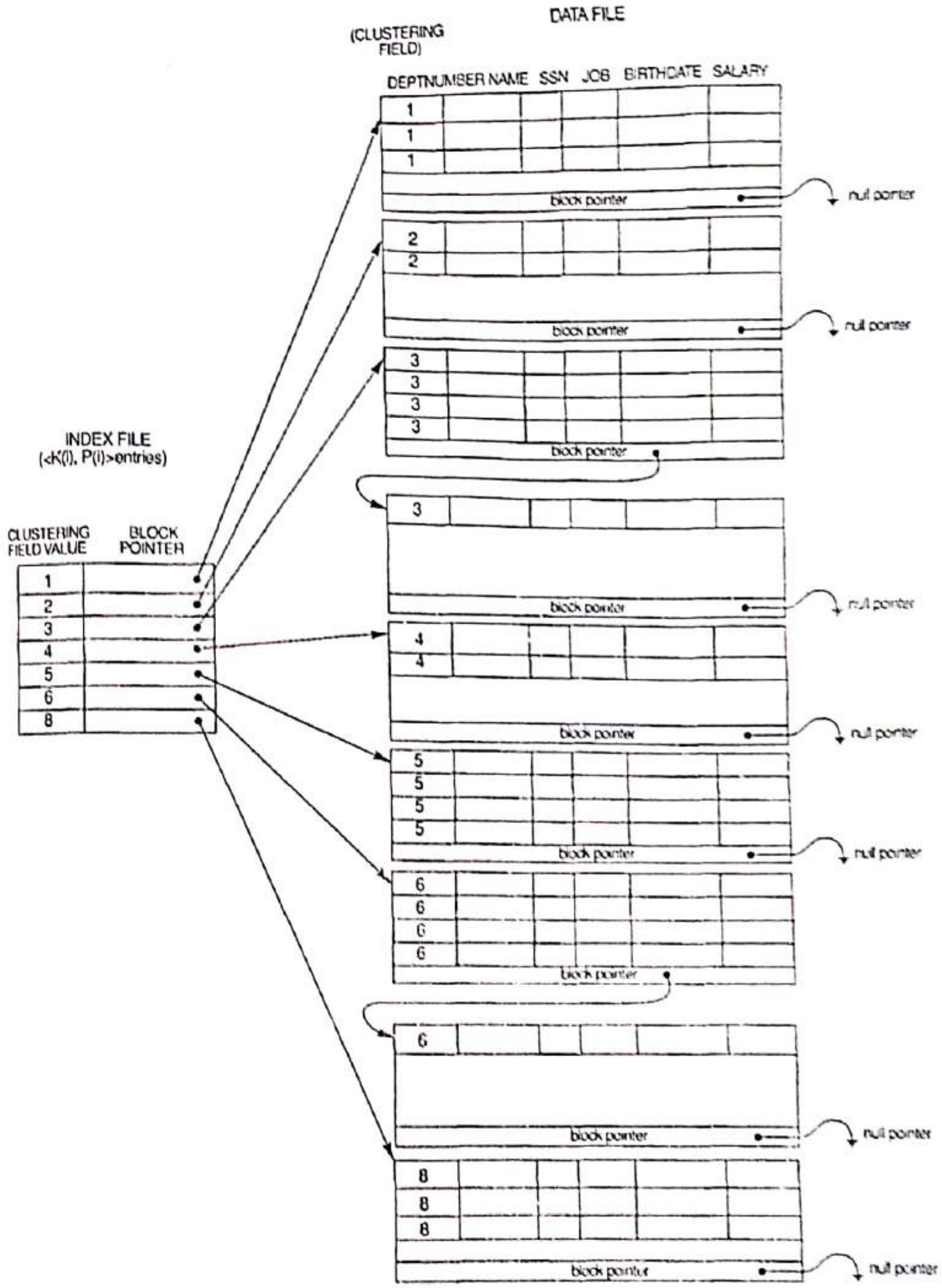


FIGURE 14.3 Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

Scanned with CamScanner

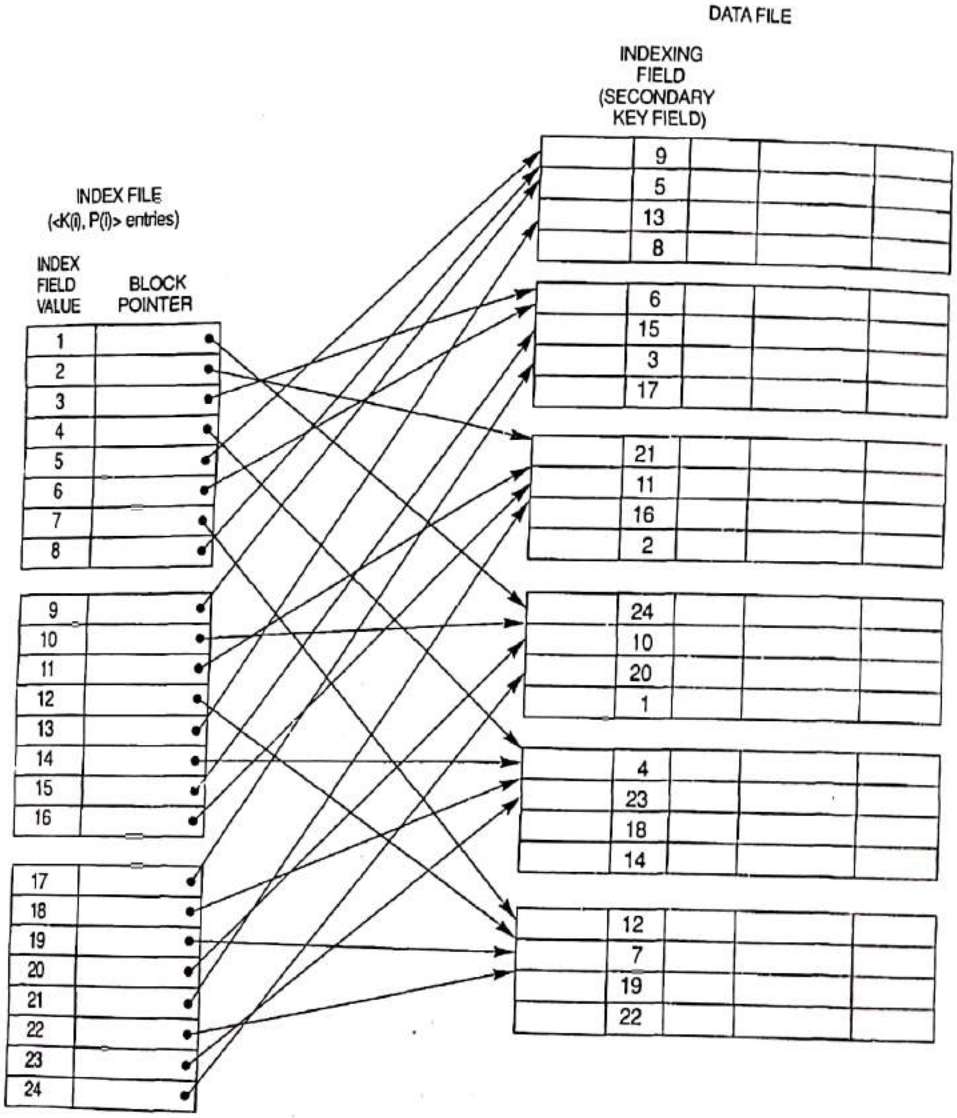


FIGURE 14.4 A dense secondary index (with block pointers) on a nonordering key field of a file.

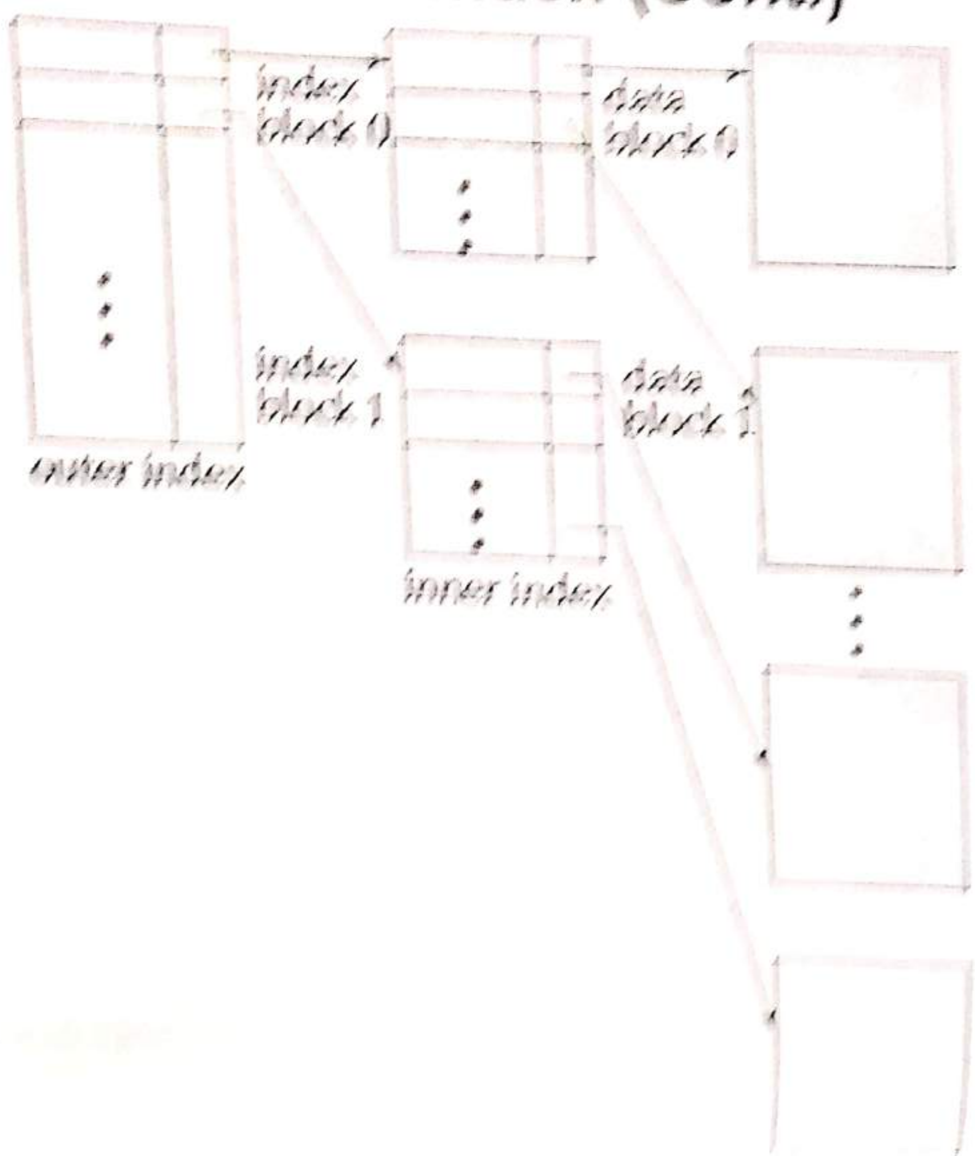
We can also create a secondary index on a *nonkey field* of a file. In this case, numerous records in the data file can have the same value for the indexing field. There are several options for implementing such an index:

- Option 1 is to include several index entries with the same K(i) value—one for each record. This would be a dense index.

Multilevel Index

- If primary index does not fit in memory, access becomes expensive.
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of primary index
  - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

Multilevel index (Cont.)



Scanned with CamScanner

Multilevel Indexes

Fig 6.5

Multilevel indexing reduces the search space. The value of bfr_i is called the fan-out of the multilevel index (f_0). The index file refer to as first level of the multilevel index. We can create a primary index for the first level; this index to the first level is called the second level of the multilevel index. Second level has one entry for each blk of the first level.

First level $bfr_i = f_0 \ (r_1)$

Second level $r_2 = \lceil r_1/f_0 \rceil$

Third level $r_3 = \lceil r_2/f_0 \rceil$

No. of levels $t = \lceil \log_{f_0} (r_1) \rceil$

Eg: $bfr_i = 68$ index entries/block. $b_1 = 442$ blks.

No. of second level blks $b_2 = \lceil b_1/f_0 \rceil$

$= \lceil 442/68 \rceil = \underline{\underline{7 \text{ blks}}}$

" Third level $b_3 = \lceil b_2/f_0 \rceil$

$= \lceil 7/68 \rceil = 1$ block.

~~t log~~ t (no. of levels) $= 3$

No. of blk accesses $= 3+1 = \underline{\underline{4}}$

~~ii) Blking factor for Index records =~~

B^+ Trees

B^+ tree, data ptrs are stored only at the leaf nodes of tree. The structure of leaf node is diff. from internal nodes.

The leaf nodes have an entry for every value of the search field, along with a data ptr to the record if the search field is a key field. For a nonkey search field, the ptr points to a blk containing ptrs to the data file records, creating an extra level of indirection.

The leaf nodes of the B^+ tree are usually linked together to provide ordered access on the search field to the records. The leaf nodes are similar to the first level of an index.

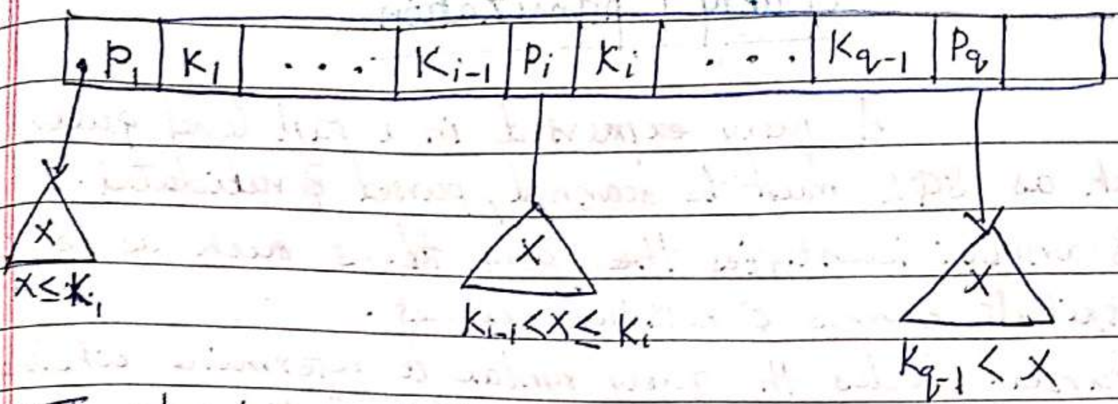
The structure of the internal nodes of a B^+ tree of order p is as follows:

1. Each internal node is of the form

$$\langle P_1, K_1, P_2, K_2 \dots, P_{q-1}, K_{q-1}, P_q \rangle$$

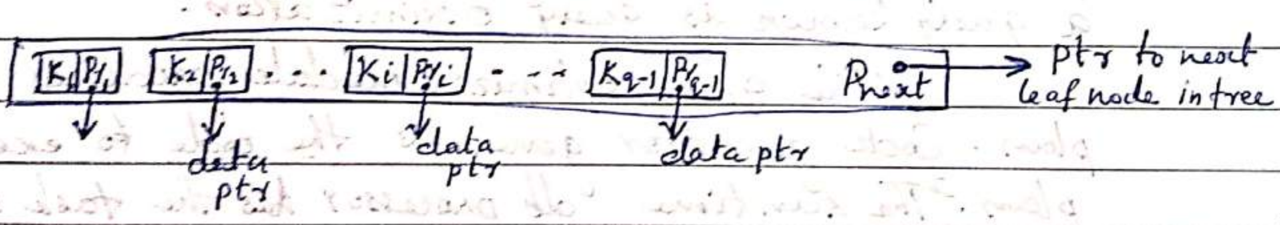
where $q \le p$ and P_i is a tree ptr.

2. Within each internal node, $K_1 < K_2 < \dots K_{q-1}$
3. For all search field values X in the subtree pointed at by P_i, we have $K_{i-1} < X \le K_i$ for $1 < i < q$; $X \le K_i$ for $i = 1$; & $K_{i-1} < X$ for $i = q$
4. Each internal node has at most p tree pointers.
5. Each internal node, except the root, has atleast $\lceil P/2 \rceil$ tree ptrs. The root node has atleast 2 tree ptrs if it is an internal node.
6. An internal node with q ptrs, $q \le p$ has $q-1$ search field values.



The structure of the leaf nodes of a B^+ tree of order p.

1. Each leaf node is of the form
$\langle\langle K_1, Pr_1\rangle, \langle K_2, Pr_2\rangle, \ldots, \langle K_{q-1}, Pr_{q-1}\rangle, P_{next}\rangle$
where $q \leq p$, each Pr_i is a data ptr and P_{next} pts to the next leaf node of the B^+ tree.
2. Within each leaf node, $K_1 < K_2 < \ldots < K_{q-1}$, $q \leq p$.
3. Each Pr_i is a data ptr that pts to the record whose search field value is K_i or to a file blk containing the record.
4. Each leaf node has at least $\lceil (p/2) \rceil$ values.
5. All leaf nodes are at the same level.



Query Optimization

A query expressed in a high level query lang. such as SQL must be scanned, parsed & validated.

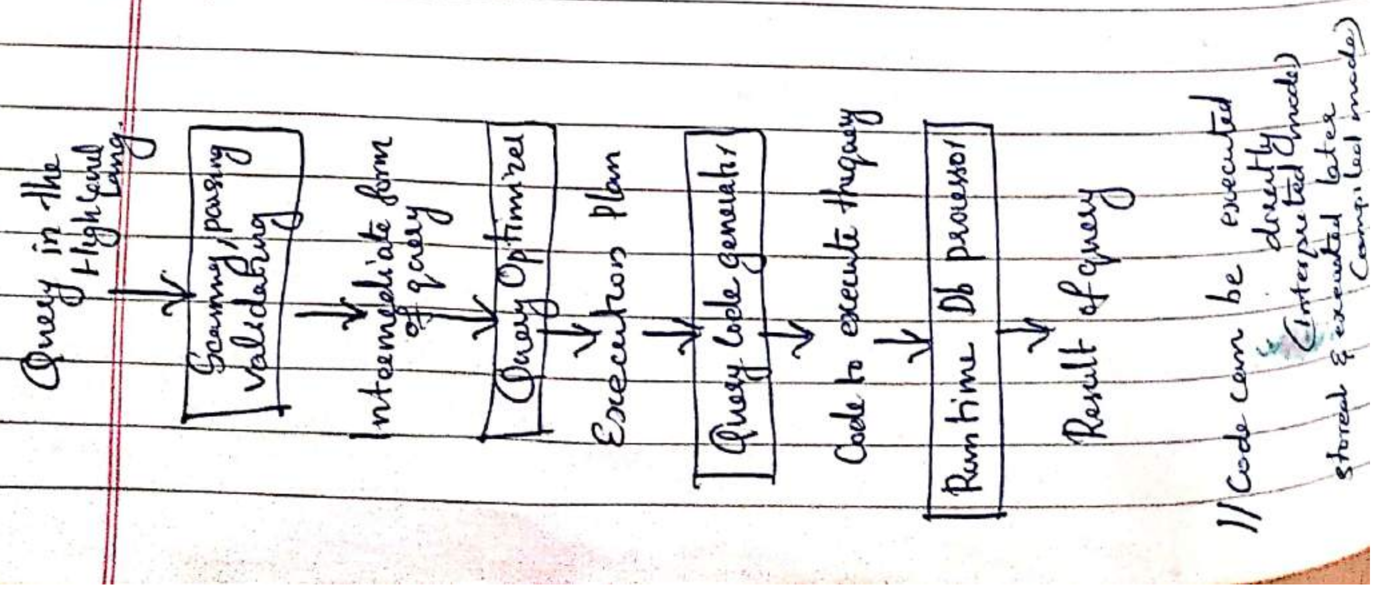
Scanner: identifies the lang. tokens such as SQL keywords, attribute names & relation names.

Parser: checks the query syntax to determine whether it is formulated acc. to the syntax rules of the query lang.

Validator: checks that all attribute & relation names are valid & semantically meaningful names in schema of the particular db being queried.

An internal repn of the query is then created as a tree data structure called a query tree. It is also possible to represent the query using a graph data structure called a query graph. The DBMS then devise an exn strategy for retrieving the result of the query from the db files. A query typically has many possible execution plans & the process of choosing a suitable one for processing a query known as query optimization.

The query optimizer module produces execution plan. Code generator generates the code to execute that plan. The runtime db processor has the task of running the query code to produce query result.



Translating SQL Queries into Relational Algebra.

An SQL query is first translated into an eqt. extended relational algebra expression. It is represented as a query tree data structure. SQL queries are then decomposed into query blks, which form the basic units that can be translated into the algebraic operators & optimized.

A query blk contains a single Select-From-Where expression as well as group by & having clauses if these are part of the blk.

Eg: select LName, FName
from Employee
where Salary > (select max(salary)
from Employee
where DNo = 5);

Decomposed into 2 blks:

The inner blk: (select max(Salary)
from Employee
where DNo = 5)

Outer block: select LName, FName
from Employee
where salary > C

where C represents the result returned from inner blk.

Relational Algebra: Inner blk: $\mathcal{F}_{max\_salary}(\sigma_{DNo=5}(Employee))$

outer blk: $\pi_{Lname, FName}(\sigma_{salary>c}(Employee))$.

The query optimizer then choose an ex' plan for each blk. The inner blk needs to be evaluated only once to produce the max. salary which is then used by the outer blk.

Heuristics in Query Optimization

→ Techniques that apply heuristic rules to modify the internal repⁿ of a query which is in the form of a query tree or a query graph data structure to improve its expected performance.

One of the main heuristic rules is to apply select and project operations before applying the join or other binary opⁿs. This is b/c the size of the file resulting from a binary opⁿ such as JOIN is usually a multiplicative fn of the sizes of i/p files. The Select & Project opⁿs reduce the size of a file & hence should be applied before a join or other binary opⁿ.

Notation for Query Trees & Query Graphs.

A query tree is a tree data structure that corresponds to a relational algebra expression. It represents the i/p rel's of the query as leaf nodes of the tree and represents the relational algebra opⁿs as internal nodes. An exⁿ of the query tree consists of executing an internal node opⁿ whenever its operands are available & then replacing that internal node by the relation that results from executing the opⁿ. The exⁿ terminates when the root node is executed & produces the result relation for the query.

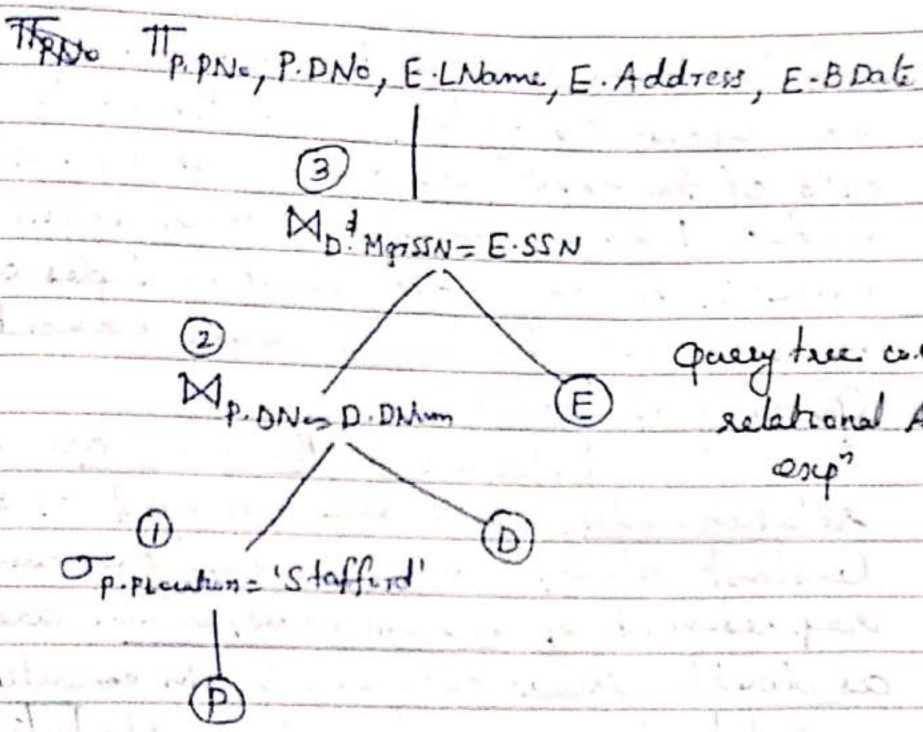
Eg: For every project located in 'Stafford', retrieve the project number, the controlling dept no, & the dept mgr's lastname, address & birth date.

Relational Algebra expⁿ:

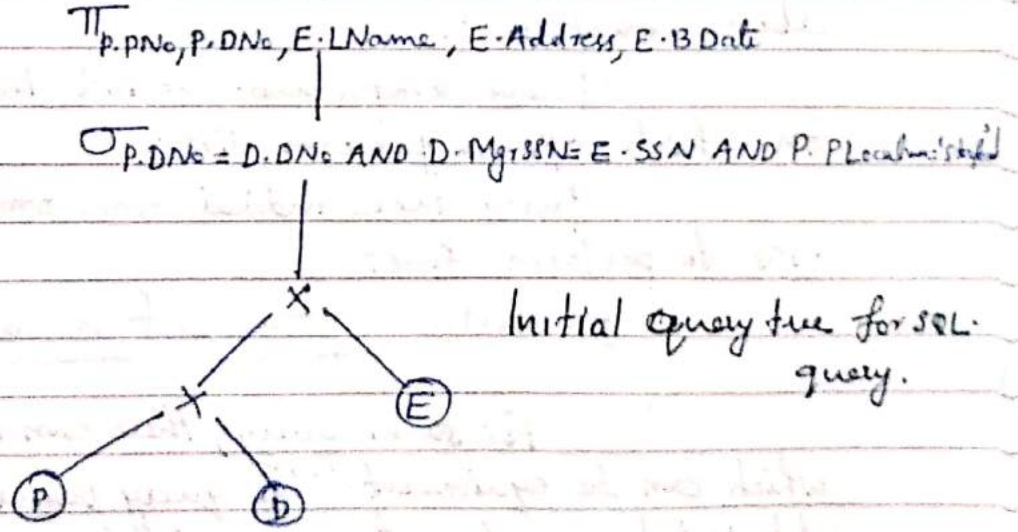
$$\pi_{PNo, DNo, LName, Address, BDate}(((\sigma_{Plocation='Stafford'}(Project)) \bowtie_{DNo=DNum} (Department)) \bowtie_{mgrSSN=SSN} (Employee))$$

SQL query:

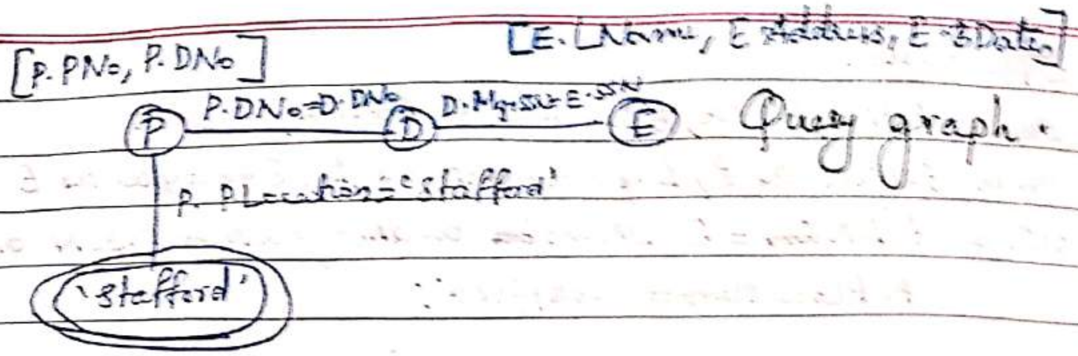
select P.PNo, P.DNo, E.LName, E.Address, E.Bdate
from Project as P, Department as D, Employee as E
where P.DNum = D.DNumber and D.MgrSSN = E.SSN and
P.PLocation = 'Stafford';



Query tree corresp. relational Algebra exp^n



Initial query tree for SQL query.



Query graph.

Fig 1a. Three relations Project, Department & Employee are represented by leaf nodes P, D, E. The rel^nal algebra op^ns of the exp^n are represented by internal tree nodes. The node marked (1) must begin ex^n before node (2) b/c some ~~set~~ resulting tuples of op^n (1) must be available before begin executing op^n (2).

Query graph notation:-

Relations in the query are represented by relation nodes, which are displayed as single circles. Constant values, from query selection conditions are represented by constant nodes, which are displayed as double circles. Selection & join conditions are represented by the graph edges. The attributes to be retrieved from each relation are displayed in square brackets above each rel^n.

Query graph rep^n doesn't indicate an order on which op^ns to perform first.

Query tree indicate the order on which op^ns to perform first.

Heuristic Optimization of Query Trees.

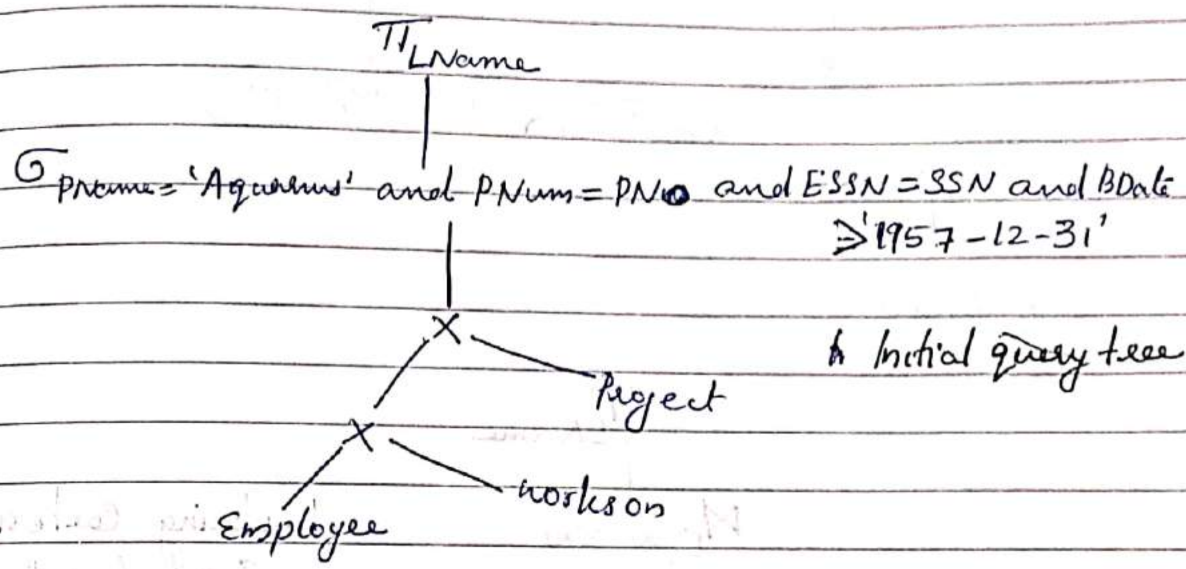
For same query, there can be many diff. queries which can be equivalent. The query parser generate std initial query tree to correspond to an SQL query without doing any optimization.

Heuristic query optimizer transform the initial query tree into a final query tree.

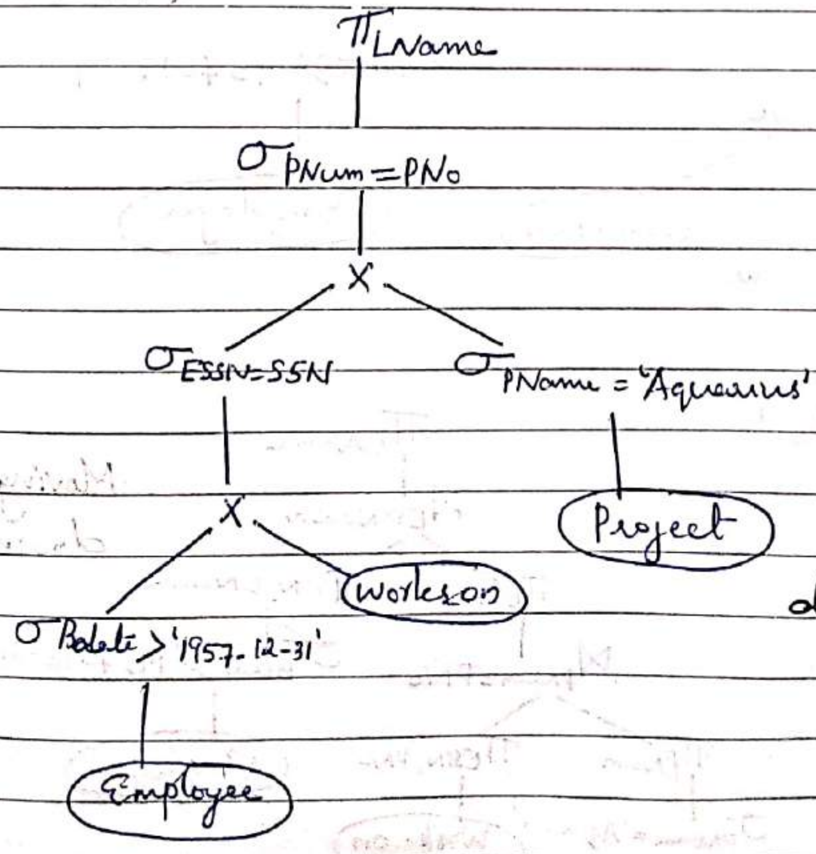
Example of Transforming a Query:

Eg: Find the last names of employees born after 1957 who work on a project named 'Aquarius'.

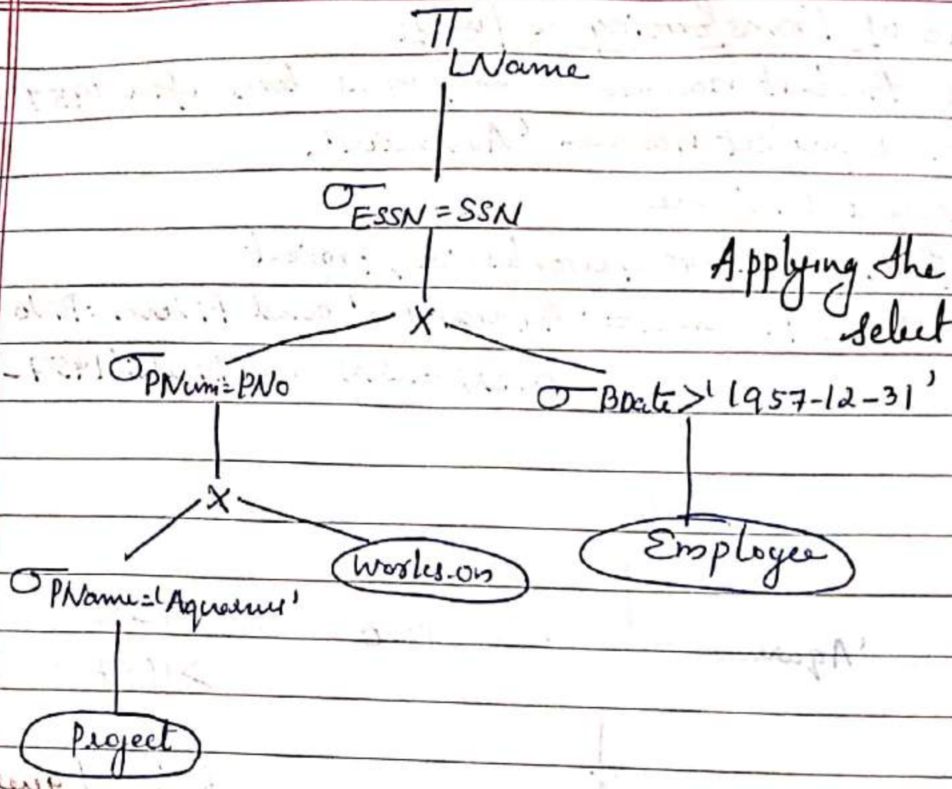
SQL: select LName
from Employee, works-on, Project
where PName = 'Aquarius' and PNum = PNo and ESSN = SSN and BDate > '1957-12-31';



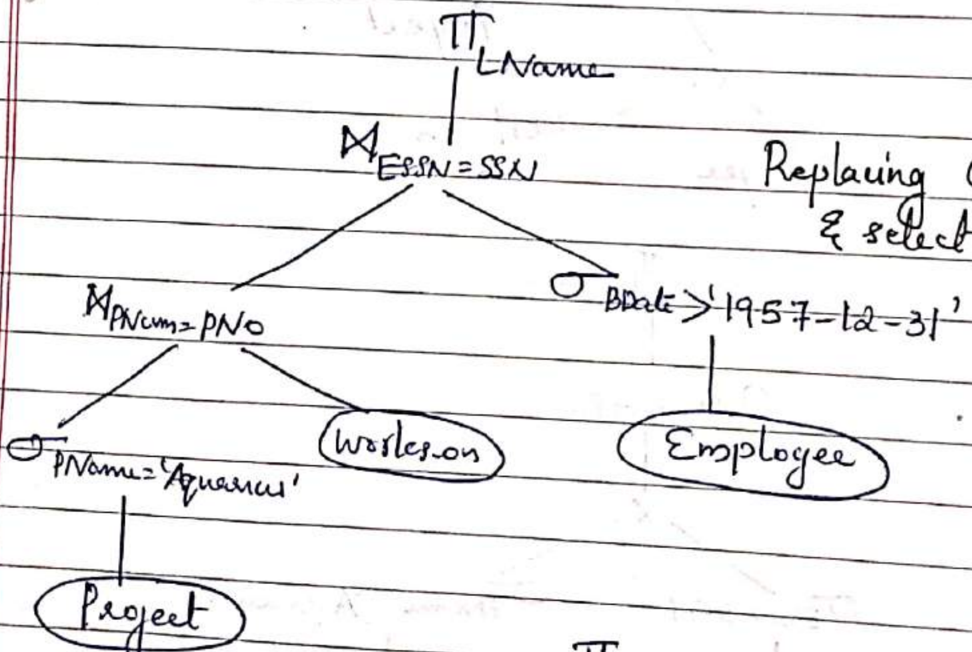
Initial query tree.



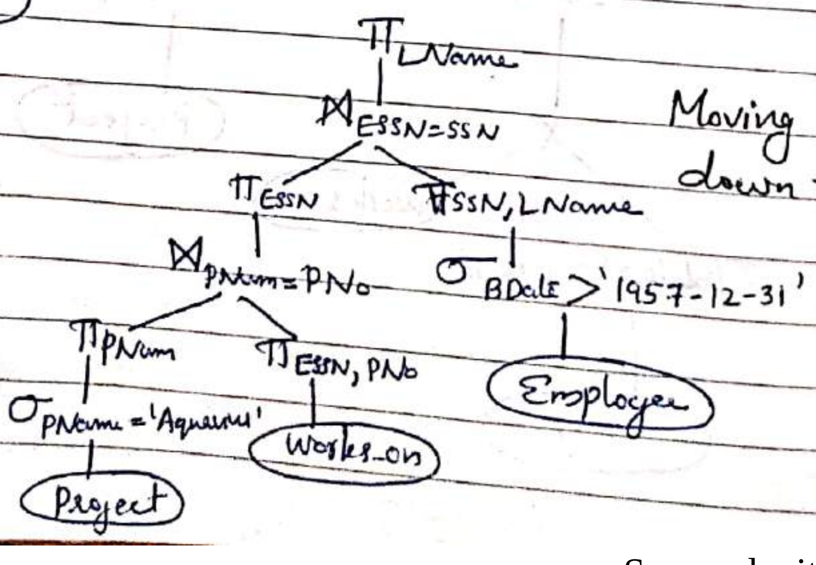
Moving Select op^n's down the query tree.



Applying the more restrictive select Op^n first.



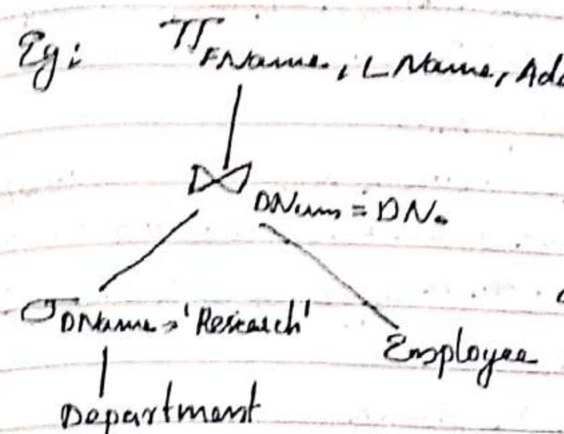
Replacing Cartesian Pdt & select with Join Operations.



Moving Project Op's down the query tree.

Converting Query Trees into Query Execution Plans

Eg:



To convert this into an exⁿ plan, the optimizer might choose an index search for the select opⁿ, a table scan as access method for Employee, a nested loop join algm for the join, and a scan of the join result for the project operator. In addition, the approach taken for executing the query may specify a materialized or a pipelined evaluation.

Materialized evaluation: The result of an opⁿ is stored as a temporary relation (physically materialised).
For eg: the join opⁿ can be computed & the entire result stored as a temporary relation, which is then read as i/p by the algm that computes the project opⁿ, which would produce the query result table.

Pipelined evaluation: As the resulting tuples of an opⁿ are produced, they are forwarded directly to the next opⁿ in the query sequence. For eg:, as the selected tuples from Dept are produced by the select opⁿ, they are placed in a buffer; the join opⁿ algm consume the tuple from the buffer & those tuples that result from join opⁿ are pipelined to the projection opⁿ algm.

The adv. of pipelining is the cost savings in not having to write the intermediate results to disks & not having to read them back for next opⁿ.

Transformation Rules for Relational Algebra Operations:

1. Cascade of σ : $\sigma_{c_1 \text{ AND } c_2 \text{ AND} \ldots \text{AND } c_n} \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots \sigma_{c_n}(R))$
2. Commutativity of σ : The σ operation is commutative
   $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R_1))$
3. Cascade of π : $\pi_{List_1}(\pi_{List_2}(\ldots(\pi_{List_n}(R))\ldots)) \equiv \pi_{List_1}(R)$
4. Commuting σ with π : ~~If the selection condition c involves only those attributes $A_1, \ldots A_n$ in~~ $\pi_{A_1, A_2 \ldots A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2 \ldots A_n}(R))$
5. Commutativity of $\bowtie$ (and X):
   $R \bowtie_c S \equiv S \bowtie_c R$, $R \times S \equiv S \times R$
6. Commuting σ with $\bowtie$ (or X) : $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$
7. Commutativity of set operations: The set operations $\cup$ & $\cap$ are commutative but '$-$' is not.
8. Associativity of $\bowtie$, X, $\cup$ & $\cap$: θ stands for any one of these op^{n}s
   $(R \theta S) \theta T \equiv R \theta (S \theta T)$
9. Commuting σ with set op^{n}s : $\sigma_c(R \theta S) \equiv (\sigma_c(R)) \theta (\sigma_c(S))$
10. The π op^n commutes with $\cup$:
    $\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$
11. Converting a (σ, X) sequence into $\bowtie$: $(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

21/3/18

classmate

Date
Page

MODULE VI

→ Transaction Processing Concepts:

* Overview of Concurrency Control & Recovery
* ACID Properties.
* Serial & Concurrent Schedules
* Conflict Serializability
* Two-phase locking
* Failure Classification
* Storage Structure - Stable Storage
* Log based Recovery
  - Deferred database modification
  - Check pointing.

→ Recent Topics (Preliminary ideas only)

* Semantic Web & RDF
* GIS, Biological Databases.

Questions

1. Draw a state diagram, and discuss the typical states that a transⁿ goes through during exⁿ.
2. What is the s/m log used for? What are the typical kinds of records in a s/m log?
3. Discuss ACID properties.
4. What is a schedule (history)?
5. What is a serial schedule? What is serializable schedule?
6. Define the violations caused by each of the following: dirty read, nonrepeatable read, Lost update problem.
7. Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules?

a) $r_1(X); r_3(X); w_1(X); r_2(X); w_3(X);$

b) $r_1(X); r_3(X); w_3(X); w_1(X); r_2(X);$

c) $r_3(X); r_2(X); w_3(X); r_1(X); w_1(X);$

d) $r_3(X); r_2(X); r_1(X); w_3(X); w_1(X);$

8. Consider the 3 trans^{n}s T_1, T_2 & T_3 and the schedules S_1 & S_2 given below. Draw the serializability (precedence) graphs for S_1 & S_2 and state whether each schedule is serializable or not. If a schedule is serializable, write down the eqt. serial schedules!

T_1 : $r_1(X)$; $r_1(Z)$; $w_1(X)$;

T_2 : $r_2(Z)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;

T_3: $r_3(X)$; $r_3(Y)$; $w_3(Y)$;

S_1 : $r_1(X)$; $r_2(Z)$; $r_1(Z)$; $r_3(X)$; $r_3(Y)$; $w_1(X)$; $w_3(Y)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;

S_2 : $r_1(X)$; $r_2(Z)$; $r_3(X)$; $r_1(Z)$; $r_2(Y)$; $r_3(Y)$; $w_1(X)$; $w_2(Z)$; $w_3(Y)$; $w_2(Y)$;

9. Illustrate generic structure of a B^+ tree.

10. Illustrate clustering index & secondary index with typical real examples?

11. Argue two phase locking ensures serializability.

12. What is the significance of checkpointing? with eg:

13. Illustrate lost update pblm & dirty read pblms with eg.

14. Determine if the follo. schedule is serializable.
$r_1(X)$; $r_2(Z)$, $r_1(Z)$, $r_3(X)$, $r_3(Y)$, $w_1(X)$, $w_3(Y)$, $r_2(Y)$ $w_2(Z)$, $w_2(Y)$

15. Write a small RDF document and show its eqt. graph structure.

16. List out any 3 salient features of Big Data.

17. How is CMS dbs diff. from conventional dbs?

Transaction Processing Concepts:

Transaction is a mechanism for describing logical units of db processing. Transaction processing s/ms are s/ms with large dbs and hundreds of concurrent users that are executing db transactions. Eg: s/ms for reservations, banking, credit card processing, stock markets etc.

Transaction includes insertion, deletion, modification or retrieval operations. ~~B~~ Transaction boundaries are begin-transaction & end transaction statements. If the db opⁿs in a transaction do not update the db but only retrieve data, the transaction is called a read only transaction.

A database is basically represented as a collⁿ of named data items. The size of a data item is called its granularity. The basic database access operations that a transaction can include are as follows:

read_item(X): Reads a db item named X into a pgm variable.

write_item(X): Writes the value of pgm variable X into the db item named X.

Eg: Two Sample Transactions: T_1 & T_2

T_1	T_2
read item(X);	read_item(X)
X = X − N;	X = X + M
write_item(X);	write_item(X);
read_item(Y);	
Y = Y + N;	
write_item(Y);	

The Lost Update Problem: This pblm occurs when 2 transⁿs that access the same db items have their opⁿs interleaved in a way that makes the value of some db items incorrect

T_1	T_2
read_item (X);	
$X := X - N$	
	read_item (X);
	$X := X + M$;
write_item (X);	
read_item (Y);	
	write_item (X); ← Item X has an incorrect value b/c its update by T_1 is "lost" (overwritten).
$Y := Y + N$;	
write_item (Y);	

T_1 & T_2 submitted at same time; their op^n's are interleaved. For eg: $X = 80, N = 5, M = 4$

T_1 transfers 5 seat reservations from X to Y; T_2 — 4 seat reservations

80 reservations initially.

∴ The final result should be

$T_1 \quad X = 80 - 5 = 75$

$T_2 \quad X = 75 + 4 = \underline{\underline{79}}$

But as per interleaved schedule

$T_1: X = 80 - 5 = 75$

$T_2 \quad X = 80 + 4 = 84$

$T_1: \text{write}(X) \rightarrow 75$

$T_2: \text{write}(X) \rightarrow 84$

So final X will be 84 incorrect.

The temporary Update (or Dirty Read) Problem:

The pblm occurs when one trans^n updates a db item and then the trans^n fails for some reason. The updated item is accessed by another trans^n before it is changed back to its original value. In Example. T_1 updates item X and then fails before completion, so the s/m must change back to its original value.

Transaction T_2 reads the "temperary" value of X which will not be recorded permanently in the db b/c of failure of T_1. The value of item X that is read by T_2 is called dirty data. This pblm is also known as dirty read problem.

Eg:

T_1	T_2
read_item(X);	
X:= X-N;	
write_item(X);	
	read_item(X);
	X:= X+M;
	write_item(X);
read_item(Y);	

Trans^n T_1 fails & must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X.

The Incorrect Summary Problem: If one trans^n is calculating an aggregate summary fn on a no. of records while other transactions are updating some of these records, the aggregate fn may calculate some values before they are updated and others after they are updated.

T_1	T_2
	sum:= 0;
	read_item(A);
	sum= sum+A;
	⋮
read_item(X);	
X:= X-N;	
write_item(X);	
	read_item(X);
	sum:=sum+X;
	read_item(Y);
	sum := sum +Y;
read_item(Y);	
Y:=Y+N;	
write_item(Y);	

← T_3 reads X after N is subtracted & reads Y before N is added; a wrong summary is the result (off by N).

Unrepeatable read: where a trans^n T reads an item twice and on the item is changed by another trans^n b/w 2 reads. Hence T recieves diff. values for its 2 reads of the same item.

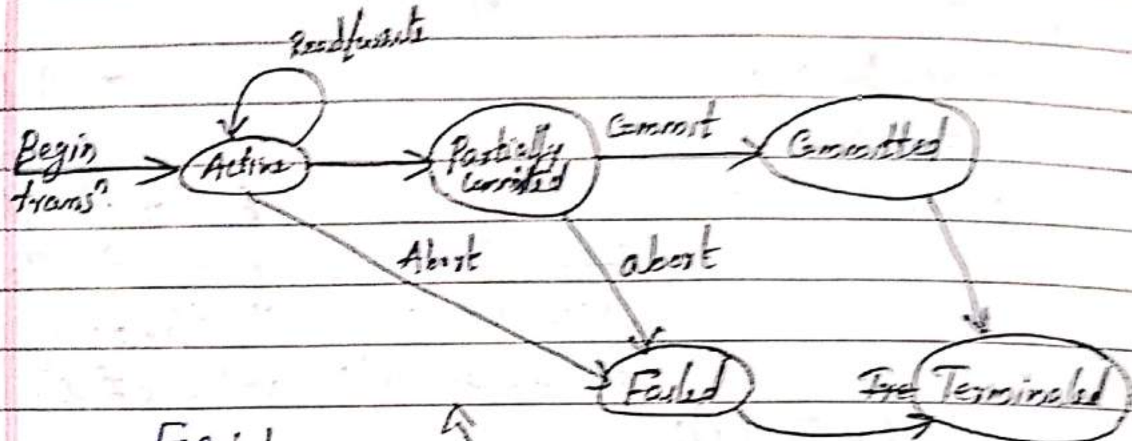


Fig: 1
Transaction States

* Begin_transaction: This marks the begining of trans^n ex
* Read or write: specify read/write op^n on db items.
* End-transaction: Read/write trans^n op^ns have ended
* Commit-transaction: Successful end of the transaction so that any changes executed by the trans^n can be safely committed to the database and will not be undone
* Rollback (or Abort): transaction has ended unsuccessfully so that any changes or effects that the trans^n may have applied to the db must be undone.

ACID Properties

Atomicity: A trans^n is an atomic unit of processing it is either performed in its entirety or not performed at all.
Consistency: A transaction is consistency preserving if its complete ex^n take the db from one consistent state to another
Isolation: The ex^n of a trans^n should not be interfered with by any other trans^ns executing concurrently.
Durability: The changes applied to the db by a committed

trans^n must persist in the db. These changes must not be lost b/c of any failure.

Serial & Concurrent Schedules:

A schedule S of n trans's $T_1, T_2 \cdots T_n$ is an ordering of the op^ns of the trans's subject to the constraint that, for each trans^n T_i that participates in S, the op^ns of T_i in S must appear in the same order in which they occur in T_i.

Eg:

T_1	T_2
read-item(X);	
X:= X-N	
	read-item(X);
	X:= X+M;
write-item(X);	
read-item(Y);	
	write-item(X);
Y:= Y+N;	
write-item(Y);	

Schedule S

$S: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$

Two op^ns in a schedule are said to conflict if they satisfy all three of the follo. cond^ns:
1) they belong to diff. transactions
2) they access the same item X;
3) atleast one of the op^ns is a write_item(X).

A schedule S of n trans's $T_1, T_2, \cdots T_n$ is said to be a complete schedule if the following cond^ns hold:

1. The op^ns in S are exactly those op^ns in $T_1, T_2 \cdots T_n$, including a commit or abort op^n as the last op^n for each trans^n in the schedule.
2. For any pair of op^n from the same trans^n T_i, their order of appearence in S is the same as their order of appearence in T_i.
3. For any 2 conflicting op^ns, one of the 2 must occur

before the ~~order~~ other in the schedule.

Serial & Concurrent Schedules:

Serial Schedule:

– op^n s of each trans^n are executed consecutively without any interleaved op^n s from the other transaction.

schedule (a)

T_1	T_2
read_item(X);	
X = X-N;	
write_item(X);	
read_item(Y);	
Y = Y+N;	
write_item(Y);	
	read_item(X);
	X := X+M;
	write_item(X);

Schedule A

Schedule (b)

T_1	T_2
~~read_item(X)~~	read_item(X);
	X := X+M;
	write_item(X);
read_item(X);	
X := X-N;	
write_item(X);	
read_item(Y);	
Y := Y+N;	
write_item(Y);	

Schedule B

Non serial schedules (Concurrent) with interleaving of op^n s:

(3)

T_1 T_j	T_2 T_i
read_item(X);	
X = X-N;	
	read_item(X);
	X := X+M;
write_item(X);	
read_item(Y);	
	write_item(X);
Y := Y+N;	
write_item(Y);	

Schedule C.

T_1	T_2
read_item(X);	
X := X-N;	
write_item(X);	
	read_item(X);
	X := X+M;
	write_item(X);
read_item(Y);	
Y := Y+N;	
write_item(Y);	

Schedule D.

Conflict Serializability

A schedule S of n transactions is serializable if it is eql. to some serial schedule of the same n transactions. Hence Serializability of schedules is used to identify which schedules are correct when transaction ex^n have interleaving their op^n's in the schedules.

Eg:

T_1	T_2	
	read(A)	Initially $A = 1000, B = 2000$
	$t = A * .1$	$t = 100$ $A = 900$
	$A = A - t$	$B = 2100$
	write (A)	Total — $2100 + 900$
	read(B)	$= 3000$
	$B = B + t$	
	write (B)	
read(A)		
$A = A - 50$	$A = 850$	Schedule (2)
write(A)	$B = 2150$	
read(B)	Total $= 850 + 2150$	
$B = B + 50$	$= 3000$	
write (B)		

T_1	T_2	
read (A)		
$A = A - 50$		
write(A)		
read(B)		Schedule (1)
$B = B + 50$		
write (B)		
	read (A)	
	$t = A * .1$	
	$A = A - t$	
	write (A)	$A = 845$
	read (B)	$B = 2145$
	$B = B + t$	$845 + 2145 = 3000$
	write (B)	

950

$= 2050$

$950 + 2050 = 3000$

T1	T2
read(A)	
A = A−50	
write(A)	
	read(A)
	t = A*·1
	A = A−t
	write(A)
read(B)	
B = B+50	
write(B)	
	read(B)
	B = B+t
	write(B)

A=950
B=2050
A+B=3000

A=855
B=2145
855+2145=3000

Schedule 3

Schedule 3 = Schedule I

So Schedule 3 is serializable.

Conflict Schedules:

Two opⁿs in a schedule are said to conflict if they satisfy all three of the follo. conditions:
1) they belong to diff. transⁿs.
2) They access same item.
3) at least one of the opⁿ is a write-item (x).

Two schedules are said to be conflict eqt. if the order of any 2 conflicting opⁿ is the same in both schedules. For eg: if a read & write opⁿ occur in the order $r_1(X), w_2(X)$ in schedule S_1 & in the reverse order $w_2(X), r_1(X)$ in schedule S_2, the value read by $r_1(X)$ can be diff. in the 2 schedules.

A schedule S to be conflict serializable, if it is equivalent to some serial schedule S'

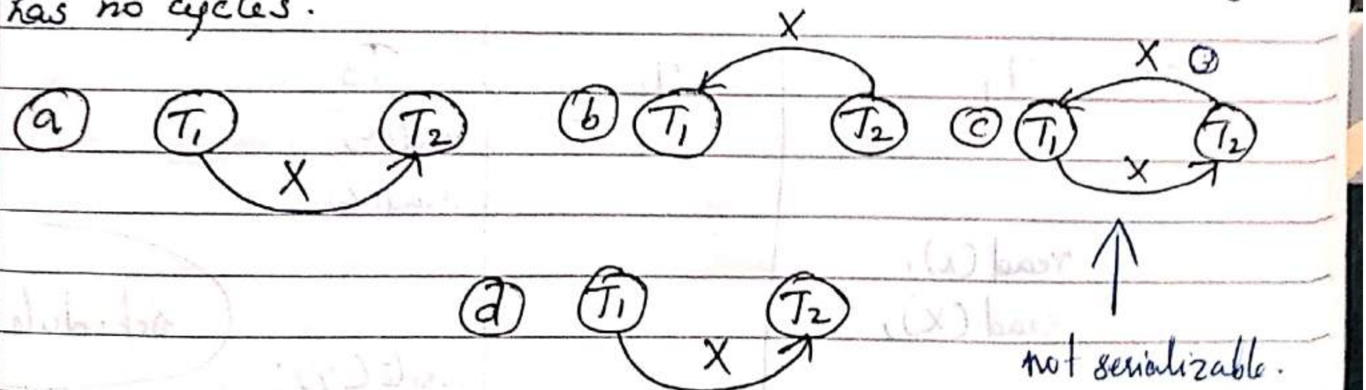
Eg: Schedule D & Schedule A

Testing for Conflict Serializability of Schedule.

The algm looks at only the read-item & write-item op's in a schedule to construct a precedence graph or (serialization graph), which is directed graph $G = (N, E)$ that consists of a set of nodes $N = \{T_1, T_2, \cdots T_n\}$ & a set of directed edges $E = \{e_1, e_2, \cdots e_m\}$.

<u>Algorithm:</u>

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a ~~write~~ read_item (X) after T_i executes a write_item (X), create an edge $(T_i \to T_j)$ in the precedence graph.
3. For each case in S where T_j executes write_item (X) after T_i executes a read_item (X), create an edge $(T_i \to T_j)$ in the precedence graph.
4. For each case in S where T_j executes a write_item (X) after T_i executes write_item (X), create an edge $(T_i \to T_j)$ in the precedence graph.
5. The schedule S is serializable iff the precedence graph has no cycles.



Eg.

Transaction T_1	Transaction T_2	Transaction T_3
read(X);	read(Z);	read(Y);
write(X);	read(Y);	read(Z);
read(Y);	write(Y);	write(Y);
write(Y);	read(X);	write(Z);
	write(X);	

The read & write opⁿˢ of 3 transⁿˢ ↑

b)

Transaction T_1	T_2	T_3
	read(Z);	
	read(Y);	
	write(Y);	
		read(Y);
		read(Z);
read(X);		
write(X);		
		write(Y);
		write(Z);
	read(X);	
read(Y);		
write(Y);		
	write(X);	

Schedule E

c)

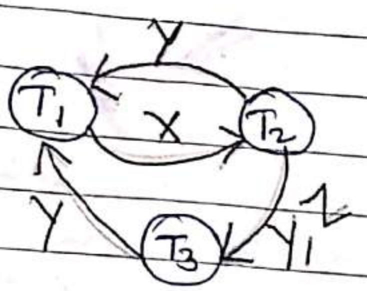
T_1	T_2	T_3
		read(Y);
		read(Z);
read(X);		
read(X);		
		write(Y);
		write(Z);
	read(Z);	
read(Y);		
write(Y);		
	read(Y);	
	write(Y);	
	read(X);	
	write(X);	

Schedule F

Serializability testing:

Schedule E Precedence Graph.

Equivalent Serial Schedule: None



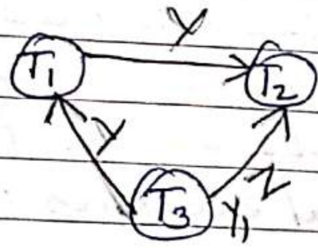
Cycle: X($T_1 \to T_2$) Y($T_2 \to T_1$)

X($T_1 \to T_2$), YZ($T_2 \to T_3$) Y($T_3 \to T_1$)

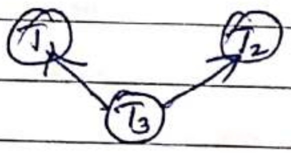
Schedule F Precedence Graph

Equivalent Serial Schedule

$T_3 \to T_1 \to T_2$



2 Equivalent Serial schedules.



$T_3 \to T_1 \to T_2$

$T_3 \to T_2 \to T_1$

Two Phase Locking

The technique of locking data items to prevent multiple trans^{n}s from accessing the items concurrently.
Another set of concurrency ctrl protocols use timestamps.
A timestamp is a unique identifier for each trans^n, generated by the s/m. It ensures serializability.
Multiversion protocols that use multiple versions of a data item. Validation/certification of a trans^n after it executes its op^{n}s — optimistic protocols.

Locking Techniques for Concurrency Control

A lock is a variable associated with a data item w.r.to possible opns that can be applied to it.

→ Types of Locks & S/m Lock Tables.

→ **Binary Locks**: A binary lock can have 2 states/values: locked & unlocked (1 & 0). Two opns, lock_item and unlock_item are used with binary locking. A transn reqst access to an item X by issuing a lock_item(X) operation. If Lock(X) = 1, the transn is forced to wait. If lock(X) = 0 it is set to 1 & the transn is allowed to access item X. When the transn is through using the item, it issues an unlock_item(X) opn which sets Lock(X) to 0. A binary lock ensures mutual exclusion on the data item.

Lock_item(X):

```
B: if Lock(X)=0 ("Item is unlocked")
   then lock(X)←1 ("lock the item)
   else begin
      wait (until lock(X)=0 and
         the lock mngr wakes up the transn);
   goto B
   end;
```

Unlock_item(X):

```
Lock(X) ← 0; ("unlock the item")
If any transns are waiting then
wakeup one of the waiting transn.
```

Binary locking s/m rules:

1. A transⁿ T must issue the opⁿ lock_item (X) before any read_item (X) or write_item(X) opⁿs are performed in T.

2. A transⁿ T must issue the opⁿ unlock_item (X) after all read_item (X) and write_item (X) opⁿs are completed in T.

3. A transⁿ T will not issue a lock_item (X) opⁿ if it already holds the lock on item X.

4. A transⁿ T will not issue an unlock item (X) opⁿ unless it already holds the lock on item X.

→ Shared / Exclusive (or Read/write) Locks:

Multiple-mode lock: shared/exclusive or read/write locks: there are 3 locking opⁿs: read_lock (X), write_lock(X) & unlock(X).

Lock Compatibility Matrix:

	S	X
S	True	false
X	false	false

Eg: T_1

Lock X(B)
read B
B = B - 50
write(B)
unlock(B)
lock X(A)
read (A)
A = A + 50
write (A)
unlock(A)

Concurrency ctrl Mngr gives the grant on lock on data item as per the compatibility matrix.

* Conversion of Locks: A transⁿ that already holds a lock on item X is allowed under certain condⁿs to convert the lock from one locked state to another. For eg:, it is possible ~~to~~ for a transⁿ T to issue a read_lock(X) & then later on to upgrade the lock by issuing a write_lock(X) opⁿ. If T is the only transⁿ holding a read lock on X at the

time it issues the write_lock(X) opⁿ, the lock can be upgraded; otherwise transⁿ must wait.

It is also possible for a transⁿ T to issue write lock & then later on to downgrade the lock by issuing a read-lock(X) opⁿ.

Two-Phase Locking

A transaction is said to follow the 2 phase locking protocol if all locking opⁿs (read-lock, write-lock) precede the first unlock opⁿ in the transaction. Such a transⁿ ~~&~~ can be devided into 2 phases:

→ expanding/growing phase, during which new locks on items can be acquired but none can be released

→ Shrinking phase during which existing locks can be released but no new locks can be acquired.

If lock conversion is allowed, then upgrading of locks must be done during the expanding phase, & downgrading of locks must be done in the shrinking phase.

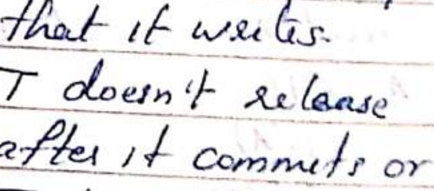
Conservative 2PL: requires transⁿ to lock all the items it accesses before the transⁿ begins exⁿ, by predeclaring its read set & write set. The readset of a transⁿ is the set of all items that the transⁿ reads, & the write set is the set of all items that it writes.

Strict 2PL: A transⁿ T doesn't release any of its Xlocks (write lock) until after it commits or aborts

Rigorous 2PL: Transⁿ T doesn't release any of its locks (X lock or shared) until after it commits or aborts.

Deadlock & Starvation

T_1	T_2
lock X (B)	
read (B)	
B = B − 50	
write (B)	
	lock S (A)
	read (A)
	lock S (B)
lock X (A)	



wait for graph.

classmate
Date
Page

Failure Classification

Whenever a transⁿ is submitted to a DBMS for exⁿ, the system is responsible for making sure that either

1) all the opⁿs in the transaction are completed successfully & their effect is recorded permanently in the db.

2) the transⁿ has no effect whatsoever on the db or on any other transⁿs.

Types of Failures:

→ Transaction failure → Local errors
→ System Crash → Concurrency ctrl enforcement
→ Disk failure → Physical Pblms & catastrophes.

→ Computer failure (System Crash):
A h/w or, s/w or n/w error occurs in the comp. s/m during transⁿ execution. Eg: M/m failure.

→ Transaction or System error:
Some opⁿ in the transⁿ may cause it to fail, such as integer overflow or division by zero. Transⁿ failure may also occur b/c erroneous parameter values or b/c of a logical Pgroming error.

→ Local errors or exception conditions detected by the transaction: During transⁿ exⁿ, certain conditions may occur that necessitate cancellation of the transaction. For eg: data for the transⁿ may not be found.

→ Concurrency ctrl enforcement: The concurrency ctrl method may decide to abort the transⁿ, to be restarted later, b/c it violates serializability or b/c of a disk read/write head crash.

→ Disk Failure: Some disk blks may lose their data b/c of read/write malfunction or read/write head crash.

→ Physical pblms: Eg: power failure, A/c failure, fire, theft etc.

Log based Recovery:

Deferred Db Modification:

Log: The data structure for recording db modification. Log is a sequence of log records, There are several types of log records.

Eg: An update log record describes a single db write. It has the follo. fields:

* Transn identifier: is a unique identifier of trans that performed the write opn.

* Data Item Identifier: is a unique identifier of the data item written:

* Old value: is the value of data item prior to write

* New value is the value that the data item will have after the write.

Other special records: <Ti, start>, <Ti, Xj, V1, V2>
<Ti Commit>, <Ti, abort>

Deferred Db Modification:

The deferred db modification technique ensures transn atomicity by recording all db modification in the log, but deferring the exn of all write opn of transaction until the transn partially commits the infn on log is written into the db. This is known as deferred writes. If the s/m crashes, before the transn completes its exn or if transn aborts then the infn on the log is simply ignored. Using the log, the s/m can handle any failure that results in the loss of infn on volatile storage. The recovery scheme uses the follo. recovery procedure.

redo (Ti) sets the value of all data items updated by transn Ti to the new values.

The set of data items updated by T_i & their respective new values can be found in the log.
Redo is performed iff the log contains both the record <T_i start> & the record <T_i Commit>. If the s/m crashes after the transn completes its exn, the recovery scheme uses the infn in the log to restore to a previous consistent state after the transn had completed.

— If <T_i Commit> is not present then the log records are deleted.

a	b	c
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
X	<T_0, Commit>	<T_0, Commit>
	<T_1, start>	<T_1, start>
	<T_1, C, 600>	<T_1, C, 600>
	X	<T_1 Commit>
		X

A = 1000
B = 2000
C = 500

Consider (a) crash occurs after write(B) When the s/m comes back up no redo actions need to be taken since no commit record appears in the log. Then A=1000 & B = 2000.

Consider (b) crash occurs after write(C), then redo (T_0) is performed. Since <T_0, Commit>. But redo(T_1) is not performed.

Consider (c) crash occurs after <T_1 Commit>. Then redo (T_0) & redo(T_1) are performed.

A = 950, B = 2050, C = 600

Deferred Update in a Single-User Environment

The algm RDU-S (Recovery using Deferred Update in a Single-user environment) uses a REDO procedure, for redoing certain write-item op^ns; it works as follows:

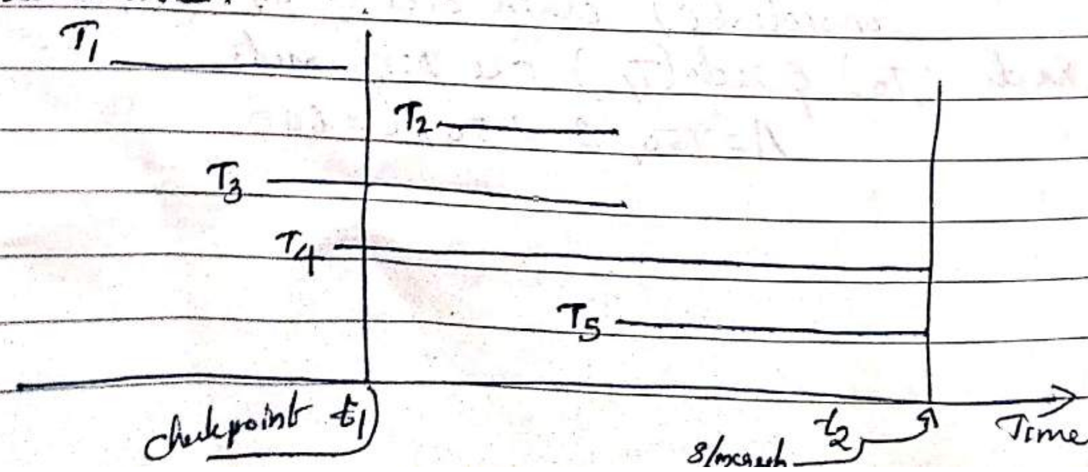
Procedure RDU-S: Use 2 lists of trans^ns: the committed transactions since the last checkpoint & active trans^ns. Apply the Redo op^n to all the write-item op^ns of the committed trans^ns from the log in the order in which they were written to the log. Restart the active trans^ns.

The redo procedure:

Redo(write-op): Redoing a write-item op^n write-op consists of examining its log entry [write-item, T, X, new value] & setting the value of item X in the db to new value, which is the after Image.

Deferred Update with ~~Concurr~~ Concurrent Execution in ~~the~~ a Multiuser Environment

Procedure RDU-M (with checkpoints): Use 2 lists of trans^ns maintained by the s/m: the committed trans^ns T since the last checkpoint (Commit list), & the active trans^ns T' (active list). Redo all the WRITE op^ns of the committed trans^ns from the log, in the order in which they were written into the log. The trans^ns that are active and didn't commit are effectively canceled & must be resubmitted.



At t_1, ⟹ T_1 committed, T_3 & T_4 not committed.
At t_2, ⟹ T_3 & T_4 Committed, but not T_4 & T_5.
At t_2, s/m crash;

T_1 — no need to redo b/c there is no need to redo write_item op^ns of transⁿ T_1 - or any transⁿs committed before the last checkpt time t_1.

T_2 & T_3 must be redone b/c both transⁿs reached their commit pts after the last checkpt.

T_4 & T_5 are ignored: They are cancelled b/c none of their write_item op^ns were recorded in the db under this protocol.

This method's main benefit is that transⁿ op^ns never need to be undone, for 2 reasons:

1. A transⁿ doesn't record any changes in the db on disk until after it reaches its commit point - ie; until it completes its exⁿ successfully.
2. A transⁿ will never read the value of an item that is written by an uncommitted transⁿ, b/c items remain locked until a transⁿ reaches its commit point.

Checkpointing

Another type of entry in the log is called a checkpoint. A [checkpoint] record is written into the log periodically at that point when the s/m writes out to the db on disk all DBMS buffers that have been modified. All transⁿs that have their <commit, T> entries in the log before a <checkpoint> entry do not need to have their write op^ns redone in case of a s/m crash, since all their updates will be recorded in the db on disk during checkpointing. The recovery mngr of a DBMS must decide at what intervals to take a checkpoint.
Taking a checkpoint consists of the follo. actions:

1. Suspend exn of transns temporarily.
2. Force-write all main mem. buffers that have been modified to disk.
3. Write a <checkpoint> record to the log, & force write the log to disk.
4. Resume executing transns.

Fuzzy Checkpointing

The time needed to force-write all modified memory buffers may delay transn processing b/c of step 1. To reduce this delay, it is common to use a technique called fuzzy checkpointing. In this technique, the s/m can resume transn processing after the <check point> record is written to the log without having to wait step 2. to finish. Until step 2 is completed, the previous <check point> record should remain to be valid. The s/m maintains a pointer to the valid checkpoint, which continues to point to the previous <checkpoint> record in the log. Once the step 2 is concluded, that ptr is changed to point to the new check point in the log.

BFIM: Before Image: The old value of the data item before updating.

AFIM: After Image ⇒ The new value of data item after updating.

Write Ahead Logging: The recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry & that the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the db on the disk

Scanned with CamScanner

Imp?

Geographic Information Systems (GIS)

→ GIS are used to collect, model, store & analyze infⁿ describing phy. properties of the geographical word. The scope of GIS leads to 2 types of data

* Spatial data, originating from maps, digital images, administrative & political boundaries, roads, transportⁿ n/ws, phy. data as rivers, soil characteristics, climatic regions, land deviations

* non spatial data such as census counts, sales or marketing infⁿ.

GIS Applications: 3 categories

- Cartographic
  - Irrigation
  - Crop yield
  - Land evaluation
  - Planning & facilities mgmt
  - Landscape studies
  - Traffic pattern analysis.
- Digital Terrain Modeling Applⁿs
  - Earth Science resource studies
  - Civil engg.
  - Military evaluatn
  - Soil Surveys
  - Air & water pollution
  - Flood ctrl
  - Water resource mgmt
- Geographic Objts Applⁿs:
  - Car navigation S/ms
  - Geographic Market Analysis
  - Utility distribution & Consumption
  - Consumer pdt & services.

Data Mgmt Rqmnts of (GIS)

* Data Modeling & Representation: GIS can be represented in 2 formats – 1) Vector: represents geometric objts such as points, lines & polygons.

2) Raster:– array of points, where each point represents the value of an attribute for a real world location.

– n dimensional images.

– 2 dimensional – pixels, 3 dimensional units voxels.

Data Analysis: GIS have various types of analysis.
Eg: geomorphometric analysis - such as slope values, gradients, profile convexity, contours & other parameters
Data Integration: GIS must integrate both vector & raster data from variety of sources. Sometimes edges & regions are inferred from a raster image to form a vector model.
Data Capture: Capture 2 or 3 dimensional geographic infⁿ in digital form.

Specific GIS Data Operations:

* Interpolation: This process derives elevation data for pt points at which no samples have been taken.
* Interpretation: Digital terrain modeling involves the interpretation of opⁿs on terrain data such as editing, smoothing, reducing details & enhancing.
* Proximity analysis: include computations of "zones of interest", around objs such as the determination of a buffer around a car on a highway.
* Raster Image Processing: 2 categories:
  - Map algebra, which is used to integrate geographic features on diff. map layers.
  - Digital Image analysis: deals with analysis of a digital image features such as edge detection & object detection.
* Analysis of Networks: * Extensibility * Data quality etc.
* Visualization.

Problems & Future Issues in GIS:

* New Architectures * Versioning & obj life cycle approach
* Data Standards * Matching appl's & data structures.
* Lack of semantics in data structures.

Biological Databases

Biological data exhibits many special characteristics that make mgmt of biological inf^n.
Bioinformatics: addresses information mgmt of genetic inf^n with special emphasis on DNA sequence analysis.
Characteristics:

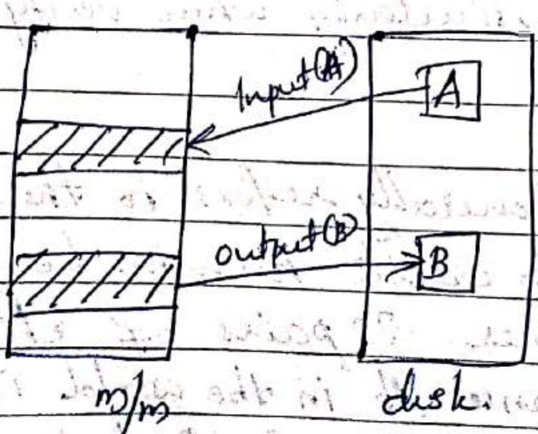
1. Biological data is highly complex when compared with most other domains. Eg: DNA & RNAs.
2. The amount & range of variability in data is high.
3. Schemas in biological databases change at a rapid pace.
4. Representations of the same data by diff. biologists will likely be different (even when using the same s/m).
5. Most users of biological data do not require write access to the database; read only access is adequate.
6. Most biologists are not likely to have any knowledge of the internal structure of the db or about schema design.
7. The context of data gives added meaning for its use in biological applications.
8. Defining & representing complex queries is extremely imp. to the biologist.
9. Users of biological inf^n often require access to "old" values of the data - particularly when verifying previously reported results.

Human Genome: Generally refers to the complete set of genes required to create human being (100000 to 300000 genes spread over 23 pairs of chromosomes)
Genbank: DNA sequence db in the world is Genbank maintained by the National Centre for Biotechnology Inf^n of National Library of Medicine (NLM).

The Genome Database (GDB): (1989) is a catalog of human gene mapping data, a process that associates a piece of infⁿ with a particular locⁿ on the human genome.

Storage Structure

Storage Structure: Volatile storage: Information residing in volatile storage doesn't usually survive s/m crashes. Eg: m/m & crash memory.
→ Access - fast & direct.
Non volatile storage: survives s/m crashes. Eg: disk & magnetic tapes.
— slower.
Stable Storage information residing in stable storage is never lost.
Data Access: Db s/m resides permanently on nonvolatile storage & is partitioned into fixed length storage units called blks.
Two main opⁿs: Input & output operations:
The blks residing in m/m are referred to as buffer blks & the blks residing in disk are referred to as physical blks.
The area of memory where blks reside temporarily called the disk buffer.
Block movements b/w disk & m/m:
1. Input(B): transfers the physical blk B to m/m.
2. Output (B): transfers the buffer blk B to the disk & replaces the appropriate physical blk there.



2 operations:

1) read(X) assigns the value of data item X to the local variable x_i. It executes this operation as follows:

a) If blk B_X on which X resides is not in m/m, it issues input(B_X).

b) It assigns to x_i the value of X from the buffer blk.

2) Write(X) assigns the value of local variable x_i to data item X in the buffer blk. It executes this op^n as follows:

a) If blk B_X on which X resides is not in m/m it issues ~~but~~ input(B_X).

b) It assigns value of x_i to X in the buffer blk.

16/4/18

CONTENT BEYOND SYLLABUS/ADVANCED TOPICS

1	Information Retrieval Query languages And their brief description
2	Latest tools used for ER diagram

1. Information Retrieval Query languages and their brief description

Information search and retrieval involves finding out useful documents from a store of information. In any information search and retrieval system an important factor which plays a role is search and selection process. Information Retrieval System (IRS) allow to find useful documents from a large volume of information by giving a query to the IRS. Information search can be made by presenting a query through the inter-mediator, or directly to the IRS. A query in general terms is a statement or series of statements made by a user to a retrieval system for the purpose of specifying what information is to be retrieved and in what form. Most of the time the query is specified in format such as artificial language hence it is called query language. A query language is the means by which the user tells the IRS what to do and what is wanted. A query is distinct from the types of documents that the user is trying to retrieve. The document and the query undergo parallel processes within the retrieval system. On the document side, someone generates or gathers some data and formulates it into a document. After creating documents they are transferred into internal representation, which then gets transferred into a format that is used for matching process. On the query side, user begins with information needs. There are two broad types of query language procedural and non-procedural or descriptive. A procedural language uses commands. If the query is written in a typical procedural query language often known as command language, little or no knowledge is required for the IRS to find what was asked for and retrieve. Natural language queries or non-procedural queries generally tend to be ambiguous in syntax and meaning. For natural language queries inter-mediator is required to formulate a query as these queries generally tend to be ambiguous. Command language queries are more structured and for IRS these queries are unambiguous.

2. Latest tools used for ER diagram:

Database Design is a collection of processes that facilitate the design, development, implementation, and maintenance of database management systems (DBMS). Properly designed databases help you to improve data consistency for disk storage.

There are a wide range of software that helps you to design your database diagrams with ease. These database design tools can be used to create a physical model or ERD of your database so that you can quickly create tables and relationships.

Following is a list of Database Diagram Design Tools, with their popular features

1.Dbdiagram.io

Dbdiagram.io is a simple database design tool to draw ER (Entity Relationship) diagrams by just writing code. It is one of the free erd tools designed for developers and data analysts.

2.SqlDBM

SqlDBM is one of the best database diagram design tools that provides an easy way to design your database on any browser. You do not require any other database engine or database modeling tools or apps to use this program.

3.Dbdesigner.net

Dbdesigner.net is an online database schema design and modeling tool. This database diagram tool allows you to create a database without wiring a single SQL code.

4.Visual Paradigm

Visual Paradigm is a database design and management tool. This database diagram tool helps the product development team to build applications faster.