Database Management System (3-1-3)

Evaluation:

	Theory	Practical	Total
Sessional	30	20	50
Final	50	-	50
Total	80	20	100

Objectives:
The objective of this course is to provide fundamental concept, theory and practices in design and implementation of DBMS.

Course Contents:

1. Introduction **(4 hrs)**
- 1.1 Concept and applications
- 1.2 Objectives and Evolution
- 1.3 Needs of DBMS
- 1.4 Data abstraction
- 1.5 Data independence
- 1.6 Schema and Instances
- 1.7 Concept of DDL, DML and DCL
- 1.8 Database Manager and users

2. Data Models **(4hrs)**
- 2.1 Logical, Physical and Conceptual Model
- 2.2 E-R Model
- 2.3 Relation with UML class diagrams
- 2.4 2.4Alternate data models (Network Data Model, hierarchical Data Model)

3. Relational Model **(4 hrs)**
- 3.1 Definitions and terminology
- 3.2 Structure of relational databases
- 3.3 The relational algebra
- 3.4 Schema and Views
- 3.5 Data dictionary

4. Relational Database Query languages **(8 hrs)**
- 4.1 SQL – features of SQL, queries and sub-queries, Join operations, set operations and other SQL constructs
- 4.2 DDL and DML queries in SQL
- 4.3 Stored procedures
- 4.4 QBE

5. Database Constraints and Relational Database Design **(8 hrs)**
- 5.1 Introduction
- 5.2 Integrity constraints
- 5.3 Referential Integrity
- 5.4 Assertions and Triggers

5.5 Functional dependencies
5.6 Normalization and Normal Forms (1NF, 2NF, 3NF, BCNF, 4NF)
5.7 Multivalued Dependencies
5.8 Decomposition of relation schemes

6. Security **(3 hrs)**
6.1 Needs of security
6.2 Security and integrity violations
6.3 Access control
6.4 Authorization
6.5 Security and Views
6.6 Encryption and decryption

7. Query Processing **(3 hrs)**
7.1 Introduction to query processing
7.2 Equivalence of expressions
7.3 Query cost estimation
7.4 Query Optimization

8. File organization and indexing **(4 hrs)**
8.1 Disks and storage
8.2 Organization of records into blocks
8.3 File organizations - The sequential and the indexed sequential file organizations
8.4 B+ Tree index
8.5 Hash index

9. Crash Recovery **(3 hrs)**
9.1 Failure classification
9.2 Concept of log-based recovery and shadow paging
9.3 Data Backup/Recovery
9.4 Remote backup system

10. Transaction Processing and Concurrency Control **(4 hrs)**
10.1 Introduction to Transactions
10.2 ACID properties of transaction
10.3 Schedules and Serializability
10.4 Concepts of locking for concurrency control

11. Advanced Database concepts **(3 hrs)**
11.1 Object-Oriented Model
11.2 Object-Relational Model (ORM)
11.3 Distributed databases
11.4 Concepts of Data Warehouses

Laboratory:
There shall be enough laboratory exercises based on some RDBMS (like ORACLE, MS-SQL server, MySQL, etc) to complement theoretical part studied. An individual project should be given to each student. 10% of sessional marks should be allocated for evaluation for lab works and project.

CHAPTER 1.

INTRODUCTION

Date
Page No.

Database :-

↳ is a repository for collection of related data and facts.

↳ a logical collection of related data that → describes the entities and their inter relationships.
→ is designed, built and populated for a specific reason.

Data

→ is a raw, unorganized facts that need to be processed.
→ itself is meaningless and worthless.

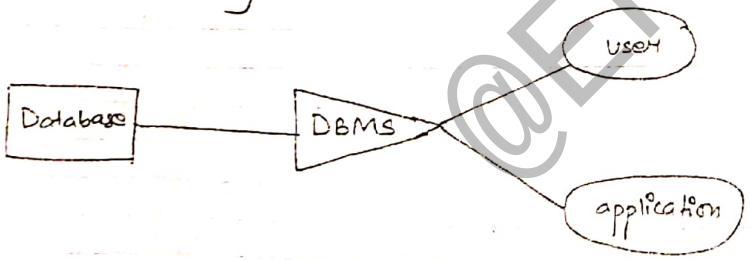
Information

→ when data is processed, organized, structured or presented in a given context so as to make it useful, is called information.
→ is output of data processing operation.

Date / /
Page No.

Database Management System (DBMS):

→ is a collections of programs that enables users to perform certain action on a particular database.
→ "define" the structure of database information.
→ "populate" the database with appropriate info.
→ "manipulate" the database.
→ "protect" the database content against accidental or deliberate corruption of contents.
→ "share" the database among multiple users.
→ DBMS is a software that defines a database, stores data, supports a query language, produce reports and creates data entry screen.
→ provide an environment that is both convinient and efficient to use in retrieving and storing database information.



Date / /
Page No. 2

Objectives of DBMS:

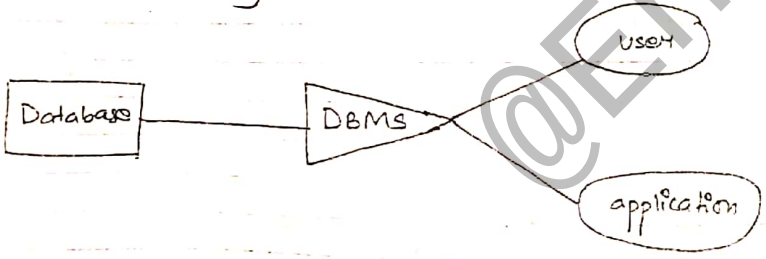
→ provide for mass storage of relevant data.
→ making easy ~~excess~~ access to data for authorized user.
→ providing prompt response to user's request for data.
→ making the latest modification visible to all database users immediately.
→ eliminate redundant data. → (duplicate)
→ allow multiple users to be active at one time.
→ allows growth of database system.
→ provide data integrity (maintain consistancy).
→ protect data from physical harm and unauthorized access.
→ serving different types of users.
→ provide security with user access privilege.
→ combining inter related data to generate report.

Normalization – to eliminate redundant data.

Date / /
Page No.

Database Management System (DBMS):

→ is a collections of programs that enables users to perform certain action on a particular database.
→ "define" the structure of database information.
→ "populate" the database with appropriate info.
→ "manipulate" the database.
→ "protect" the database content against accidental or deliberate corruption of contents.
→ "share" the database among multiple users.
→ DBMS is a software that defines a database, stores data, supports a query language, produce reports and creates data entry screen.
→ provide an environment that is both convinient and efficient to use in retrieving and storing database information.



Date / /
Page No. 2

Objectives of DBMS:

→ provide for mass storage of relevant data.
→ making easy ~~excess~~ access to data for authorized user.
→ providing prompt response to user's request for data.
→ making the latest modification visible to all database users immediately.
→ eliminate redundant data. → (duplicate)
→ allow multiple users to be active at one time.
→ allows growth of database system.
→ provide data integrity (maintain consistancy).
→ protect data from physical harm and unauthorized access.
→ serving different types of users.
→ provide security with user access privilege.
→ combining inter related data to generate report.

#Normalization – to eliminate redundant data.

Date / /
Page No.

Key Terms in db

1) field :- one piece of info of an entity in db.
2) Record :- One full set of fields
3) Table :- complete collection of records.

fields

Roll	Name	Add	Phone
1	Ram	Ktm	9841475529
2	Sita	Pkr	9860802469

records →

← table →

Application Areas of DBMS:

1) Universities
2) Airlines
3) Human resources
4) Education
5) Banking
6) e-commerce etc.

Disadvantages:

→ costly
→ fast changing technology
→ Requires trained manpower
→ chance of data leakage & hacking

Date / /
Page No. 3

Advantages of DBMS

→ Data storing
→ Reduced data redundancy
→ Improved data integrity
→ Increased security
→ Time saving
→ report generation.

Applications Areas

1) Universities : For student information, course details and grades. Examinations are done online today and universities and colleges maintain these all through DBMS
2) Airlines : For reservations and schedule information
3) Human Resource Management
Big firms have many workers working under them. Human resource management department keeps records of each employee's salary, tax and work through DBMS
4) Telecommunications :
For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards and storing information about the communication networks

Date / /
Page No.

5) Banking

We make thousands of transactions through banks daily and we can do this without going to the bank. So now banking has become so easy that by sitting at home we can send or get money through banks.

2073/09/27
Date / /
Page No. 4

Data Abstraction

- ensuring easy, smooth and efficient data structures in such a way that every type of database user is able to access its desired information efficiently.
- hides details of data storage that are not needed by most database users and applications.
- different types of data abstraction
  i) internal level (physical level)
  ii) conceptual level (logical level)
  iii) external level. (view level)

i) Internal / Physical level:

- is the lowest level of data abstraction that is closest to physical storage.
- it describes:
  → how the data are actually stored in storage devices?
  → what will be storage technique?

ii) Conceptual / logical level

→ is the next higher level
→ it describes
→ what data are actually stored in db?
→ what are relationship between data entities.

Date / /
Page No.

iii) External level or view level.
- Is closest to the users.
- It describes:
→ what is the way of viewing information to the concerned users?

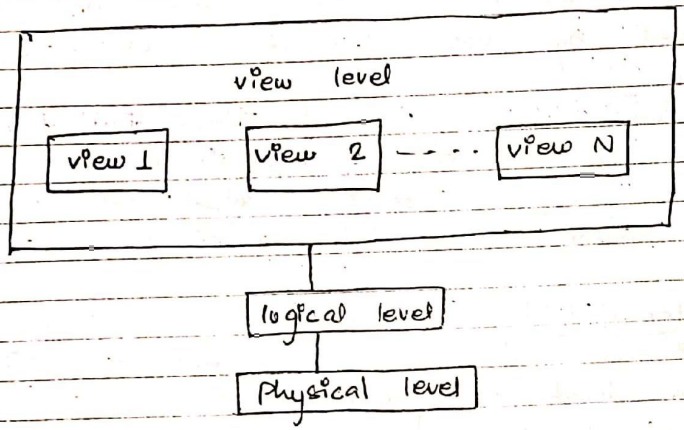


fig: three levels of data abstraction

Schema and Instances:

Schema
- is the overall design of database

Instances
- the collection of information stored in db at particular moment.

Date / /
Page No. 5

Physical Schema:
- Is database design at physical level of abstraction.

logical schema:
- Is the database design at logical level of abstraction.

Sub Schemas
- Schemas at view level that describes different view of database.

Data Independence
- the ability to modify a schema definition in one level without affecting schema definition in next high level is called data independence.

- each higher level of data architecture is immune to changes of next lower level of architecture.

- we can change the structure of a db without affecting data required by users and programs.

Date / /
Page No.

Physical Data Independence:

→ refers to ability to modify schema followed at physical level without affecting schema followed by conceptual level.

Logical Data Independence

→ refers to ability to modify conceptual schema without causing any changes in schemas at view level.

→ it is more difficult to achieve than physical data independence because application programs are dependent on logical structure of data that user access.

Concepts of DDL, DML, DCL

DDL (Data Definition Language)

→ used to define database structure or schema

e.g.

CREATE ⇒ to create objects in db
ALTER ⇒ to alter structure of db
DROP ⇒ to delete objects from db
TRUNCATE ⇒ to remove all records from table, including allocated spaces.

Date / /
Page No. 6

DML (Data Manipulation language):

→ used for managing data within schema object.

e.g.

DRL (DML) ⇐ SELECT ⇒ to retrive data from db
(Data Retrived language)
INSERT ⇒ to insert data into table
UPDATE ⇒ to update within table
DELETE ⇒ to delete records from table

DCL (Data control language)

→ used to control db

e.g.

GRANT ⇒ to give user's access privileges to db.
REVOKE ⇒ to withdraw access privileges.

TCL (Transaction control language)

– used to manage changes made by DML statements e.g.

COMMIT ⇒ save work
ROLLBACK ⇒ restore db to original since commit

CHAPTER 2: Data Models

2078/04/30

Date / /
Page No. 7

↳ Data model is a collection of conceptual tools for describing data, data relationships, semantics etc.

↳ Data model is simple representation of complex real world data structure.

↳ It is a communication tool to facilitate interaction among designer, application programmer and end user.

↳ It define how data is connected to each other and how it will be processed and stored inside the system.

3 levels of data modeling

1) Conceptual data model:
- Identifies highest level relationship between the different entities.
- It include important entities and their relationship.
- No attribute and primary key is specified

2) Logical Data Model
↳ describes the data in as much detail as possible without saying much about their physical implementation.
↳ it includes all entities and relationship among them.
↳ all attributes for each entity are specified.

Date / /
Page No

↳ primary key and foreign key are also specified.

3) Physical Data Model
→ represents how the model will be built in db.
→ shows all table structure including column name, data type, constraints, primary key, foreign key and relationship between tables.

Feature	Conceptual	logical	physical
1) Entity name	✓	✓	
2) Entity Relationship	✓	✓	
3) Attributes	–	✓	
4) primary keys	–	✓	✓
5) Foreign keys	–	✓	✓
6) Table name	–	–	✓
7) Column name	–	–	✓
8) Column datatypes	–	–	✓

This Copy Scaned by bhum
http://t.me/engineers_np

e.g:

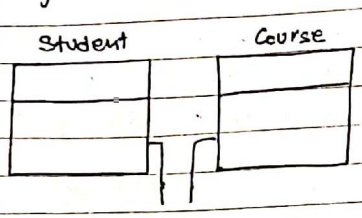


fig: 1) Conceptual Data Model

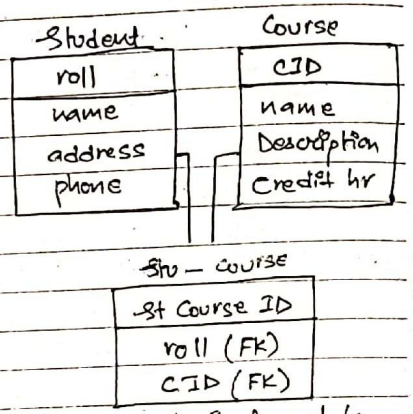


fig: 2) logical data model

tbl – Student
roll : integer pk
name : varchar (25)
address : varchar (25)
phone : varchar (20)

tbl – Course
CID : integer pk
name : varchar (20)
Description : varchar (50)
credit hr : int

tbl – Stu Course
St Course ID : integer
roll : integer FK
CID : integer FK

fig: 3) physical data model

E-R (Entity Relationship) Model: (EAR)

→ defines the conceptual view of database.
→ works around real world objects called entities and relationship among them.

Entity
→ a real world thing that is easily identifiable and distinguishable.

Entity Set
→ a collection of similar types of entities.
→ entity sets do not need to be disjoint.
→ e.g. entity set of employee and entity set of customer may have members in common.

Attributes
→ are the properties of the entities.
→ **Simple attribute:**
– which can't be divided further (e.g. phone no)
→ **Composite attribute:**
– made of more than one simple attribute. (e.g. name (∵ name ⇒ first name, last name)

→ Derived attribute
– do not exist in physical database but values are derived from other attribute in db (e.g. age)

→ Single valued attribute:
e.g: citizenship no.

→ Multivalued attribute:
e.g: phone no, email_address.

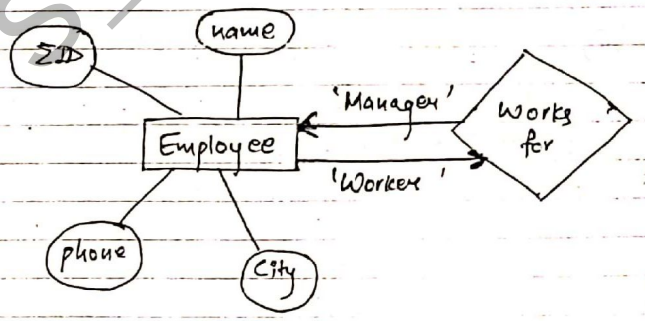
E-R (Entity-Relationship) ~~attribute~~ model:

Domain of attribute is set of permitted values
↳ formally, an attribute is a function which maps an entity set into a domain.
↳ every entity is described by a set of (attribute, datavalue) pairs.
e.g: student entity
{(Roll, 1), (name, Ram)}

Relationship
→ association among entities.
→ Relationship set is a set of relationship of same type.
→ formally, it is a mathematical relation on $n \geq 2$ entity sets.
→ if $E_1, E_2, \ldots E_n$ are entity sets then relationship set R is subset of

$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots e_n \in E_n\}$
where $(e_1, e_2, \ldots e_n)$ is relationship.

↳ The role of an entity is the function it plays in a relationship.

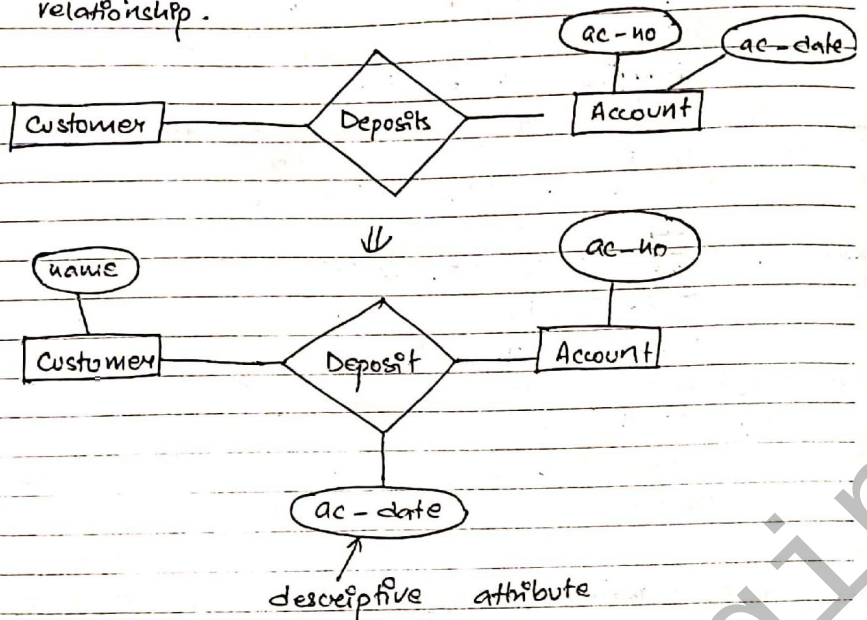


– here, 'manager' & 'worker' are roles of entity employee in 'works for' relationship.

– roles are optional and clarify semantic of a relationship.

Date / /
Page No. 10

Descriptive attributes are the attributes of relationship.



Degree of relationship

↳ It refers no. of entity sets that participate in a relationship.
↳ Unary relationship is of degree 1 that involves only one entity set.
↳ Binary relationship is of degree 2.
↳ Ternary relationship is of degree 3.
↳ Relationship in db are often binary.
↳ It is possible to replace a non binary (n-ary, $n > 2$) relationship set by number of distinct binary relationship set.

Mapping Cardinalities

↳ express the no. of entities to which another entity can be associated through a relationship set.
↳ For binary relationship set R between entity set A and B, the mapping cardinality must be one of following

1) **one to one:**
- an entity in A is associated with at most one entity in B and vice versa.

2) **one to many:**
→ an entity in A can be associated with more than one entities of entity set B but from entity set B one entity is associated with at most one entity in A.

3) **Many to one**
↳ More than one entities from entity set A can be associated with at most one entity of entity set B but one entity from entity set B can be associated with more than one entity from entity set A

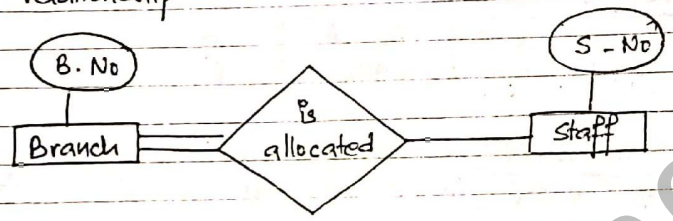
Date / /
Page No. 11

4) Many to many
— one entity from A can be associated with more than one entity from B and vice versa

Participation Constraints

↳ total participation
— if every entity in entity set E participates in at least one relationship in R.

↳ Partial participation
— if only some entities in E participate in relationship in R.



Here,
↳ every branch office is allocated members of staff i.e. total participation of entity 'branch'.
↳ A member of staff need not work at branch office i.e. partial participation of 'staff'

Date / /
Page No.

Keys
↳ one or more columns in db table that is used to sort and for identify rows in a table is called key.
↳ a key is a single or combination of fields in a table.
↳ is used to fetch or retrieve records/ data from table according to condition.
↳ different keys are:

1) Superkey
↳ is a set of one or more attributes whose value uniquely determine each entity.

2) Candidate key
↳ is a minimal superkey.

3) Primary key
↳ is a candidate key chosen as principal means of identifying entities within an entity.

4) Foreign key
↳ is a field in one table that uniquely identifies a row of another table.
↳ it is used to establish and enforce a link between two tables.

Date / /
Page No. 12

Strong and Weak entity set

	Strong entity	Weak entity
1)	It has its own prim- any key.	1) it doesn't have sufficient attributes to form a primary key of it's own.
2)	It is represented by a rectangle.	2) It is represented by double rectangle.
3)	Contains a primary key represented by underline.	3) Contains a partial key represented by dashed underline.
4)	Its member is called dominant entity set.	4) Its member is called subordinate entity set.
5)	Primary key is one of attribute that uniquely identifies its member.	5) Primary key is combination of partial key & primary key of strong entity set.
6)	relationship between two strong entity set is represented by diamond symbol.	6) relationship between one strong and one weak entity set represented by double diamond symbol.
7)	single line connects strong entity set with relation ship.	7) Double line connects weak entity set with relationship.

Date / /
Page No.
21st August 2016

ER Diagram

- Illustrates the logical structure of database.
- is a data modeling technique that creates a graphical representation of entities and the relationship between entities within an information system.
- major components are:

1) Rectangles -  ⇒ to represent entity set
2) Ellipses —  ⇒ to represent attributes
3) Diamond -  ⇒ to represent relationship
4) Lines - ——— ⇒ to link attribute to entity and entity sets to relationships.
5) Double ellipses -  ⇒ to represent multivalued attributes.
6) Dashed ellipses - ⇒ to represent derived attribute

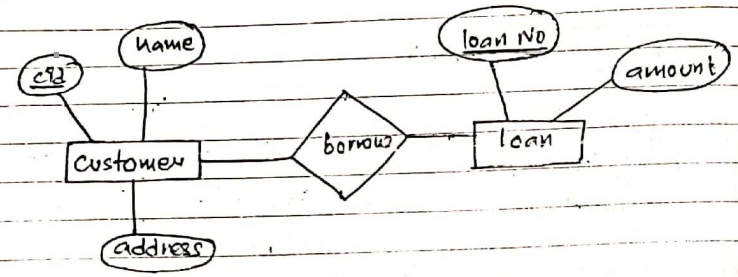
Relationship type –

Page No 13

1) Double rectangle – ▭ => to represent weak entity set.

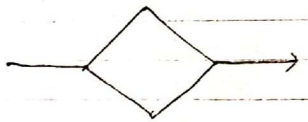
e.g:

2)



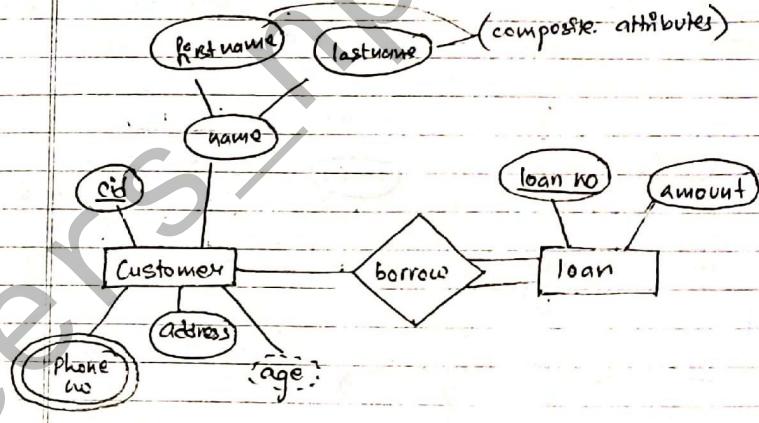
* one to one relationship

* one to many



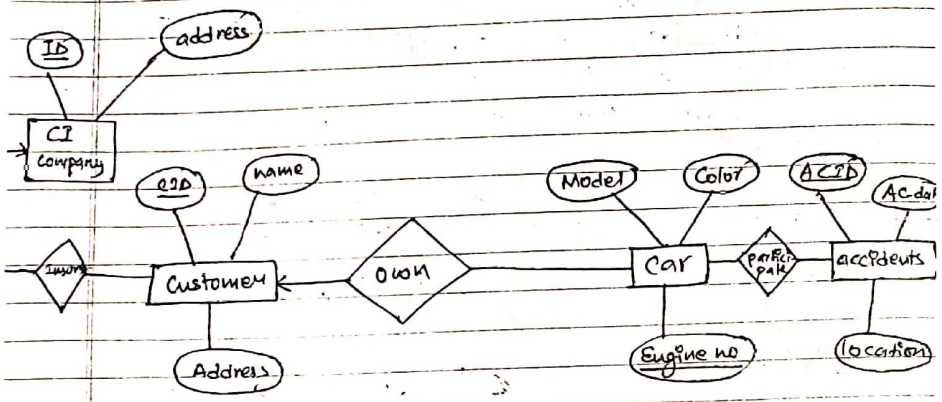
* many to one

* many to many



Example

Construct an E-R Diagram for a car – Insurance Company whose customers own one or more cars. Each car has associated with it zero to any no of recorded accidents. Assume attributes yourself.



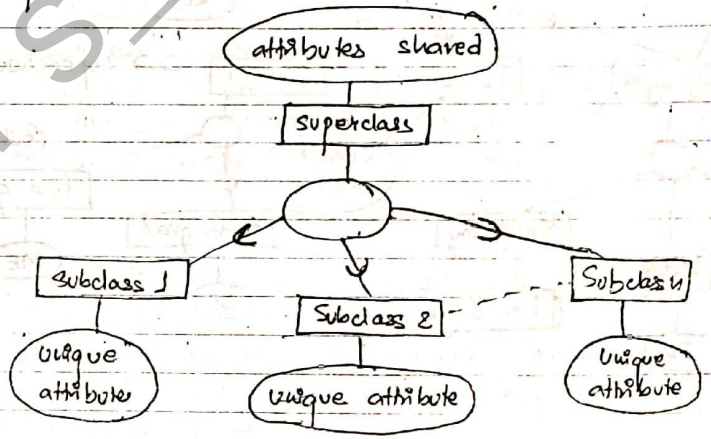
1) Library Management System
2) Hospital System.

EER Model (Extended or Enhanced ER)

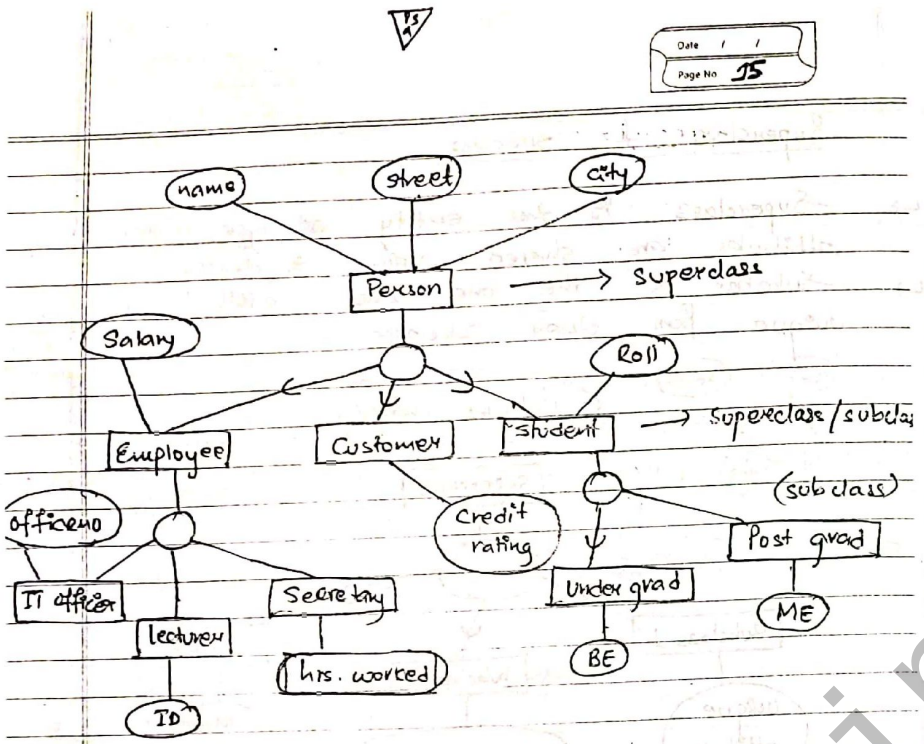
- is the ER model combined with the additional features of semantic concepts.
- shows complex relationships between objects in database.
- Some additional concepts in EER model are
  a) Specialization by Generalization
  b) categorization

Superclass & Subclass

↳ Superclass is the entity of type whose attributes are shared among subclasses.
↳ Subclass is the one whose attribut is unique for other subclasses.



e.g: Basic notation for Superclass Subclass relationship



22nd August 2016

Specialization

↳ is a means of identifying sub groups within an entity set which have unique attributes.

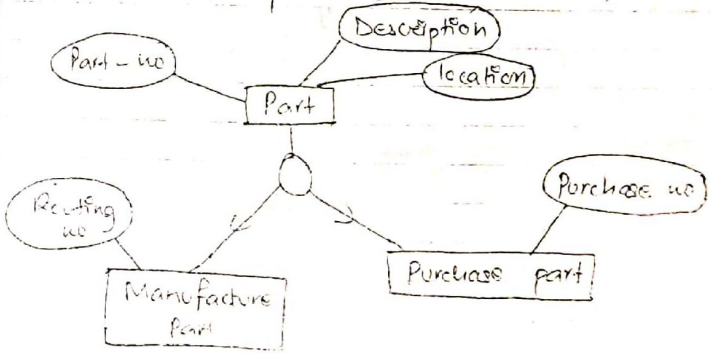
↳ is a top down approach.

↳ is a process of defining one or more subclasses from the superclass and forming superclass-subclass relationship.

↳ in EER diagram, it is represented by a triangle component labeled 'ISA' or simply circle with U symbol.

e.g:

- specialization of part as manufactured part and purchase part.
- Part (Part-no, Description, location)
- Manufactured part (Routing no)
- Purchase part (Purchase no)

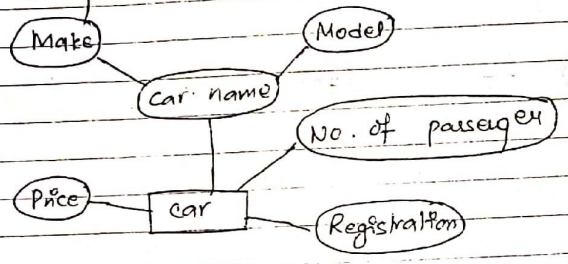


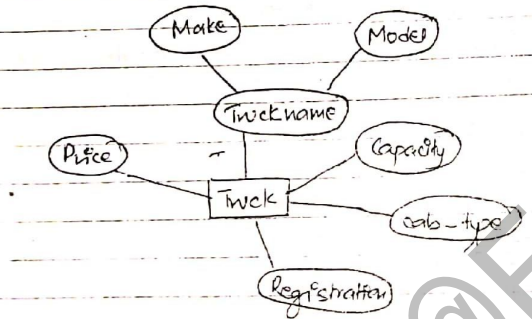
Date / /
Page No. 16

Generalization

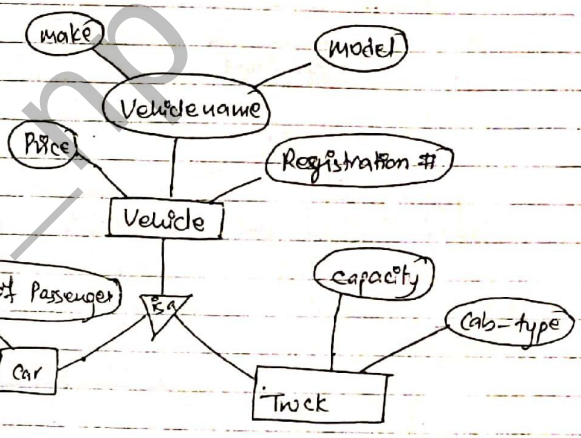
↳ is a bottom up approach.
↳ is the process of forming superclass.
↳ it emphasize the similarities among lower level entity set and hide the difference.

Eg





Date / /
Page No.



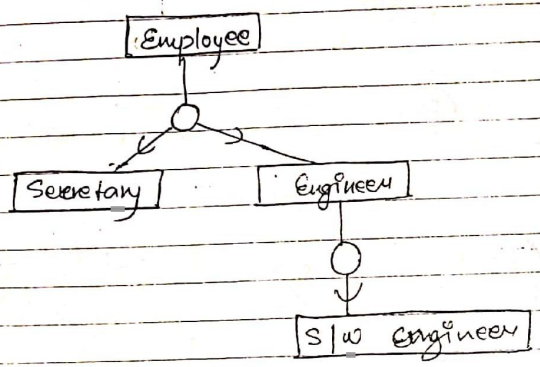
Attribute Inheritance

↳ is a crucial property of superclass and subclass.
↳ attribute of superclass are said to be inherited by subclasses

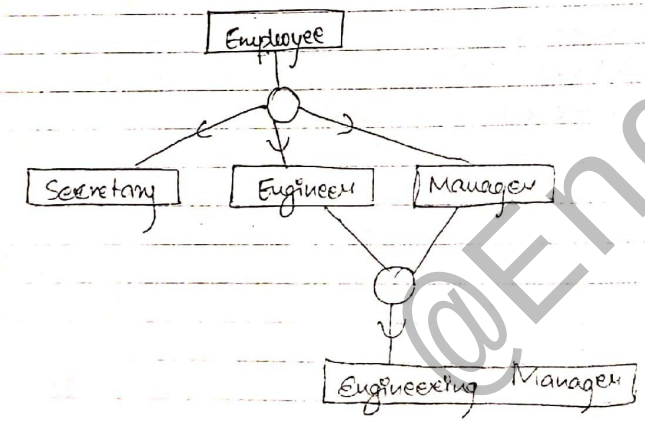
⇒ B 'ISA' A
⇒ B inherits features of A
⇒ A is generalization of B.

Date / /
Page No. 17

Single Inheritance (Hierarchy)

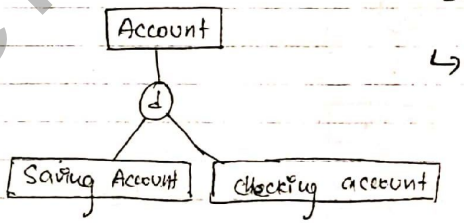


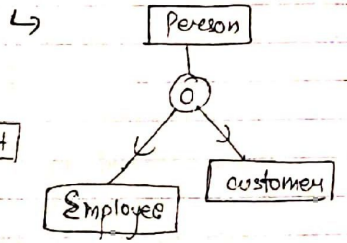
Multiple Inheritance (Lattice)



Date / /
Page No.

Disjoint Constraint	Overlapping Constraint
↳ Subclass belong to no more than one lower level entity set.	↳ Subclass may belong to more than one lower level entity set.
↳ is represented by placing 'd' in circle or simply ISA relationship.	↳ is represented by placing 'O' in circle or simply by ISA relationship.



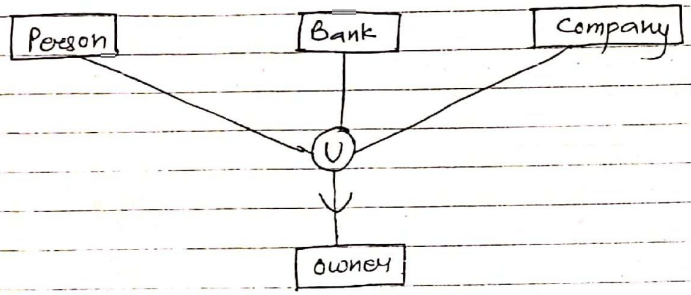


Total generalization/specialization	Partial generalization/specialization
↳ every entity in superclass must belong to subclass ↳ double lines are used to connect the superclass and generalization or specialization.	↳ every entity in superclass may not belong to subclass ↳ single line is used.

Date / /
Page No. 18

Categorization (Union type)

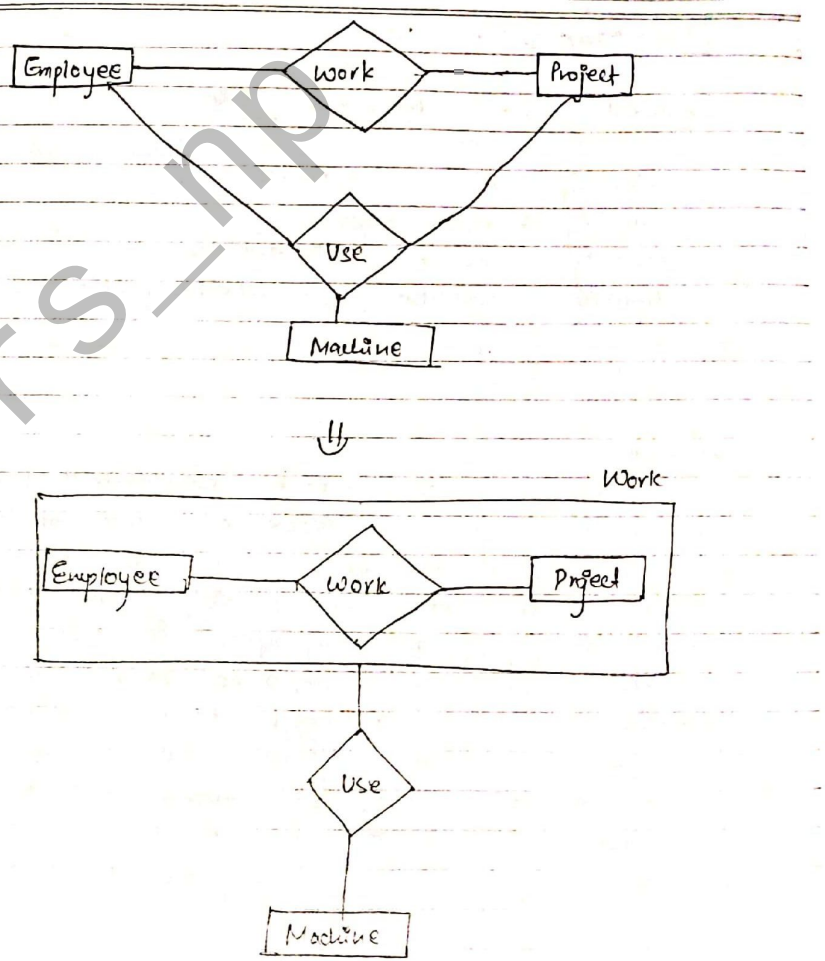
↳ model a class / subclass with more than one superclass of distinct entity sets.
↳ Subclass is called category.
↳ only one subclass exists in categorization.



- here, category 'owner' is subclass of union of person, Bank, company.
- category member must exist in at least one of its superclass.

Aggregation

- is an abstraction through which relationships are treated as high level entities.



Date / /
Page No 19

ER diagram

General steps to create ERD

- Identify entity
- Identify entity's attributes
- Identify primary keys.
- identify relation betⁿ entities.
- Identify cardinality constraints (relationship type)
- Draw ERD
- check ERD

Example

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one but possibly more departments. At least one employee is assigned to a project but an employee may be on vacation and not assigned to any projects. The important (data fields) are names of departments, projects, supervisors and employees. As well as the supervisor and employee no. & a unique project number.

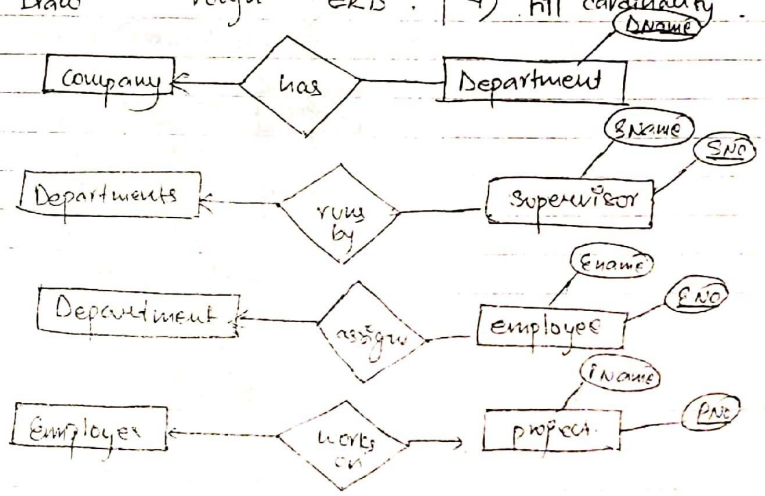
Date / /
Page No.

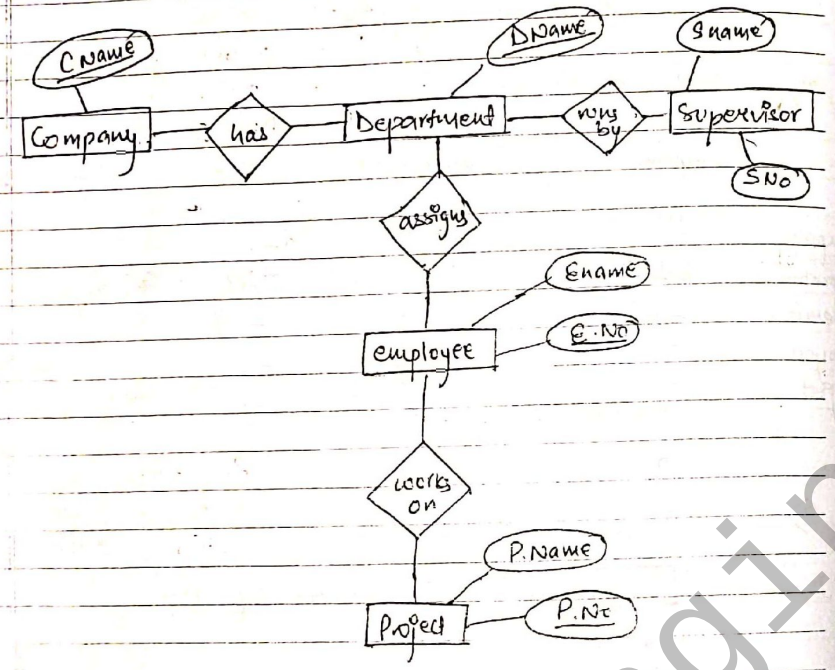
1) Identify entities :- Company, Department, Supervisor, Employee, Project.

2) Find relationships:

	Company	Department	Supervisor	Employee	Project
Company	—	has	—	—	—
Department	belongs.	—	Run by	is assigned	—
Supervisor	—	runs	—	—	—
Employee	—	belongs to	—	—	works on
Project	—	—	—	uses	—

3) Draw rough ERD. 4) Fill cardinality.





CHAPTER 3

Relational Model

2023/5/3
4th August

Relation data model

- Is the primary data model widely used for data storage and processing.
- was proposed by Dr. EF Codd, IBM in 1970.
- data are organized in two dimensional tables called relation.
- tables are related to each other.

RDBMS

- Is dbms based on relational model

Some terminology for RDBMS.

	record or relation oriented	table oriented
1)	↳ relation	table
2)	↳ Domain	set of permitted values
3)	↳ Tuple, record	Row
4)	↳ Attribute, field	column
5)	↳ Cardinality	No. of rows
6)	↳ Degree, Arity	No. of columns
7)	↳ Primary key	UNIQUE + NOT NULL

Table oriented

1) ↳ table
2) ↳ Set of permitted values
3) ↳ Row

Date / /
Page No 21

4) ↳ Column
5) ↳ No. of rows.
6) ↳ No. of columns
7) ↳ Unique + NOT NULL

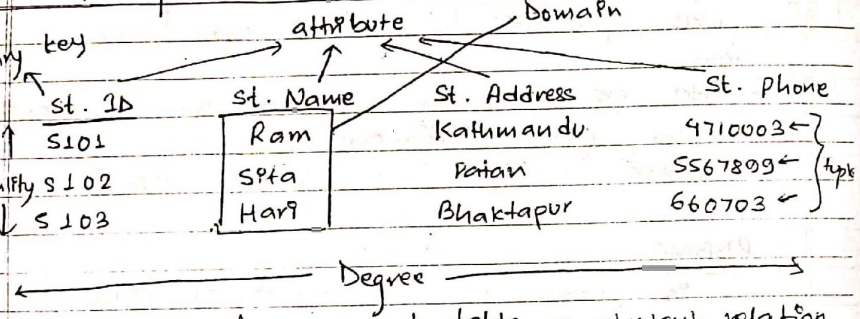


fig: student table or student relation

here,

Let D_1 is set of St. ID
Let D_2 is set of St. name
Let D_3 is set of St. address
D_4 is set of St. phone

Hence,

relation 'student' is subset of

$D_1 \times D_2 \times D_3 \times D_4$

in general,

a table of n attributes must be subset of

$D_1 \times D_2 \times \ldots \times D_n$

for above eg.

Degree of student = 4
Cardinality of student = 3

→ relation is a set of tuples.
→ $t \in r$ denotes that tuple t is in relation r.
→ for relation r, domains of all attributes of r be atomic.

Integrity Constraints.

- Integrity refers to accuracy or correctness of data in db.
- Integrity constraints ensures no result in loss of data. Consistency although changes are made to db by authorized users.
- integrity constraints are defined by some key constraints like primary key, foreign key.
- two general integrity rules are
  1) Entity Integrity
  2) Referential integrity.

1) Entity Integrity:

- ensures that there are no duplicate records within the table and the fields that identifies each record is ~~eg.~~ unique and not null.

Date / /
Page No. 22

- It is the mechanism to provide and maintain primary key.
- ensures two properties for primary key
  - → primary key for a row is unique.
  - → primary key is not null.

2) Referential Integrity

- is concerned with foreign key.
- it designates a column or combination of column as foreign key and establishes relationship between primary key.
- It states that, if in two relations R and S, if an attribute A, in relation R is primary key and foreign key in S. Then the value of foreign key in a tuple S must either be equal to primary key of tuple in R or be entirely null.

SQL (Structured Query language)

→ is the database language designed for managing data in RDBMS.

→ Its scope include data insert, query, update and delete. schema creation and modification and data access control.

→ can execute queries against databases.

→ Different types of SQL statements are

a) DDL (Data Definition language)
   → CREATE, ALTER, DROP, TRUNCATE.

b) DML (Data Manipulation language)
   → INSERT, UPDATE, DELETE

c) DQL (Data Query Language)
   → SELECT

d) DCL (GRANT, REVOKE)

e) TCL (COMMIT, ROLLBACK)

Basic SQL data types

	Datatype	Access	SQL server	Oracle	MYSQL
1)	Boolean	Yes/No	Bit	Byte	N/A
2)	Integer	Number	Int	Number	int, integer
3)	Float	Number	float, real	Number	float
4)	Currency	Currency	Money	N/A	N/A
5)	String (fixed)	N/A	char	char	char
6)	String (variable)	text	Varchar	varchar	varchar Text

Date / /
Page No 23

History of SQL

→ SEQUEL (Structured English Query Language) in 1970's
→ dveloped at IBM
→ SQL-86 (first ANSI standard)
→ SQL-89, SQL1
→ SQL-92, SQL2 (Most dbms use it)
→ SQL-99, SQL3 (recursive queries, triggers and OO features)
→ SQL-2003, windows function, XML related features
→ SQL-2006, XML query support
→ SQL-2008
→ SQL-2011

DDL

→ **to create database**

Syntax:

```
CREATE DATABASE dbname;
```

e.g.

```
CREATE DATABASE cms;
```

→ **to create table**

Syntax

```
CREATE TABLE tablename
(
    column1 datatype;
    .
    .
    .
    column n datatype;
)
```

```
CREATE TABLE tablename
(
    column1 datatype constraints,
    .
    .
    .
);
```

E.g

```
CREATE TABLE STUDENT
(
    SID int,
    Name varchar(20),
    address varchar(25)
);
```

```
CREATE TABLE Student
(
    SID int primary key,
    name varchar(20),
    address varchar(25)
);
```

```
CREATE TABLE Student
(
    SID int not null,
    name varchar(20) not null,
    address varchar(25),
    CONSTRAINTS PK_SID
    primary key (SID, name)
);
```

Date 28th May 2019
Page No 24

To alter table

Syntax

i) ALTER TABLE tablename
ADD Columnname datatype
[e.g: ALTER TABLE student ADD email text ;]

ii) ALTER TABLE tablename
DROP COLUMN columnname
[e.g DROP COLUMN address]

iii) ALTER TABLE tablename
// Modify // ALTER COLUMN columnname datatype
[e.g. ALTER COLUMN email text]

→ ALTER TABLE student
ADD primary key (SID)
or
ADD constraint PK - SID Primary key (SID)

→ ALTER TABLE Student
DROP Constraint PK - SID

To delete table

Syntax

DROP TABLE tablename;

To delete data inside table, not table itself

Syntax:

TRUNCATE TABLE tablename

DML (Data Manipulation language)

1) Insert
— to insert record use INSERT statement

Syntax

→ INSERT INTO tablename (column1, ...)
VALUES (Val1, ...);
→ INSERT INTO tablename VALUES (val1, ...);
→ INSERT INTO tablename VALUES (Val1, Val2, ...), (Val1, Val2, ...) NULL
→ INSERT INTO tablename VALUES (Val1, ~~Val2~~, Val2, ... Valn);

Student

SID	name	address	phone

Date 28th Aug 2019
Page No. 25

4) UPDATE

- to modify records use UPDATE statement.

Syntax:

i) UPDATE tablename SET Column1 = Value1;

ii) UPDATE tablename SET Column1 = Value1
WHERE condition

iii) UPDATE tablename SET Column1 = Value1,
Column2 = Value2 WHERE condition
(SID = 1)

Student

SID	name	address	phone
1	Ram	ktm	9602...
2	Shyam	Ptr	9860...
3	Shyam	Dharan	9814...
4	Sita	Patan	98510...
5	Gita	Bhaktapur	9023...

e.g.

```
UPDATE  student  SET  phone = 4710946;
UPDATE  student  SET  phone = 4710346.
        WHERE  SID = 1;
UPDATE  student  SET  name = 'Gita', phone
        = 4445678 WHERE  SID = 2;
```

Date / /
Page No.

5) DELETE:

- to delete use DELETE statement

Syntax

i) DELETE FROM ~~TABLE~~ tablename

ii) DELETE FROM tablename WHERE condition.

e.g: DELETE FROM student
DELETE FROM Student WHERE SID=5

DQL (Data Query language)

Query:

- a query is an operation that retrieves data from one or more tables or views.
- query nested within another SQL statement is called subquery.
- use 'SELECT' statement to retrieve info from table.

Syntax

i) SELECT * FROM tablename

ii) SELECT ~~col~~ col1, col2 FROM tablename;

iii) SELECT * FROM tablename WHERE condition

Date / /
Page No. 26

iv) SELECT * FROM tablename WHERE Col1 = Val1 AND Col2 = Val2.

v) SELECT * FROM tablename WHERE Col1 = 'Val1' OR Col2 = 'Val2'.

vi) SELECT * FROM tablename WHERE Col1 = 'Val1' AND (Col2 = 'Val2' OR Col3 = 'Val3')

~~DQL (Data~~

vii) SELECT * FROM tablename ORDER BY Columnname ASE | DESC ;

viii) SELECT Column1 (as → optional) NewCol FROM tablename

ix) SELECT TOP {number | percent} Columnname(s) FROM tablename;

e.g: SELECT TOP 50 percent * FROM student

Date / /
Page No.

Wild Card Characters:

→ are used with SQL LIKE operator.
→ are used to search for data within table.
→ % => a substitute for zero or more characters
→ _ => a substitute for single character.
→ [charlist] => sets and range of characters to match
→ [! charlist] => Match only characters not specified in brackets.

x) select columnname(s) FROM tablename WHERE Columnname LIKE pattern;

e.g:
→ select * FROM customer WHERE city LIKE '%s';
→ select * from customer where country like '%land%';
→ select * from customer where country NOT LIKE '%land%';
→ select * from customer where city like 'ber%'
→ select * from customer where city like '_erlin';
→ select * from customer where city like '[L_N_ON]';

[ ] → for multiple options. 28th August

Date / /
Page No. 27

→ select * from customer where city like '[bsp]%'
→ " " " " " " " " '[a-c]%'
→ " " " " " " " " '[!bsp]%'
⇒ " " " " " " " " NOT like '[bsp]%'

Employee

EID	Name	Address	Age	Salary
1	Ram	Patan	35	20000
2	Hari	Pokhara	40	25000
3	Sita	Dharan	32	22000
4	Shyam	Kathmandu	40	20000
5	Gita	Palpa	28	17000

1. Find name of employee whose age is less than 40

⇒ SELECT name from employee where Age < 40

2. Display all employee whose name starts with S

⇒ SELECT * FROM Employee where name LIKE 'S%';

Date / /
Page No.

3. Display all employees whose age is less than 40 and salary greater than 20,000.

⇒ SELECT * FROM employee WHERE age < 40 AND Salary > 20000.

EID	Name	Address	Age	Salary
3	Sita	Dharan	32	22000

4. Display name of employee whose address starts with either P or K.

⇒ SELECT name FROM employee WHERE address LIKE '[PK]%'

5. Display all employees whose salary not starts with 2.

SELECT * FROM employee WHERE salary NOT LIKE '2%'

Date / /
Page No. 28

XI) Select Columnname (s) from tablename where Columnname IN (val1, val2, ...)

e.g:
Select * from Customer where city IN ('London', 'Paris'); {where city = 'London', city = 'Paris'}

XII) Select Columnname (s) from tablename where Columnname BETWEEN value 1 AND value 2,

e.g:
→ Select * from products where price between 10 and 20.
→ Select * from ~~cust~~ products where price not between 10 and 20.
→ Select * from products where (price between 10 and 20) and not category IN (1, 2, 3);
→ Select * from products where productname between 'C' and 'M'.
→ Select * from products where productname not between 'C' and 'M'
→ SELECT * FROM Orders WHERE Orderdate between #07/04/2016# and #05/05/2016#.

XIII) Select * from tablename As Newtablename;
e.g Select * from Customer As C;

Date / /
Page No.

SQL Functions

⇒ AVG() → returns average value of numeric column.
— Select AVG (Columnname) from tablename

⇒ Count() → returns the no. of rows.
— Select Count (columnname) from tablename;
— Select Count (*) from tablename;

⇒ Max() → returns the maximum value of selected column.
— Select max (columnname) from tablename;

⇒ Min() → return the minimum value of selected column.
→ select min (columnname) from tablename;

⇒ Sum() → returns the total sum of numeric column
— SELECT Sum (columnname) from tablename;

⇒ Upper() → converts value of field to uppercase
— SELECT upper (Columnname) from tablename;

⇒ lower() → Converts value of field to lower case

Date / /
Page No. 29

- Select lower (columnname) from tablename;

⇒ SQRT() → finds square root of any no.
- select sqrt (Columnname) from tablename;

⇒ RAND() → to produce random numbers beth 0 and 1.
- Select * from employee order by RAND();

⇒ concat() → to concatenate two strings to form single string.
- Select concat (Column1, Column2) from tablename.

Groupby and "Having" ~~Cate~~ Clause

- used to divide the rows in table in groups
- is used with SQL aggregate functions like Sum() Avg()

e.g:
- Select job, Sum(salary) from employee group by job.
- Select job, max(salary) from employee group by job having sum(salary) >= 30000

Date / /
Page No

Subquery:

- is a query within another SQL query and embedded within 'where' clause.
- is used to return data that will be used in main query as a cond^n to further restrict data to be retrieved.
- can be used with select, insert, update and delete statements along with operators like =, >=, <=, <, >, in, between etc.

Subquery

Rules for subquery:

→ must be enclosed within parenthesis
→ can have only one column in 'select' clause.
→ An 'order by' can't be used but 'Group by' can be.
→ Subqueries that return more than one row can only be used with multiple value operator like 'IN'.
→ A subquery can't be immediately enclosed in a set function.
→ 'Between' operator can't be used with subquery, however it can be used within subquery.

Subqueries with select statement

Syntax:

```
Select Columnname(s) from table where
columnname operator
(select Columnname(s) from table where..)
```

e.g:

```
Select * from Customer
where ID IN
(select ID from Customer where Salary >4500)
```

Subqueries with INSERT Statement

e.g:

```
Insert into Customer1 select * from
Customer where ID IN (Select ID from
Customer);
```

Subqueries with update Statement

e.g:

```
UPDATE Customer SET Salary = Salary * 0.25
where age in (select age from customer
where age >= 27);
```

Subqueries with Delete Statement

e.g:

```
Delete from customer where age in
(-select age from customer where age>27)
```

Customer

ID	name	age	Salary
1	Ram	28	10000
2	Shyam	32	5000
3	Hari	49	8000
4	Sita	43	7000

SQL using join

- join is used to combine records from two or more tables in a database based on a common field between them.

Different types of join

a) Inner join (or simply join)

Select all rows from both tables as long as there is a match between the columns in both tables.

Syntax

```
Select Columnname(s) from table1
Inner join table2
on table1 column1 = table2.column2
```

eg:

```
Select Customer.CustomerName, Order.OrderID
from Customer inner join order
on customer.CustomerID = Order.CustomerID.
```

b) Left join (left outer join)

- returns all rows from the left table (table1) with the matching rows in right table (table2)
- The result is null in right side where there is no match.

Syntax

```
Select Columnname(s) from table1
left join table2
```

e.g

```
Select Customer.CustomerName, Order.OrderID
from customer left join order on
customer.CustomerID = Order.CustomerID
```

c) Right join (right outer join) (second table)

- returns all rows from right table (table2) with the matching rows in left table (table1)
- the result is null in left side where there is no match.

Syntax

```
Select Columnname(s) from table1
right join table2
on table1.column1 = table2.column2
```

e.g:

```
Select Customer.Customername, Order.OrderID
from Customer right join order
on customer.CustomerID = Order.CustomerID
```

d) Full outer join

- returns all rows from left table and from right table
- combine the result of both left and right join.

Syntax:

```
Select Columnname(s) from table1
Full outer join table2
```

on table1. column1 = table2. column2

e.g:

```
Select Customer. Customername, Order. OrderID
from Customer full outer join order
on Customer. CustomerID = Order. CustomerID.
```

Set operation

1) Union operation

Syntax

a)
```
Select Columnname (s) from table1
Union
Select Columnname (s) from table2.
```

b)
```
Select Columnname (s) from table1
Union All
Select Columnname (s) from table2.
```

2) Intersect operator

Syntax:
```
Select Columnname (s) from table1
Intersect
Select Columnname (s) from table2
```

3) Except operator

Syntax:
```
Select Columnname (s) from table1
Except
```

Views

→ View is a virtual table based on the result set of an SQL statement.
→ Views are logical representation of data.
→ It contains rows and columns just like real table.
→ SQL functions, where and join statements can be added to a view and data can be presented as if data are coming from a single table.
→ View allows users to
- structure data in a way that users want.
- restrict access to data such that user can see and sometimes modify exactly what they need and no more.
- Summarize data from various tables, which can be used to generate reports.

Syntax to create view

```
CREATE VIEW viewname AS
SELECT columnname (s) FROM TABLE
WHERE CLAUSE
```

to query view

```
SELECT * FROM viewname;
```

to update view

```
UPDATE viewname SET columns =
WHERE
```

to drop view

DROP VIEW viewname ;

Note:

It shouldnot contain aggregate function or distinct or Groupby clause while updating views.

Example

Customer (cid , cname , caddress)
order (oid , cid , date)

⇓

CREATE VIEW V customerorder
SELECT cname , date FROM CUSTOMER
INNER JOIN ORDER
ON CUSTOMER . cid = ORDER . cid .

Relational Algebra: 8 marks.

→ is a procedural query language which takes instances of relations as input and gives instance of relation as output.
→ helps to understand query execution and optimization in RDBMS.
→ Basic operations of relational algebra are
a) selection
b) projection
c) Union
d) set difference

1) Selection

- It selects a subset of rows from relation that satisfies some condition.
- it is denoted by σ.
- here, schema of result is identical to schema of input relation.
- syntax:

σ condition (relation)

← result relation can be input for another relational algebra operation.

employee (eid , ename , eaddress , salary)
find all employees whose salary is greater than 10000.

σ salary > 10000 (employee)

eid	ename	eaddress	salary
1	Ram	Ktm	15000
2	Shyam	Patan	10000

2) Projection:

- It deletes the attribute that are not in project list.
- is denoted by Π.
- Schema of result contains exactly the

Date / /
Page No. 31

fields in projection list, with the same names that they had in input relation.

- Syntax:

Π attribute list (relation)

e.g:

Π ename, salary (employee)

eid	ename	eaddress	salary
1	Ram	kim	15000
2	Shyam	patan	10000
3	Hari	Bhaktapur	12000

Π ename, salary (σ salary > 10000 (employee))

σ salary > 10000 (Π ename, salary (employee))

3) Union

- it builds a relation from tuples appearing in either or both of the specified relation.
- it eliminates duplicates
- 'r' is denoted by 'U'.
- two relation are union compatible if
  - → they have same no. of attributes.
  - → attribute domain must be compatible.

Date / /
Page No.

→ names of attribute are same in both.

Syntax:

Π attribute list (relation) U Π attribute list (relation)

→ R U S = S U R

4) Set Difference // except

→ it results tuples present in 1st relation but not in second relation.

→ is denoted by -

→ union compatible should exists.

→ Syntax:

Π attributelist (relation) − Π attributelist (relation)

→ R − S ≠ S − R

Page No. 32

5) Cartesian Product

↳ It builds a relation with concatenation of every row in first relation with every row in second relation.

↳ also known as Cross product or ~~Cross~~ Cross join.

↳ is denoted by 'X'.

Syntax :-

R X S

eg.

Book

BID	BName	Price	PID
B1	DBMS	550	P1
B2	Math	600	P2
B3	SE	750	P3

Publisher

PID	PName	Paddress
P1	ABC	ktm
P2	XYZ	Patan

Book X Publisher

BID	BName	Price	Book.PID	Publisher.PID	PName	Paddress
B1	DBMS	550	P1	P1	ABC	ktm
B1	DBMS	550	P1	P2	XYZ	Patan
B2	Math	600	P2	P1	ABC	ktm
B2	Math	600	P2	P2	XYZ	Patan
B3	SE	750	P3	P1	ABC	ktm
B3	SE	750	P3	P2	XYZ	Patan

Date / /
Page No.

B-name, price, publisher

⇒ $\Pi_{Bname, price, pname}$ (Book × publisher)

(|| condition . (Book.PID relation P.PID))

$\Pi_{Bname, price, Pname}(\sigma_{Book.PID = publish.PID}$ (Book × publisher))

~~Relational~~ A

6) Intersection

→ results tuples that appear in both relation. → denoted by ∩.

→ relation must be union compatible

→ Syntax:

$\Pi_{attributelist}$ (relation1) ∩ $\Pi_{attributelist}$ (relation2)

→ R ∩ S = S ∩ R

12th Sept.

Date / /
Page No. 33

Join operation (⋈) [cartesian product with selection]

↳ allow users to combine two relation in a specified way.
↳ The join operation can be stated in terms of a cartesian product followed by a selection operation.
↳ is very important as used frequently while specifying database queries.
↳ three types of join:
  a) Theta join (condition Join)
  b) Equijoin (Inner join)
  c) Natural join

a) Condition Join (Theta join)

↳ join two relation together on basis of some comparision operator.
↳ θ is a predicate & consists of one of comparison operator (=, <, >, <=, >=, <>) and specifies join condition.
↳ if θ is =, then it is called equijoin.
↳ defined as cross product followed by selection.
↳ for any two relations R and S, theta join is

$R \bowtie_\theta S$ OR

$\sigma_\theta (R \times S)$

Date / /
Page No.

e.g.

R1

Sid	bid	day
22	101	10/10/2015
58	103	9/9/2016

S1

Sid	S name	rating	age
22	Ram	7	45
31	shyam	8	50
58	Hari	10	35

Now,

$S1 \bowtie_{S1.Sid < R1.Sid} R1$.

R1.Sid	S1.Sid
22	22
22	31
22	58
58	22 ✓
58	31 ✓
58	58

Date / /
Page No. 34

o/p

Sid	Sname	rating	age	Sid	bid	day
22	Ram	7	45	58	101	9/9/2016
31	Shyam	8	50	58	103	9/9/2016

b) Equi join or Inner join:
- ↳ if θ is '=', then theta join becomes equijoin.
- ↳ the (=) condition is performed by primary and foreign keys.
- ↳ The result must include two attributes with property that values of these two attributes are equal every tuple in relation.
- ↳ ~~Eg~~ ~~Syntax~~ Syntax:

$$R \bowtie_{(R.PK = S.FK)} S$$

e.g:

$$R1 \bowtie_{R1.Sid = S1.Sid} S1.$$

Sid	bid	day	Sid	Sname	Rating	Age
22	101	10/10/2015	22	Ram	7	45
58	103	9/9/2016	58	Hari	10	35

Date / /
Page No.

c) Natural join:
- ↳ is represented by symbol ⋈
- ↳ only relates those rows which have common attribute value.
- ↳ For relation R and S

R⋈S = Column (R) ∪ Column (S)

Employee

EID	name	Deptname
101	Ram	Finance
102	Shyam	Marketing
103	Hari	Finance

Department

Deptname	Manager
Finance	Gopal
Marketing	Gita

employee ⋈ Department.

EID	name	Deptname	Deptname	Manager
101	Ram	Finance	Finance	Gopal
102	Shyam	Marketing	Marketing	Gita
103	Hari	Finance	Finance	Gopal.

Date / /
Page No. 35

Hierarchical Model:

↳ It organizes data in tree structure where there is a hierarchy of parent and child data segments.
↳ It has a single root segment connected to lower level segments.
↳ each segment may connect to other lower level segments.
↳ Here, rule is that one parent can have many children but children are allowed only one parent.
↳ this allows information to be repeated through parent child relation.

Advantages

↳ allows easy condition and deletion of new information.
↳ fast access to data at top
↳ relates well to anything that works on one to many relationship.

Disadvantages:

↳ allows data redundancy
↳ Searching for lower level information is slow.
↳ Many to many relationship are not supported.

Date / /
Page No.

Network Model:

↳ replaces the hierarchical tree with graph.
↳ allows to have

Advantages

↳ simple and easy to implement.
↳ can handle 1:1 and many to many N:N relationship.
↳ data access is easier than hierarchical model.

Disadvantages:

↳ System complexity.
↳ Lack of structural independence.
i.e. change in structure makes change in application too.

Relational Model

↳ all data are stored in tables where each table consists of rows and columns.
↳ here, keys are used to order data or relate data to other tables
↳ defines set of rules to ensure data integrity known as integrity constraints
↳ also defines how data are manipulated.
↳ defines special feature called normalization to ensure efficient data structure

12th Sept 2016
Date
Page No. 36

Advantages :-
↳ easy of use
↳ flexibility
↳ security
↳ data independence.

Disadvantages :-
↳ ease of design can lead to bad design.
↳ H/w overheads.

QBE (Query By Example)

↳ developed at IBM in 1970's.
↳ QBE has a two dimensional syntax.
↳ Queries looks like table.
↳ QBE queries are expressed by example instead of giving a procedure.
↳ Queries in QBE are expressed by using skeleton tables, which show the relational schema of database
↳ Skeleton table looks like.

Relation name	Column1	Column 2	. .	Column N
Tuple or row operation.				

↳ For relation college (collegeId, Name), QBE is as

College	College Id	Name.

12th Sept 2016
Date
Page No. 36

Advantages :-
↳ easy of use
↳ flexibility
↳ security
↳ data independence.

Disadvantages :-
↳ ease of design can lead to bad design.
↳ H/w overheads.

QBE (Query By Example)

↳ developed at IBM in 1970's.
↳ QBE has a two dimensional syntax.
↳ Queries looks like table.
↳ QBE queries are expressed by example instead of giving a procedure.
↳ Queries in QBE are expressed by using skeleton tables, which show the relational schema of database.
↳ Skeleton table looks like.

Relation name	Column1	Column 2.		Column N
Tuple or row operation.				

↳ For relation College (collegeId, Name),

College	College Id	Name

Date / /
Page No.

- use of P to get list of relation.
- P. for point command.
- data retrieval command is P. <variablename> where variablename is optional.
- QBE automatically eliminates duplicate values.
- to see duplicate values, P. ALL
- QBE supported by microsoft Access is Graphical QBE.

e.g:

employee (ename, city, salary, rating)

1) to find all employee names who live in city kathmandu.

employee	ename	city	salary	rating
	P. or P. ALL	kathmandu		

2) to find entire employee

employee	ename	city	salary	rating
P.				

3) to find employee name where salary is greater than 10000.

employee	ename	city	salary	rating
	P. or P. ALL		>10000	

Date / /
Page No. 37

4) to find all employees whose salary between 5000 and 10000.

employee	ename	city	salary	rating
P.			-X	

Condition
-X >= 5000
-X <= 10000

Database modification in QBE.

→ Insert

Employee	ename	city	salary	rating
I.	Ram	Pokhara	15000	5
I.	Shyam			6

2) Delete

employee	ename	city	salary	rating
D.	Ram			

employee	ename	city	salary	rating
	Ram	D.		

3) Update

employee	ename	city	salary	rating
	Ram			U. 10)

Relational Calculus

- is non procedural query language that describe the set of answers, without being explicit about how they should be computed.
- two types,
  - ↳ Tuple relational calculus (TRC)
  - ↳ Domain " " (DRC)

(1) TRC

- a TRC query has the form $\{T \mid P(T)\}$ where
  - T is a tuple variable.
  - P(T) is a formula that describes T.
  - ,
- it results set of all tuples t to which P(T) evaluates to true when $T = t$.

eg:

2) $\{T.name \mid Author(T)$ and $T.article = 'dbms'\}$

- it returns tuples with values from Author who has written article on 'dbms'

b) $\{t \mid student(t)$ and $t.percentage > 60\}$

c) $\{t \mid employee(t)$ and $t.deptID = 10\}$

d) DRC

- ↳ it uses domain variables that take on values from attribute domain rather than values for an entire tuple.
- ↳ query is expressed in the form of $\{<x_1, x_2, \cdots x_n> \mid P(x_1, x_2, \cdots x_n)\}$ where $x_1, x_2, \cdots x_n$ represent domain variables and P represents formula.
- ↳ an atom in DRC has the form.
  - → $<x_1, x_2, \cdots x_n> \in r$, where r is relation of n attributes and $x_1, x_2, \cdots x_n$ are domain variables.
  - → $X \ominus Y$, where X and Y are domain variable and $\ominus$ is comparison operator
  - → $X \ominus c$, where X is domain variable, $\ominus$ is comparison operator, c is constant.

Date / /
Page No.

⇒ We build up formula from atom by using following rules

↳ an atom is a formula.

↳ If P1 is a formula then so are ¬P1 and (P1)

↳ if P1 and P2 are formula then so are
P1 ∧ P2, P1 ∨ P2 and P1 ⇒ P2.

↳ If P1(x) is a formula in x, where x is domain variable then ∃x(P1(x)) and ∀x(P1(x)) are also formula.

e.g.

i) $\{ < l, b, a > \mid < l, b, a > \in loan \wedge a > 20000 \}$

ii) $\{ < c, a > \mid \exists_l (< c, l > \in borrower \wedge \exists_b (< l, b, a > \in loan \wedge b = 'ktm')) \}$

Note:

Expression power of TRC and DRC is equivalent to relational algebra.

Stored Procedure

↳ is a set of SQL statements with an assigned name that is stored in db in compiled form so that it can be shared by no of programs

↳ is nothing more than prepared SQL code that is saved and can be reused over and over again

Date / /
Page No. 39

Creating Simple Stored Procedure

Syntaxe:

```
CREATE PROCEDURE procedureName
As
SQL Statement
```

e.g:

```
CREATE PROCEDURE getInfo
As
select * from Employee
```

To execute stored procedure
→ getInfo
→ EXEC getInfo

Stored procedure with a parameter

```
CREATE procedure getInfo @city varchar(25)
As
select * from employee
where city = @city
```

e.g:

getInfo @city = "Berlin"

⇒ employee (eid, name, salary)
department (did, dname, eid).

```
CREATE PROCEDURE getInfo
As
SELECT name, salary, dname from employee inner
join department
on employee.eid = department.eid
```

Relational Database Constraints

Chapter Outline
- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

Relational Model Concepts
- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.
- We review the essentials of the relational approach in this chapter.

Relational Model Concepts
- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.
  The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.

INFORMAL DEFINITIONS
RELATION: A table of values
- A relation may be thought of as a **set of rows**.
- A relation may alternately be though of as a **set of columns**.
- Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
- Each row has a value of an item or set of items that uniquely identifies that row in the table.
- Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
- Each column typically is called by its column name or column header or attribute name.

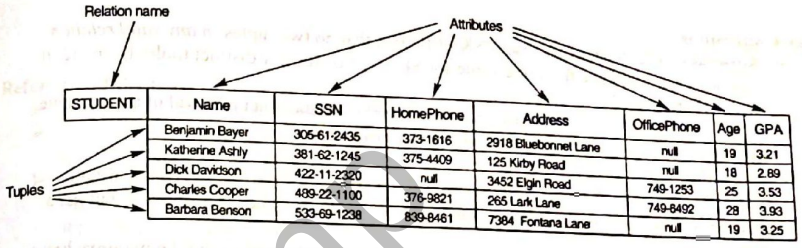
FORMAL DEFINITIONS
- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: *R* (A1, A2, ...An)
  Relation schema *R* is defined over **attributes** A1, A2, ...An
  For Example -
  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  Here, CUSTOMER is a relation defined over the fourattributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

- A **tuple** is an ordered set of values
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.

- **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">** is a tuple belonging to the CUSTOMER relation.
- A relation may be regarded as a ***set of tuples*** (rows).
- Columns in a table are also called attributes of the relation.
- A **domain** has a logical definition: e.g.,"USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.
- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddddddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes "Invoicedate" and "Payment-date".
- The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.
- For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.
- Formally,
  Given R(A1, A2, ..., An)
  r(R) ⊂ dom (A1) X dom (A2) X ....X dom(An)
- R: schema of the relation
- r of R: a specific "value" or population of R.
- R is also called the **intension** of a relation
- r is also called the **extension** of a relation
- Let S1 = {0,1}
- Let S2 = {a,b,c}
- Let R ⊂ S1 X S2
  Then for example: r(R) = {<0,a> , <0,b> , <1,c> }
  is one possible "state" or "population" or
  "extension" r of the relation R, defined over domains
  S1 and S2. It has three tuples.

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column	Attribute/Domain
Row	Tuple
Values in a column	Domain
Table Definition	Schema of a Relation
Populated Table	Extension



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Characteristics of Relations

- **Ordering of tuples in a relation r(R)**: The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
- **Ordering of attributes in a relation schema R** (and of values within each tuple): We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be *ordered* .
- (However, a more general *alternative definition* of relation does not require this ordering).
- **Values in a tuple**: All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.
- Notation:
  - We refer to **component values** of a tuple t by t[Ai] = vi (the value of attribute Ai for tuple t).

  Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively.

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Relational Integrity Constraints

Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:

1. **Key** constraints
2. **Entity integrity** constraints
3. **Referential integrity** constraints

Key Constraints

- **Superkey** of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t0 and t2 in r(R), t1[SK] ≠ t2[SK].
- **Key** of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.
  Example: The CAR relation schema:
  CAR(State, Reg#, SerialNo, Make, Model, Year) has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.
- If a relation has *several* **candidate keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are *underlined*.

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.
  S = {R1, R2, ..., Rn}
- **Entity Integrity**: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples.
  t[PK] ≠ null for any tuple t in r(R)
- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

BE Computer Semester VI (DBMS) Note-4

Referential Integrity

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R2. A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity Constraint

Statement of the constraint:
The value in the foreign key column (or columns) FK of the **referencing relation** R1 can be either:
(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R2,, or..
(2) a null.
In case (2), the FK in R1 should not be a part of its own primary key.

Other Types of Constraints

Semantic Integrity Constraints:

- based on application semantics and cannot be expressed by the model per se
- E.g., "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"
- A *constraint specification language* may have to be used to express these
- SQL-99 allows triggers and ASSERTIONS to allow for some of these

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

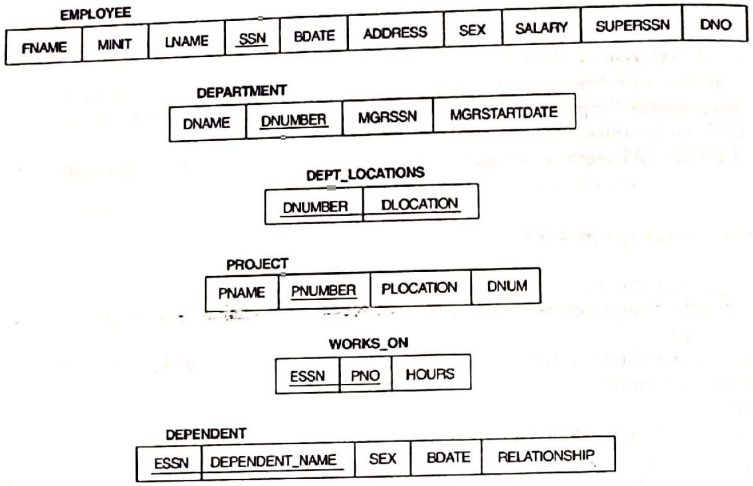


Figure 7.6 One possible relational database state corresponding to the COMPANY schema.

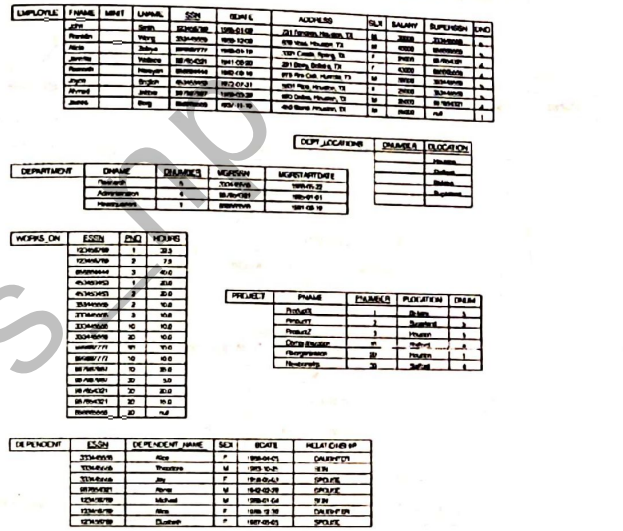
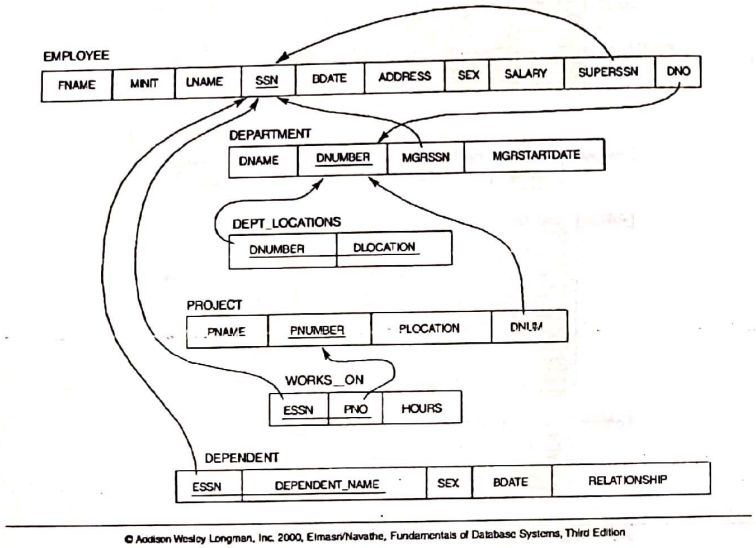


Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may *propagate* to cause other updates automatically. This may be necessary to maintain integrity constraints.
- In case of integrity violation, several actions can be taken:
  - ✓ Cancel the operation that causes the violation (REJECT option)
  - ✓ Perform the operation but inform the user of the violation
  - ✓ Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - ✓ Execute a user-specified error-correction routine

In-Class Exercise

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)
COURSE(Course#, Cname, Dept)
ENROLL(SSN, Course#, Quarter, Grade)
BOOK_ADOPTION(Course#, Quarter, Book_ISBN)
TEXT(Book_ISBN, Book_Title, Publisher, Author)

Draw a relational schema diagram specifying the foreign keys for this schema.

CHAPTER 5

Database Constraints and Relational Database Design.

Date / /

- Constraints are nothing but rules on data that help to define what data is valid and what is invalid, helps to maintain integrity of data.

- Database constraints are user defined structure that let you restrict the behaviour of columns.

- Different types of database constraints are
  i) primary key constraint
  ii) foreign " "
  iii) NOT NULL " "
  iv) Unique " "

Integrity Constraints:

- ensures that changes are made to the database by authorized users do not result in loss of data consistency.
- guards against accidental damage of database

Referential Integrity

- ensures that a value that appears in one relation for a given set of attributes also appear for certain set of attributes in another relation.
- ensures that foreign key value = primary key value.

Date / /
Page No 40

Functional dependencies:

↳ are the fundamental to the process of normalization.

↳ FD plays key role in differentiating good database design from bad db design.

↳ FD describes the relationship between attributes in a table.

↳ for attributes X & Y in relation R, functional dependencies between X & Y is shown as

$$X \longrightarrow Y$$

here,

X is determinant
Y is functionally dependent on X.

↳ the functional dependency $X \longrightarrow Y$ holds on table R if in any legal relation r(R), for all pair of tuples t_1 & t_2 in r such that $t_1[X] = t_2[X]$ it is also case that $t_1[Y] = t_2[Y]$

eg:

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b2	c2	d3
a3	b3	c2	d4

here,
$A \rightarrow C$ is satisfied
$C \rightarrow A$ is not satisfied
$t_1[C] = t_2[C]$
but
$t_1[A] \neq t_2[A]$

E.g:

eid	ename	age	salary
101	Ram	25	10000
102	Shyam	28	15000
103	Hari	30	18000

here,
eid → ename ; True as eid uniquely determines ename
ename → eid ; false as it is not always true
eid → age ; True
eid → salary ; false
age → eid ; false.

eid	project No	hrs	ename	projectname	projectlocation

here,
eid → ename
project No → { project Name , projectlocation }
{eid , project No} → hrs.

Types o

Chapter 5

Types of functional dependency.

1) Full functional dependency.

X → Y is a full functional dependencies if the removal of any attribute A from X removes the dependency.

↳ A full functional dependency occurs when it is are already functional dependency and the set of attributes on the left side of functional dependency cannot be reduced anymore.

e.g.

orderno.	lineno	qty	price
A001	001	10	200
A002	001	20	400
A002	002	30	800
A004	001	15	300

here,
{ orderno, lineno } → qty
{ orderno, lineno } → price
} are full functional dependency.

2) Partial functional dependency.
→ A functional dependency X → Y is partial dependency if some attribute A belongs to X can be removed from X and the dependency still holds

e.g:
{eid, phone} ⟶ name
here,
removing phone.
eid ⟶ name

3) Transitive Dependency
Occurs when there is an indirect relationship that causes a functional dependency.
if $X \to Y$ and $Y \to Z$ then transitive dependency exists is
$X \to Z$.

4) Trivial ~~Trial~~ Dependency
↳ Occurs when functional dependency of an attribute is described on a collection of attributes that includes the original attribute.

↳ Functional dependency is trivial if RHS is subset of LHS.
e.g.
{name, SSN} ⟶ SSN

5) Non Trivial Dependency
↳ are the one that is not trivial.
e.g:

{S#, P#} ⟶ S# — trivial
{S#, P#} ⟶ {S#, qty} — Non trivial

Axioms or Rules of inference for functional dependency.

i) Reflexivity Rule:
If Y is subset of X ($Y \subseteq X$) then $X \to Y$ holds.

ii) ~~Trans~~ Augmentation Rule:
if $X \to Y$ holds and z is set of attributes then $zX \to zY$ holds.

iii) Transitivity Rule:
If $X \to Y$ and $Y \to Z$ holds then $X \to Z$ holds.

iv) Union Rule
If $X \to Y$ and $X \to Z$ holds then $X \to YZ$ holds

v) Decomposition Rule:
if $X \to YZ$ holds then $X \to Y$ and $X \to Z$ holds

vi) Pseudo transitivity Rule:
if $X \to Y$ and $ZY \to P$ holds then
$XZ \to P$ holds.

vii) Self determination

$A \rightarrow A$

e.g:

$R = (A, B, C, G, H, I)$

Set of F.D $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

Then we can have

i) $A \rightarrow H$; with transitivity Rule

ii) $CG \rightarrow HI$; " Union "

iii) $AG \rightarrow I$; " pseudotransitivity rule -

e.g:

$F = \{A \rightarrow B, C \rightarrow X, BX \rightarrow Z\}$

prove or disprove

$AC \rightarrow Z$

$\Rightarrow$ $C \rightarrow X$, $BX \rightarrow Z$ (Pseudo transitivity)

$A \rightarrow B$ and $BC \rightarrow Z$

$AC \rightarrow Z$

e.g. $F = \{A \rightarrow B, C \rightarrow D, C \subseteq B\}$

prove or disprove $A \rightarrow C$.

$B \rightarrow C$

$\Rightarrow A \rightarrow C$.

Closure of a set of functional dependencies

→ the set of functional dependencies that is logically implied by F is called closure of F and written as F^+.

→ the closure of F denoted by F^+ is set of all FD's entailed by F.

$F^+ = \{X \rightarrow Y \mid F \Rightarrow FX \rightarrow Y$

$R = \{A, B, C, D\}$

$F = \{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$

list several members of F^+.

Soln

i) $A \rightarrow B$, $A \rightarrow C$ ii) $A \rightarrow BC$, $BC \rightarrow D$

$\therefore A \rightarrow BC$ $A \rightarrow D$

iii) $A \rightarrow C$, $BC \rightarrow D$

$\therefore AB \rightarrow D$

$F^+ = \{A \rightarrow BC, A \rightarrow D, AB \rightarrow D\}$

e.g:

i) $R = \{A, B, C, G, H, I\}$

$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

1Pst element of F^+.

i) $A \to B$, $A \to C$.

$\equiv A \to BC$

ii)

iii) $R = (A, B, C, D, E, F)$

$F = \{A \to BC, B \to E, CD \to EF\}$

list F^+ & prove $AD \to F$

Closure of attribute set

- Given a set of attributes $A_1 \dots A_n$ the closure $\{A_1 \dots A_n\}^+$ is set of all attributes B such that

$$A_1 \dots A_n \to B$$

an algorithm to compute α^+

result = α,
while (changes to result) : do
  for each Functional dependency $\beta \to \gamma$
  begin
    if $\beta \subseteq$ result then
      result = result $\cup \gamma$
  end

e.g:

$R = (A, B, C, G, H, I)$

$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}$

Compute $(AG)^+$

Soln

result = AG

for $A \to B$

$A \subseteq AG$ true ; result = $\{ABG\}$

for $A \to C$

$A \subseteq ABG$; true ; result = $\{ABCG\}$

for $CG \rightarrow H$

$CG \subseteq ABCG$ true ; result = {ABCGH}

for $CG \rightarrow I$

$CG \subseteq ABCGH$ true ; ∴ result = {ABCGHI}

for $B \rightarrow H$

$B \subseteq ABCGHI$ true ; ∴ result = {ABCGHI}

∴ $(AG)^+ = ABCGHI$

eg. 2.

$R = (A, B, C, D, E, F)$

$F = \{A \rightarrow BC, E \rightarrow CF, B \rightarrow E, CD \rightarrow EF\}$

$\{AB\}^+ = ?$

Unit 4

Normalization

- Database normalization is a technique of organizing data in the database.
- It is the process of removing data redundency and undesirable characteristics like insertion, update and deletion anomolies.
- Mainly, two purposes of normalization are
  i) Eliminating redundant data
  ii) Ensuring data dependencies make sense.
- It involves dividing table into two or more tables and defining relationship between the tables.
- Normal forms represents guidelines for design
- Different types of normal forms are 1NF, 2NF, 3NF, BCNF, 4NF etc.

Unnormalized Form.

- It contains one or more repeating groups or each row may contain multiple set of values for some columns.
- Multiple values in a single row are called as atomic values.

eg.

CID	CName	Caddress	Cphone
C101	Ram	Kathmandu	98 C 3346813, 5 3346814
C102	Shyam	Patan	9851010257
C103	Hari	Pokhara	56 94820 9803329

Fig: UNF

First Normal Form (1NF)

A relation is in 1NF if and only if all columns are atomic i.e. no repeating items in columns

↳ here, Fig. 1 will become.

CID	cname	caddress	cphone
C101	Ram	kathmandu	9803346813
C101	Ram	kathmandu	5534681 4
C102	Shyam	patan	9851010257
C103	Hari	pokhara	9803343437
C103	Hari	pokhara	9819003214
C103	Hari	pokhara	5534 6832
		~~pokhara~~ .	

Fig. 1NF

↳ There shouldn't be any repeating columns.

id	name	email	child 1	child 2	child 3
1	Ram	- - -	Hari	Sita	Gita
2	Shyam	- - -	Shyam	John	rita.

Second Normal Form (2NF)

For a relation to be in 2NF,

a) it must be in 1NF

b) all attributes depend on full primary key.

eg. table person project

Person ID	Proj ID	PersonName	Project Code	Proj Name	Mob.
1	1	Ram	Pro 1	DB	981...
2	1	Shyam	Pro &1	DB	980...
1	2	Ram	Pro 2	Web	981...
2	2	Shyam	Pro 2	web	980...

1. Person

Person ID	Person Name	Phone
1	Ram	981...
2	Shyam	980...

2. Project

Proj ID	Proj Code	Proj Name
1	Pro 1	DB
2	Pro 2	Web

3. PerProj

Person ID	Project ID
1	1
2	1
1	2
2	2

55

Third Normal Form (3NF)

- For a relation to be in 3NF,
  a) it must be in 2NF,
  b) There is no transitive functional dependency i.e. there shouldn't be case that non prime attribute is determined by another non-prime attribute.

eg. Student info

St-ID	StName	DOB	City	Zip
1	Ram	1984/1/1	Ktm	01
2	Shyam	1985/2/1	Pkr	05
3	Hari	1989/2/1	Pkr	05
4	Gita	1990/5/5	Ktm	01

1. Student

St-ID	St Name	DOB	Zip
1	Ram	1984/1/1	01
2	Shyam	1985/2/1	05
3	Hari	1989/2/1	05
4	Gita	1990/5/5	01

2. Address

Zip	City
01	Kathmandu
05	Pokhara

BCNF (Boyce Codd Normal Form)

- For a relation to be in BCNF
  - it must be in 3NF
  - Every determinant in that table is superkey.
- if table contain only one candidate key, 3NF and BCNF are equivalent.

eg. Student

Student	Course	Teacher
Ram	DB	John
Ram	PT	Tedd
Shyam	DB	John
Shyam	PT	Tedd

where, key = {student, course}

{student, course} → Teacher

Teacher → course

but, Teacher is not superkey, but determine course. So, not in BCNF.

Table1

Teacher	Course
John	DB
Tedd	PT

Table 2

Student	Course
Ram	DB
Ram	PT
Shyam	DB
Shyam	PT

Table3

Course

DB

PT

Denormalization

- It is process of attempting to optimise the performance of a database by adding redundant data or by grouping data.
- In relational database, denormalization is an approach to speed up read performance in which the administrator selectively adds back specific instances of redundant data after the data structure has been normalized.
- It is needed when multiple joins in same query can have negative impact on performance.
- Denormalization should take place after satisfactory level of normalization has taken and that any required constraints or rules have been created to deal with inherent anamolies in design

Decomposition

- refers to breaking down of one table into multiple tables.
- desirable properties of decomposition are:
  i) attribute preparation
  ii) Lossless description
  iii) Dependency prevention
  iv) Lack of redundancy

Assertion

- A condition that we want database to always follow.
- It means to specify integrity constraints.
- An assertion in SQL takes the form:

CREATE ASSERTION <assertion-name> CHECK <predicate>

- When an assertion is made, the system tests it for validity and ~~test~~ it again on every update that may ~~update~~ violate assertion.
- Assertion should be used with great care as testing may cause significant amount of overhead.

eg. the sum of all loan amounts for each branch must be less than sum of all account balances at branch.

CREATE assertion sum-constraints check (not exists (select * from branch where (select sum(amount) from loan where loan.branchname = branch.branchname) >= (select sum(balance) from account where account.branchname = branch.branchname)))

Date / /
Page No.

Trigger

- It is a special kind of stored procedure that executes in response to certain action on table like insertion, deletion or update of data.
- It is db object bound to table and is executed automatically.
- We cannot explicitly invoke triggers.

Multivalued Dependency

- It is a functional dependency where determinant can determine more than one value.
- For MVD,
  i) There must be 3 attributes in relation say. A, B, C.
  ii) Given A one can determine multiple values of B and also one can determine multiple values of C.
  iii) B & C are independent of one another.

eg

SID	Major	Activities
1	DB	Football
2	DB	Basketball
2	Math	Football
1	Math	Basketball

Here, SID → Major, SID → Activities.

CHAPTER 6: Security

Date / /
Page No.

- unauthorized access or manipulation of database creates problem for organization.
- Security refers to protection of data against unauthorized disclosure, alteration or destruction
- Main objectives of designing secured database system are integrity, availability and secrecy.
- Database security is about controlling access to information. ie. some information be available freely and other informations be available to certain authorized people or groups.
- Database security system stores authorization rules and enforces them for database access.
- Database security can be ph

1) **Physical security**

It refers to security of hardware and protection of sites where computer resides.

2) **Logical Security**

It refers to software safeguards for organization, systems, including user id verification, password access, access rights and authority levels.

Date / /
Page No.

Security Violation:

↳ Among the forms of malicious access are unauthorized reading of data, unauthorized modification of data and unauthorized destruction of data.

Database Security levels

↳ To ensure database security, security at different levels must be maintained.

1) Database System:
Database system have to ensure that authorization, restrictions are not violation.

2) Operating System:
Weakness in operating system security is concerned with unauthorized access to database

3) Network
Software level security within network software is important.

4) Physical
Sites with computer system must be physically secure against armed entry by intruders.

5) Human
Users must be authorized carefully.

↳ Security is protecting data against unauthorized users.

↳ Integrity is protecting data against authorized users.

Authorization:

↳ Authorization is a security mechanism used to determine user/client privilege or access level related to system resources.

↳ During authorization system verifies authenticated users access role and either grant or revoke resource access.

↳ Authorization includes
i) Permitting only certain users to access, process or alter data.
ii) Applying different limitations on users access or actions. Here limitations placed on users can apply to object such as tables, rows etc

↳ Authorization on data include
i) Authorization to read data
ii) Authorization to insert new data
iii) Authorization to update data.
iv) Authorization to delete data

↳ Each of these type of authorization is called

privilege.

↳ Users are authorized all or none or combination of these types of privileges on specified parts of database such as relation or views

↳ In addition to authorization data, users may be granted database schema, allowing them to create modify or drop relations.

↳ The ultimate form of authority is that given to a DBA (Database Administrator).

↳ DBA may authorize new users, restructured the database etc.

Granting and Revoking of privilege

↳ SQL standard include the privileges select, insert, update, delete.

↳ all privileges can be used for all allowable privileges.

↳ a user who creates a new relation is given all privileges automatically.

- Grant statement:

```
grant <privilege list>
on <relation or view>
to <user|role list>
```

eg:

1)
```
grant select
ON employee
to Ram, Hari;
```

* to read tuples in relation select authorization is required.

eg: 2) i) grant update
on employee
to Ram, Hari;

ii) grant update (salary)
on employee
to Ram, Hari;

* to update any tuple in relation 'update' authorization is used.
* 'update' authorization may be given either on all attributes of relation or only some.

↳ To insert tuples into relation, 'insert' authorization is used.

↳ 'insert' privilege may also specify ~~only those~~ a list of attributes.

↳ any inserts to the relation must specify only these attributes.

↳ The system either gives default value or NULL for remaining attributes.

eg: 3) i) grant insert
on employee
to Ram, Hari;

ii) grant insert (name, address on employee
to Ram, Hari;

↳ To delete tuples from relation 'delete' authorization is used.

eg 4) grant delete
on employee
to Ram, Hari;

↳ privileges granted to 'public' are implicitly granted to all current and future users.

↳ To revoke an authorization, we use revoke statement

revoke <privilege list>
on <relation or view>
from <user/role list>;

eg:
i) revoke select
on employee
from Ram, Hari;

ii) revoke update (salary)
on employee
from Ram, Hari;

↳ System can support user groups also known as 'roles' and can thus provide a way of allowing all with same role to share same object

Ute:
- create role instructor;
- grant select
on employee
to instructor;

↳ role is a database object that groups one or more privileges.

- roles can be assigned to users or groups or other roles by using Grant statement.
- users that are members of roles have privileges that are defined for the role with which to access data.

Data Encryption / Crypto System.

→ is storing and transmitting data in encrypted form.
→ original data is called plain text.
→ plaintext is encrypted using encrypted algorithm (whose inputs are plaintext and encryption key)
→ output is called ciphertext.
→ Encryption refers to the process of transforming data into a form that is unreadable unless the reverse process of decryption is applied.
→ encryption algorithm use an encryption key to perform encryption and requires decryption key to perform decryption.
→ encryption is widely used today for protecting data in transit in a variety of applications such as data transfer in internet, cellular phone n/ws.
→ encryption is also used to carry out other tasks like authentication.

→ In db, encryption is used to store data in secure way so that even if data is acquired by unauthorized users the data will not be accessible without decryption key.

→ a good encryption technique has following properties

a) It is relatively simple for authorized users to encrypt and decrypt data.

b) It depends on encrypts key used to encrypt data.
- In symmetric key encryption, encryption key is also used to decrypt.
- In assymetric key encryption, two different keys public and private key are used to encrypt and decrypt data.

c) Its decryption key is extremely difficult for an intruder to determine even if intruder has encrypted data.
- In assymetric key encryption, its difficult to infer private key even if public key is available.

e.g:
AES (Advanced Encryption Standard)
DES (Data Encryption Standard)

→ In symmetric key encryption, authorized users must be provided with encrypted key via secure mechanism.

→ In public key encryption
- two keys private & public.
- public key are published.
- private key is known only to user to whom key belongs.

CHAPTER 7: QUERY PROCESSING

Date / /
Page No.

Introduction to Query processing

- is transforming a query written in high level language typically in SQL into correct and efficient execution strategy expressed in a low level language and to execute the strategy to retrieve required data.
- Basic steps in query processing
  i) Parsing and Translation
  ii) Optimization
  iii) Evaluation.

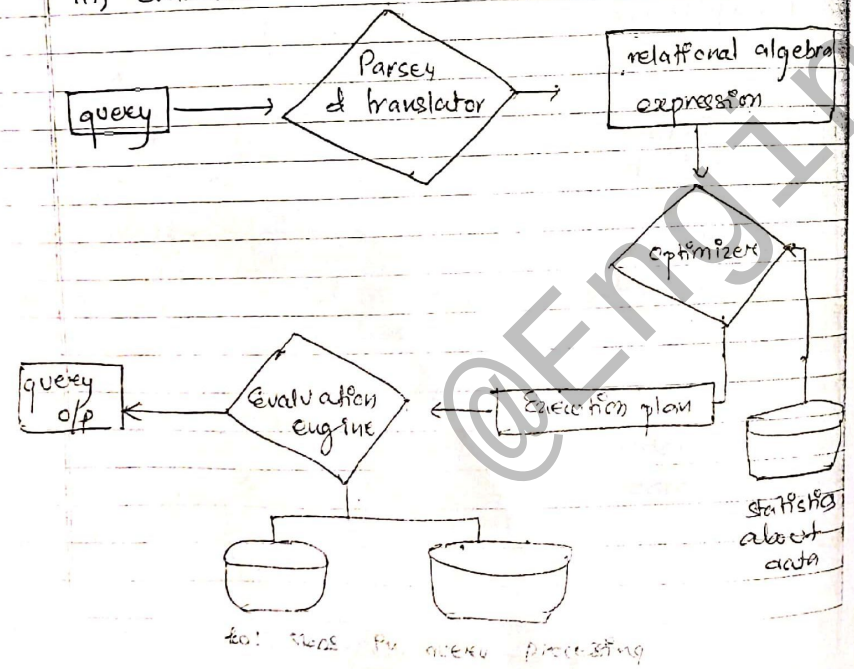


fig: steps in query processing

Date / /
Page No.

Parsing and Translating:

→ translate the query into a internal form which is then translated into relational algebra.
→ parser checks syntax, verifies relation.
→ the relational algebra representation of a query sep specifies only partially how to evaluate a query.
→ a relational algebra expression may have many equivalent expressions.

Consider a query
- Select salary from employee where salary > 20000. Now, equivalent relational algebra expressions are

$$\sigma_{salary > 20000}(\Pi_{salary}(employee))$$

or

$$\Pi_{salary}(\sigma_{salary > 20000}(employee))$$

→ a sequence of primitive operations that can be used to evaluate a query is called query execution plan or query evaluation plan.

Evaluation

- the query execution engine takes a query evaluation plan, executes that plan and return answers to query.

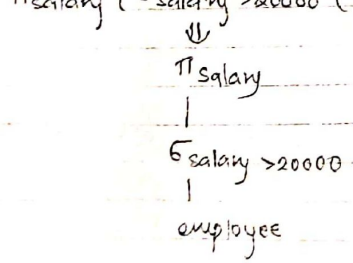
Query optimization

- amongst all equivalent evaluation plans choose the one with lowest cost.
- cost is estimated using statistical information from database catalog.
  e.g. no. of tuples in each relation, size of tuple etc.
- once the query plan is choosen, the query is evaluated with that plan and result of query is output.

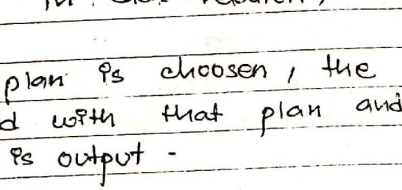
Construction of Parse Tree

→ a leaf node is created for each base relation in query.
→ a non leaf node is created for each intermediate relation produced by relational algebra expression.
→ the root node represents result of query.
→ the sequence of operation directed from leaves to root.

e) $\Pi_{salary}(\sigma_{salary>20000}(employee))$



e) $\Pi_{name, street}(\sigma_{status='open'}(order \bowtie Customer))$



Measure of Query Cost:-

↳ Cost is generally measured as total elapsed time for answering query.
- many factors contribute to time cost like disk accesses, CPU or even n/w communication

↳ Typically, disk access is the predominant cost and is also relatively easy to estimate. It is measured by taking into account:
- Number of seeks * average seek cost.
- Number of blocks read * average block read cost
- Number of blocks written

↳ Cost to write a block is greater than cost to read a block because data is read back after being written to ensure that the write was successful

↳ For simplicity we just use number of block transfers from disk and the no. of seeks as the cost measures
- t_T = time to transfer one block.
- t_S = time to one seek
- cost for b block transfers plus S seeks

$$b * t_T + S * t_S$$

↳ We ignore CPU cost for simplicity as real systems do take CPU cost into account. ↳ We do not include cost to writting in our cost formula.

Query Operation

* **Selection Operation**
- The lowest level query processing operator for accessing data is file scan.
- Search and retrieve records for a given selection condition.

Linear Search
- Scan each file block and test all records to see whether they satisfy selection condition
- Cost estimation = br block transfer + 1 seek

Here, br denotes no. of block containing records from relation r.

↳ If selection is on a key attribute, can stop on finding record.
- Cost = (br/2) block transfer + 1 seek.

↳ Linear Search can be applied regardless of
- selection condition or
- ordering of records in file or
- availability of indices.

Selections using Indices.

↳ **Index Scan**
- Search algorithm that use an index
- selection condition must be on search key of index.

↳ **A2 (primary Index, equality on key)**
- retrieve a single record that satisfies the corresponding equality condition
- Cost = $(h_i + 1) * (t_T + t_S)$

↳ **A3 (primary Index, equality on non key)**
- retrieve multiple records.
- records will be on consecutive blocks
- let b = no. of blocks containing matching record.
- Cost = $h_i * (t_T + t_S) + t_S + t_T * b$

→ A4 (Secondary Index, equality on non key)
- retrieve a single record if search key is candidate key.
  * cost = $(h_i + 1) * (t_T + t_S)$
- retrieve multiple records if search key is not candidate key.
- each of n matching records may be on different block
- cost = $(h_i + n) * (t_T + t_S)$
- can be very expensive.

Sorting

- A query may specify that the output should be sorted.
- The processing of some relational query operations can be implemented more efficiently based on sorted relations (join operation).
- For relations that fit into memory, Techniques like quick sort can be used
- For relations that do not fit into memory, external merge sort can be used.

External Merge Sort

- Let M denote memory size (in pages)

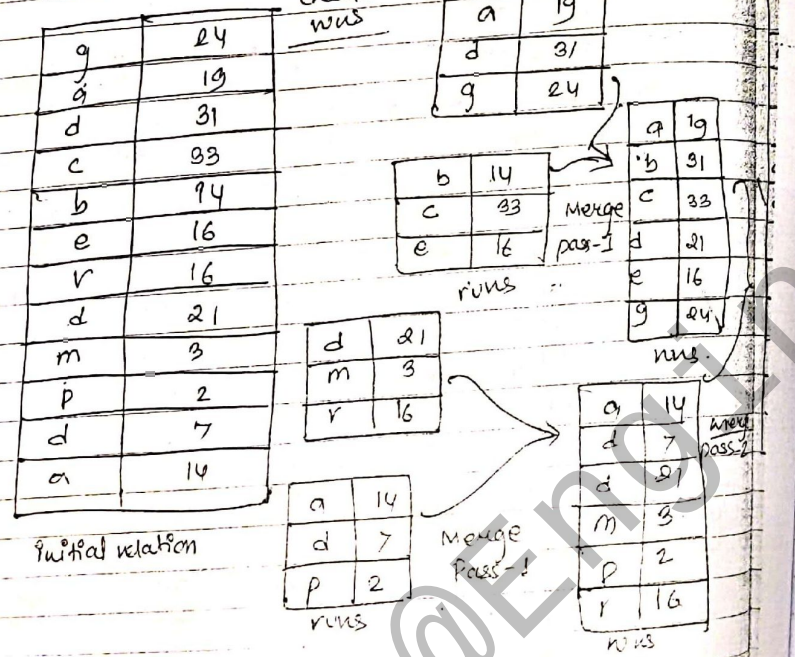
1. Create sorted runs.
   - Let i be 0 initially.
   - repeatedly do the following till the end of relation:
     (a) Read M blocks of relation in memory.
     (b) Sort the in-memory blocks.
     (c) Write sorted data to Run R_i, Increment i.
   - Let the final value of i be N.

2. Merge the runs (N-way merge):-
   - We assume $N < M$.
   - Use N blocks of memory to buffer input runs and 1 block to buffer output. Read the first block of each run into its buffer page.
   - repeat
     * Select the first record (in sort order) among all buffer pages
     * Write the record to output buffer. If output buffer is full write it to disk.
     * Delete the record from its input buffer page If the buffer page becomes empty then read next block of run into buffer.
   - until all input buffer pages are empty.

3. If $N \geq M$, several merge passes are required
   - In each pass, contiguous groups of M-1 runs are merged
   - A pass reduces the number of runs by factor of M-1 and creates runs longer by same factor.

e.g: if ~~M = 11~~ M = 11, and there are 90 runs, one pass reduces the no. of runs to 9, each 10 times the size of initial runs.

→ Finally, repeated passed are performed till all runs have been merged into one.



Query optimization

→ Is an important aspect of query processing.
→ Is the activity of choosing an efficient execution strategy for processing a query.
→ Aim is to choose one of memory equivalent transformation that minimizes resource usage.
→ Reduce the total execution time of the query, which is the sum of execution times of all individual operators that make up the query.
→ There can be enormous difference in terms of performance between different evaluation plan for same query.
e.g: seconds vs days to execute same query.

→ cost based query optimization.
⇒ generate logically equivalent expressions by using set of equivalent rules
– annotate the expressions to get alternative query evaluation plans
– select the cheapest plan based on estimated cost.
→ Estimation of query evaluation cost based on statistical info from catalogue manager in combination with expected performance of algo.

Equivalent Rules:

1. Conjunctive selection operations can be deconstructed into sequence of individual selections.
   - $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. Selection operations are commutative.
   - $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. Selection can be combined with cartesian products and theta joins.
   - $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   - $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
4. Theta join (and natural join) operations are commutative
   - $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
5. Natural join operations are associative
   - $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
6. Union and intersection operations are commutative
   - $E_1 \cup E_2 = E_2 \cup E_1$
   - $E_1 \cap E_2 = E_2 \cap E_1$
7. Union and intersection operations are associative
   - $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$
   - $(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$
8. Selection operation distributes over union, intersection and set difference
   - $\sigma_p(E_1 - E_2) = \sigma_p(E_1) - \sigma_p(E_2)$
9. projection distributes over union.
   - $\Pi_A(E_1 \cup E_2) = (\Pi_A(E_1)) \cup (\Pi_A(E_2))$

Heuristic Optimization

- Cost based optimization is expensive even with dynamic programming.
- System's may use Heuristics to reduce the no. of choices that must be made in cost based fashion.
- Heuristic optimization transforms the query tree by using a set of rules that typically improve execution performance.
  - perform selection early. (reduces the no of tuples)
  - perform projection early (reduces the no. of attributes)
  - perform most restrictive selection and join operations ~~selection~~ before other similar operations
  - some systems use only heuristics, others combine heuristic with partial cost based optimization

Steps in Heuristic Optimization

↳ Deconstruct conjunctive selections into a sequence of single selection operations.

↳ Move selection operations down the query tree for the earliest possible execution.

↳ Execute first those selection and join operation that will produce the smallest relation.

↳ replace cartesian product operation that are followed by selection condition by join operations.

↳ Deconstruct and move as far down the tree as possible list of ~~poss~~ projection attributes, creating new projections where needed.

↳ Identify those sub-trees whose operations can be ~~part~~ pipelined & execute them.

1) Consider a relational schema.
Teacher (TeacherID, TeacherName, office)
- write SQL statement for the following task.
i) To create a table.
ii) To eliminate duplicate rows.
iii) To add new column 'Gender' in table.
iv) To sort data in table.
v) To delete row.

⇒

1) create table Teacher
(
TeacherID ~~text~~ INT PRIMARY KEY,
TeacherName text,
Office text,
);
~~INSERT INTO Teacher VALUES~~

ii) Select distinct (*) from Teacher.

iii) ALTER table Teacher
Add Gender text

iv) Select * from Teacher
order by TeacherID desc.

v) delete from Teacher.

2.

Employees (emp-ID, Fname, Lname, email, phone, hiredate, jobid, salary, manager-id, Department id)

Departments (Departmentid, dname, manager-id, location_id)

locations (location_id, street, postcode, city, county_id, state)

i) Show lastname, salary and department id of employee whose salary is between 5000 & 8000.

ii) Show the records of those employees whose name starts with letter 'S' and works in department 100.

iii) List all employees firstname, lastname, email, department name whose department name is 'programming'.

iv) List records of all employees department name, managerid, street whose employeeid = 110.

i) ⇒ Select Employees.lastname, Employee.Salary, Departments.Departmentid. ~~inner~~ join from Emplo

→ Select lastname, salary, departmentid from employees where salary between 500 & 800.

ii) Select * from employee where fname like 's%' and department_id = 100.

iii) Slect employees.fname, employees.lname, employee.email, department.departmentname from employees inner join department on

department ~~dname~~ department programming.
employees. ~~emp~~ id = department. department id where department.dname = 'programming'.

iv) Select department. ~~departmentname~~ dname, department.manager_id, location.street. from ~~department~~ location inner join ~~location~~ department on department.location_id = locations.location_id inner join employee ~~where~~ on department.departmentid = employee.department id
where employees.empid = 110.

3) Employee (EID, Name, Post, Age),
Post (Post_title, Salary)
Project (PID, PName, Duration, Budget)
Work (PID, EID, joinDate)

i) List name of employees whose age is greater than average age of employees.

ii) Display all employee numbers of those employees who are not working in any project.

iii) List name of employee and their salary who are working in project 'DBMS'

iv) Update database to that 'Rishab' now lives in 'Butwal'.

Answers

1) select name from employee where age > ~~avg(age)~~ (select avg(age) from employee);

2) Select count (EID) from employee
inner join work
on employee. EID = work. EID
where work. PID = NULL.

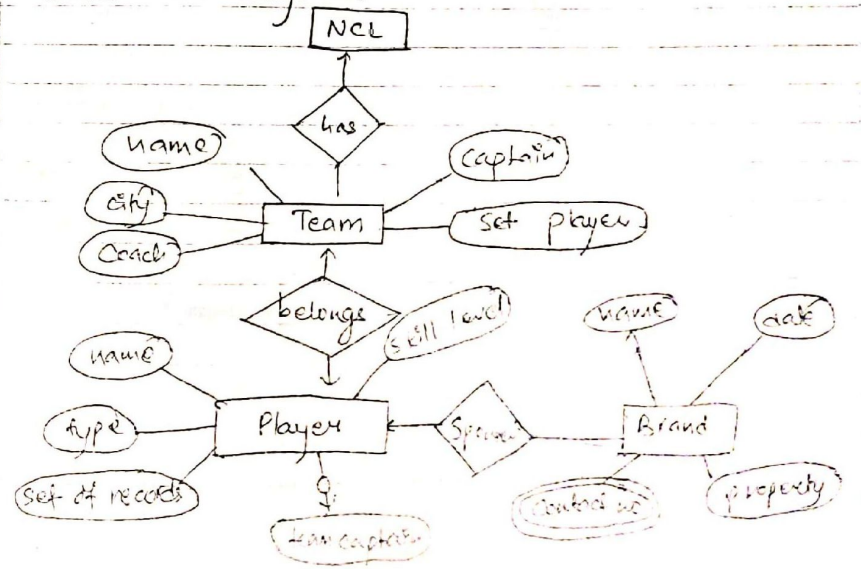
3) select employee.name, post.salary from employee inner join ~~employee~~ post on employee.postID = post.postID
inner join ~~work~~ on
Work on ~~employee~~
work. EID = employee. EID
inner join ~~work~~ on project
on work. PID = Project. PID where
Project. Pname = 'DBMS'.

4) ALTER table employe add location text

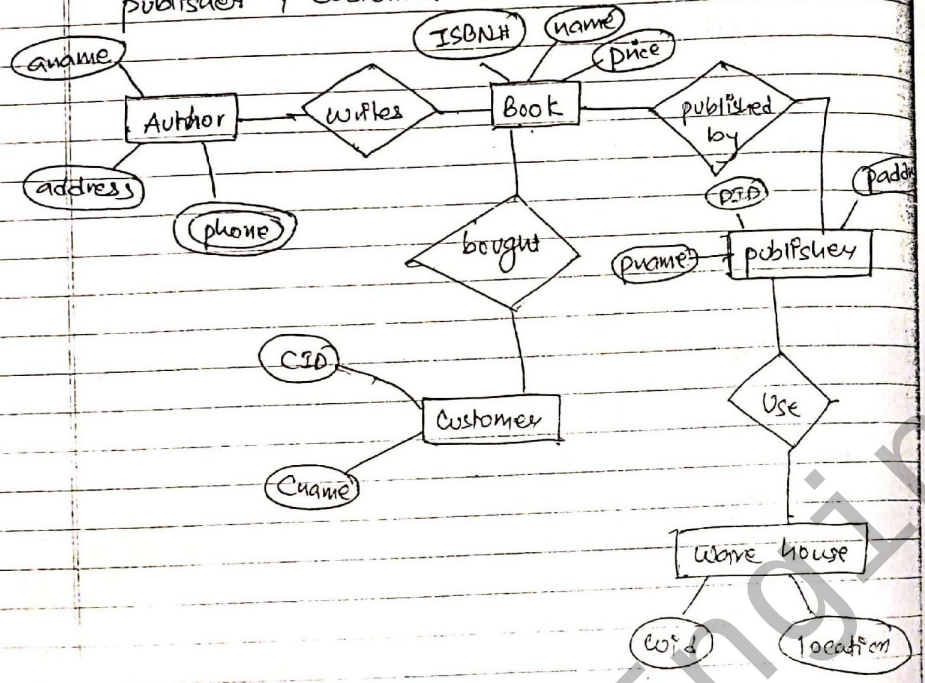
5) UPDATE employee set location = 'Butwal'
where employee. name = 'Rishab'.

6) Suppose you are given following requirements for simple database National Cricket League (NCL)
- the NCL has many teams.
- each team has a name city, coach, captain, set of players
- each player belongs to only one team.
- each player has a name, type, skill level, set of records.
- a team captain is also a player.
- each player is sponsored by at least one brand.
- a brand has its name, established date property, multiple contact no.

⇒ Draw ER diagram:



Construct an E-R diagram for online Book store where entities are author, book, publisher, customer & ware house.



Consider following relational database schema:

Employee (EName, street, city)
Works (EName, CompanyName, Salary)
Company (CompanyName, City)
Manager (EName, ManagerName)

Write relational algebra for following

→ Find names of all employees who work for Hamro Bank.

⇒ $r_1 \leftarrow \Pi_{EName}(\sigma_{CompanyName = 'HamroBank'}(Employee \bowtie Works))$

OR

Equijoin. $(Employee \bowtie_{Employee.Ename = Works.ename} Works.)$

→ Give all employees of Hamro Bank a 5percent salary ~~use~~. raise.

$r_2 \leftarrow \Pi_{EName, CompanyName, salary * 1.05}(\Pi_{EName}(r_1))$

~~$\sigma_{CompanyName = 'HamroBank'}(Employee \bowtie Works)$~~

Insert record

Date / /
Page No.

→ Delete all tuples in works relation for employee of Hamro Bank.

Delete

works ← works − σ companyName 'HamroBank' (works)

new. (original − selection)

↳ Insert record into employee

employee ← employee ∪ ('Hari', 'new record', 'ktm')

↳ Modify database so that 'Harish now lives in Biratnagar.

⇒ employee ← π EName, Street, 'Biratnagar' (σ EName = 'Harish') ∪ (employee − σ name = 'Harish' (employee))

Or

r1 ← π EName, Street, City (σ EName = 'Harish' (employee))

or employee.

r2 ← π EName, Street, 'Biratnagar' (r1) (employee).

Date / /
Page No.

Given,

Student (CRN, Name, gender, Address, Telephone)
Course (CourseID, CName, hour, TID)
Teacher (TID, TName, Office)
Registration (CRN, CourseID, Date)

↳ Count the number of student registered subject in year 2015 genderwise

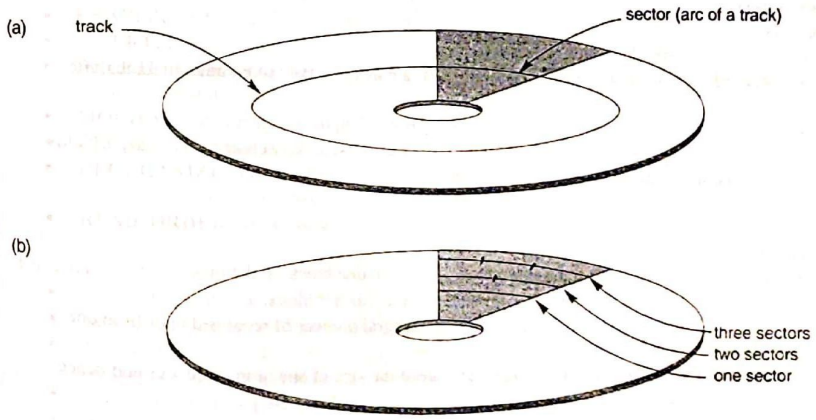
G count (gender) (Student ⋈ σ Date = 2015 Registration)

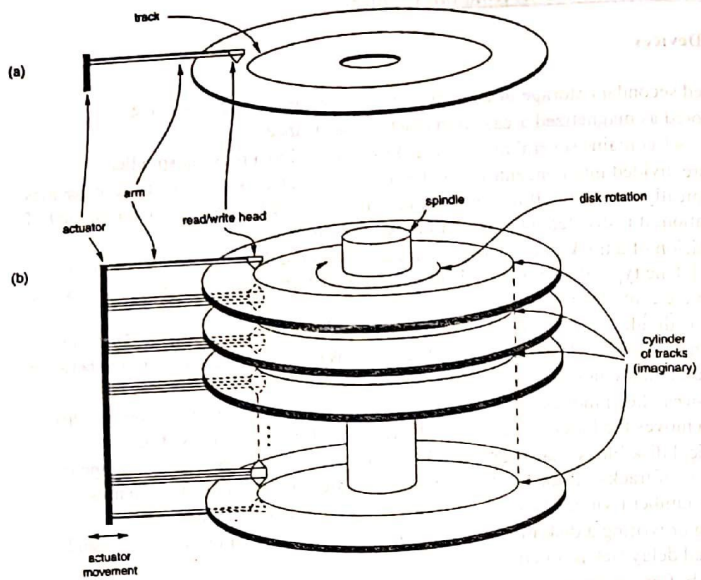
Chapter 8

Filing and File Structure
(Read well, ½ Question 7-8 marks from this chapter)

Disk Storage Devices

- Preferred secondary storage device for high storage capacity and low cost.
- Data stored as magnetized areas on magnetic disk surfaces.
- A *disk pack* contains several magnetic disks connected to a rotating spindle.
- Disks are divided into concentric circular *tracks* on each disk *surface*. Track capacities vary typically from 4 to 50 Kbytes. Because a track usually contains a large amount of information, it is divided into smaller *blocks* or *sectors*.
- The division of a track into *sectors* is hard-coded on the disk surface and cannot be changed. One type of sector organization calls a portion of a track that subtends a fixed angle at the center as a sector.
- A track is divided into *blocks*. The block size B is fixed for each system. Typical block sizes range from B=512 bytes to B=4096 bytes. Whole blocks are transferred between disk and main memory for processing.
- A *read-write* head moves to the track that contains the block to be transferred. Disk rotation moves the block under the read write head for reading or writing.
- A physical disk block (hardware) address consists of a cylinder number (imaginary collection of tracks of same radius from all recorded surfaces), the track number or surface number (within the cylinder), and block number (within track).
- Reading or writing a disk block is time consuming because of the seek time s and rotational delay (latency) **rd**.
- Double buffering can be used to speed up the transfer of contiguous disk blocks.





Typical Disk Parameters

Records

- Fixed and variable length records
- Records contain fields which have values of a particular type (e.g., amount, date, time, age)
- Fields themselves may be fixed length or variable length
- Variable length fields can be mixed into one record: separator characters or length fields are needed so that the record can be "parsed".

Blocking

- Blocking: refers to storing a number of records in one block on the disk.
- Blocking factor (*bfr*) refers to the number of records per block.
- There may be empty space in a block if an integral number of records do not fit in one block.
- *Spanned Records*: refer to records that exceed the size of one or more blocks and hence span a number of blocks.

Files of Records

- A file is a *sequence* of records, where each record is a collection of data values (or data items).
- A *file descriptor* (or *file header*) includes information that describes the file, such as the *field names* and their *data types*, and the addresses of the file blocks on disk.
- Records are stored on disk blocks. The *blocking factor bfr* for a file is the (average) number of file records stored in a disk block.
- A file can have *fixed-length* records or *variable-length* records.
- File records can be *unspanned* (no record can span two blocks) or *spanned* (a record can be stored in more than one block).
- The physical disk blocks that are allocated to hold the records of a file can be *contiguous*, *linked*, or *indexed*.
- In a file of fixed-length records, all records have the same format. Usually, unspanned blocking is used with such files.
- Files of variable-length records require additional information to be stored in each record, such as *separator characters* and *field types*. Usually spanned blocking is used with such files.

(Imp)

Operation on Files

Typical file operations include:

- **OPEN:** Readies the file for access, and associates a pointer that will refer to a *current* file record at each point in time.
- **FIND:** Searches for the first file record that satisfies a certain condition, and makes it the current file record.
- **FINDNEXT:** Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
- **READ:** Reads the current file record into a program variable.
- **INSERT:** Inserts a new record into the file, and makes it the current file record.
- **DELETE:** Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
- **MODIFY:** Changes the values of some fields of the current file record.
- **CLOSE:** Terminates access to the file.
- **REORGANIZE:** Reorganizes the file records. For example, the records marked deleted are physically removed from the file or a new organization of the file records is created.
- **READ_ORDERED:** Read the file blocks in order of a specific field of the file.

Unordered Files

- Also called a *heap* or a *pile* file.
- New records are inserted at the end of the file.
- To search for a record, a *linear search* through the file records is necessary. This requires reading and searching half the file blocks on the average, and is hence quite expensive.
- Record insertion is quite efficient.
- Reading the records in order of a particular field requires sorting the file records.

Ordered Files

- Also called a *sequential file*.
- File records are kept sorted by the values of an *ordering field*.
- Insertion is expensive: records must be inserted in the *correct order*. It is common to keep a separate unordered *overflow* (or *transaction*) file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.
- A *binary search* can be used to search for a record on its *ordering field value*. This requires reading and searching log2 of the file blocks on the average, an improvement over linear search.
- Reading the records in order of the ordering field is quite efficient.

	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
	⋮					
block n – 1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
block n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

File Organization

A file organization essentially means organization of records in the file. Some basic file organization techniques are given below:

1. Heap or Pile
2. Sequential
3. Indexed Sequential
4. Direct or Hashed

Heap or Pile

In a heap or a pile file records are collected in the order they arrive. The blocks used in a heap are linked by pointers. A table lookup approach can also be adopted to locate the blocks. When a new record is inserted, it is placed in the last block if there is space. If last block cannot accommodate the record, a new block is allocated and record to be inserted is placed. Deletion is performed by setting a flag called deletion bit in the deleted record.

Fig:

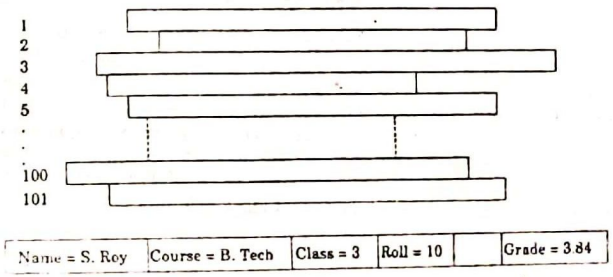


Fig. 2.21 A heap file with a representative record

2.7 THE SEQUENTIAL FILE ORGANIZATION

In a sequential file records are placed sequentially onto the storage media. In addition, the records in the file are usually ordered according to the values of key attributes. The key, as defined in Section 2.4, is used to identify the records uniquely. If key consists of more than one attribute, as in the case of student records in Fig. 2.5, then one attribute belonging to the key is called primary key (or high order sort key) and the remaining attributes constitute secondary keys. Thus for the student record in Fig. 2.5, the course may be taken as primary key and the class and roll no. constitute secondary keys. In the case of Employee record in Fig 2.6. Employee ID is a key as well as the primary key. A typical sequential file with employee records is shown in Fig. 2.22. Along with the attribute names, the

EMP. ID	Name	Other fields					
000 - 121	A. Roy	. . . . .	001 - 125	B. Saha	. . . . .	001 - 130	T. Khan
R_1			R_2			R_3	

Fig. 2.22 A sequential file

information regarding the data type of the attribute values (e.g. string, integer, etc.) and their permissible range (an integrity constraint) may be stored in the directory. If a new attribute is to be added to a record, then the entire file needs reorganization. In order to avoid this, an extra space is sometimes allocated in sequential files. The fixed length records are generally preferred as processing programs in that case are simpler. However, if fixed length records are used, then the length (in words or bytes) of the individual fields of the record must be sufficiently large to accommodate the corresponding attribute values within their permissible range.

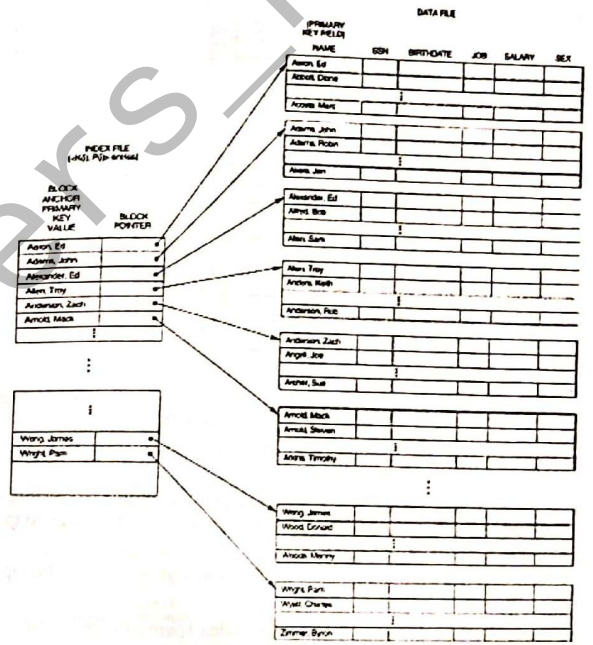
Indexed File Organization

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries <field value, pointer to record>,which is ordered by field value
- The index is called an access path on the field
- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller

- A binary search on the index yields a pointer to the file record

Primary Index

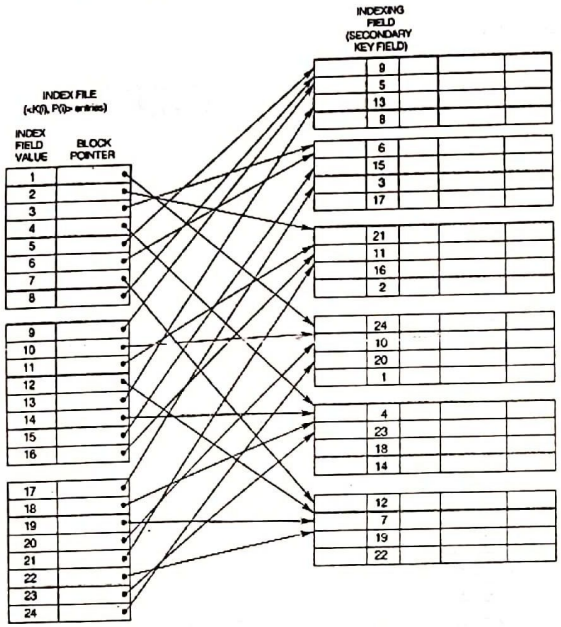
- Defined on an ordered data file
- The data file is ordered on a key field
- Includes one index entry for each block in the data file; the index entry has the key field value for the first record in the block, which is called the block anchor
- A similar scheme can use the last record in a block.
- A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.



Secondary Index

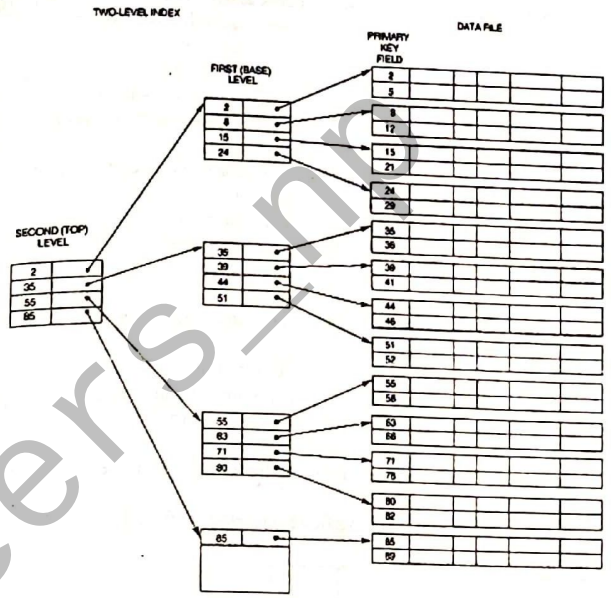
- A secondary index provides a secondary means of accessing a file for which some primary access already exists.
- The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non key with duplicate values.
- The index is an ordered file with two fields.
  - The first field is of the same data type as some non ordering field of the data file that is an indexing field.

- The second field is either a block pointer or a record pointer. There can be many secondary indexes (and hence, indexing fields) for the same file.



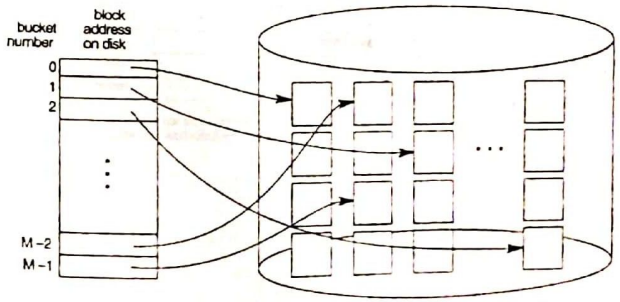
Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index to the index itself; in this case, the original index file is called the first-level index and the index to the index is called the second-level index.
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the top level fit in one disk block
- A multi-level index can be created for any type of first level index (primary, secondary, clustering) as long as the first-level index consists of more than one disk block



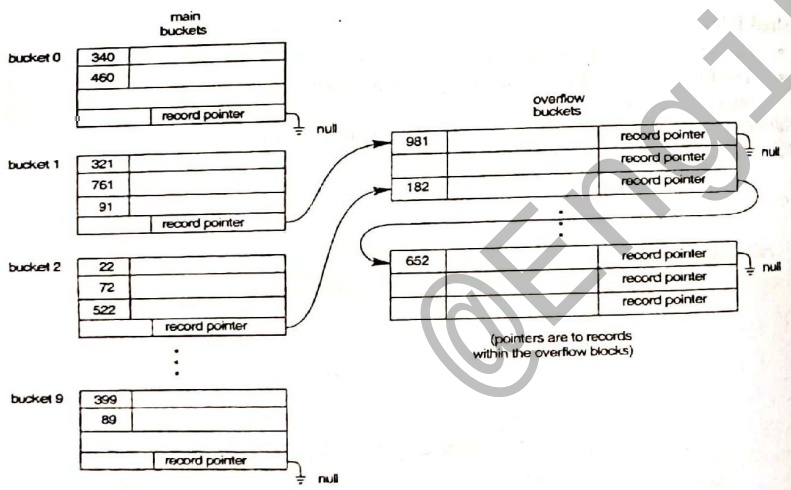
Hashed Files

- Hashing for disk files is called *External Hashing*
- The file blocks are divided into M equal-sized *buckets*, numbered bucket0, bucket1, ..., bucket M-1. Typically, a bucket corresponds to one (or a fixed number of) disk block.
- One of the file fields is designated to be the hash key of the file.
- The record with hash key value K is stored in bucket i, where i=h(K), and h is the *hashing function*.



- Search is very efficient on the hash key.
- Collisions occur when a new record hashes to a bucket that is already full. An overflow file is kept for storing such records. Overflow records that hash to each bucket can be linked together. There are numerous methods for collision resolution, including the following:
- ***Open addressing:*** Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.
- ***Chaining:*** For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. In addition, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.
- ***Multiple hashing:*** The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.
- To reduce overflow records, a hash file is typically kept 70-80% full.
- The hash function h should distribute the records uniformly among the buckets; otherwise, search time will be increased because many overflow records will exist.
- Main disadvantages of static external hashing:
  - Fixed number of buckets M is a problem if the number of records in the file grows or shrinks.
  - Ordered access on the hash key is quite inefficient (requires sorting the records).

Hashed Files - Overflow handling

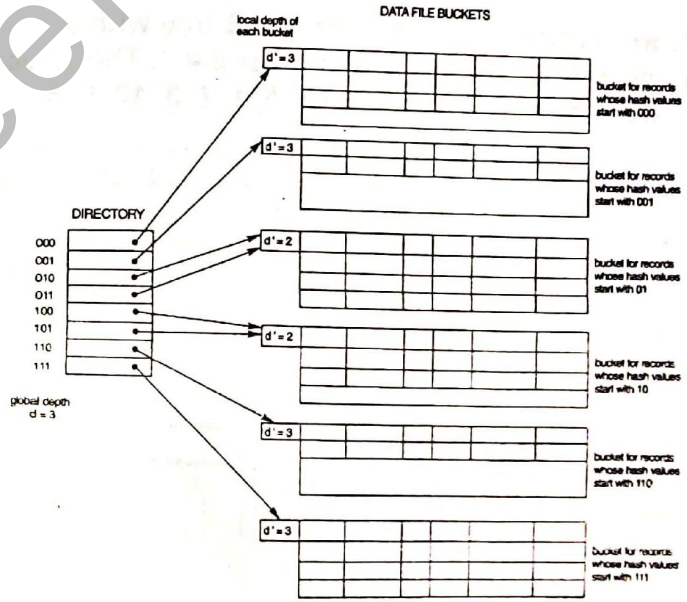


Dynamic and Extendible Hashed Files

Dynamic and Extendible Hashing Techniques

- Hashing techniques are adapted to allow the dynamic growth and shrinking of the number of file records.
- These techniques include the following: *dynamic hashing* , *extendible hashing* , and *linear hashing* .
- Both dynamic and extendible hashing use the *binary representation* of the hash value h(K) in order to access a *directory*. In dynamic hashing the directory is a binary tree. In extendible hashing the directory is an array of size 2d where d is called the *global depth*.
- The directories can be stored on disk, and they expand or shrink dynamically. Directory entries point to the disk blocks that contain the stored records.
- An insertion in a disk block that is full causes the block to split into two blocks and the records are redistributed among the two blocks. The directory is updated appropriately.
- Dynamic and extendible hashing do not require an overflow area.
- Linear hashing does require an overflow area but does not use a directory. Blocks are split in *linear order* as the file expands.

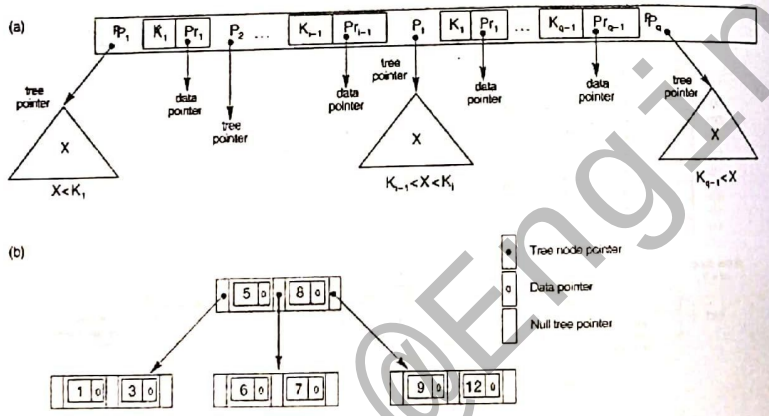
Extendible Hashing



B-Tree Index File Organization

- Because of the insertion and deletion problem, most multi-level indexes use B-tree or B+-tree data structures, which leave space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full
- An insertion into a node that is not full is quite efficient; if a node is full the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- A deletion is quite efficient if a node does not become less than half full
- If a deletion causes a node to become less than half full, it must be merged with neighboring nodes

B-tree structures. (a) A node in a B-tree with q – 1 search values. (b) A B-tree of order p = 3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



Data Dictionary Storage

1. The database also needs to store information about the relations, known as the data dictionary. This includes:
   - Names of relations.
   - Names of attributes of relations.
   - Domains and lengths of attributes.
   - Names and definitions of views.
   - Integrity constraints (e.g., key constraints).

   plus data on the system users:

   - Names of authorized users.
   - Accounting information about users.

   plus (possibly) statistical and descriptive data.

   - Number of tuples in each relation.
   - Method of storage used for each relation (e.g., clustered or non-clustered).
2. When we look at indices (Chapter 11), we'll also see a need to store information about each index on each relation:
   - Name of the index.
   - Name of the relation being indexed.
   - Attributes the index is on.
   - Type of index.
3. This information is, in itself, a miniature database. We can use the database to store data about itself, simplifying the overall structure of the system, and allowing the full power of the database to be used to permit fast access to system data.

2.5 BUFFER MANAGEMENT

Since the files are stored in mass storage devices such as disks, to retrieve information stored in a file, the appropriate portion of the file is brought into the main memory in units of disk blocks. The performance of a file system depends on how fast the basic operations (e.g., retrieve, insert, update and delete) can be performed on a file. Since the access time to a disk is a hardware parameter and cannot be reduced further until the technology developed further, the performance of a file system can be upgraded if the number of block accesses required for a sequence of basic file operations is minimized. Such minimization is possible if the chance that a block being accessed is already in the main memory is maximized.

The buffer is that part of the main memory available for the storage of the contents of some of the blocks. The subsystem responsible for the allocation of buffer space is called the *buffer manager*.

The buffer manager services all requests made by the file management system for blocks of the files currently being operated upon by the DBMS. If a requested block is already in the buffer, the address of the block in the main memory is passed on to the file manager and subsequently to the DBMS. If the block is not in the buffer, the buffer manager reads the block from the disk into an empty buffer and passes the address of the block in the main memory (i.e., address of the buffer).

The most commonly used technique for buffer management is the **buffer cache** (cache is taken from the French word 'Cacher' meaning to hide). A *cache* is basically a collection of disk blocks that are kept in the memory for performance reasons. We give below a brief account of the buffer cache implemented in the UNIX system.

The buffer cache is maintained by UNIX as a pool of internal data buffers which contain the data in recently used disk blocks. Each buffer is an in-memory copy of a disk block. A disk block can never map into more than one buffer at a time.

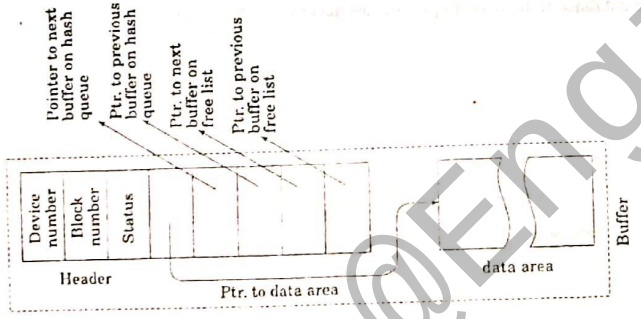


Fig. 2.16 Structure of a buffer

A buffer consists of two parts (Fig. 2.16) namely, a *buffer header* and a few bytes of contiguous memory having the same size as a disk block. The buffer header is used to identify a buffer and to dynamically create and maintain a multi-linked list of buffers. As it will be shown latter, the buffer pool is maintained as a *dynamic multi-linked list of buffers*. The other part of a buffer, i.e., the associated contiguous memory is used to contain data from a disk block. The values of device number and block number fields are combined to form a key. A buffer is uniquely identified by its key value. By device number we mean here a logical device number that specifies the concerning file system and block number is the logical block number with respect to the file system.

The buffer cache module (Buffer manager) in UNIX manages the buffer pool as two overlapping, doubly linked circular lists of buffers. One doubly linked circular list of buffers (known as free list) is used to keep all the available buffers. The other doubly linked, circular list is used as a separate queue, hashed as a function of the device and block number. Usually a hashing function that distributes the buffers uniformly across the set of queues (called hash queues) is used. When the buffer manager searches for a buffer with the appropriate device-block number combination it hashes the key value (device-block no) to find the appropriate queue. It then searches the particular queue instead of searching the entire buffer pool. If the buffer is not found in the designated queue, the corresponding disk block is not in the buffer pool. In that case, a buffer should be obtained from the free list.

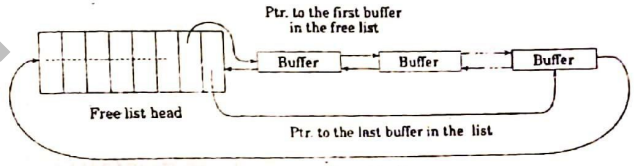


Fig. 2.17 Free list of buffers

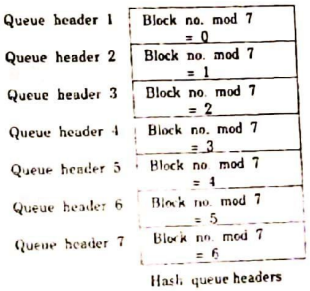


Fig. 2.18 Empty hash queue at the initial stage

Chapter 8: File Organization and Indexing

Rev. Jan 20,2014

Chapter 8: File organization and indexing

- Overview of Physical Storage Media
- Storage Access
- File Organization
- Organization of Records in Files
- Data-Dictionary Storage

Classification of Physical Storage Media

Several types of data storage exist in most computer system.These storage are classified by

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage:**
    - Contents persist even when power is switched off.

Physical Storage Media

Among the media typically available are these:

- **Cache** – fastest and most costly form of storage; volatile; managed by the computer system hardware
  - (Note: "Cache" is pronounced as "cash")
- **Main memory**:
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
  - **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.

Physical Storage Media (Cont.)

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles.
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower

Physical Storage Media (Cont.)

- Magnetic-disk
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for access, and written back for storage
  - It is **direct-access** – possible to read data on disk in any order, unlike magnetic tape
  - Survives power failures and system crashes
    - disk failure can destroy data: is rare but does happen

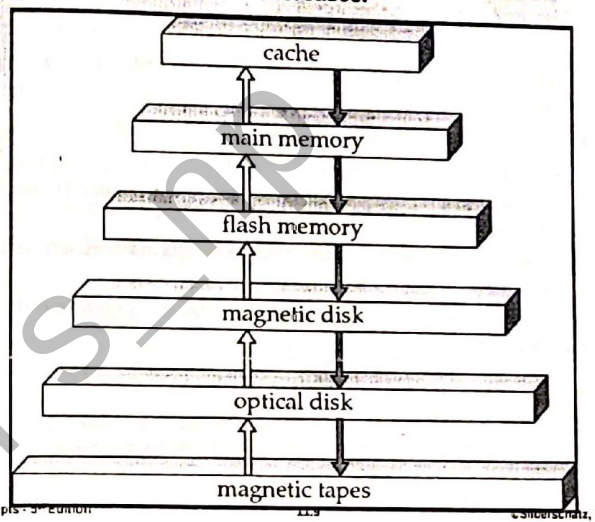
Physical Storage Media (Cont.)

- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk

Such disks are useful for archival storage of data as well as distribution of data.

Storage Hierarchy

Storage medai can be organized in hierarchy according to their speed and their cost.The higher level are expensive,but are fast.As we move down the hierarchy,the cost per bit decrease where as the access time increases.



Physical Storage Media (Cont.)

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **It is sequential-access** – much slower than disk
  - very high capacity (40 to 300 GB tapes available)
  - tape can be removed from drive ⇒ storage costs much cheaper than disk, but drives are expensive

Storage Hierarchy (Cont.)

- **primary storage:** Fastest media but volatile (cache, main memory).
- **secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
  - also called **on-line storage**
  - E.g. flash memory, magnetic disks
- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage**
  - E.g. magnetic tape, optical storage

Optimization of Disk Block Access (Cont.)

- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
  - E.g. Store related information on the same or nearby blocks/cylinders.
    - File systems attempt to allocate contiguous chunks of blocks (e.g. 8 or 16 blocks) to a file
  - Files may get **fragmented** over time
    - E.g. if data is inserted to/deleted from the file
    - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
  - Some systems have utilities to defragment the file system, in order to speed up file access

Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**. Blocks are units of both storage allocation and data transfer.
- Database system seeks to minimize the number of block transfers between the disk and memory. We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.

Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
- Buffer manager does the following:
  - If the block is already in the buffer, return the address of the block in main memory
  1. If the block is not in the buffer
     1. Allocate space in the buffer for the block
        1. Replacing (throwing out) some other block, if required, to make space for the new block.
        2. Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
     2. Read the block from the disk to the buffer, and return the address of the block in main memory to requester.

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records*. A record is a sequence of fields.
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

This case is easiest to implement; will consider variable length records later.

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where n is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries
- Deletion of record *i*: alternatives:
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - do not move records, but link all free records on a *free list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Free Lists

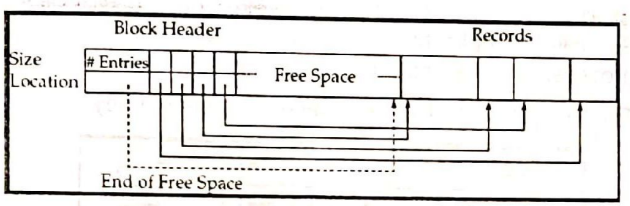
- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they "point" to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)

header			
record 0	A-102	Perryridge	400
record 1			
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4			
record 5	A-201	Perryridge	900
record 6			
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
  - Record types that allow repeating fields (used in some older data models).

Variable-Length Records: Slotted Page Structure



- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

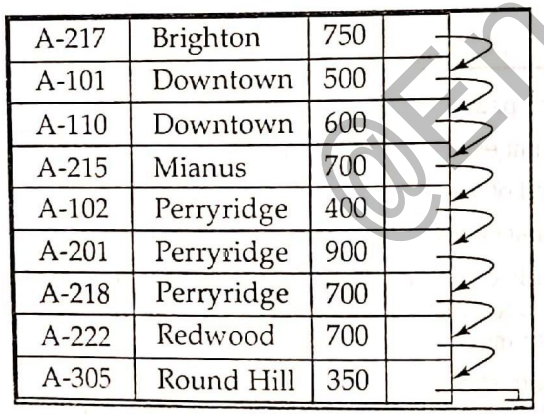
Organization of Records in Files

Several of the possible ways of organizing records in files are:

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

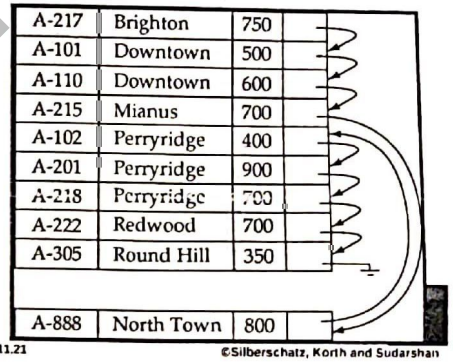
Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Date / /
Page No. 56

÷ Starvation happens if same transaction is always chosen as victim.
must ensure that transaction can be picked as a victim only a small no. of times.

CHAPTER 9. CRASH RECOVERY

→ DBMS is highly complex system with hundreds of transactions being executed every second.
→ availability of DBMS depends on its complex architecture and underlying h/w or system s/w.
→ if it fails or crashes amid transactions being executed, it is expected that the system would follow some sort of algo. or techniques to recover from crashes or failures.
→ There may be several causes of failure, including power failure, s/w bugs etc.

Types of failure

i) Transaction Failure
→ logical errors — transaction can't complete due to some internal condition.

→ System errors :- The database system must terminate an active transaction due to an error condition (like deadlock).

2) System Crash
→ a power failure or other h/w, s/w failure cause the system to crash.
→ fail stop assumption :-
- non volatile storage contents are assumed not to be corrupted by system crash.
- DB system have numerous integrity checks to prevent corruption of dist data.

3) Disk Failure
- a head crash or similar dist failure. destroys all part of disk storage.
- destruction is assumed to be detectable, disk drives use check sums to detect failure.
- Multiple copies or archival tapes are solution.

Recovery and atomicity

→ modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state.
→ Consider transaction T_i that transfer Rs 100 from account A to B here goal is either to perform all database modification made by T_i or none at all.
→ If T_i performed multiple database modification several output operations may be required for T_i and a failure may occur after one of these modification have been made but before all of them are made.
→ To ensure atomicity despite failures, we first o/p information describing the modification to stable storage without modifying database itself
→ two approaches
i) log based recovery
ii) shadow paging

here, assume that transaction run serially.

Log based Recovery

→ The log is a sequence of log records and maintains a record of update activities on the database, which is kept in a stable storage.

→ It works as follows:

→ the log file is kept on stable storage media.

→ when a transaction enters the system and starts execution, it write a log about it.
< T_i start >

→ when the transaction modifies an item x, it writes log as follows:
< T_i, x, V_1, V_2 >
i.e. T_i has changed value of x from V_1 to V_2

→ when the transaction finishes.

→ Two approaches using log
i) Deffered database modification
ii) Immediate " "

Deffered database Modification

→ It ensures transaction atomicity by recording all db modification in the log but deferring execution of all write operations of a transaction until the transaction partially commits

→ when transaction partially commits, the information on the log associated with transaction is used in executing deferred values.

→ If the system crashes before transaction completes its execution, then the information on the log simply ignored.

→ Transaction starts by writing < T_i, start > record to log.

→ A write (X) operation results in a log record < T_i, X, V > being written where V is new value of X. Here, old value is not needed.

→ Deffered update ~~statement~~ steps
i) 'write' operation is not performed on X at this time but is deffered.
ii) when T_i partially commits < T_i, commit > is written to log.
iii) finally, by records are read and used to actually execute previously 'deffered writes'.

→ REDO only scheme:-
↳ during recovery after crash, a transaction needs to be ~~the~~ redone if and only if both < T_i, start > and < T_i commit > are there in log.
↳ redoing a transaction T_i (i.e redo(T_i)) sets the value.

Date / /
Page No.

crashes can occur while
- the transaction is executing the original updates or while recovery action is being taken.

e.g:
transaction T_0 and T_1 such that T_0 executes before T_1.
Let A = 1000, B = 1000, C = 500

T_0 :	T_1 :
read (A)	read (C)
A = A - 50	C = C - 100
write (A)	write (C)
read (B)	
B = B + 50	
write (B)	

log
$< T_0$, start$>$
$< T_0$, A, 950$>$
$< T_0$, B, 1050$>$
$< T_0$, commit$>$

$< T_1$, start$>$
$< T_1$, C, 400$>$
$< T_1$, commit$>$

Database
A = 950 } // write operations
B = 1050 } executed

C = 400

Date / /
Page No. 59

Below is the log as it appears at three instances of time.

(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, 1050$>$	$< T_0$, B, 1050$>$	$< T_0$, B, 1050$>$
	$< T_0$, commit$>$	$< T_0$, commit$>$
	$< T_1$, start$>$	$< T_1$, start$>$
	$< T_1$, C, 400$>$	$< T_1$, C, 400$>$
		$< T_1$, commit$>$

If log on stable storage at time of crash is as in case
(a) → No redo action need to be taken
(b) → redo(T_0) must be performed
(c) → {redo(T_0), redo(T_1)} must be performed

Immediate Database Modification

allows database update of an uncommitted transaction to be made as the writes are issued

since, undoing is needed, update logs must have new and old values

update log record must be written before database

item is written.

→ output of updated blocks can take place at any time before or after transaction commit.

→ order in which blocks are output can be different from order in which they are written in buffer.

→ Example as before.

then,

log	Database
<T0, start>	
<T0, A, 1000, 950>	
<T0, B, 1000, 1050>	
	A = 950
	B = 1050
<T0, commit>	
<T0, start>	
<T1, C, 500, 400>	
	C = 400
<T1, commit>	

→ here recovery procedure has two operations

i) Undo (Ti) :- restores value of all data items updated by Ti to old values, going backward from last log record for Ti.

ii) Redo (Ti) :- Sets the value of all data item updated by Ti to new values, going forward from first log record for Ti.

→ Both operations must be idempotent.

undo (Ti) = ~~undo(Ti)~~ undo(undo (Ti)) = undo (undo(Ti))

i.e. even if the operation is executed multiple times the effect is the same as if it is executed once.

← it is needed since operations may get re-executed during recovery.

→ while recovering after failure, Transaction Ti needs to be redone if the log contains both the record <Ti, start> and <Ti, commit>

→ undo operations are performed first then only redo.

→ Before, we show log as it appears at three instance of time.

(a)	(b)	(c)
<T0, start>	<T0, start>	<T0, start>
<T0, A, 1000, 950>	<T0, A, 1000, 950>	— —
<T0, B, 1000, 1050>	<T0, B, 1000, 1050>	— —
	<T0, commit>	— —
	<T1, start>	— —
	<T1, C, 500, 400>	— —
		<T1, Commit>

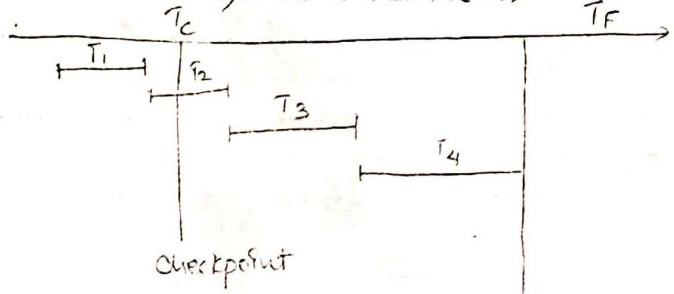
– recovery actions in each case of above are

(a) :- Undo(T_0)
(b) :- undo(T_1) then redo(T_0)
(c) :- redo(T_0) and redo(T_1)

Checkpoints

→ when more than one transactions are being executed in Parallel, the logs are interleaved.
→ at the time of recovery it would become hard for recovery algo. to backtrack all logs and then start recovering.
→ To ease this situation, most modern DBMs use concept of checkpoints
→ problems in recovery procedure as discussed earlier
i) Searching entire log is time consuming.
ii) might necessarily redo transactions which have already output their updates to database.
→ Streamline recovery procedure by periodically performing checkpointing
i) output all log records currently residing in main memory to stable storage.
ii) output all modified buffer blocks to disk.
iii) write a log record < checkpoint > on stable storage.

→ During checkpointing, other transactions cannot be processed.
→ During recovery we need to consider only the most recent transaction T_i that started before checkpoint and transactions that started after checkpoint.
→ Scan backward from end of log to find the most recent <'checkpoint'> record.
→ Continue scanning backward till a record $< T_i$, start > is found.
→ Need only consider the part of log following above 'start' record.
– earlier part of log can be ignored during recovery and can be erased whenever desired.
→ for all transactions (starting from T_i or later) with no $< T_i$, commit > execute undo(T_i).
→ Scanning forward in the log, for all transactions starting from T_i or later with $< T_i$, commit >, execute redo(T_i).



Here,
- Ignore T_1
- Undo T_4
- Redo T_2, T_3

Shadow Paging

→ Is an alternative to log based recovery.
→ Is useful if transaction execute serially.
→ Idea is maintain two page tables during the lifetime of transaction, on is current page table, another is shadow page table.
→ Store shadow page table in non-volatile storage, such that state of the db prior to transaction execution
- Here, shadow page table is never modified during execution.

→ To start with, both page tables are identical.
- only current page table is used for data item access during execution of transaction

→ Whenever any page is about to be written for first time
- a copy of this page is made onto an unused page.
- the current page table is then made to point to copy.
- the update is performed on the copy.

→ To commit transaction.
i) flush all modified pages in main memory to disk.
ii) output current pagetable to disk.
iii) Make current pagetable the new shadow page table as follows.
a) keep a pointer to shadow page table at a fixed known location on disk.
b) Simply update pointer to point current page table on disk.

→ Once pointer to shadow page table has been written, transaction is committed
→ No recovery is need after the crash, new transaction can start right away using shadow page table.
→ Pages not pointed to from current / shadow page table should be free.

Shadow Paging

→ Advantages of shadow paging over log based
→ no overhead of writing log records
→ recovery is trivial

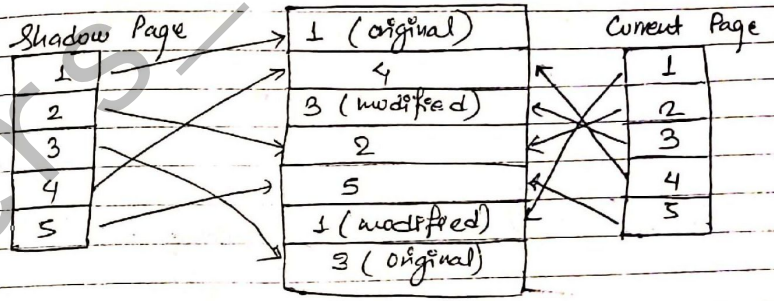
→ Disadvantages:
→ Copying the entire page is very expensive.
→ Commit overhead is high
→ Data gets fragmented (related pages get

→ To commit transaction.
i) flush all modified pages in main memory to disk.
ii) output current pagetable to disk.
iii) Make current pagetable the new shadow page table as follows.
a) keep a pointer to shadow page table at a fixed known location on disk.
b) simply update pointer to point current page table on disk.
→ once pointer to shadow page table has been written, transaction is committed.
→ No recovery is need after the crash, new transaction can start right away using shadow page table.
→ Pages not pointed to from current / shadow page table should be free

Shadow Paging

→ Advantages of shadow paging over log based
→ no overhead of writing log records.
→ recovery is trivial

→ Disadvantages:
→ Copying the entire page is very expensive.
→ Commit overhead is high.
→ Data gets fragmented (related pages get separated on disk).
→ after every transaction, old version of modified data need to be garbage collected.
→ hard to extend algo to allow transaction run concurrently.



Data Backup and Recovery

→ Is to protect data.
→ backup files can protect against accidental loss of data, database corruption and other failures.
→ with backup and recovery plan, we can easily recover our important data from any type of disasters.
→ To create backup plan, deal with following
i) how important is data on system?
ii) what type of information does data contain?

iii) How often does data change?
iv) How quickly data need to recover?
v) Have the equipment to perform backup?
vi) Who will be responsible for backup and recovery plan?
vii) What is best time to schedule backup?
viii) Is there need of storing backup off site?

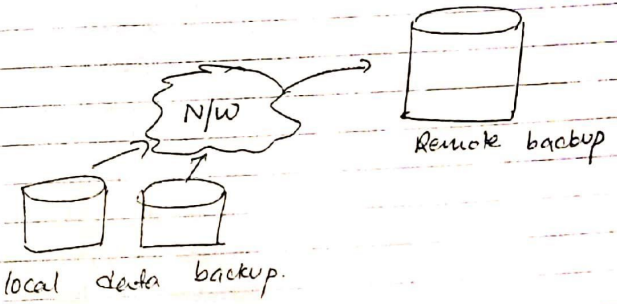
→ Backup plans depends on factors like
a) Capacity
b) Reliability
c) Extensibility
d) Speed
e) cost

→ common backup solution are
a) Tape drive
b) Disk drive
c) Magnetic optical drive
d) removable disk etc.

Remote Backup or Online Backup

→ Here, files, folders or entire contents of hard drive are regularly backed up on a remote server with n/w connection.

→ Here, risk of catastrophic data loss as a result of fire, theft, file corruption or other disaster is eliminated.
→ encryption and password protection system helps to ensure privacy and security.
→ For large organization and for practically valuable data, online backup strategy is wise investment.
→ If is backup performed on data even though it is actively accessible to users and may currently be in state of being updated.
→ Cloud computing, Remote login method are for remote backup and recovery.



25th Sept.

Date / /
Page No. 42

CHAPTER 10

Transaction Processing and Concurrency Control

Transaction System

→ group of operations that form a single logical unit of work is called transaction.
→ a transaction is a logical unit of work that must be either entirely completed or aborted.
→ a transaction is a unit of program execution that access and possibly updates various data items.
→ a transaction is the DBMS abstract view of a user program that consists of sequence of reads and writes.
→ Transaction access data using two operations
  i) read (x)
  ii) write (x)
→ Transaction processing is information processing that is devided into individual, indivisible operations called transactions.
→ each transaction must succeed or fail as a complete unit but can't remain in inter-mediate state.
→ Transaction processing maintains a system in as consistent state.

Date / /
Page No

e.g.:

```
T1 : read (A)
     A = A - 100
     write (A)
     read (B)
     B = B + 100
     write (B)
```

Properties of Transaction (ACID properties)

1) **Atomicity**: either all operations of transaction are reflected properly in database or none are.

2) **Consistency**: execution of a transaction in isolation preserves the consistency of db.

3) **Isolation**: even though multiple transactions may execute concurrently, the system guarantees that for every pair of transactions T_i & T_j for T_i either T_j finished execution before T_i started or T_j started execution after T_i finished.

4) **Durability**: after successful completion of transaction, the changes in db persist, even if there are system failures.

Date / /
Page No 43

Transaction State:

i) Active state:
- It is the initial state.
- transaction stays in this state while it is executing.

ii) Partially Committed:
- transaction is in P.C state after the final statement has been executed.

iii) Failed:
- transaction is in failed state after normal execution can no longer proceed

iv) Aborted
- transaction is in aborted state after it has been rolled back & the database has been restored to its state prior to the start of transaction.

v) Committed:
- transaction is in committed state after successful completion.

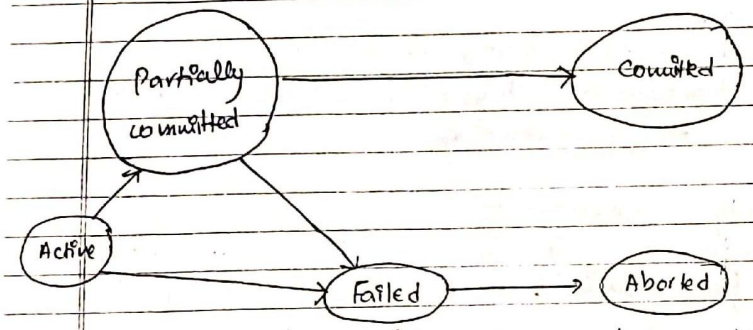


fig: state diagram of transaction.

→ a transaction has committed only if it enter committed state

↳ a transaction has aborted only if it enter aborted state.

↳ a transaction has terminated if either committed or aborted.

→ after transaction being in aborted state, system has two options either it can restart transaction.

→ Transaction processing system usually allow multiple transactions to run concurrently.

→ This cause several complications with consistency of data.

↳ though, transactions can be run serially to ensure consistency, concurrent execution of transaction has following advantages:

i) Improved throughput and resource utilization.

ii) Reduced waiting time.

→ database system control the interaction among the concurrent transactions to prevent from destroying consistency of db through variety of mechanisms called concurrency control scheme

→ when transactions are executing concurrently in an ~~unbalanced~~ interleaved fashion, then the ~~act~~ order of execution of operations from various transactions is known as schedule.

↳ A schedule is specific the chronological order in which the operations of concurrent transactions are executed

↳ e.g:
Let transaction T1 transfer Rs 100 from account A to B.
Transaction T2 transfer 20% of balance from A to B.
let A = 1000, B = 1000

Now,

T1 :- read(A)
A = A - 1000
write(A)
read(B)
B = B + 100
write(B)

T2 : read (A)
temp = A * 0.2
A = A - temp
write (A)
read (B)
B = B + temp
write (B)

Serial Schedule.

i)

T1	T2
read (A)	
A = A - 100	
write (A)	
read (B)	
B = B + 100	
write (B)	
	read (A)
	temp = A * 0.2
	A = A - temp
	write (A)
	read (B)
	B = B + temp
	write (B)

ii)

T1	T2
	read (A)
	temp = A * 0.2
	A = A - temp
	write (A)
	read (B)
	B = B + temp
	write (B)
read (A)	
A = A - 100	
write (A)	
read (B)	
B = B + 100	
write (B)	

Non - Serial Schedule

i)

T1	T2
read (A)	
A = A - 100	
write (A)	
	read (A)
	temp = A * 0.2
	A = A - temp
	write (A)
read (B)	
B = B + 100	
write (B)	
	read (B)
	B = B + temp
	write (B)

ii)

T1	T2
read (A)	
A = A - 100	
	read (A) → 1000
	temp = A * 0.2 → 200
	A = A - temp → 800
	write (A) → 800
	read (B) → 1000

write (A) → 800
read (B) → 1000
B = B + 100 → 1100
write (B) → 1100

B = B + temp

Date / /
Page No.

here, both ① & ⑪ are non serial schedule.
i) results correct state
but
ii) doesn't result correct state.

Serializability.

→ Basic assumption in each transaction preserves data consistency.
→ Thus serial execution of a set of transaction preserves database consistency.
→ Main objective of serializability is to search non serial schedules that allow transaction to execute concurrently without interfering one another transaction and produce the result of db state that could be produced by serial execution.
→ We can conclude that a non serial schedule is correct if it produce same result as serial execution.
→ a schedule is serializable if it is equivalent to a serial schedule.

Rules:

→ Ordering of read/write is important
→ if two transactions only read data item they do not conflict and order is not important.
→ if two transactions either read or

write completely separate data items, they do not conflict and order is not important.

→ if one transaction write a data item and another read or write same data item, order of execution is important.

Schedule 1			Schedule 2		
T_1	T_2	T_3	T_1	T_2	T_3
R(a)			R(a)		
	R(b)		W(a)		
		R(c)		R(b)	
W(a)				W(b)	
	W(b)				R(c)
		W(c)			W(c)

→ Here, actions of transactions in schedule 1 are not executed as same as in schedule 2 but at the end schedule 1 gives same result as that of schedule 2.

→ Thus it is considered as serializable.

Conflict serializability.

→ Instructions I_i and I_j of transaction T_i and T_j respectively conflict if and only if there exists same item Q accessed by both I_i & I_j and at least one of these instruction wrote Q.

→ If a schedule S can be transformed into a schedule 's' by a series of swaps of non conflicting instructions, we say that S and s' are conflict equivalent.

→ we say that a schedule S is conflict serializable if it is conflict equiv. to a serial schedule.

→

Instruction I_i	Instruction I_j	Result
Read (Q)	Read (Q)	No conflict
Read (Q)	Write (Q)	Conflict
Write (Q)	Read (Q)	"
Write (Q)	Write (Q)	"

Eg.1) let we have schedule A as.

T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

and schedule B as,

T1	T2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

→ Here, schedule A can be transformed into serial schedule B by series of swaps of non conflict instructions.

→ Hence, schedule A is conflict serializable.

eg.2) Let we have,

Schedule X		Schedule Y.	
T1	T2	T1	T2
read (Q)		read (Q)	
	write (Q)	write (Q)	
write (Q)			write (Q)

Here, we can't transform schedule X into schedule Y by swapping non conflict instruction

∴ Schedule X is non conflict serializable.

View serializability:

→ two schedules are said to be view equivalent if following 3 condition holds

1) For each data item Q, if transaction T_i reads initial value of Q in schedule S, also read initial value of Q

2) For each Q, if T_i executes read (Q) in S and if value was produced by write (Q) of T_i then read (Q) of T_i must in S' also read value of Q produced by same write (Q) of T_i.

3) For each Q, the transaction that performs the final write (Q) operation in S; must perform final write (Q) operation in S'.

→ a schedule is view serializable if it is view equiv. to a serial schedule.

eg: Let,

Schedule A			Schedule B		
T_1	T_2	T_3	T_1	T_2	T_3
read(Q)				read(Q)	
	write(Q)			write(Q)	
write(Q)					write(Q)

∴ here, schedule A is view serializable because it is view equivalent to serial schedule B

→ here write (Q) of T2 and T3 are called blind writes because it is performed without having performed read (Q).

→ every conflict serializable schedule is also view serializable but not vice versa.

Testing for serializability

- construct a directed graph called precedence graph from S.
- Precedence graph consists of a pair $G = (V, t)$ where,
  - V is set of vertices. it consists of all transactions in schedule
  - E is set of edges
    - set of edges consists of all edges $T_i \to T_j$ for which one of 3 conditions hold.
      1.
      2.
      3.

→ if an edge $T_i \rightarrow T_j$ exists in precedence graph then in any serial schedule S' equivalent to S, T_i must appear before T_j.

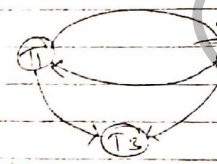
→ if the precedence graph has a cycle, then the schedule is not conflict serializable.

→ if the precedence graph has not any cycle then schedule is conflict serializable.

eg.1)

T_1	T_2	T_3
read (A)		
	write (A)	
write (A)		
		write (A)

Now,

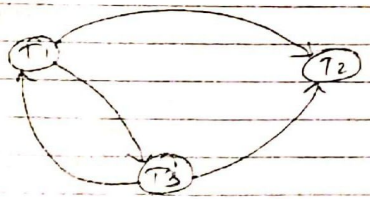


Equivalent precedence graph

→ Here graph contains a cycle, so above schedule is non conflict serializable.

eg.2)

T_1	T_2	T_3
write (A)		
	read (A)	
		write (B)
write (B)		
		write (B)
	write (A)	
		read (B)
	read (B)	



equivalent P.G.

- Here also above schedule is non conflict serializable.

Test for following.

(1) $r_1(x), w_2(x), r_1(x)$.

(2) $r_1(A), r_3(A), w_1(A), r_2(A), w_3(A)$

(3) $r_3(A), r_2(A), w_3(A), w_1(A), w_1(A)$

(4) $r_1(A), w_3(A), w_3(A), w_1(A), r_2(A)$.

Concurrency ~~control~~ control.

→ ensures that database transactions are performed concurrently without violating data integrity of database.

→ different concurrency control schemes can be used to ensure the isolation property when multiple transactions are executed in parallel.

→ DBMS must guarantee that only serializable schedule are generated and that no effect of committed transaction is lost and no effect of aborted transaction remains in db.

→ Different methods like locking method, timestamp methods etc are used

10th November

Page No. 49

Lock based protocol

↳ a lock is a mechanism to control concurrent access to data item.

↳ Data items can be locked in two modes.

a) Exclusive mode (X)

→ Data item can be both read as well as written.

→ X-lock is requested using lock-x instruction

b) Shared mode (S)

→ Data item can only be read.

→ S-lock is requested using lock-S instruction.

→ lock requests are made to concurrency control manager.

→ Transaction can only be proceed once request is granted.

→ A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on item by other transactions.

	S	X
S	True	False
X	False	False

fig: lock compatibility matrix.

- here, shared mode is compatible only with shared mode.
- if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.
→ If a lock can't be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released the lock is then granted.
- A transaction can unlock a data item Q by instruction unlock (Q).
- transaction must hold a lock on data item as long as it access item.

e.g:

```
T :  lock_S (A)
     read (A)
     unlock (A)
     lock-S (B)
     read (B)
     unlock (B)
     display (A+B)
```

- A locking protocol is set of rules followed by all transactions while requesting and releasing locks.
- locking protocol restricts the set of possible schedules.
- locking protocol must ensure serializability.

e.g:

Let, value of account A and B are 100 and 100 respectively.

T1 transaction transfer 20 from A to B.

T2 transaction display sum of A and B

Now, here,

T1 :	T2 :-
lock – X (A)	lock – S(A)
read (A)	read (A)
A = A – 20	unlock (A)
write (A)	lock – S (B)
unlock (A)	read (B)
lock – X (B)	unlock (B)
read (B)	display (A+B)
B = B + 20	
write (B)	
unlock – X (B)	

→ If T1 and T2 are executed serially, we get correct result.

→ But concurrent execution of T1 and T2 may result incorrect value.

like

T1	T2	Concurrency Control Manager
lock – X (A)		grant – X (A, T1)
read (A)		
A = A – 20		
write (A) //update		
unlock (A)		

T1	T2	Concurrency Control Manager
	lock - S(A)	
		grant - S(A, T2)
	read (A)	
	unlock (A)	
	lock - S(B)	
		grant - S (B, T2)
	read (B)	
	unlock (B)	
	display (A+B)	
lock - X(B)		
		grant - X (B, T1)
read (B)		
B = B + 20		
write (B)		
unlock (B)		

Fig: Schedule X.

→ Assume that unlocking is delayed at the end of transaction then ~~T1 becomes~~ T2 becomes

T1 becomes

```
T3 : lock - X(A)
     read (A)
     A = A - 20
     write (A)
     lock - X(B)
     read (B)
     B = B + 20
```

```
T4 : lock - S(A)
     read (A)
     lock - S(B)
     read (B)
     display (A+B)
     unlock (A)
     unlock (B)
```



```
write (B)
unlock (A)
unlock (B)
```

→ with T3 and T4, we can't get schedule x as above and for any other schedule, it produce correct value Plz:

T1	T2	T3	T4
lock - x(A)			~~read (B)~~
read (A)			~~display (A+B)~~
A = A - 20			~~unlock (A)~~
write (A)			~~unlock (B)~~
	lock - S(A)	lock - X(B)	
	read (A)	read (B)	
unlock (B)		B = B + 20	
	lock - S(B)	write (B)	
	read (B)	unlock (A)	
	display (A+B)		
	unlock (A)		
	unlock (B)		

Date / /
Page No.

Pitfalls of lock based protocol

- consider partial schedule as

T3	T4
lock - X(B)	
read (B)	
B = B - 50	
write (B)	
	lock - S(A)
	read (A)
	lock - S(B)
lock - X(A)	

fig: Schedule

- here, neither T3 nor T4 can make progress executing lock - S(B) cause T4 to wait for T3 to release its lock on B.
  while executing lock - X(A) cause T3 to wait for T4 to release its lock on A.
- Such situation is called deadlock.
- To handle deadlock one of transactions T3 or T4 must be rolled back and its lock must be released.
- If we do not use locking, we may get inconsistent state
- Similarly, if we do not unlock data item before requesting lock on another item,

Date / /
Page No. 52

deadlock occurs.

- However, deadlocks are necessary evil, more preferable than inconsistent state.
- The potential for deadlock exists in most locking protocol.
- Starvation is also possible,
  → if a transaction may be waiting for an X-lock on item while, a sequence of other transactions request S-lock and are ~~guarantd~~ granted on same item.
  → Same transaction is repeatedly rolled back due to deadlock.

Two phase locking protocol

→ ensures serializability.
→ here, transaction issue lock and unlock requests in two phases

i) Growing phase :-
   → transaction may obtain locks.
   → transaction may not release locks.

ii) Shrinking phase:
   → transaction may release locks
   → transaction may not obtain locks.

→ Initially, transaction is in growing phase
→ after transaction release a lock, it enters shrinking phase.

Date / /
Page No.

e.g. T3 and T4 are two phase but not T1 and T2.

→ two phase locking does not ensure freedom from deadlocks.

→ cascading roll back is possible under two phase locking.

the phenomenon in which a single transaction failure leads to series of transaction rollbacks is called cascading roll back.

e.g.:

T1	T2	T3
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)

↳ here, T2 is dependent on T1, and T3 is dependent on T2.

↳ if T1 failed, T1 must be rolled back. Similarly, as T2 is dependent on T1, T2 also be rolled back again as T3 is dependent on T2. T3 also must be rolled back

14th November

Date / /
Page No. 58

↳

T_a	T_b	T_c
lock – X(A)		
read (A)		
lock – S(B)		
read(B)		
write (A)		
unlock (A)		
	lock – X(A)	
	read (A)	
	write (A)	
	unlock (A)	
		lock-S(A)
		read (A)

fig: partial schedule X under two phase locking

- in above schedule X,
- if T_a fails after read(A) of T_c then there must be cascading rollback of T_b & T_c

1) Strict two phase locking protocol

→ here, transaction must hold all its exclusive mode locks till it commits

→ this prevents any other transaction from reading data.

→ No cascading rollback

2) Rigorous two phase locking
→ Here, all locks (S and X) are held till commits.
→ No cascading roll back.
→ Here, transaction can be serialized in order in which they commit.

Deadlock Handling

↳ System is deadlocked if there is set of transactions such that every transaction in the set is waiting for another transaction in the set.

↳ Let,

T_1	T_2
write (X)	write (Y)
write (Y)	write (X)

↳ Here, Schedule with deadlock

T_1	T_2
lock-X(X)	
write (X)	
	lock-X(Y)
	write (Y)
	write (X)
write (Y)	

↳ to deal with deadlock, we can use
1) Deadlock prevention protocol

Page No. 54

- ensure that system will never enter deadlock state.

2) Deadlock detection and recovery scheme
- try to recover system once it entered deadlock state.

→ Both methods may result in transaction rollback.
→ prevention is used if probability of system entering deadlock state is relatively high.
→ otherwise, detection and recovery are more efficient.

Deadlock prevention

↳ deadlock prevention protocol ensure that the system will never enter into deadlock state.

↳ some prevention strategies
i) require that each transaction locks all data item before it begins execution.
ii) Impose partial ordering of all data item.
- require that a transaction can lock data items only in order specified by partial order.
iii) Timeout based schems.
- a transaction waits for a lock only for specified amount of time

29

Date / /
Page No.

- after the wait time is out, transaction is roll back.
→ Simple to implement but starvation is possible.
→ following schemes use transaction timestamps for deadlock prevention.

i) Wait - die scheme (Non preemptive)
→ older transaction may wait for younger one to release data item.
→ younger transactions never wait for older ones; they are rolled back instead.
→ a transaction may die several times before acquiring needed data item.

ii) ~~Deadlo~~ Wound - wait scheme (pre emptive)
→ older transaction wounds (force rollback) younger transaction instead of waiting for it.
→ younger transaction may wait for older ones.
→ may be fewer rollbacks than wait die scheme.

↳ in both schemes, rollback transaction is restarted with its original timestamp.
↳ older transaction thus have precedence over newer ones in these schemes and starvation is avoided

Date / /
Page No. 55

Deadlock Detection and Recovery

- is used if no protocol is used to ensure deadlock freedom.
- some algo are used to check whether deadlock occurs, if deadlock occurs, then must be recovered.

Deadlock detection:

- deadlock can be described as wait for graph which consists of pair $G_1 = (V, E)$
→ V is set of vertices (transaction)
→ E is set of edges, each edge is ordered pair $T_p \rightarrow T_j$

- when transaction T_p request a data item held by T_j then $T_p \rightarrow T_j$ is inserted in wait for graph
- this edge is removed only when T_j no longer holding data item needed by T_p.
- the system is in deadlock state if and only if wait for graph has a cycle.
- the system invokes deadlock detection algorithm periodically to look for cycle.

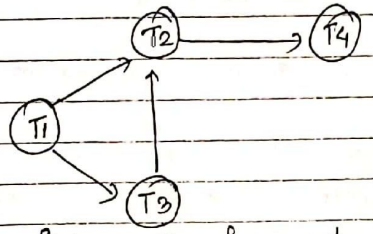


fig: wait for graph without cycle.

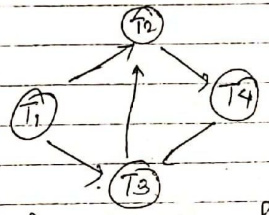


fig: wait for graph with cycle.

Deadlock Recovery

When deadlock is detected

→ Some transactions will have to roll back to break deadlock (i.e. victim selection)

→ Rollback :- determine how far to rollback transaction

a) total rollback
   - abort transaction and then restart it.

b) Partial rollback
   - rollback transaction only as far as necessary to break deadlock.
   - is more effective.

- Starvation happens if same transaction is always chosen as victim.
- must ensure that transaction can be picked as a victim only a small no. of times.

Chapter-11
Advanced Database Concepts
½ Question/(7-8 Marks)

Topics

- Object Oriented Model
- Object Relational Model
- Distributed Databases
- Concept of Data Warehouses

Object Oriented Model

- Need for Complex *Data Types*
- ***Traditional*** database ***applications*** in data processing had conceptually ***simple data types***
  - Relatively ***few data types***,
    - ***first normal form*** holds
- ***Complex data types*** have grown more important in ***recent years***
  - E.g. *Addresses* can be viewed as a
    - Single string, or
    - *Separate attributes* for each part, or
    - *Composite attributes* (which are not in first normal form)
  - E.g. it is often ***convenient*** to store ***multivalued attributes as-is***,
    - ***without*** creating a ***separate relation*** to store the values in 1FN
- ***Applications:***
  - computer-aided design, computer-aided software engineering
  - multimedia and image databases, and document/hypertext databases.

Object-Oriented *Data Model*

- *Loosely speaking*, an **object** *corresponds to*:
  - an ***entity*** in the ***E-R model***.
- The ***object-oriented paradigm*** is based on:
  - ***encapsulating code*** and ***data*** related to an object *into* ***single unit***.
  - Inheritance, Polymorphism
- The ***object-oriented data model*** is:
  - a ***logical data model*** (*like* the ***E-R model***).
- ***Adaptation*** of the ***object-oriented programming paradigm***:
  - (e.g., Smalltalk, C++)
  - to ***database systems***.

Object Structure

- An object has associated with it:
  - A set of **variables** that contain the ***data*** for the object. The value of each variable is itself an object.
  - A set of **messages** to which the object responds; each message may have zero, one, or more *parameters.*
  - A set of **methods**, each of which is a ***body of code*** to implement a message; a method returns a value as the *response* to the message
- The physical representation of data is visible only to the implementor of the object
- ***Messages*** and responses *provide* the only *external interface* to an object.
- The term message does *not necessarily* imply physical *message passing*.
  - ***Messages*** can be *implemented* as ***procedure invocations.***

Object-Relational Data Models

- ***Extend*** the *relational data model* by
  - including object orientation and constructs to deal with added data types.
- ***Allow*** attributes of tuples to have ***complex types***,
  - including ***non-atomic values*** such as ***nested relations***.
- ***Preserve*** relational *foundations*,
  - in particular the ***declarative access*** to data,
  - while *extending* ***modeling power***.
- ***Upward compatibility*** with existing relational languages.

Nested Relations

- ***Motivation***:
  - Permit ***non-atomic*** domains (atomic ≡ indivisible)
  - Example of non-atomic domain: *set of integers*, or *set of tuples*
  - Allows more ***intuitive*** modeling for applications with ***complex*** data

- *Intuitive definition*:
  - allow relations whenever we allow atomic (scalar) values:
    - ***relations within relations***
  - Retains *mathematical foundation* of relational model
  - Violates *first normal form.*

Example of a Nested Relation

- Example: library information system
- Each book has
  - title,
  - a set of authors,
  - Publisher, and
  - a set of keywords
- Non-1NF relation *books*

title	*author-set*	*publisher*	*keyword-set*
		(name, branch)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

Distributed Database

- A distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU.
- It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.
- Collections of data (e.g. in a database) can be distributed across multiple physical locations.
- A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks.
- The replication and distribution of databases improves database performance at end-user worksites.

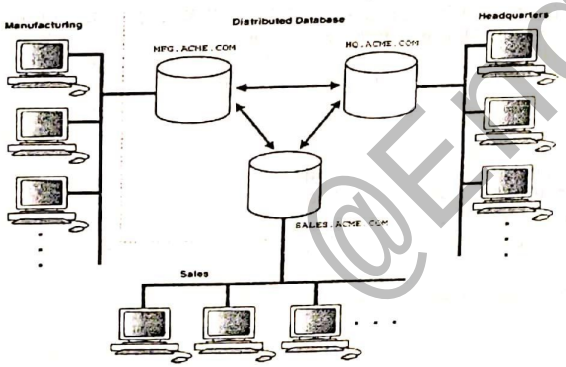
Features

- The features of the Heterogeneous Services include:
- *Distributed Transactions.* A transaction can span both Oracle and non-Oracle systems, while still guaranteeing, through Oracle's two phase commit mechanism, that changes are either all committed or all rolled back.
- *Transparent SQL access.* Integrate data from non-Oracle systems into the Oracle environment as if the data is stored in one single, local database. SQL statements issued by the application are transparently transformed into SQL statement understood by the non-Oracle system.
- *Procedural Access.* Procedural systems, like messaging and queuing systems, are accessed from an Oracle8 server using PL/SQL remote procedure calls.
- Fragmentation: Horizontal/Vertical

Distributed Database Architecture

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems
- Client/Server Database Architecture

Homogeneous Distributed Database



Advantages

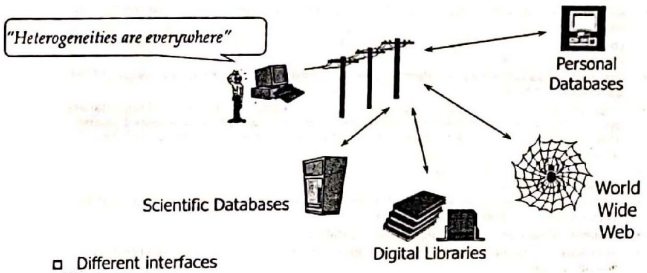
- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.
- Increase reliability and availability.
- Easier expansion.
- Reflects organizational structure — database fragments are located in the departments they relate to.
- Local autonomy or site autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).
- Reliable transactions - Due to replication of database.
- Hardware, Operating System, Network, Fragmentation, DBMS, Replication and Location Independence.
- Continuous operation.
- Distributed Query processing.
- Distributed Transaction management.

Disadvantages

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites)
- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- Database design more complex — besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.
- Additional software is required.
- Operating System should support distributed environment.
- Concurrency control: it is a major issue. It can be solved by locking and timestamping.

Data Warehouses

Problem: Heterogeneous Information Sources



- Different interfaces
- Different data representations
- Duplicate and inconsistent information

What is a Data Warehouse?

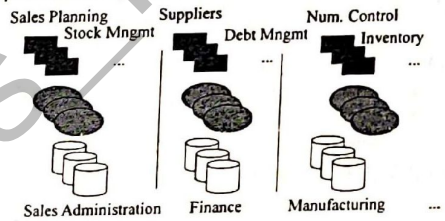
A Practitioners Viewpoint

"A data warehouse is simply a single, complete, and consistent store of data obtained from a variety of sources and made available to end users in a way they can understand and use it in a business context."

A Data Warehouse is a
subject-oriented,
integrated,
time-variant,
non-volatile
collection of data used in support of management decision making processes

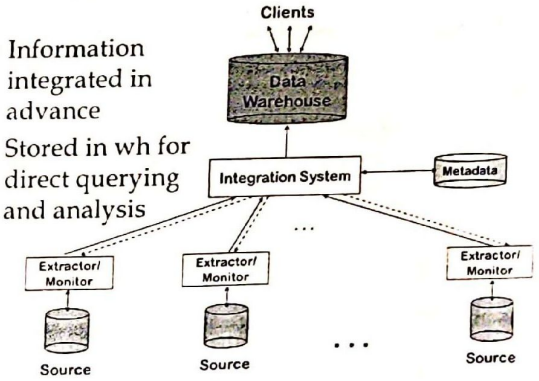
Problem: Data Management in Large Enterprises

- Vertical fragmentation of informational systems (vertical stove pipes)
- Result of application (user)-driven development of operational systems



The Warehousing Approach

- Information integrated in advance
- Stored in wh for direct querying and analysis



Advantages of Warehousing Approach

- High query performance
  - But not necessarily most current information
- Doesn't interfere with local processing at sources
  - Complex queries at warehouse
  - OLTP at information sources
- Information copied at warehouse
  - Can modify, annotate, summarize, restructure, etc.
  - Can store historical information
  - Security, no auditing
- *Has* caught on in industry

Distributed Database System

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

Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets orextranets, or on other company networks. The replication and distribution of databases improves database performance at end-user worksites.

To ensure that the distributive databases are up to date and current, there are two processes: replication and duplication. Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be very complex and time consuming depending on the size and number of the distributive databases. This process can also require a lot of time and computer resources. Duplication on the other hand is not as complicated. It basically identifies one database as a master and then duplicates that database. The duplication process is normally done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, changes to the master database only are allowed. This is to ensure that local data will not be overwritten. Both of the processes can keep the data current in all distributive locations.[2]

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

Architecture

A database user accesses the distributed database through:

Local applications

applications which do not require data from other sites.

Global applications

applications which do require data from other sites.

A distributed database does not share main memory or disks..

Features

The features of the Heterogeneous Services include:

- *Distributed Transactions.* A transaction can span both Oracle and non-Oracle systems, while still guaranteeing, through Oracle's two phase commit mechanism, that changes are either all committed or all rolled back.
- *Transparent SQL access.* Integrate data from non-Oracle systems into the Oracle environment as if the data is stored in one single, local database. SQL statements issued by the application are transparently transformed into SQL statement understood by the non-Oracle system.
- *Procedural Access.* Procedural systems, like messaging and queuing systems, are accessed from an Oracle8 server using PL/SQL remote procedure calls.

Data Dictionary translations. To make the non-Oracle system appear as another Oracle server, SQL statementscontaining references to Oracle's data dictionary tables are transformed into SQL statements containing references to a non-Oracle system's data dictionary tables.

Pass-through SQL. Optionally, application programmers can directly access a non-Oracle system from an Oracle application using the non-Oracle system's SQL dialect.

Accessing stored procedures. Stored procedures in SQL-based non-Oracle systems are accessed as if they were PL/SQL remote procedures.

National Language Support. Gateways supports multi-byte character sets, and translate character sets between a non-Oracle system and the Oracle8 server.

Global query optimization. Cardinality and indexes on tables at the non-Oracle system are taken into account by the Oracle8 query optimizer and decomposed to produce efficient SQL statements to be executed at the non-Oracle system.

Note: Not all features listed above are necessarily supported by your Heterogeneous Services agent or Oracle Gateway. Please see your Heterogeneous Services agent or Oracle Open Gateway documentation for the supported features.

Distributed Database Architecture

A distributed database system allows applications to access data from local and remote databases. In a homogenous distributed database system, each database is an Oracle Database. In a heterogeneous distributed database system, at least one of the databases is not an Oracle Database. Distributed databases use a client/server architecture to process information requests.

This section contains the following topics:

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems

- Client/Server Database Architecture

Homogenous Distributed Database Systems

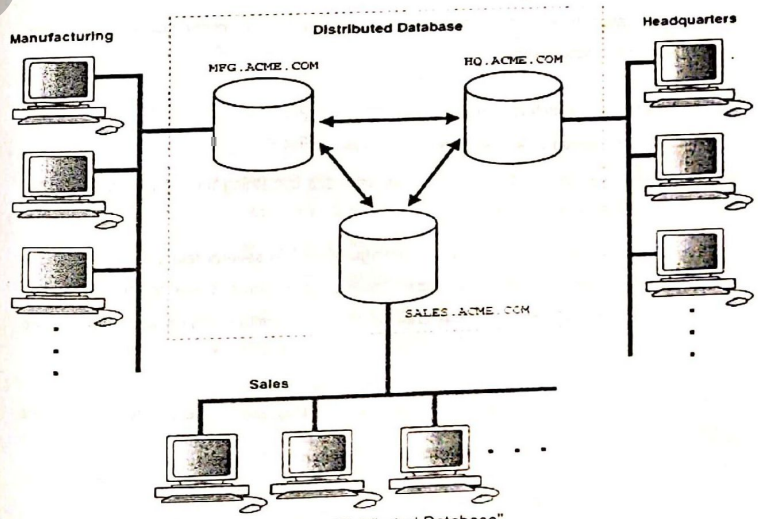
A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more machines. Figure 29-1 illustrates a distributed system that connects three databases: hq, mfg, and sales. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database mfg can retrieve joined data from the productstable on the local database and the dept table on the remote hq database.

For a client application, the location and platform of the databases are transparent. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database mfg but want to access data on database hq, creating a synonym on mfg for the remote dept table enables you to issue this query:

```
SELECT * FROM dept;
```

In this way, a distributed system gives the appearance of native data access. Users on mfg do not have to know that the data they access resides on remote databases.

Figure 29-1 Homogeneous Distributed Database



Description of "Figure 29-1 Homogeneous Distributed Database"

An Oracle Database distributed database system can incorporate Oracle Databases of different versions. All supported releases of Oracle Database can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle Database.

Important considerations

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

- The distribution is transparent — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access among other things.
- Transactions are transparent — each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub-transactions, each sub-transaction affecting one database system.

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

A) Replication

B) Fragmentation

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

- Data is more available in this scheme.
- Parallelism is increased when read request is served.
- Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.

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

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

Advantages

- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.
- Increase reliability and availability.
- Easier expansion.
- Reflects organizational structure — database fragments are located in the departments they relate to.
- Local autonomy or site autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).
- Reliable transactions - Due to replication of database.
- Hardware, Operating System, Network, Fragmentation, DBMS, Replication and Location Independence.
- Continuous operation.
- Distributed Query processing
- Distributed Transaction management.

Single site failure does not affect performance of system. All transactions follow A.C.I.D. property: A-atomicity, the transaction takes place as a whole or not at all; C-consistency, maps one consistent DB state to another; I-isolation, each transaction sees a consistent DB; D-durability, the results of a transaction must survive system failures. The Merge Replication Method is popularly used to consolidate the data between databases

Disadvantages

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.,
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- Database design more complex —> besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.
- Additional software is required.
- Operating System should support distributed environment.
- Concurrency control: it is a major issue. It can be solved by locking and timestamping.

Object-Oriented Database

Definition

An object-oriented database is a database that subscribes to a model with information represented by objects. Object-oriented databases are a niche offering in the relational database management system (RDBMS) field and are not as successful or well-known as mainstream database engines.

As the name implies, the main feature of object-oriented databases is allowing the definition of objects, which are different from normal database objects. Objects, in an object-oriented database, reference the ability to develop a product, then define and name it. The object can then be referenced, or called later, as a unit without having to go into its complexities. This is very similar to objects used in object-oriented programming.

A real-life parallel to objects is a car engine. It is composed of several parts: the main cylinder block, the exhaust system, intake manifold and so on. Each of these is a standalone component; but when machined and bolted into one object, they are now collectively referred to as an engine. Similarly, when programming one can define several components, such as a vertical line intersecting a perpendicular horizontal line while both lines have a graded measurement. This object can then be collectively labeled a graph. When utilizing the ability to plot components, there is no need to first define a graph; but rather the instance of the created graph can be called.

Examples of object-oriented database engines include db4o, Smalltalk and Cache.

Date: / /

2018 (Fall)

1a) Data independence is the capacity to change the schema at one level of a database system without having to change schema at the next higher level.

Logical data independence is the ability to change the logical schema without changing the external schema. eg addition or removal of new entities, attributes or relationships to the conceptual schema should be possible without having to change existing external schemas.

physical data independence is the ability to change physical schema without changing the logical schema. eg. a change to the internal schema, such as using different file organization or storage structure, storage devices, or indexing strategy, should be possible without having to change the conceptual or external schemas.

Major steps that I should take in setting up a database for a particular enterprise are:-

(i) Define the high level requirements of the enterprise. (This step generates a document known as the system requirement).

(2) Define a model containing all appropriate types of data and data relationships.

(3) Defining the integrity constraints on the data.

Date: / /

(4) Define the integrity constraints on the data.

(5) Define the physical level.

(6) For each known problem to be solved on a regular basis define user interface to carry out the task, and write the necessary application programs to implement the user interface.

(7) Create/initialize the database.

2(a)

(i) Department Name $\mathcal{G}$ count (department id)

(ii) Π empid, empName, gender, Department ID (σ salary > 5000 (employee ⋈ designation))

(iii) Π empid (σ allowance name = 'House' (employee ⋈ Allowance ⋈ Allowance Details)).

2(b)

(i) select name from Doctor where name not in (select name from works).

(ii) Update doctor
set address = 'Pokhara'
where name = 'Dr. Hari'

Date: / /

(iv) select name, depart_no from works where count (depart_no) ≥ 2;

3a.

	SQL	MYSQL.
(i)	SQL is a standard language for	MYSQL is a database manging accessing and manipulating system like SQL, create the database.
(ii)	SQL is not open source.	My MYSQL is an open source.
(iii)	SQL offers indexed views which are much more powerful, able views.	My MYSQL offers only updat- performing wise.

SQL is a direct query language, as such, it has limitations. Some of the reasons why access to database via programming languages is needed.

a) Complex computational process of the data.

b) Specialized user interface.

c) Access to more than one database at a time.

Desirable features of such systems.

(i) Ease of use
(ii) Simplicity of implementation
(iii) Conformance to standards for existing programming languages and database query languages.
(iv) Interoperability (the ability to use a common interface to diverse database systems).

Date: / /

3(b)

- 1NF :- It States that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be simple value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values or a combination of both as an attribute value for a single tuple.

2NF :- It is based on the concept of full functional dependency. A functional dependency $X \to Y$ is a full functional dependency if removal of any attribute 'A' from X means that the dependency doesn't hold any more; that is, for any attribute $A \in X$, $(X - \{A\})$ doesn't functionally determine A. Functional dependency $X \to Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and the dependency still holds; that is for some $A \in X$, $(X - \{A\}) \to Y$.

A relation schema R is in 2NF if every non-prime attribute A in R is fully functional dependent on primary key of R.

3NF :- It is based on the concept of transitive dependency. A functional dependency $X \to Y$ in a relation schema R; a transitive dependency if there exist a set of attribute Z in R that is neither candidate key nor subset of any key of R, and both $X \to Y$ and $X \to Z$ hold.

Date: / /

A relation schema R is in 3NF if it satisfies 2NF and no non-prime attribute of R is transitively dependency on primary key.

Normalization of data can be considered a process of analysing the given relation schema based on their FDs and primary keys to achieve the desirable properties of:

(i) Minimizing redundancy and
(ii) minimizing the insertion, deletion and update anomalies

4(b)

→ Database often hold the backbone of an organization; its transactions, customers, employee information, financial data for both the company and its customers, and much more are all held in database, often left to the power of database administrator with no security training. Database security and integrity are essential aspects of an organization's security posture.

Using views, it is possible to restrict data accessible to a user, the type of operation the user can perform through the views or both.

Consider the following DDL SQL statements.

```
CREATE VIEW V_Customer_Status
AS ( SELECT Cust_name_s, Cust_Status_s FROM
     customer)
```

This view selects only two fields from the table

Date: / /

customer which has a total of seven fields. This is called vertical restriction.

The other field might contain confidential information that should be accessible only to upper mangement. If you grant SELECT privilege to the view to some role(for example ROLE("staff")), then everyone who belongs to that role would be able to see customer's names and status, while the rest of the information that the table contains remains inaccessible to them.

4(a)

⇒ As assertion is a predicate expressing a condition that ~~we~~ we wish the database always to satisfy. When an assertion is made, the system tests it for validity, and test it again on every update that may violate the assertion. This testing may introduce a significant amount of overhead. Hence, assertions should be used with great care.

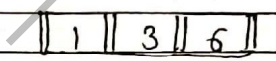
A trigger is a statement that is executed automatically by the system as a side effect of modification to the database. By using triggers, we don't have to wait to run the scheduled tasks because the triggers are invoked automatically before or after a change is made to the data in the table.

Date: / /

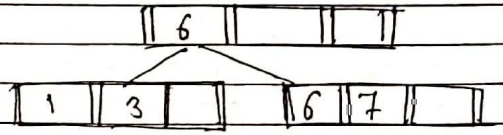
CREATE ASSERTION kote CHECK
(NOT EXIST (SELECT * from branch
WHERE branch_name = 'koteshwor' AND
assets != (SELECT SUM(amount)
FROM Loan
WHERE branch_name = 'koteshwor')));

5a.

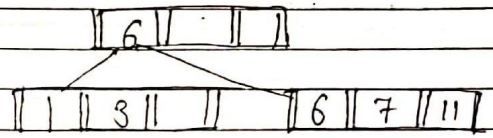
Insert 1, 3, 6.



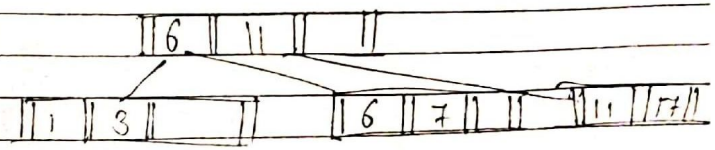
Insert 7 - Node is full. Applying rule 1.



Insert 11

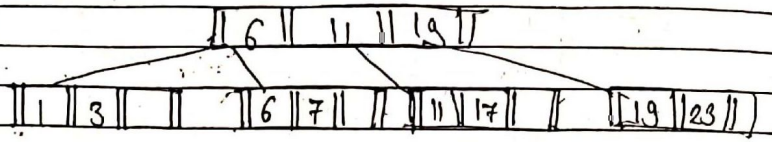


Insert 17; Node is full Applying rule 1.



Date: / /

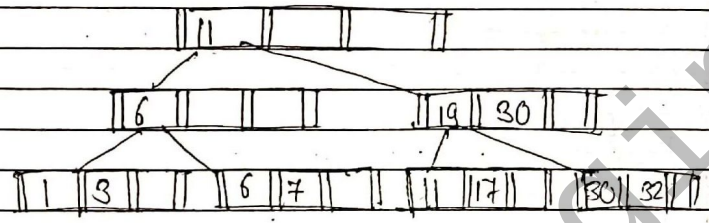
Inserting 19, 23, 23



Insert 80, 32, 88, 32 will make node full
Applying rule 1, and copy 80 to parent node
But this time parent node will also overflow so applying rule 2.

$$(\text{Ceil of } N/2) - 1 = \frac{4}{2} - 1 = 2 - 1 = 1.$$

Final tree will be.



6a)

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions we can say that S and S' are conflict equivalent. We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Concurrency control protocols allow concurrent schedules, but ensure that the schedule are conflict view serializable and are recoverable and cascadeless.

A lock is a mechanism to control concurrent access a data item. Data item can be locked in two modes.

(i) exclusive (X) mode
- Data item can be both read as well as written X-lock is requested using lock-X instruction.

(ii) Shared (S) mode.
- Data item can only be read. S lock is requested using lock-S instruction.

Transaction can proceed only after request is granted lock lock-compatibility matrix.

	S	X
S	true	false
X	false	false

6)

- Various types of failure that can occur in data base are.

(i) transaction failure:-
- Logical error - transaction cannot complete due to some internal error condition.
- System errors - the database system must

Date: / /

terminate an active transaction due to an error condition.

(ii) System Crash.

- A powerful failure or other hardware or software failure causes system to crash.

(iii) Disk failure

- A head crash or similar disk failure destroys all or part of disk storage.

A log is kept on stable storage. The log is a sequence of log records, and maintains a record of update activities on the database.

- When transaction T_i starts, it registers itself by writing a $\langle T_i \text{ start} \rangle$ log record.
- Before T_i executes write(X), a log record $\langle T_i, X, V_1, V_2 \rangle$ is written, where V_1 is the value of X before the write and V_2 is the value to be written to X.
- When T_i finishes its last statement, the log record $\langle T_i \text{ commit} \rangle$ is written.
- We assume for now that log records are written directly to stable storage (that is, they are not buffered).

7(a) A(I_1) property.

- There are four important properties of transaction that a DBMS must ensure to maintain data in the case of concurrent access and system failure

Atomicity (all or nothing)

- A transaction is said to be atomic if a transaction always executes all its action in one step or not execute any actions at all.

Consistency (No violation of integrity constraints)

- A transaction must preserve the consistency of a database after the execution.

Isolation (concurrent changes invisibles)

- The transaction must behave as if they are executed in isolation. It means that if several transactions are executed concurrently the result must be same as if they were executed serially in some order.

Durability.

- The effect of completed or commited transactions should persist even after a crash. It means once a transaction commits, the system must guarante that the result of its operations will never be lost, in spite of subsequent failure.

Date: / /

5 ORM

The Object Relational Model (ORM) is designed to provide a relational database management that allows developes to integrate database with their data types and methods. It is exential relational model that allows users to integrate object-oriented features into it.

Some of the benifits offered by object-Relational Model includes:-

- Extensibility
  Uses are able to extend the capability of the database server; this can be done by defining new data types, as well as user-defined patterns. This allows user to store and manage data.

- Complex types
  - It allows users to define new data types that combines one or more of the currently existing data types.

- Inheritance
  - Users are able to define objects or types and tables that procure the properties of other objects as wells add new properties that are specific to the object that has been defined.

- A field may also contain an object with attributes and operations.

2017 : Spring

Date: / /

10Y

=) Database is a collection of interrelated data and set of programs to acess those data. The purpose of the database management is to store and retrive data from the database that are convinent and efficient.

The advantage of DBMS are as following ways.

(a) Data Sharing
The DBMS helps to create an environment in which user have better access to managed data. which such access make easy for uses to respond the changes.

(b) Data Security.
More user access to data means high risk of data security breaches. DBMS provides framework for better enforcement of data privacy and security policies.

(c) Data Integration.
Wider access to well-managed data promotes an wider integrated view of organization's operation.

(d) data access.
The DBMS make large amount of data to operate quick possible by a database query.

(e) Decision making.
Better-managed data and improved data access make it better possible to generate better-quality information, on which better decision are based.

Date: / /

f) Increased end-user productivity

By developing best data quality tools and database management system the end user is increased which helps to make quick & informed decision.

Data Independance: is the ability to modify a schema definition in one level without affecting schema definition in next high level. Each higher level is immune to change of next lower level.

Importance of Data Independance.

a) It helps to improve the performance.

b) It helps to maintain security and hide data from person who doesn't need to know those information.

c) It helps to reduce the incongruity.

d) It helps to maintain standard

e) The state of being undamaged or undivided can be improved.

1(b)

→ A data model is a collection of conceptual data tools for describing a data relationships, semantics. It is a simple representation of complex real world data structure. It is a communication tool to facilitate interaction among ~~the~~ designer application

Date: / /

programmer and end users.

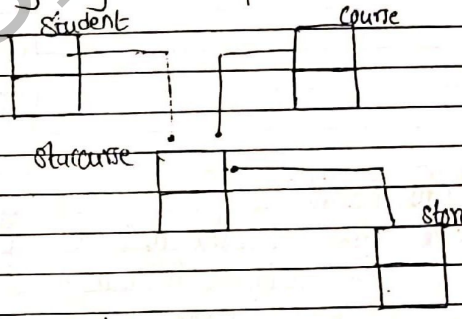
The type of data model are as following ways.

(a) Conceptual Data Model.

- It identifies the highest level relationship between the different entities.

Features

a) It includes the entity and relation between entities.
b) No attribute are specified.
c) No primary key are specified.



(b) Logical Data Model

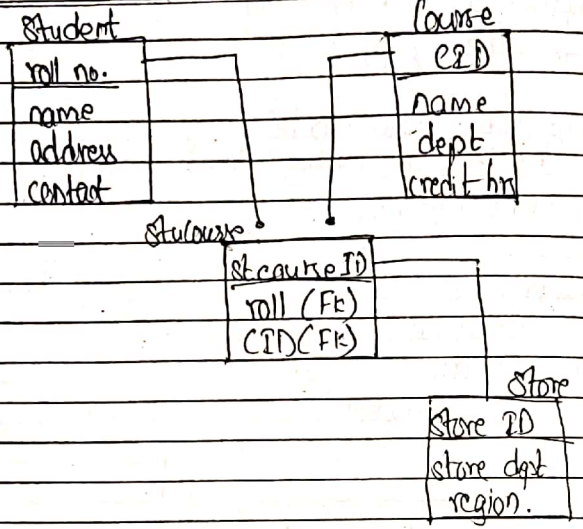
- It describes the data in as much detail as possible without saying much about their physical implementation.

Feature:-

a) It includes entity and relation between entities.
b) All attributes for each entity are specified.
c) Primary key are specified.
d) Normalization occurs at this level.

→ → →

Date: / /



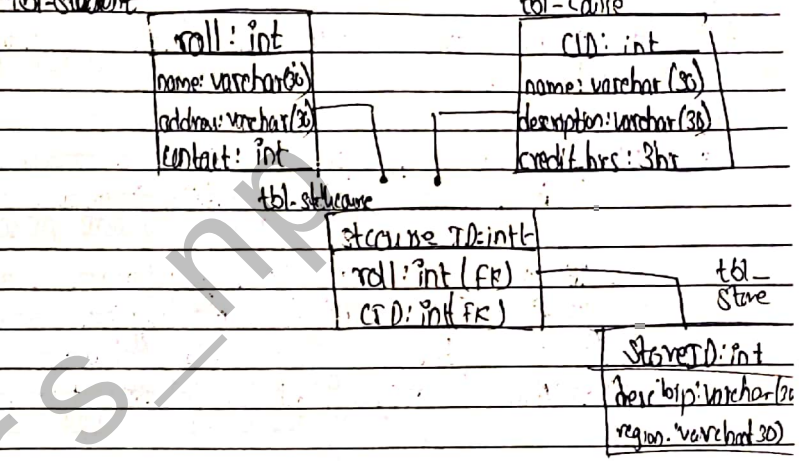
(c) Physical Data Model

- It represents how the data will be built in data. It shows in table structure including column name, data type, constraints, primary key, foreign key relationship between them.

Steps for physical data model design.

1(a) Convert entity into table
1(b) Convert attribute into column
1(c) Convert relationship into foreign key
1(d) Modify physical data model based on Physical constraints.

Date: / /



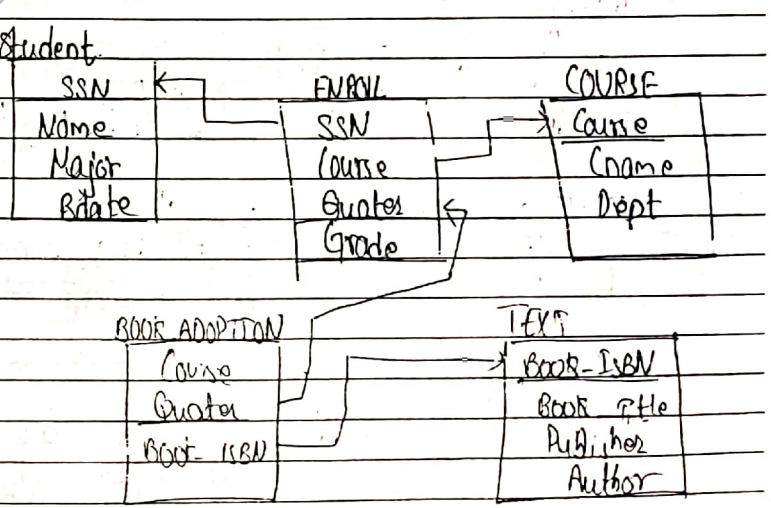


fig: schema-diagram.

Date / /

2(b)

→ The several parts of Structured Query Language are as following ways:-

(i) DDL (Data Defination language)
It is used to define database structure or schema

CREATE ⇒ to create objects in database
ALTER ⇒ to alter structure of database
DROP ⇒ to delete objects from database
TURNCATE ⇒ to remove all records from table including allocated space

(2) Interactive DML
It is used to manage data within schema objects. It is used for retriving of information and insert, ~~dat~~ delete or modify information in the database.

INSERT ⇒ to insert data in database
UPDATE ⇒ to update data within table
DELETE ⇒ to delete records from table

(3) View Defination
In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in ~~this~~ the database.

(4) TCL (Transaction Control Language)
It is used to manage changes made by DML

Date / /

Statement:

COMMIT ⇒ save work
ROLLBACK ⇒ restore database to original since commit

5) Integrity
Integrity in SQL means the consistency and the accuracy of the data that has been stored. It can be used to describe state, process or a function and is often used as proxy for data quality.

(6) Authorization
Authorization is the process of giving permission There are different forms of authorization on part of database
- Read Authorization - allows to read but not modify
- Insert Authorization - allows to insert
- Update " - allows to update
- delete " - allows to delete.

Some Domain types of SQL are:
(1) Varchar (n): fixed length varying character string with user specified length.
(2) Char (n): fixed length character string with user specified length.
(3) int : an integer
(4) smallint: a small integer
(5) numeric (p,d): a fixed point number with user specified precision, consist of 'p' digits and d of p digits to the right of the decimal point.
(6) float (n): floating point with user-specified precision
(7) Date : a calendar date containing four digit year.

Date: / /

month and days of the month.

⑧ time: the time of day in hours, minutes, seconds.

3a)

→ The basic structure of SQL queries consist of three clauses; select, from and where

~~The basic structure of~~
Typical sql query has

select $A_1, A_2, A_3 \cdots A_n$
from $r_1, r_2, r_3 \cdots r_m$
where P

Each A_i represents attribute
r_i represents relation
P is a predicate.

SELECT CLAUSE
~~If relations~~ It returns a set of record stored in data base.

WHERE CLAUSE
It consist of condition for returning the records.

FROM CLAUSE
It defines a cartesian product of the relation in the clause.

SQL Query for different type of set operation are

Date: / /

① UNION
select customer_name
from depositor
union
select customer_name
from borrower.

② INTERSECT
select customer_name
from depositor
~~except~~ intersect
select customer_name
from borrower

③ Except
select customer_name
from depositor
except
select customer_name
from borrower

④ except all
select customer_name
from depositor
except all
select customer_name
from borrower

⑤ UNION
select customer_name
from depositor
UNION all
select customer_name
from borrower

Date / /

(6) INTERSECT ALL
select customer_name
from depositor
intersect all
select customer_name
from borrower.

4(a)

→ Normalization of analysing the given relation schema based on this FDs and primary key to acheive the desirable properties of
(i) minimizing redundancy
(ii) minimizing insertion, deletion and update anomalies.
It is a process of filtering.

A) 1NF (First Normal form)
For 1NF the domain of the attribue must have the atomic values which means the 1NF doesn't allow to have a set of values.

Dname	Dnumber	DNGR. SSN	Dlocation.
Research	5	1234	{Ktm, Bkt, Pkh}
Admin	4	2345	{Birgunj}
Head	1	3656	{Bhairahwa}

① Department

Dname	Dnumber	DNGR SSN
Research	5	1234
Admin	4	2345
Head	1	3456

② Dlocation.

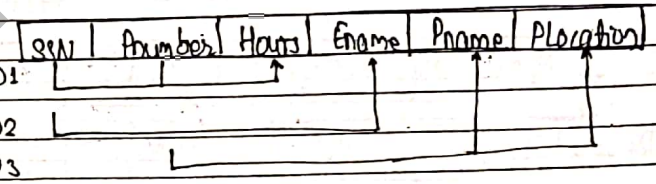
Dnumber	Dlocation
5	Ktm
5	Bkt
5	Pkh
4	Birgunj.

Date

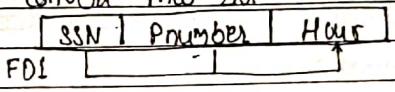
③ Depart

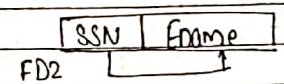
Dname	Dnumber	DNGR SSN	Dlocation.
Research	5	1234	Ktm
Research	5	1234	Pkh
Research	5	1234	Bkt
Admin	4	2345	Birgunj
Head	1	3656	Bhairahwa.

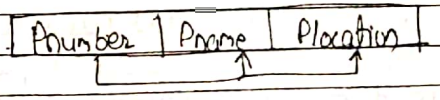
B) 2NF (Second Normal form)
- Firstly the given attribute must be in 1NF. The 2NF is based on the functional dependency. Every non-prime attribute 'A' in 'R' is fully FD on primary key of R.



It is not 2NF due to FD2 and FD3.
To convert into 2NF







→ → → →

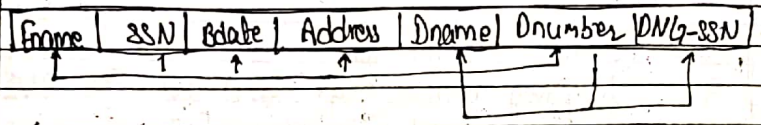
Date: / /

3NF (Third Normal Form)

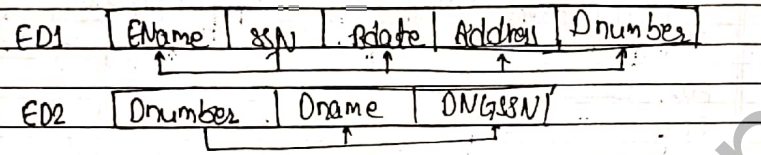
It is based on transitive dependency. The relation schema is in 3NF attribute if it satisfies.

(i) 2NF

(ii) No nonprime attribute of R is transitively dependent on primary key.



It is not in 3NF because of transitively dependancy of DN(s SSN and Dname on SN via Diagram.



4(b)

→ The encryption techniques to secure application are as following ways:-

(i) Data Encryption Standard (DES)

DES is an implementation of Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bit of the key are not used by the encryption algorithm.

DES has proved to be very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Date: / /

(ii) Advanced Encryption Standard.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

Some features of AES are:-

a) Symmetric key symmetric block cipher
b) 128-bit data, 128/192/256 bit keys.
c) Stronger & faster than Triple DES
d) Provide full specification and design details.

iii) Public key Encryption.

Or Public Key Encryption is a cryptographic system that ~~use~~ uses two keys. A public which is known to everyone and a private secrete key known only to the recipient of the message. An important and private keys are related in such a way that only the public key can be used to encrypt message.

5(a)

→

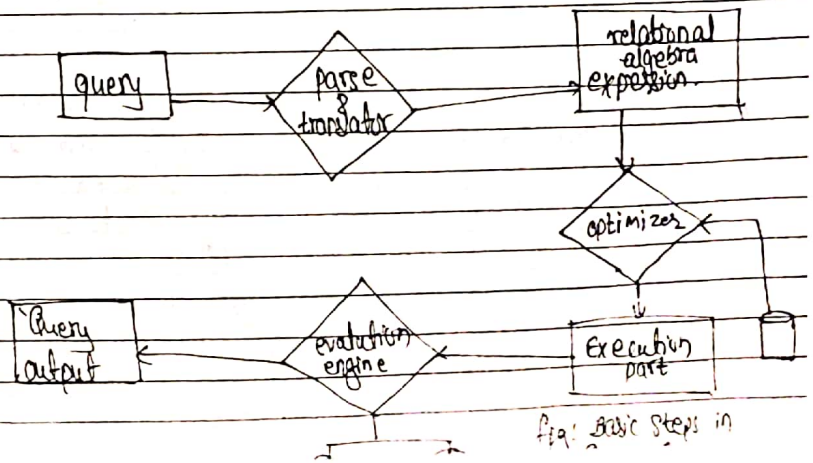


Fig: Basic steps in

Date: / /

Parsing and translation:
- translate the query into its internal form. This is then ~~transf~~ translated into relational algebra.
- parser checks syntax, verifies relations

Evaluation:
- The query execution engine takes a query
- evaluation plan, executes that plan and returns the answer to the query.

Optimization
- A relational algebra expression ~~that~~ may have many equivalent expression.
  eg:- $\sigma_{salary < 100000}(\pi_{salary}(instructor))$ is equivalent to $\pi_{salary}(\sigma_{salary < 75000}(instructor))$

Each relational algebra operation can be evaluated using one of several algorithm. Annoted expression specifying detailed evaluation strategy is called as evaluation plan.

5(b)

→ Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when query is processed. An index is a data structure which is used to quickly locate and access the data in a database table.

B^+ tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of same length.

The B^+ tree is a rooted tree satisfying ~~following~~

Date: / /

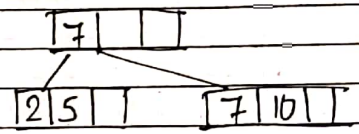
property.
i) All path from root to leaf are of same length.
ii) Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.

Example.
Consider no. of pointer 4.
2, 4, 7, 10, 13, 16, 20, 22, 23, 24.

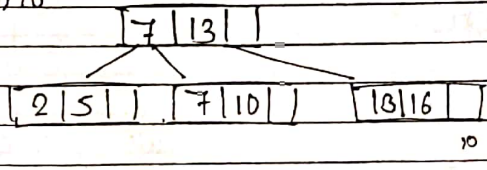
no. of pointer = 4
no. of node = 4-1 = 3

Insert 2, 5, 7

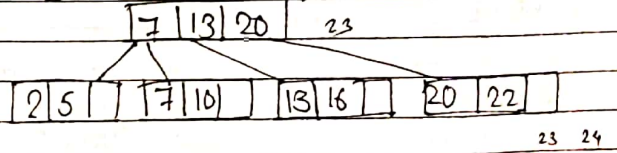
Insert 10,



Insert 13, 16

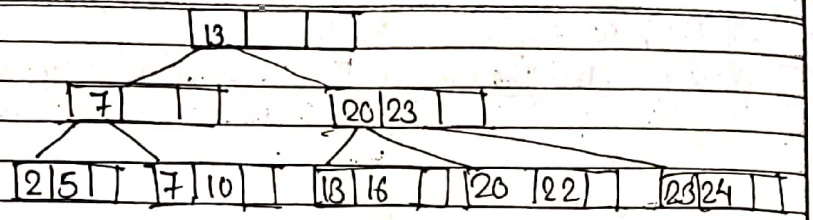


Insert 20, 22



Insert 23, 24, node over flow.

Date: / /



6b)

→ A transaction can be defined as a ~~re~~ group of tasks. A single task is the minimum processing unit which cannot be divided further.
A transaction is a very small unit of program and it may contain several low level tasks. A transaction must maintain Atomicity, Consistancy, Isolation and Durability commonly known as ACID property.

Atomicity
This propery states that a transaction must be created as an atomic unit, that is either all of its operations are executed or none.

Consistancy
The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on data

Isolation
In a database system where more than one transaction are being executed simultaneously and in parallel.

Durability:
The database should be durable enough to hold all its latest updates even if the system fails or restarts.

The two-phase locking protocol (2PL) for concurrency control are:-
Phase 1: Growing Phase
- transaction may obtain lock but may not release lock

Phase 2: Shrinking Phase
- transaction may release lock but may not obtain locks

(7)
(a) Data Dictionary:-
A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects and other data.
The data dictionary is a crucial component of any relational ~~be~~ database. Ironically, because of it's importance, it is invisible to most database users. Typically, only database administrations interact with the data dictionary.

(c) Functional Dependencies.
They are the fundamentals to the process of normalization. Functional Dependency plays key role in differentiating good database design from bad database design. Functional Dependencies describes the relationship between attributes in a table.

Date: / /

for attributes X and Y in relation R.
$X \to Y$ is full functional dependency if removal of any attribute A from X means dependency doesn't hold any more and $X \to Y$ is partial dependency if removal of any attribute A from X and still holds dependency.

2016 (Spring)

1a)

→ In file system data are scattered in various files, and files may be in different formats; writing new application programs to retrive the appropriate data is difficult. There is also chance of data duplication. Also changing the files would lead to change in application programs. Thus, we need DBMS. DBMS can help in removal of redundency using data normalization. Each user has a different set of access. DBMS helps in easy access to data, easy recovery and is more flexible than file system.

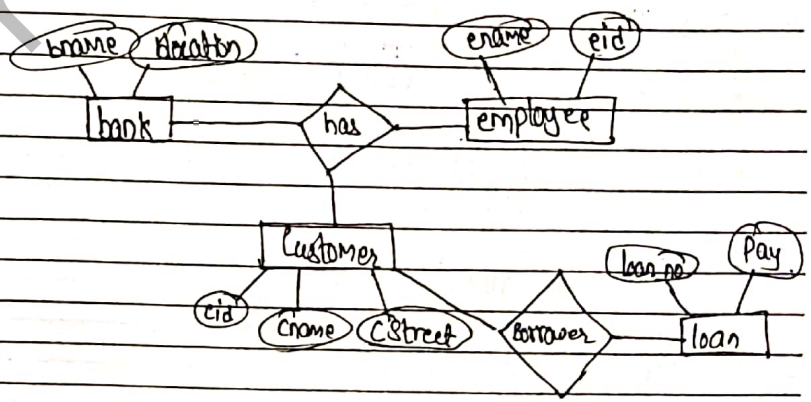
To simplify the user interaction with the system, data abstraction is implied. There are three levels of data abstraction:

(i) Physical level
The lowest level of abstraction describes how the data are actually stored.

(ii) Logical level
- The next higher level of abstraction describes what data are stored in the database, and what relationship exist among those data

iii) View level
- The highest level of abstraction describes only part of entire database. Many users of database system do not need its entire information, instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

1b)



3(a)

⇒ A stored procedure is prepared SQL code that you can save, so the code be reused over and over again. We can also pass parameters to stored procedure. So that stored procedure can act based on the parameter value(s) that is passed.

Advantages:
Stored procedure can be used to maintain data

Date: / /

Integrity and ~~en~~ enforce database policy without relying on an external program to do so
- Easier to expand a system.

Disadvantages:-
- can make debugging more complex.
- can be sensitive to being dropped during copy operations if not done correctly.

Store procedure are created using following syntax:

Store procedre syntax

```
CREATE PROCEDURE procedurename
AS
  Sql_Statement
  GO;
```

Execute a stored Procdure

```
EXEC procedure_name;
```

4(a)

=> Integrity constraints guard against the accidental damage to the database, by ensuring that authorized changes the database do not result in a loss of data consistency.

Assertion is a predicate expressing a condition that we use the database always to satisfy. when an assertion is made, the system tests it for validity, and tests it again on every update that may violate the asser. This testing may introduce a significant amount of overhead, assertions should be used with great care.

Syntax: CREATE ASSERTION <assertion_name> CHECK <predicate>

A trigger is a statement that is executed automatically by system as a side effect of a modification to the database. To design a trigger mechanism we must
(i) Specify the condition under which the trigger is to be executed.
(ii) Specify the actions to be taken when the triggers executes.

Syntax: CREATE [OR REPLACE] TRIGGER trigger_name
[BEFORE | AFTER | INSTEAD OF]
[INSERT | UPDATE | DELETE] [:OF col_name]
ON table_name
[REFERENCING OLD AS | NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration_statements
BEGIN
  Executable_statements
EXCEPTION
  Exception_handling_statements
END;

(b)

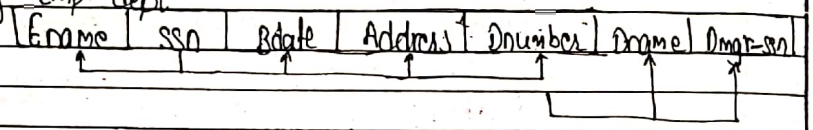
=> A functional dependency is a constraint between two sets of attributes in a relation from database. In other word, functional dependency is a constraint that describes relationships between attributes in a relation.

Date: / /

3NF: It is based on the concept of transitive dependency. A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there exist a set of attributes Z in R, that is neither candidate key nor subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

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

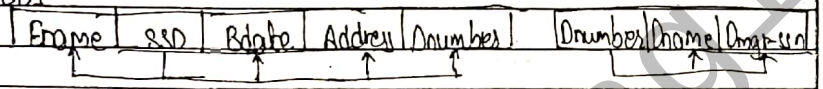
eg: Emp-dept



It is not in 3NF because of transitive dependency and Dmgr-ssn and Dname on SSN via Dnumber.

Using 3NF normalization.

FD1



BCNF: A relation is in BCNF if every determinant is a candidate key.

Consider following example

Funds

FUNDID	Investment Type	Manages
99	Common Stock	Smith
33	Common Stock	Green
22	Growth Stock	Brown
11	Municipal stock	Smith

Date: / /

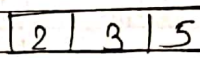
FD1 : Fund ID, Investment Type → manages
FD2 : FundID, manages → Investment Type
FD3 : Manages → Investment Type

In this case combination of FundID and Investment type form a candidate key. Similarly combination of fundIDs and manages also form candidate key but manger itself is not a candidate key.

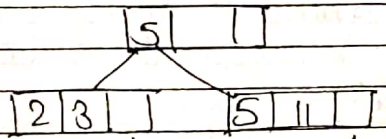
Consider what happens if we delete the tuple with fundID and manages also from candidate key but manager itself is not a candidate key. 22. We loose the fact that Brown manages the investment Type. "Growth Stock". Hence, it is not a BCNF because not all determinats (Manager in FD3) are candidate keys.

Sol Construct a B+

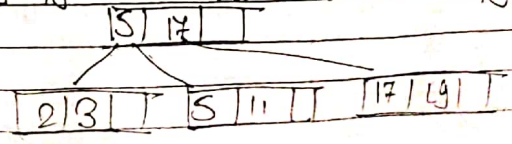
Insert 2, 3, 5



Insert 11: But Node is full, Apply rule 1.



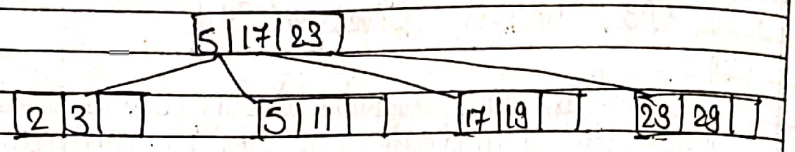
Insert 17, 19. 19 will make node full, Applying rule 1.



144

Date: / /

Insert 23, 29. 29 will make node full.



(6)

a) Query processing includes translation of high-level queries in low-level expressions that can be used at the physical level of the file system, query optimization and execution of query submitted by the user.

SQL is a high level language. The user specify what to search for, not how the search is actually done. The algorithms are chosen automatically by the DBMS. For a given SQL query there may be many possible execution plans. Amongst all equivalent plans choose the one with lowest cost. Cost is estimated using statistical information from the database catalog eg: number of tuples in each relation, size of tuples etc.

(b)

→ Log is a sequence of log records and maintains a record of update activities on the database. Log is kept on stable storage

Deferred database modification scheme records all modification to the log, but defers all the writes to after partial commit. Assume that transactions execute serially. Transaction starts by writing $\langle T_i \text{ start} \rangle$ record to log. A write(X) operation results in a log record $\langle T_i, X, V \rangle$ being written, where V is the new value of X. The write is not performed on X at the time, but is differed. When T_i partially commits, $\langle T_i \text{ commit} \rangle$ is written to the log records are read and used to actually execute the previously deffered writes.

Immediate database modification scheme allows database updates of an uncommitted transaction to be made as the writes are issued. Update log record must be written before database item is written. Output of updated blocks can take place at any time befor or after transaction commit. Order in which blocks are output can be different from the order in which they are written.

(b)

	Exclusive lock	Shared lock
(i)	Data item can be both read as well as written.	Data item can only be read.
(ii)	X-lock is requested using lock-X instruction.	S-lock is requested using lock-S instruction.

Conflict serializability

- Instructions I_i and I_j of transaction T_i and T_j respectively conflict if and only if there exist some item accessed by both I_i and I_j and atleast one of these instructions write(Q).

Consider the following operation.

(i) I_i = read(P). I_j = read(P). I_i and I_j don't conflict.

145

Date: / /

ii) L_i = read(P), L_j = write(P). They conflict.
iii) L_i = write(P), L_j = read(P). They conflict.
iv) L_i = write(P), L_j = write(P). They conflict.

- A conflict between L_i and L_j forces a temporal order between them.
- If L_i and L_j are consecutive in a schedule and they don't conflict, their result would remain the same even if they had been interchanged in the schedule.

→ If a schedule S can be transformed into schedule S' by a series of swaps of non-conflicting instructions, then S and S' are conflict equivalent.

View ~~serializability~~ serializability

- S and S' are view equivalent if the following three conditions are met:

i) For each data item P, if transaction T_i reads the initial value of P in schedule S, then transaction T_i must, in schedule S' also read the initial value of P.

ii) For each data item P, if transaction T_i executes read(P) in schedule S, and that value was produced by T_j (if any), then in schedule S' also transaction T_i must read the value of P that was produced by transaction T_j.

iii) For each data item P, the transaction that performs the final write(P) operation in schedule S must perform final write(P) operation in schedule S'.

Date: / /

(7) Short Notes.

(a) Distributed Database

- A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions those transactions that access data is only that single site. The general structure of a distributed system appears in figure below.

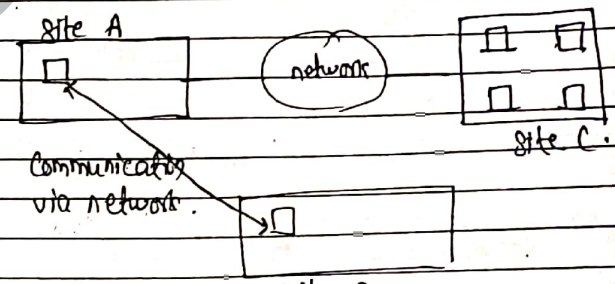


Fig Distributed System.

There are several reasons for building distributed system, including sharing of data, autonomy, and availability.

(c) Cryptography

Cryptography is a science of encoding information before sending via unreliable communication paths so that only an authorized recieves can decode and use it.

The coded message is called cipher ~~tip~~ text.

146

Date: / /

and they original message is called plain text. The proces of converting cipher text to plain text by the receiver is called decoding or decryption. The entire procedure of communicating using cryptography can be illustrated through the following diagram

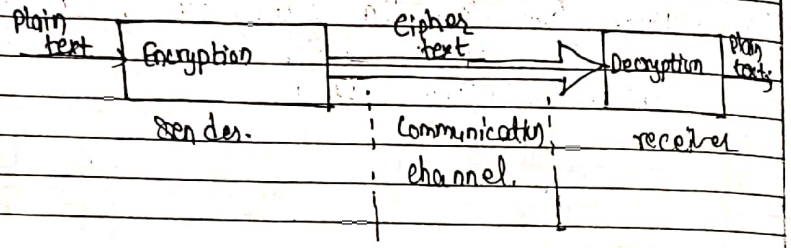


Fig: Cryptography.

2(b) $\Pi_{customer\_name}(borrower) \cup \Pi_{customer\_name}(loan)$

(iii) account ← account − $\sigma_{branch\_name = "B1"}$ (account)

iv) Account ← $\Pi_{account\_number, branch\_name, balance * 1.05}$ (account)

3(b)

(i)
```
SELECT PatientID
FROM Patients, Hospitals
WHERE Patients.PatientID = Hospitals.PatientID
AND location = 'Butwal' AND Patient Name LIKE 'X.a';
```

(ii)
```
DELETE FROM Doctor As D, Doctor As A
WHERE D.salary > AVG(A.salary)
```


(iii)
```
UPDATE Doctors
SET salary = salary * 1.185
WHERE Department = 'OPD'
```