APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY Fourth Semester B.Tech Degree Examination July 2021 (2019 Scheme)

Course Code: CST204

Course Name: DATABASE MANAGBMENT SYSTEMS

1. List any three categories of database users, highlighting any one important characteristic of each category.

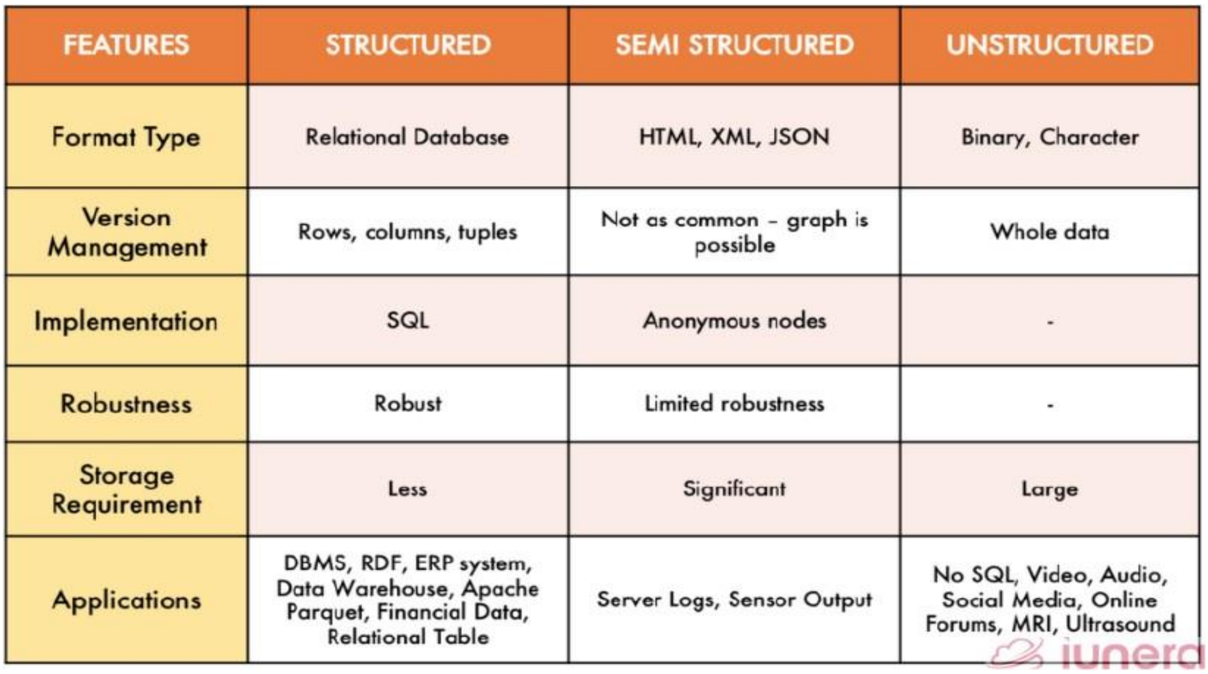
a. Database Administrators
A person who has central control over the system is called DBA. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA).

b. Database Designers
Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.

c. System Analyst and Application Programmers

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements. **Application programmers** implement these specifications as programs. Then they test debug, document and maintained these canned transactions

2. What are the major difference between structured, unstructured and semi structured data

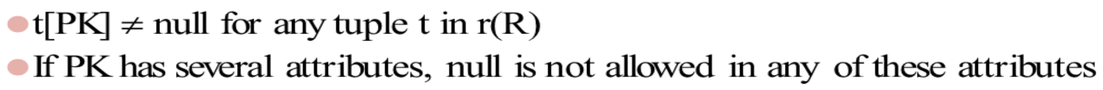


FEATURES	STRUCTURED	SEMI STRUCTURED	UNSTRUCTURED
Format Type	Relational Database	HTML, XML, JSON	Binary, Character
Version Management	Rows, columns, tuples	Not as common – graph is possible	Whole data
Implementation	SQL	Anonymous nodes	-
Robustness	Robust	Limited robustness	-
Storage Requirement	Less	Significant	Large
Applications	DBMS, RDF, ERP system, Data Warehouse, Apache Parquet, Financial Data, Relational Table	Server Logs, Sensor Output	No SQL, Video, Audio, Social Media, Online Forums, MRI, Ultrasound

3. What is entity integrity? Why is it important?

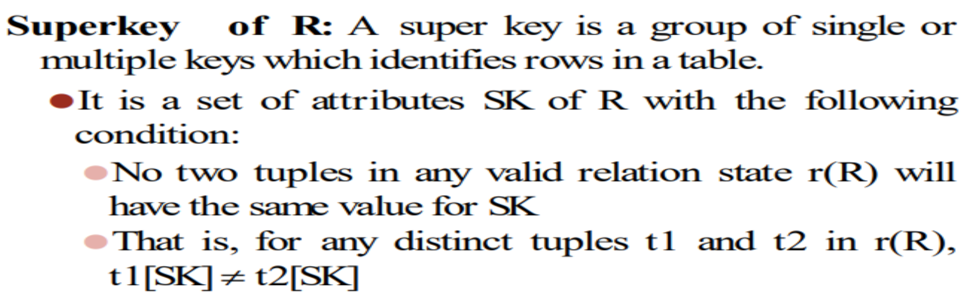
The entity integrity constraint states that primary key value can't be null.

This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.



- t[PK] ≠ null for any tuple t in r(R)
- If PK has several attributes, null is not allowed in any of these attributes

4. Distinguish between Super key, Candidate key, and Primary key using a real convincing example.



Superkey of R: A super key is a group of single or multiple keys which identifies rows in a table.

- It is a set of attributes SK of R with the following condition:
  - No two tuples in any valid relation state r(R) will have the same value for SK
  - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK]

Candidate Key

A candidate key is a set of one or more columns that can uniquely identify a row within a table. A table can have multiple candidate keys, but can only have one primary key.

A relation may have more than one key. In such cases each of the keys are called candidate keys

Primary Key

A primary key constraint is a column that uniquely identifies every row in the table of the relational database management system.

Example: Consider a relational schema : student(Roll_no, Name, Registration_no)

Super Key : { Roll_no, Name}

In the above example, we saw that we have two candidate keys i.e (Roll_no) and (Registration_no).

Primary Key: Any candidate key can become a primary key (Roll no or Registration no)

5.Illustrate the concept of trigger in SQL with example.

A trigger is a statement that the system executes automatically as a side effect of a modification to the database.

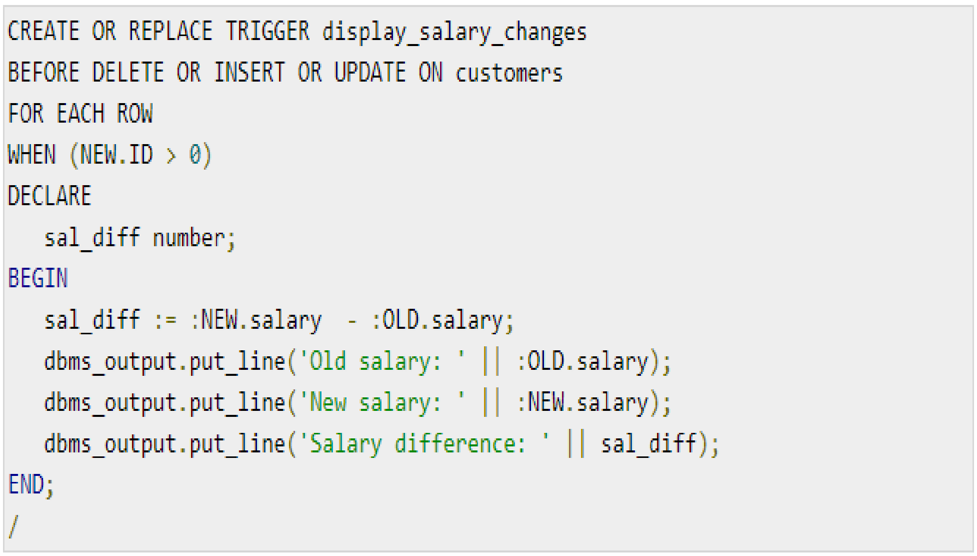
A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.

Action to be taken when certain events occur and when certain conditions are satisfied.

For example,

- a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- it may be useful to specify a condition that, if violated, causes some user to be informed of the violation.
- When an event occur, database trigger is fired and a predefined PL/SQL block will perform necessary action.

Create a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.



```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

6. Compare DDL and DML with the help of an example.

DDL is a set of SQL commands used to create, modify, and delete database structures but not data.

The common DDL commands are create table, alter table, drop table, truncate ,rename

eg: CREATE TABLE <table name> (column1 datatype(size), column2 datatype(size), column3 datatype(size),………, PRIMARY KEY(coulmn name));

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

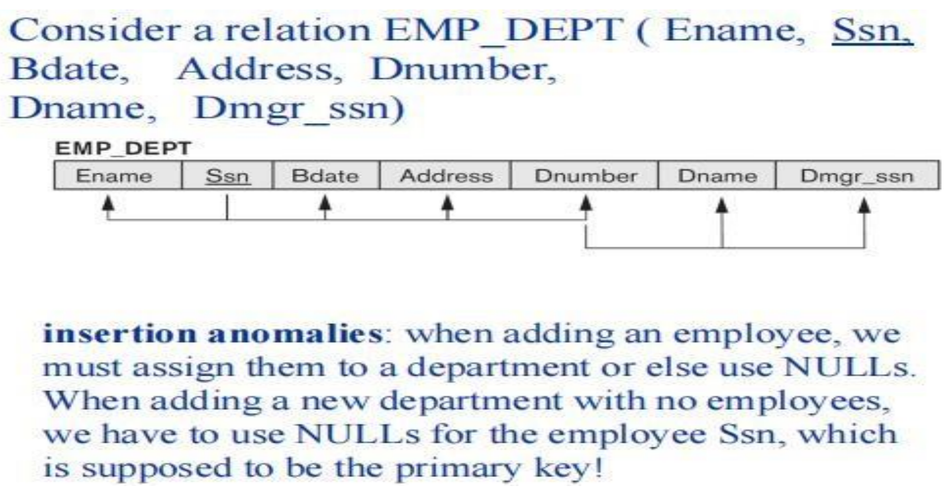
The DELETE Statement in SQL is used to delete existing records from a table.

We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Eg: DELETE FROM <table_name> WHERE <condition>;

7. Illustrate different anomalies in designing a database

Insertion Anomalies


Consider a relation EMP_DEPT (Ename, Ssn, Bdate, Address, Dnumber, Dname, Dmgr_ssn)

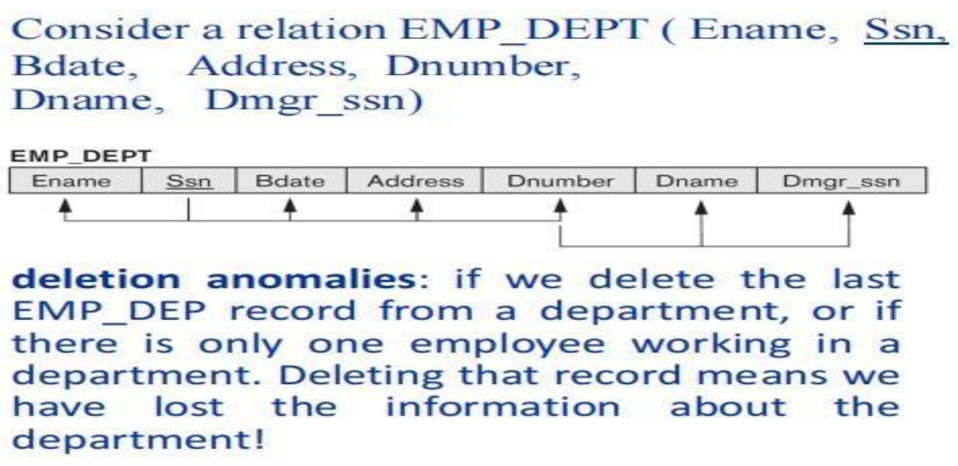
EMP_DEPT
Ename
Ssn
Bdate
Address
Dnumber
Dname
Dmgr_ssn

insertion anomalies: when adding an employee, we must assign them to a department or else use NULLs. When adding a new department with no employees, we have to use NULLs for the employee Ssn, which is supposed to be the primary key!

Deletion Anomalies

When a project is deleted, it will result in deleting all the employees who work on that project.

Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.


Consider a relation EMP_DEPT (Ename, Ssn, Bdate, Address, Dnumber, Dname, Dmgr_ssn)

EMP_DEPT
Ename
Ssn
Bdate
Address
Dnumber
Dname
Dmgr_ssn

deletion anomalies: if we delete the last EMP_DEP record from a department, or if there is only one employee working in a department. Deleting that record means we have lost the information about the department!

Updation Anomalies

- EMP_DEPT, if we change the value of one of the attributes of a particular department say,

  the manager of department 5 we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

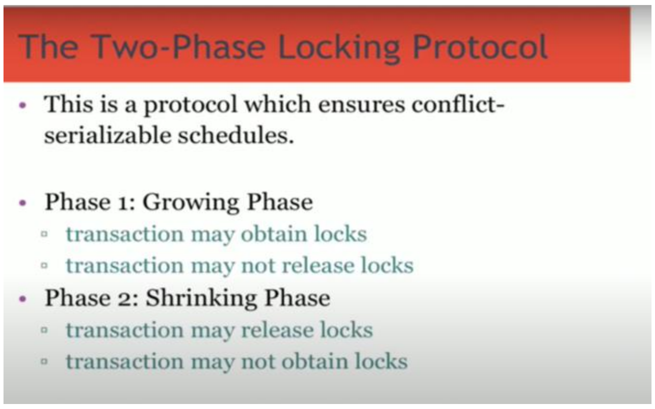
8. How can we conclude two FDs are equivalent?

Two sets of functional dependencies E and F are equivalent if E+ = F+. Therefore, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions—E covers F and F covers E— hold.

Let FD1 and FD2 are two FD sets for a relation R.

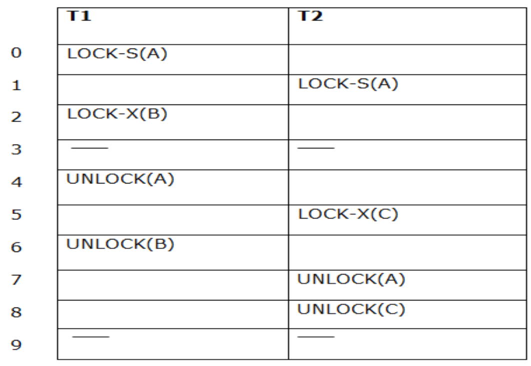
1. If all FDs of FD1 can be derived from FDs present in FD2, we can say that FD2 ⊃ FD1.
2. If all FDs of FD2 can be derived from FDs present in FD1, we can say that FD1 ⊃FD2.
3. If 1 and 2 both are true, FD1=FD2.

9. Illustrate two phase locking



The Two-Phase Locking Protocol

- This is a protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks



	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

Lock point: The Point at which the growing phase ends

Transaction T1:

- **Growing phase:** from step 1-3

- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

10. How conversion of locks are achieved in concurrency control?

Changing the mode of a lock that is already held is called *lock conversion*.

Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the access mode requires a more restrictive lock than the one already held. A process can hold only one lock on a data object at any given time, although it can request a lock on the same data object many times indirectly through a query.

If lock conversion is allowed, then upgrading of lock(from S(a) to X(a)) is allowed in the Growing Phase, and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

PART B

11. a A company has the following scenario: There are a set of salespersons. Some of them manage other salespersons. However, a salesperson cannot have more than one manager. A salesperson can be an agent for many customers. A customer is managed by exactly one salesperson. A customer can place any number of orders. An order can be placed by exactly one customer. Each order lists one or more items. An item may be listed in many orders. An item is assembled from different parts and parts can be common for many items. One or more employees assemble an item from parts. A supplier can supply different parts in certain quantities. A part may be supplied by different suppliers.

- Identify and list entities, suitable attributes, primary keys, and relationships to represent the scenario.

  Entities & attributes

Sales_person - sp_id, name, age, address	PK- sp_id
Customer - cust_id,name,address,phone	PK -cust_id
Part - p_id,pname,ptype	PK- p_id
Item – i_code,iname	PK - i_code

Order – order id,order date PK – order id

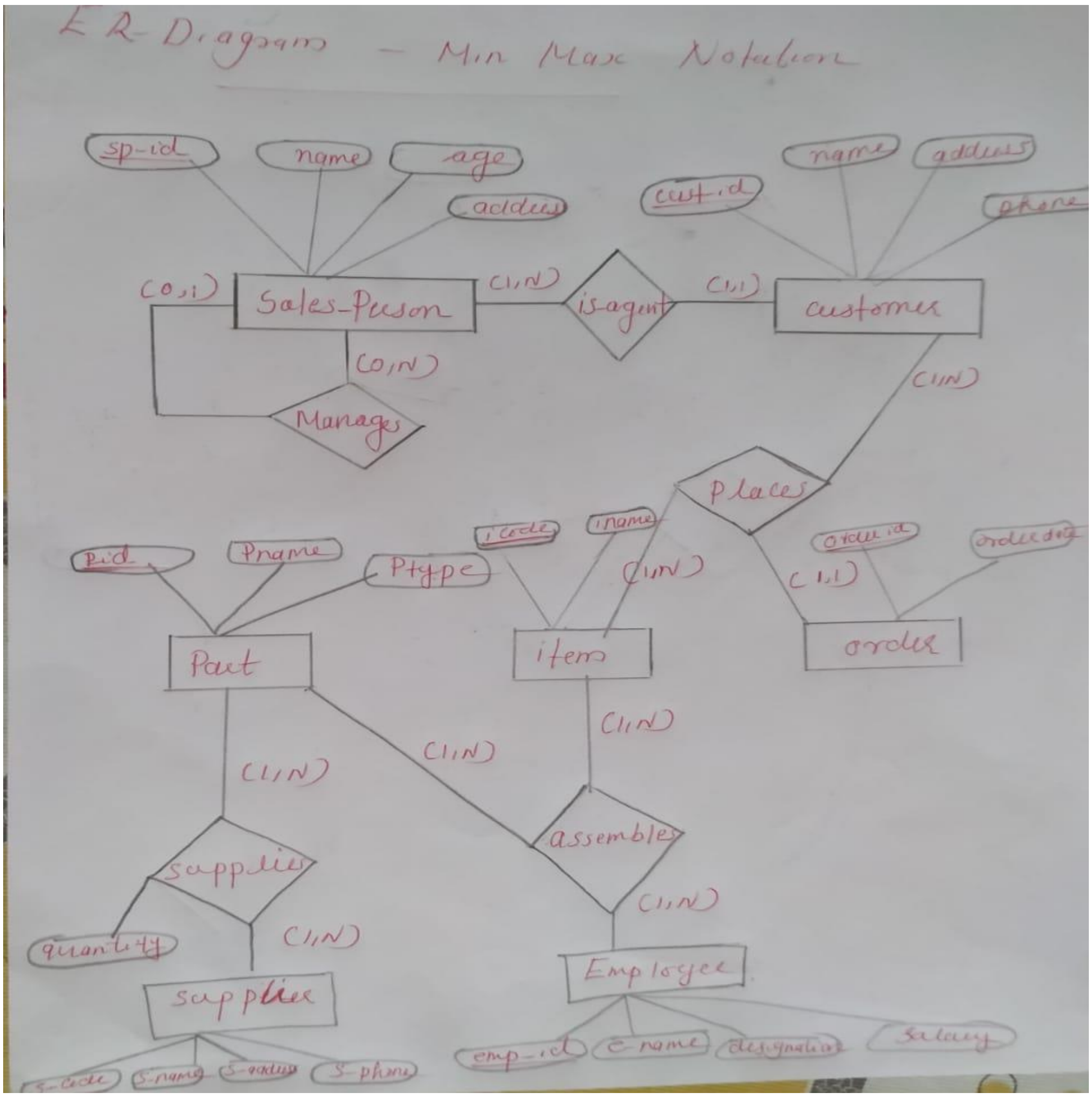
Supplier -s_code,s-name,s_address,s_phone PK - s_code

Employer – emp_id,e_name,designation,salary PK -emp_id

Relationships

Is agent,places,supplies,assembles,manages

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



11.b Explain three schema architecture with figure

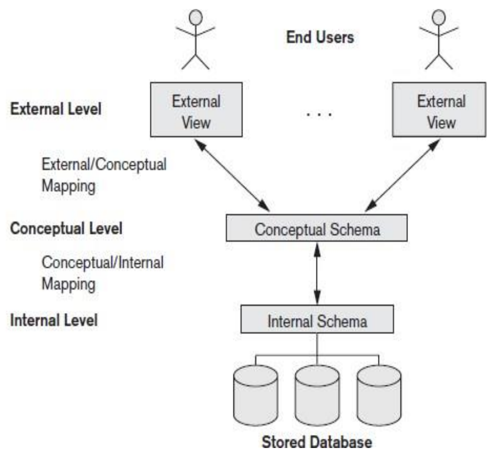
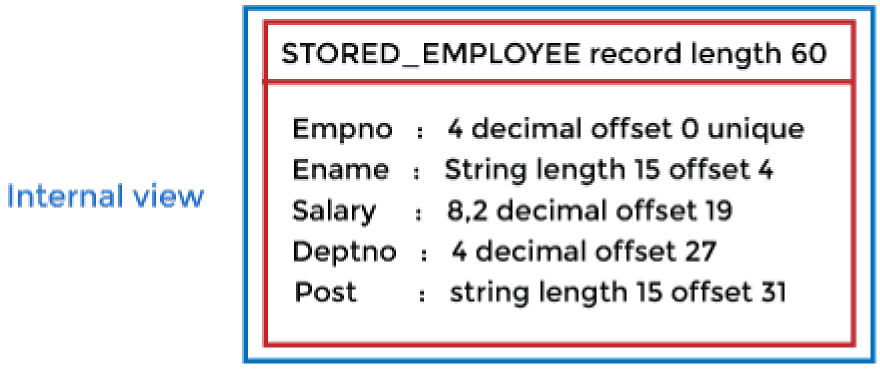


Figure 2.2
The three-schema architecture.

- Internal Level
- The internal level has an internal schema which describes the physical storage structure of the database.
- The internal schema is also known as a physical schema.
- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.



Internal view

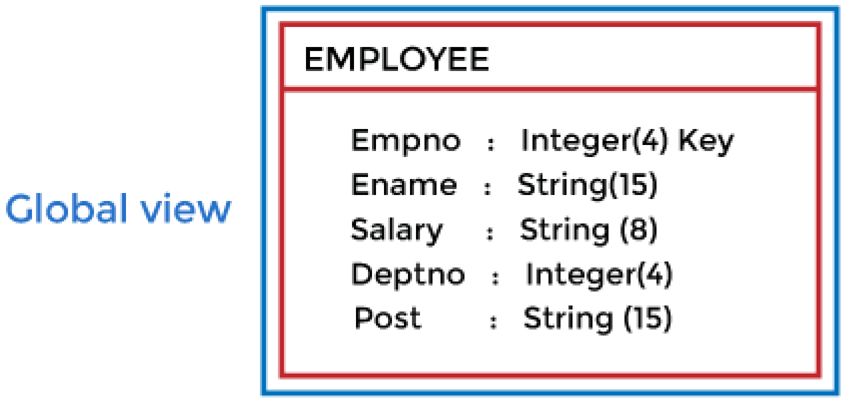
STORED_EMPLOYEE record length 60

Empno	:	4 decimal offset 0 unique
Ename	:	String length 15 offset 4
Salary	:	8,2 decimal offset 19
Deptno	:	4 decimal offset 27
Post	:	string length 15 offset 31

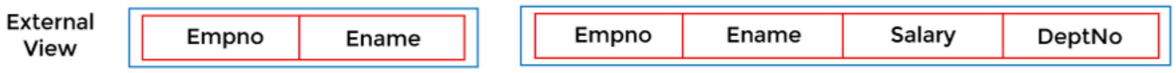
Conceptual Level

- The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- The conceptual schema describes the structure of the whole database.
- The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.

- In the conceptual level, internal details such as an implementation of the data structure are hidden.
- Programmers and database administrators work at this level.



- External or view level
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
- The view schema describes the end user interaction with database systems.



- In a DBMS based on the three-schema architecture, each user group refers only to its own external schema.
- Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.

If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view

There are basically two types of mapping in the database architecture:

- Conceptual/ Internal Mapping
- External / Conceptual Mapping

Conceptual/ Internal Mapping

The Conceptual/ Internal Mapping lies between the conceptual level and the internal level. Its role is to define the correspondence between the records and fields of the conceptual level and files and data structures of the internal level.

External/ Conceptual Mapping

The external/Conceptual Mapping lies between the external level and the Conceptual level. Its role is to define the correspondence between a particular external and the conceptual view.

12 a. Illustrate Database architecture with a neat diagram

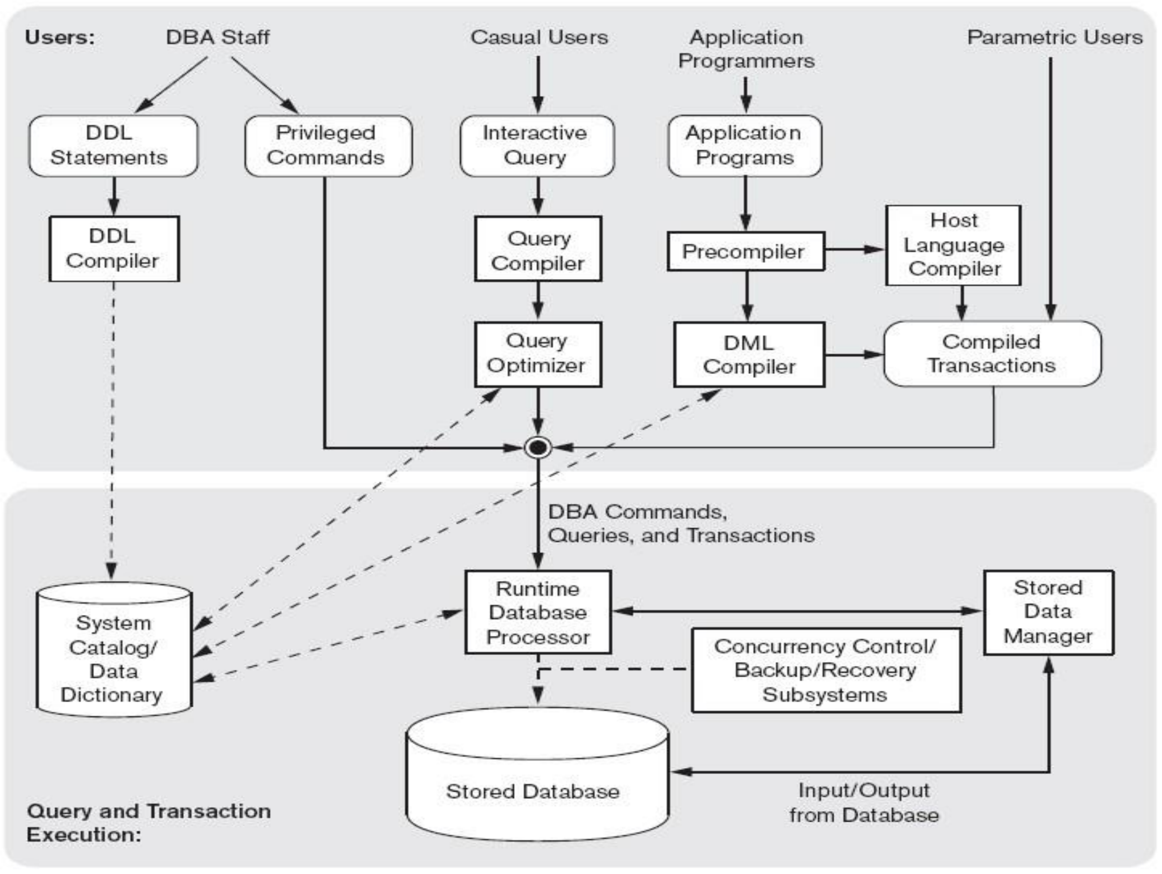


Figure 2.3
Component modules of a DBMS and their interactions.

The figure is divided into two parts.
The top part of the figure refers to the various users of the database environment and their interfaces.
The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.

1. Users

The top part shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries, application programmers who create programs using some host programming languages, and parametric users who do data entry work by supplying parameters to predefined transactions. The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

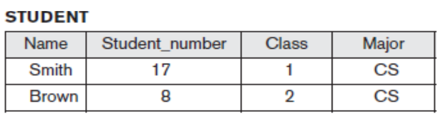
2. DDL compiler :The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta- data) in the DBMS catalog
3. Query compiler :The interactive queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form
4. Query optimizer :After query compilation, the query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.

5. Precompiler :Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The precompiler extracts DML commands from an application program written in a host programming language.
6. **DML compiler :** It collects the DML commands from precompiler and compiles them to produce object code for database and the rest of the program is sent to the **host language compiler**.
7. **Compiled/Canned transactions:** The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction. An example is a bank withdrawal transaction where the account number and the amount may be supplied as parameters.
    - In the lower part of Figure , the **runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters.
    - It works with the **system catalog** and may update it with statistics.
    - It also works with the **stored data manager,** which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
    - **The runtime database processor** handles other aspects of data transfer, such as management of buffers in the main memory.
    - The **concurrency control and backup and recovery systems** are integrated into the working of the runtime database processor for purposes of transaction management.

b. Explain the characteristics of Database system.

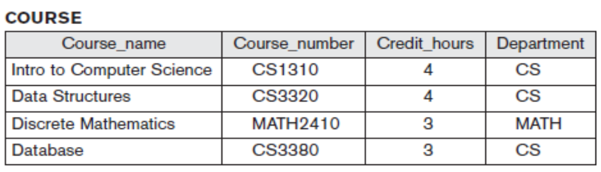
1. Self describing nature of a database system

- Database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- This definition is stored in the DBMS catalog
- Information stored in the catalog is called metadata and it describes the structure of the primary database.



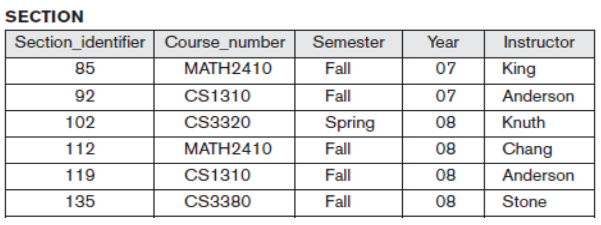
STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS



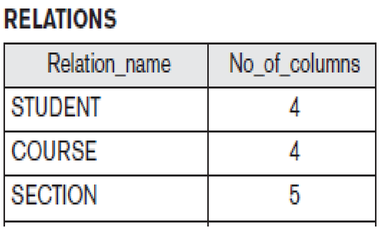
COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS



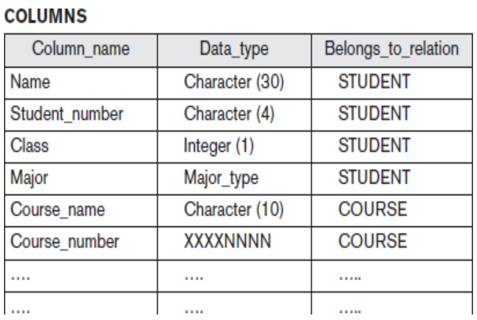
SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone



RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5



COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....

2. Insulation between Programs and Data, and Data Abstraction

- The structure of data files is stored in the DBMS catalog separately from the access programs. This property is called **program-data independence**
- An operation (also called a function or method) is specified in two parts.

Interface

The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).

Implementation

The implementation (or method) of the operation is specified separately and can be changed without affecting the database

3.Support of Multiple Views of the Data

- A database has many users, each user may require a different perspective or view of the database.
- A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.

4.Sharing of Data and Multiuser Transaction Processing

- DBMS must include concurrency control software

▫ to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct

- DBMS must enforce several transaction properties

▫ Isolation property

- ensures that each transaction appears to execute in isolation from other transactions

- even though hundreds of transactions may be executing concurrently.

▫ Atomicity property

- ensures that either all the database operations in a transaction are executed or none are.
- Any mechanical or electrical device is subject to failure, and so is the computer system. In this case we have to ensure that data should be restored to a consistent state.
- For example an amount of Rs50 has to be transferred from Account A to Account B
- Let the amount has been debited from account A but have not been credited to Account B and in the meantime, some failure occurred. So, it will lead to an inconsistent state. So, we have to adopt a mechanism which ensures that either full transaction should be executed or no transaction should be executed ie, the fund transfer should be atomic.

13 a. Study the tables given below and write relational algebra expressions for the queries that follow.

STUDENT(<u>ROLLNO</u>, NAME, AGE, GENDER, ADDRESS, ADVISOR)

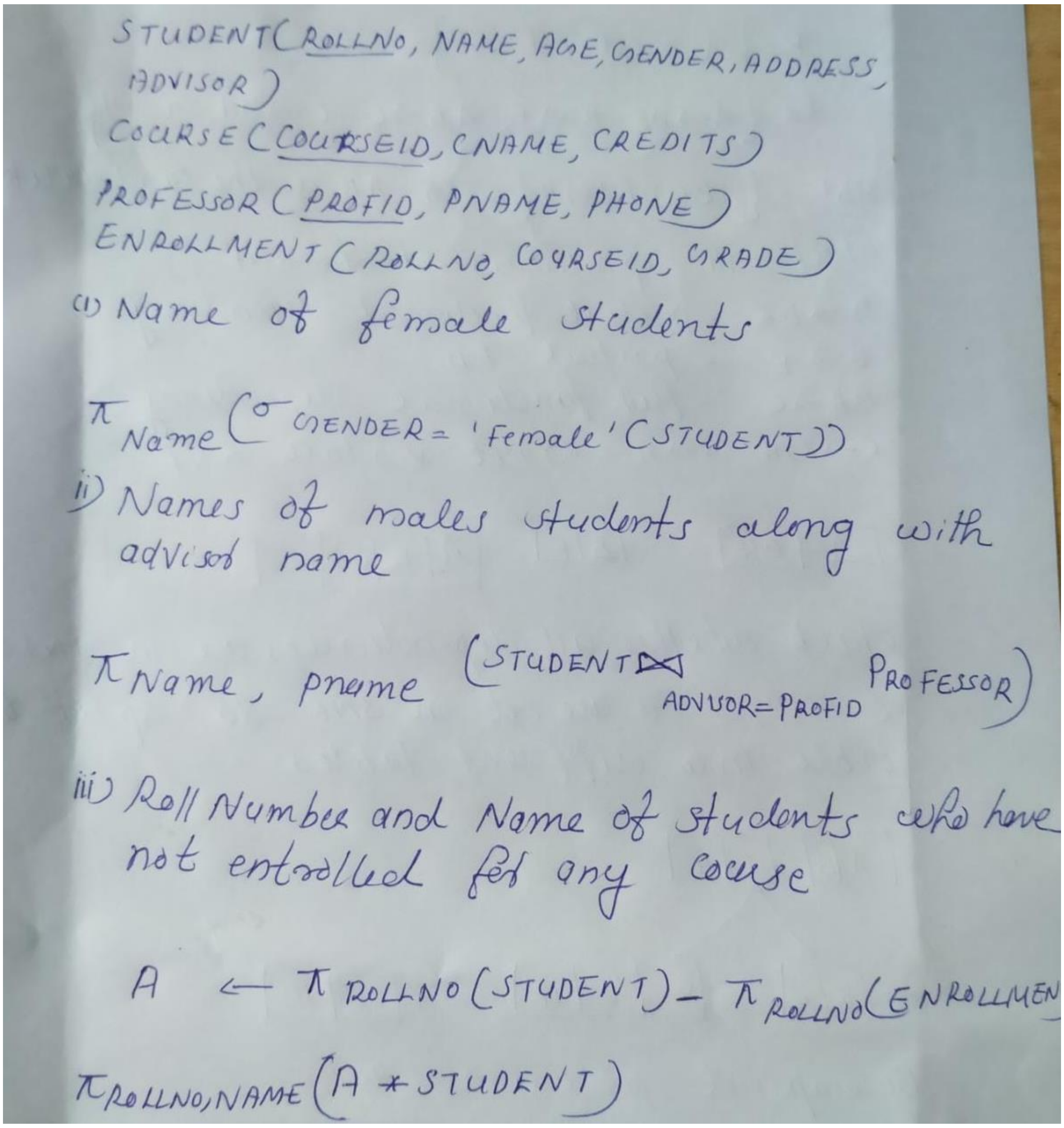
COURSE(<u>COURSEID</u>, CNAME, CREDITS)

PROFESSOR(PROFID,PNAME, PHONE)

ENROLLMENT(ROLLNO, COURSEID, GRADE)

Primary keys are underlined. ADVISOR is a foreign key referring to PROFESSOR table. ROLLNO and COURSEID in ENROLLMENT are also foreign keys referring to the primary keys with the same name.

- Names of female students
- Names of male students along with adviser name
- Roll Number and name of students who have not enrolled for any course.



STUDENT(ROLLNO, NAME, AGE, GENDER, ADDRESS, ADVISOR)

COURSE(COURSEID, CNAME, CREDITS)

PROFESSOR(PROFID, PNAME, PHONE)

ENROLLMENT(ROLLNO, COURSEID, GRADE)

(i) Name of female students

$\pi_{Name}(\sigma_{GENDER='Female'}(STUDENT))$

ii) Names of males students along with advisor name

$\pi_{Name,\ pname}(STUDENT \bowtie_{ADVISOR=PROFID} PROFESSOR)$

iii) Roll Number and Name of students who have not enrolled for any course

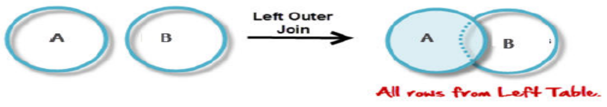
$A \leftarrow \pi_{ROLLNO}(STUDENT) - \pi_{ROLLNO}(ENROLLMEN$

$\pi_{ROLLNO,NAME}(A * STUDENT)$

13.b Explain the left outer join, right outer join, full outer join operations with examples

Left Outer Join (A ⟕ B)

- In the left outer join, operation allows keeping all tuple in the left relation.
- if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Student

S_id	Name	Class	Age	C_type
1	Andrew	5	25	A
2	Angel	10	30	A
3	Anamika	8	35	C

Course

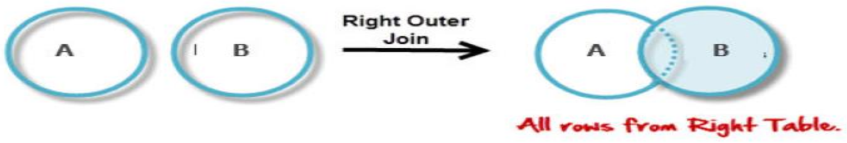
C_type	C_name
A	Foundation C
B	C++

Student ⟕ Course

S_id	Name	Class	Age	C_type	C_name
1	Andrew	5	25	A	Foundation C
2	Angel	10	30	A	Foundation C
3	Anamika	8	35	C	-

Right Outer Join (A ⟖ B)

- In the right outer join, operation allows keeping all tuple in the right relation.
- However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



Student

S_id	Name	Class	Age	C_type
1	Andrew	5	25	A
2	Angel	10	30	A
3	Anamika	8	35	C

Course

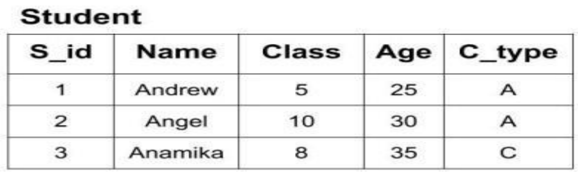
C_type	C_name
A	Foundation C
B	C++

Student ⟖ Course

S_id	Name	Class	Age	C_type	C_name
1	Andrew	5	25	A	Foundation C
2	Angel	10	30	A	Foundation C
-	-	-	-	B	C++

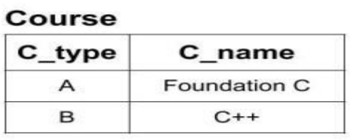
Full Outer Join (⟗)

- Full Outer Join is a type of join in which all the tuples from the left and right relation which are having the same value on the common attribute. Also, they will have all the remaining tuples which are not common on in both the relations.
- ***Notation: R1 ⟗R2*** where R1 and R2 are relations.
-



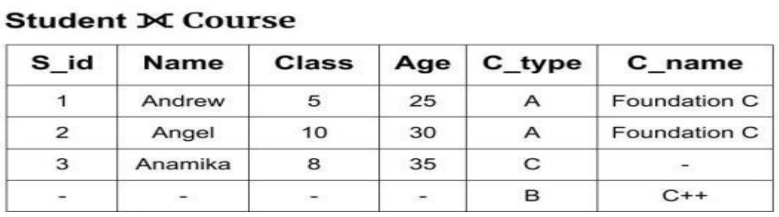
Student

S_id	Name	Class	Age	C_type
1	Andrew	5	25	A
2	Angel	10	30	A
3	Anamika	8	35	C



Course

C_type	C_name
A	Foundation C
B	C++



Student ⟗ Course

S_id	Name	Class	Age	C_type	C_name
1	Andrew	5	25	A	Foundation C
2	Angel	10	30	A	Foundation C
3	Anamika	8	35	C	-
-	-	-	-	B	C++

14 a. Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:

SALESPERSON(Ssn, Name, StartYear, DeptNo)

TRIP(Ssn, FromCity, ToCity, DepartureDate, ReturnDate, TripId)

EXPENSE(TripId, AccountNo, Amount)

- A trip can be charged to one or more accounts. Specify the foreign keys for this schema, stating any assumptions you make.
- Write relation algebra expression to get the details of salespersons who have travelled between Mumbai and Delhi and the travel expense is greater that Rs.50000.
- Write relation algebra expression to get the details of salesperson who had incurred the greatest travel expenses among all travels made.

Foreign Key

Ssn in TRIP relation is the foreign key of that relation.

TripId in EXPENSE is the foreign key of that relation.

Write relation algebra expression to get the details of salespersons who have travelled between Mumbai and Delhi and the travel expense is greater that Rs.50000.

$\Pi_{\text{ssn, Name,Startyear,DeptNo}}(\sigma_{\text{FromCity='Mumbai' AND ToCity='Delhi' AND Amount=50000}}(\text{SALESPERSON*TRIP*EXPENSE}))$

Write relation algebra expression to get the details of salesperson who had incurred the greatest travel expenses among all travels made.

$\Pi_{\text{ssn, Name,Startyear,DeptNo}}(\sigma_{\text{ Maximun(Amount)}}(\text{SALESPERSON*TRIP*EXPENSE}))$

b. List the basic data types available for defining attributes in SQL?

a)CHAR(n) -Character string, fixed length n

b) VARCHAR(n) - Variable length character string, maximum length n.

c) NUMBER(p) - Integer numbers with precision p.

d) BOOLEAN - Boolean variable used to store TRUE, FALSE, NULL

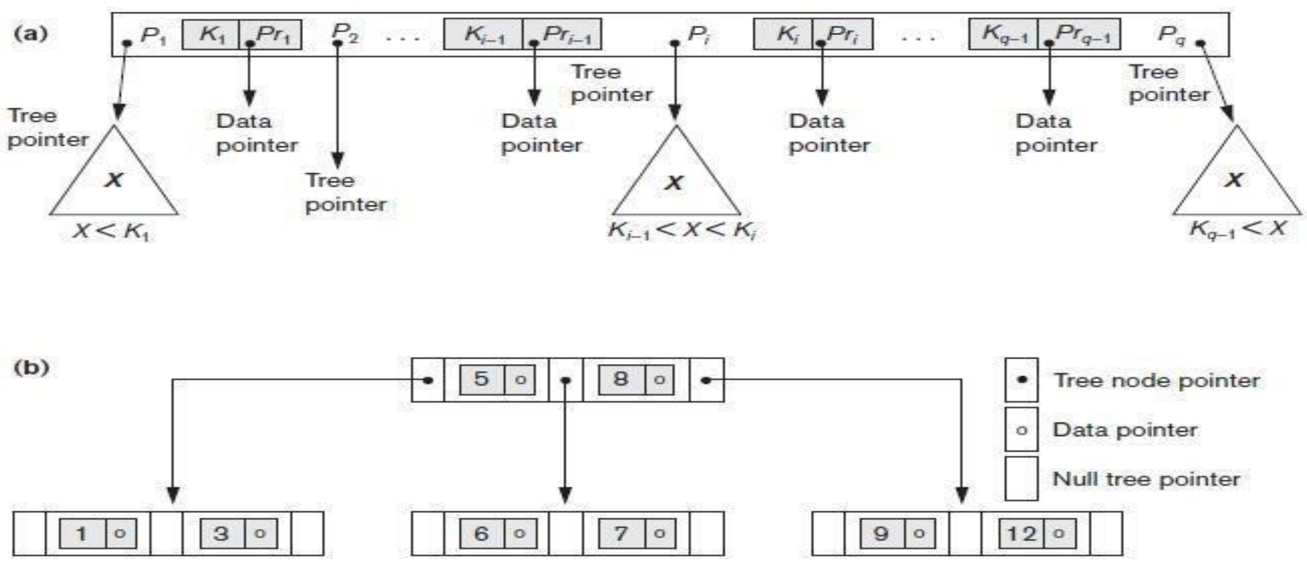
e) DATE - Stores year, month, day, hour, minute and second values

f) BLOB - Binary Large Object (the data type in Oracle for storing binary files like executables, images etc.)

15 a. Illustrate structure of B-Tree and B+ Tree and explain how they are different?

The B-tree has additional constraints that ensurethat the tree is always balanced.
A B-tree of order p, when used as an access structureon a key field to search for records in a data file, can be defined as follows:

1. Each internal node in the B-tree is of theform
   - <P1, <K1, Pr1>, P2, <K2, Pr2>, ..., <Kq–1, Prq–1>,
   - Pq> where q ≤ p. Each Pi is a tree pointer—apointer to another node in the Btree. Each Pri is a data pointer—a pointer to the record whose search key field value is equal to Ki.
2. Within each node, K1 < K2 < ... < Kq−1.
3. For all search key field values X in the subtree pointed at by Pi, we have: Ki–1 < X < Ki for 1 < i < q; X < Ki for i= 1; and Ki–1 < X for i = q.
4. Each node has at most p tree pointers.
5. Each node, except the root and leaf nodes, has at least ⌈(p/2)⌉ tree pointers. The root node has at least two tree pointers unless it is the only node in the tree.
6. A node with q tree pointers, q ≤ p, has q – 1 search key field values (and hence has q – 1 data pointers).
7. All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes except that all of their tree pointers Pi are NULL



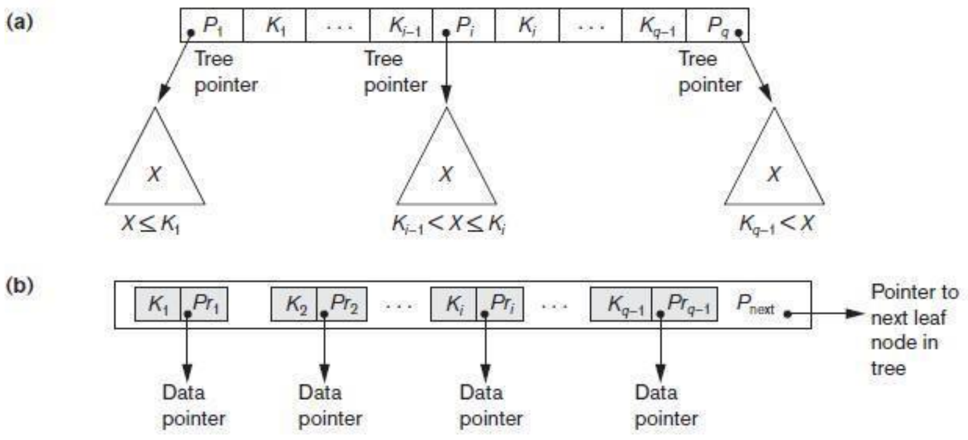
- The structure of the internal nodes of a B+ tree of order p is as follows:

Each internal node is of the form <P1, K1, P2, K2, ..., Pq– 1, Kq –1, Pq> where q ≤ p and

each Pi is a tree pointer.

2. Within each internal node, K1 < K2 < ... < Kq−1.
3. For all search field values X in the subtree pointed atby Pi, we have Ki−1 < X≤ Ki for 1 < i < q; X ≤ Ki for i = 1; and Ki−1 < X for i = q.
4. Each internal node has at most p tree pointers.
5. Each internal node, except the root, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers if it is an internal node.
6. An internal node with q pointers, q ≤ p, has q – 1 search field values.

- The structure of the leaf nodes of a B+ tree of order p is as follows:
  1. Each leaf node is of the form <<K1, Pr1>, <K2, Pr2>, ..., <Kq−1, Prq−1>, Pnext> where q ≤ p, each Pri is a data pointer, and Pnext points to the next leaf node of the B+-tree.
  2. Within each leaf node, K1 ≤ K2 ... , Kq−1, q ≤ p.
  3. Each Pri is a data pointer that points to the record whose search field value is Ki or to a file block containing the record
  4. Each leaf node has at least $\lceil (p/2) \rceil$ values.
  5. All leaf nodes are at the same level.


(a)
P_1 K_1 ... K_{i-1} P_i K_i ... K_{q-1} P_q
Tree pointer
Tree pointer
Tree pointer
X
X
X
$X \le K_1$
$K_{i-1} < X \le K_i$
$K_{q-1} < X$
(b)
K_1 Pr_1 K_2 Pr_2 ... K_i Pr_i ... K_{q-1} Pr_{q-1} P_{next}
Pointer to next leaf node in tree
Data pointer
Data pointer
Data pointer
Data pointer

Difference between B tree and B+ tree

- In B+trees, search keys can be repeated but this is not the case forB-trees
- B+trees allow satellite data to be stored in leaf nodes only, whereas B-trees store data in both leaf and internal nodes

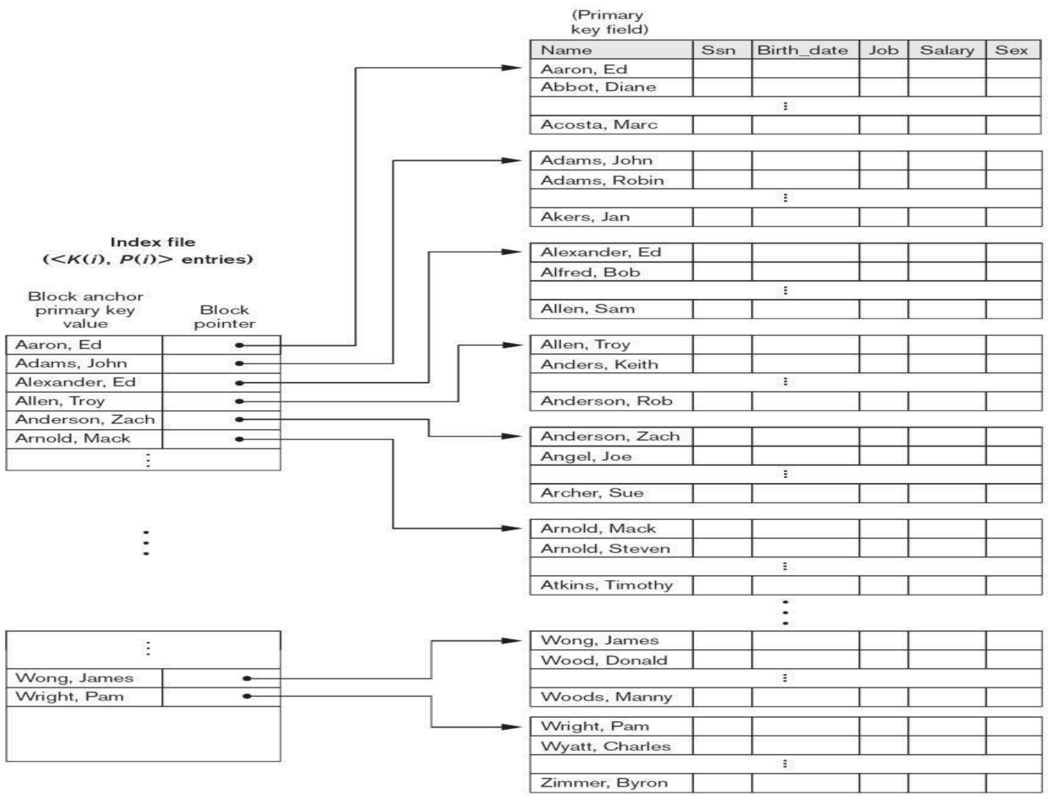
- In B+trees, data stored on the leaf node makes the search moreefficient since we can store more keys in internal nodes
- this means we need to access fewer nodes
- Deleting data from a B+tree is easier and less time consuming because we only need to remove data from leaf nodes
- Leaf nodes in a B+tree are linked together making range search operations efficient and quick
- Finally, although B-trees are useful, B+trees are more popular. In fact, 99% of database management systems use B+trees for indexing.
- This is because the B+tree holds no data in the internal nodes.
- This maximizes the number of keys stored in a node thereby minimizing the number of levels needed in a tree.
- Smaller tree depth invariably means faster search.

15.b What are the different types of single-level ordered indices? Explain.

Primary Indexes

- Primary index is defined on an ordered data file whose records are of fixed length with two fields.
- The first field is of the same data type as the ordering key field(primary key) of the data file.
- The second field is a pointer to a disk block (a block address).
- There is one index entry (or index record) in the index file for each block in the data file.
- The two field values of index entry i are referred as

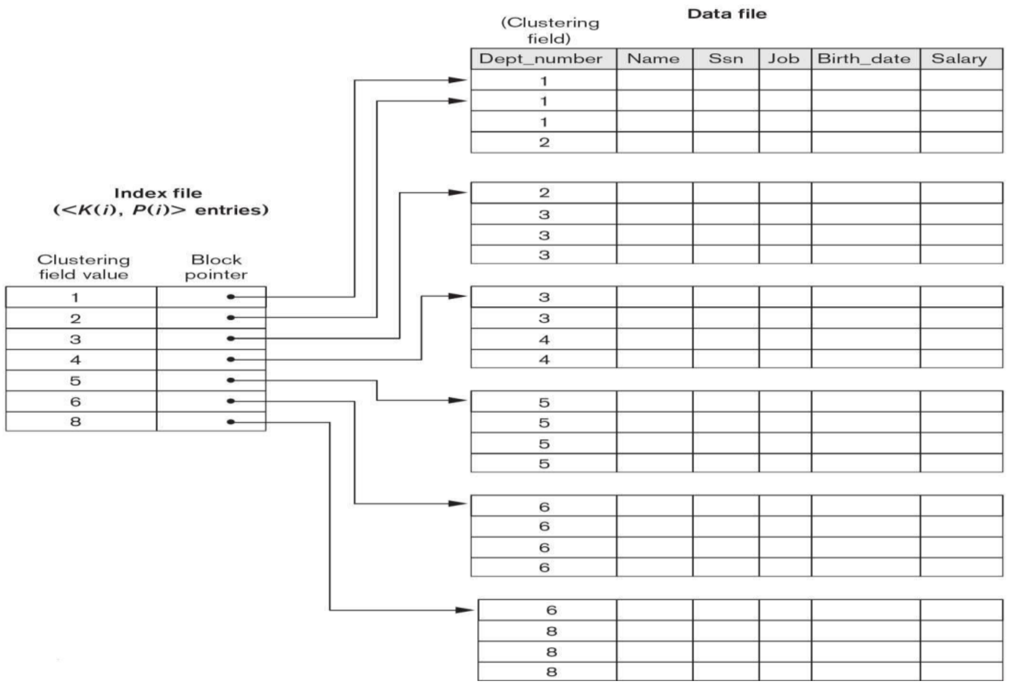
<K(i),P(i)>



- 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.
- Thus index file for primary indexing occupies a much smaller space than data file for two reasons:
    1. There are fewer index entries than the records in the data file.
    2. Each index entry is smaller in size than a data record because it has only two fields. So more index entries can fit in one block than data records.

Clustering Index

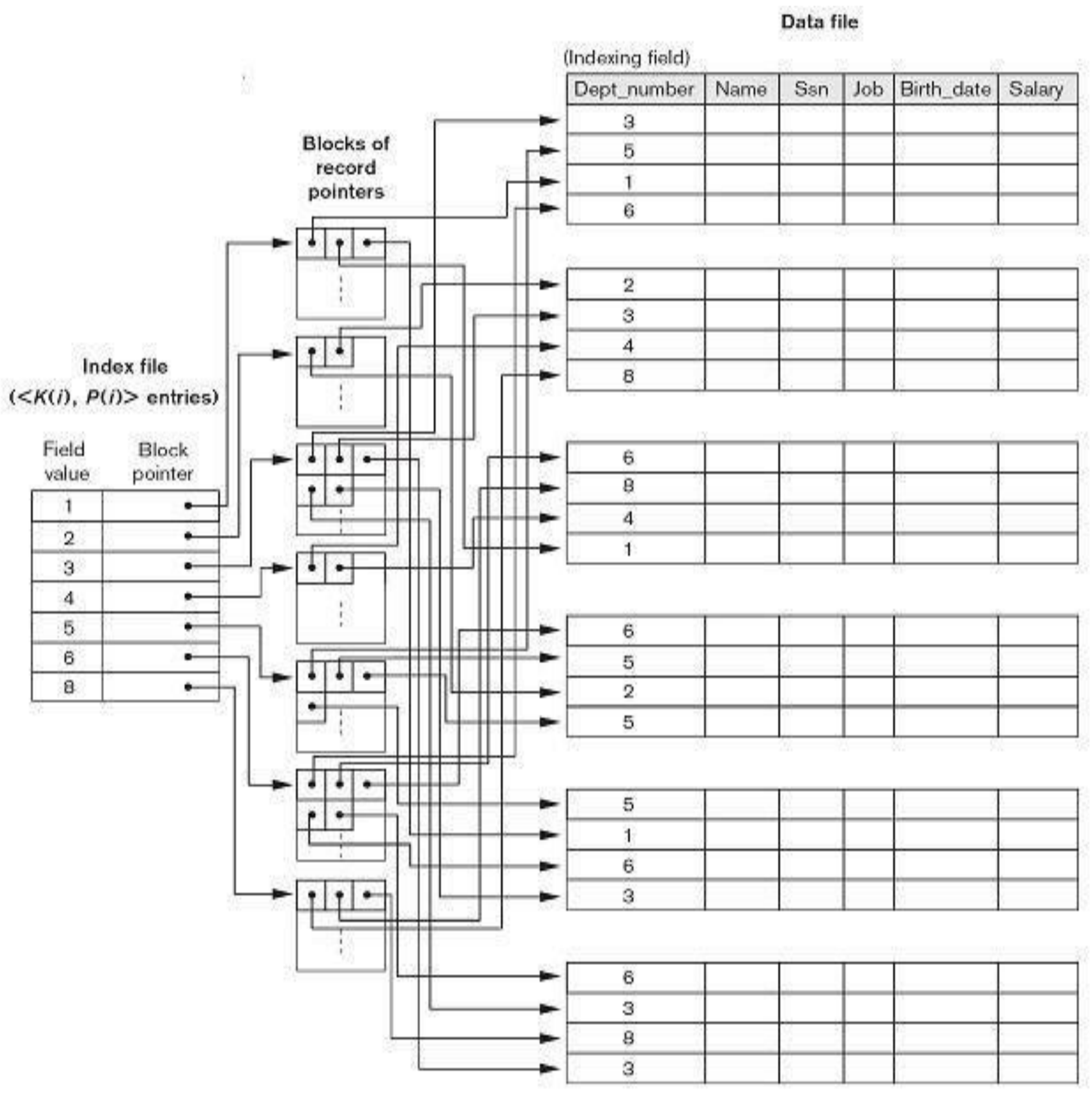
- Defined on an ordered data file.
- The data file is ordered on a non-key field unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
- Includes one index entry for each distinct value of the field;
- The index entry points to the first data block that contains records with that field value.
- It is another example of nondense index where Insertion and Deletion is relatively straightforward with a clustering index.

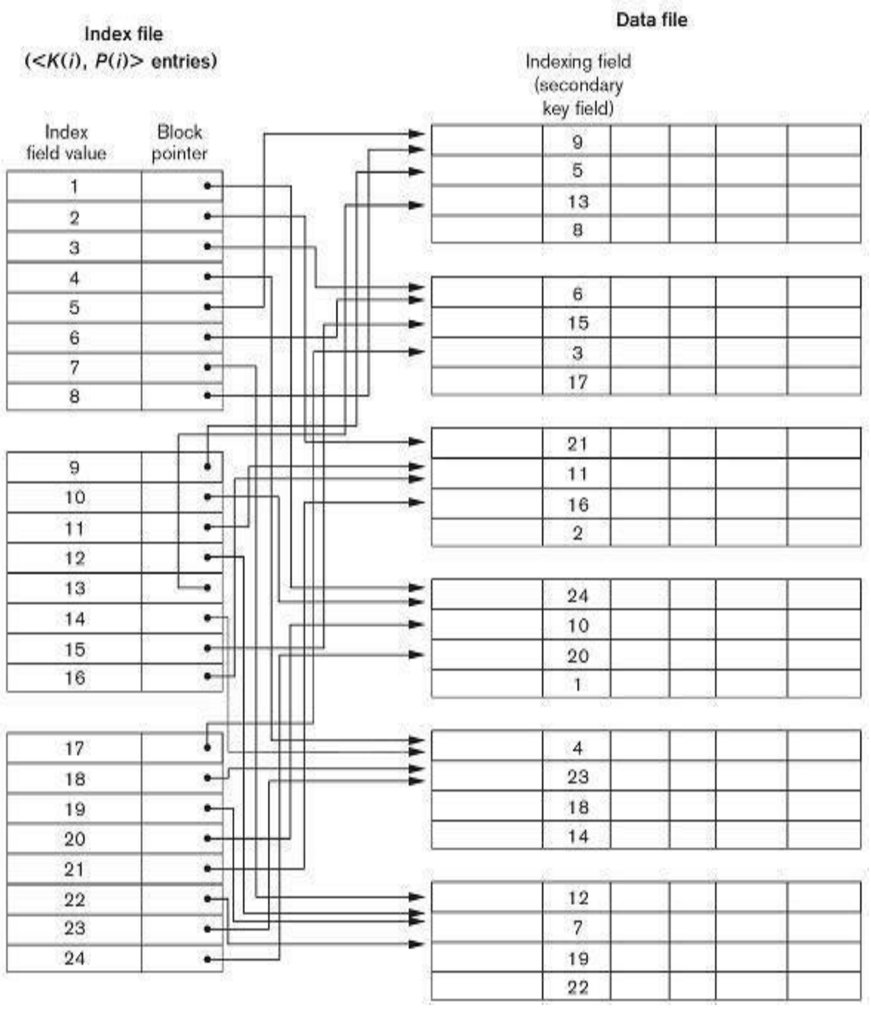


Secondary Index

- 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.
- Includes one entry for each record in the data file; hence, it is a dense index.





16. a Differentiate between static hashing and dynamic hashing.

Static Hashing

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values(ie. 0,1,2,3,4). The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.

Operations

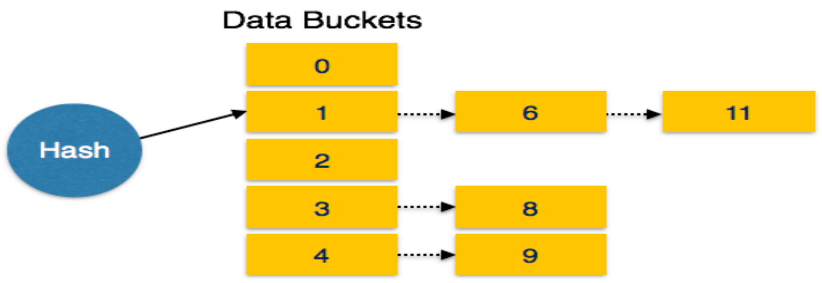
Insertion − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.

Bucket address = h(K)

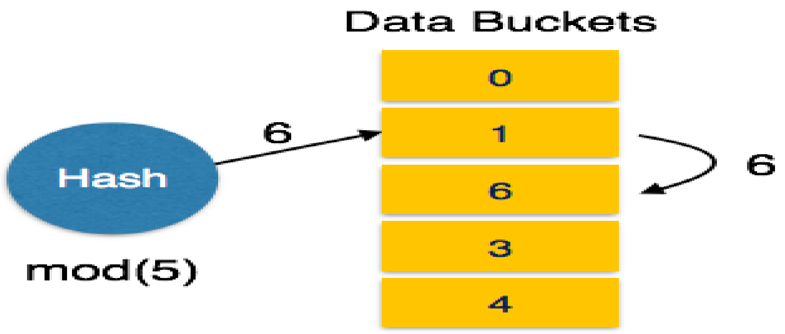
Search − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

Delete − This is simply a search followed by a deletion operation.

Bucket Overflow
The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.
Overflow Chaining – When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.



Linear Probing – When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.

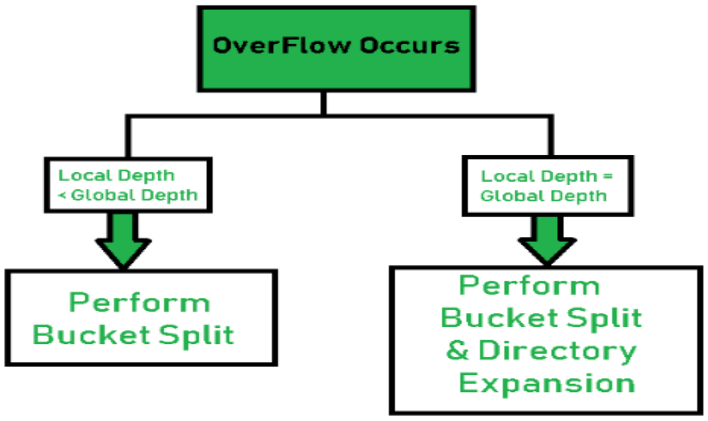


The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. **Dynamic hashing** is also known as **Extendible Hashing**.

Main features of Extendible Hashing: The main features in this hashing technique are:
Directories: The directories store addresses of the buckets in pointers. An id is assigned to each directory which may change each time when Directory Expansion takes place.
Buckets: They store the hashed keys. Directories point to buckets. A bucket may contain more than one pointers to it if its local depth is less than the global depth.

Global Depth: It is associated with the Directories. They denote the number of bits which are used by the hash function to categorize the keys. Global Depth = Number of bits in directory id.
Local Depth: It is the same as that of Global Depth except for the fact that Local Depth is associated with the buckets and not the directories. Local depth in accordance with the global depth is used to decide the action that to be performed in case an overflow occurs. Local Depth is always less than or equal to the Global Depth.
Bucket Splitting: When the number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.

Directory Expansion: Directory Expansion Takes place when a bucket overflows. Directory Expansion is performed when the local depth of the overflowing bucket **is equal to** the global depth.


OverFlow Occurs
Local Depth < Global Depth
Local Depth = Global Depth
Perform Bucket Split
Perform Bucket Split & Directory Expansion

16.b Write a short note on nested queries

In **nested queries**, a query is written inside a query. The result of inner query is used in execution of outer query.

In nested queries, the comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V.

In addition to 'IN' operator, all other comparison operators such as (>,>=,<,<= and < >) can be combined with the keywords ANY or ALL.

Correlated Nested Aueries

Whenever a condition in the WHERE-clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be **correlated**

Eg: Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
Q16:  SELECT  E.FNAME, E.LNAME
      FROM    EMPLOYEE AS E
      WHERE   E.SSN IN    (SELECT  ESSN
                           FROM    DEPENDENT
                           WHERE   E.FNAME=
                                   DEPENDENT_NAME AND
                                   E.SEX=SEX);
```

Non correlated nested queries

In **non-correlated query** inner **query** does not dependent on the outer **query**

Eg: Retrieve the name and address of all employees who work for the 'Research' Department

SELECT FNAME,LNAME, ADDRESS FROM EMPLOYEE WHERE DNO IN (SELECT DNUMBER FROM DEPARTMENT WHERE DNAME='Research')

17. a. i. What are Amstrong's axioms

The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of functional dependencies

If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F.

Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- Axiom of reflexivity

  If A is a set of attributes and B is subset of A, then A holds B. If $B \subseteq A$ then $A \rightarrow B$

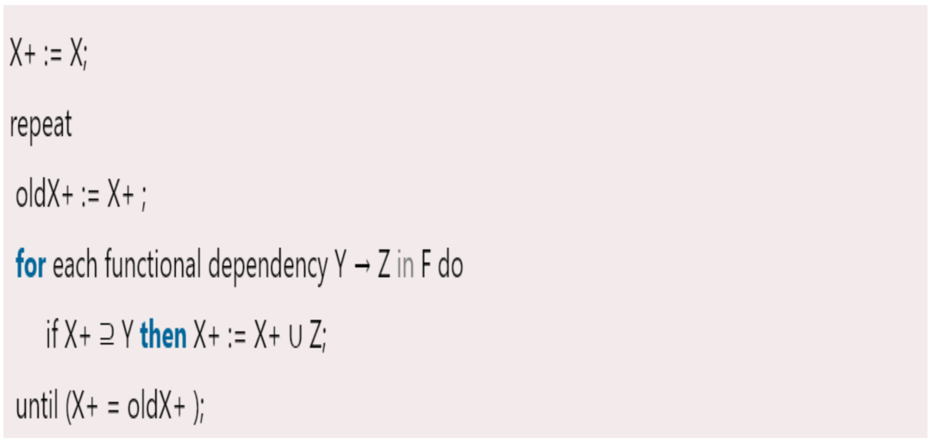
  This property is trivial property.
- Axiom of augmentation

  If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds.
- Axiom of transitivity

  Same as the transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds.

ii. Write an algorithm to compute the attribute closure of a set of attributes (X)under a set of functional dependencies(F)

Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.



```
X+ := X;
repeat
 oldX+ := X+ ;
 for each functional dependency Y → Z in F do
    if X+ ⊇ Y then X+ := X+ ∪ Z;
 until (X+ = oldX+ );
```

iii. Explain three uses of attribute closure algorithm.

- Used to check whether an attribute (or set of attributes) forms a key or not. ...
- Used to find easily all the functional dependencies hold in a relation. ...
- Used as an alternate way to find Closure of FDs (F^+).

17.b Explain the difference between BCNF and 3NF with an example

A relation is in the third normal form if there is no transitive dependency for non-prime attributes as well as it is in the second normal form.

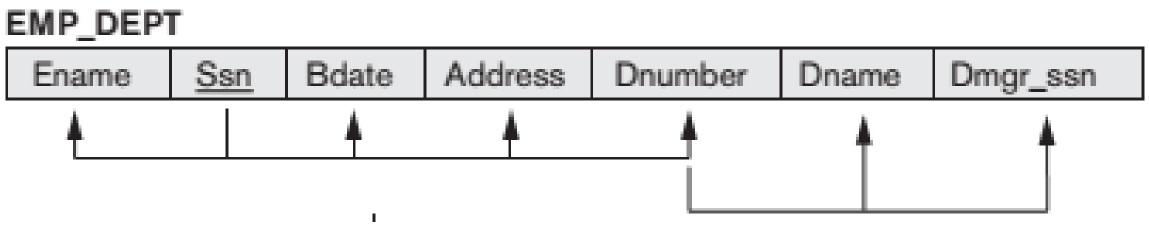
Two Conditions for 3NF

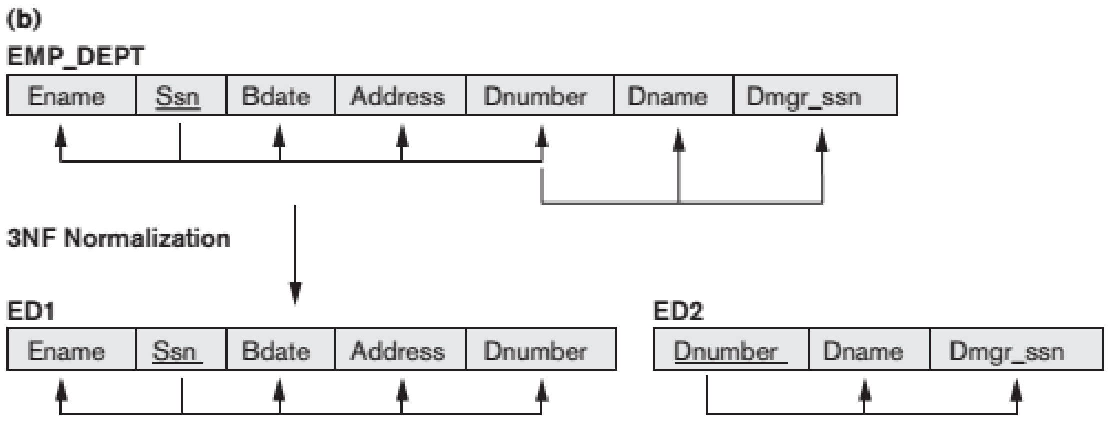
- It should be in 2NF
- No transitive Dependency

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

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

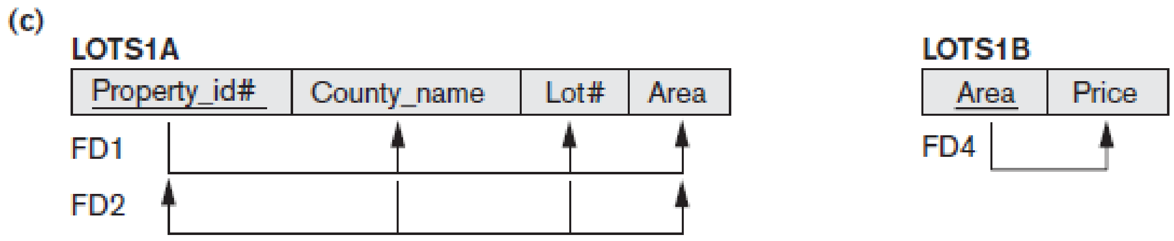
- The dependency Ssn → Dmgr_ssn is transitive through Dnumber in EMP_DEPT, because both the dependencies Ssn → Dnumber and Dnumber → Dmgr_ssn hold *and* Dnumber is neither a key itself nor a subset of the key of EMP_DEPT.





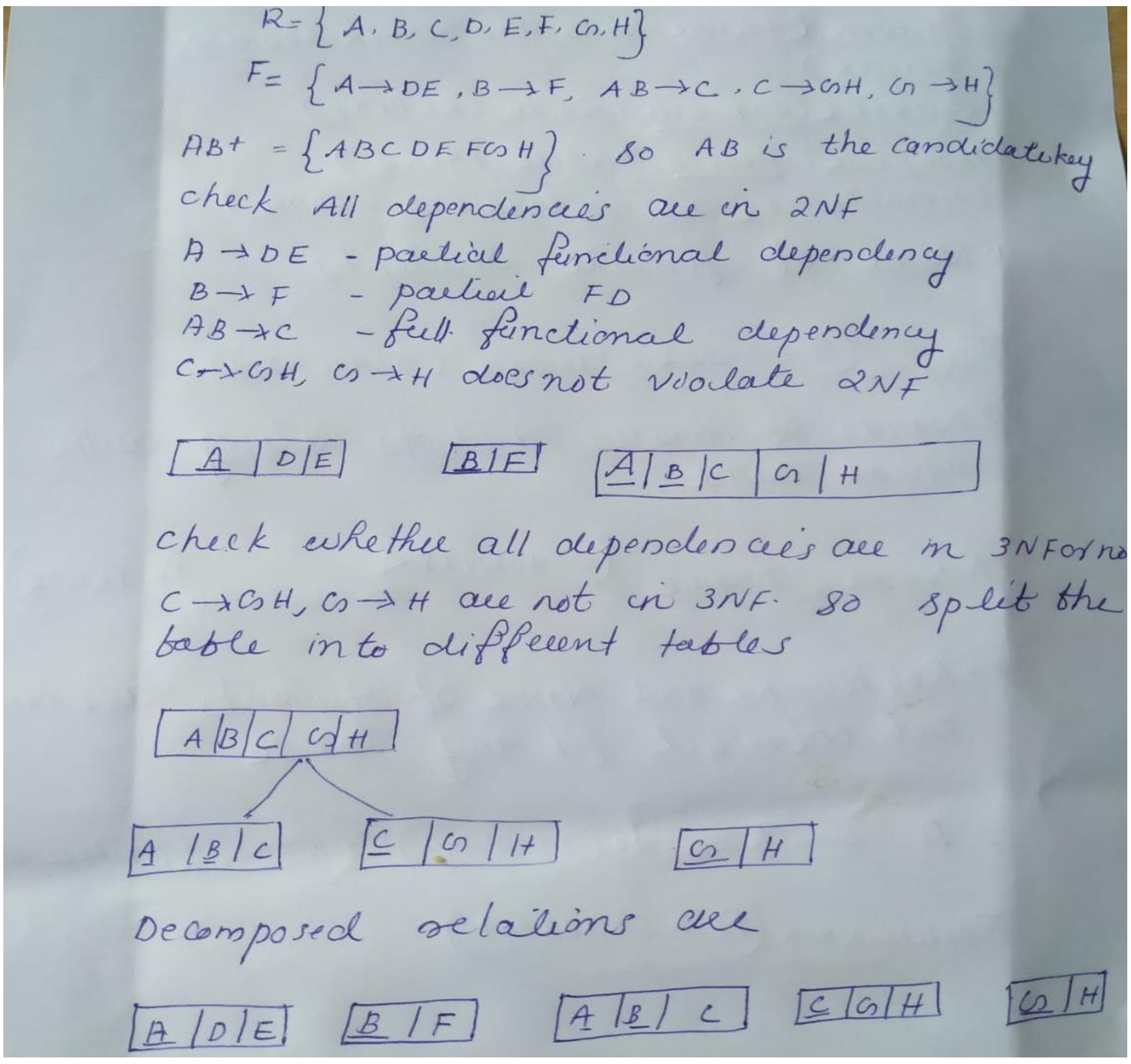
BCNF

- A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.
- That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF



-

18. a Consider the relation R = {A, B, C, D, E, F, G, H} and the set of functional dependencies F = {A→DE, B→F, AB→C, C→GH, G→H}. What is the key for R? Decompose R into 2NF and then 3NF relations.



R = {A, B, C, D, E, F, G, H}

F = {A→DE, B→F, AB→C, C→GH, G→H}

AB+ = {ABCDEFGH} so AB is the candidate key

check All dependencies are in 2NF

A→DE - partial functional dependency

B→F - partial FD

AB→C - full functional dependency

C→GH, G→H does not violate 2NF

A	D	E

B	F

A	B	C	G	H

check whether all dependencies are in 3NF or no

C→GH, G→H are not in 3NF. so split the table into different tables

A	B	C	G	H

A	B	C

C	G	H

G	H

Decomposed relations are

A	D	E

B	F

A	B	C

C	G	H

G	H

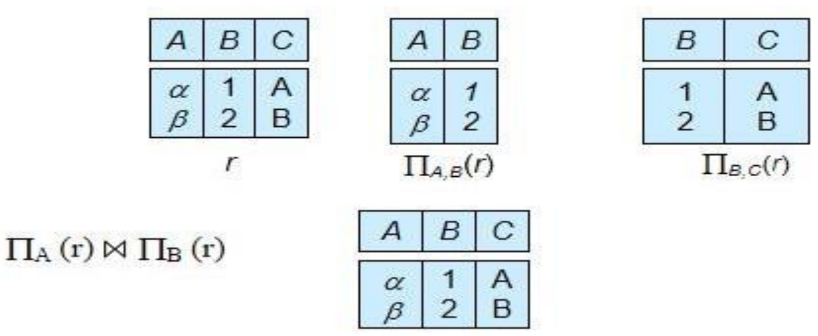
18.b What is the lossless join property of decomposition? Why is it important?

A decomposition D = {R1, R2, ..., Rm} of R has the lossless(nonadditive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all the relations in D: *(πR1(r), ..., πRm(r)) = r.

The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition

Decomposition of *R = (A, B, C)* R_1 *= (A, B)* R_2 *= (B, C)*

After PROJECT (π) and NATURAL JOIN (*) operations are applied on R1 and R2, no spurious tuples are there in the result.



A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B	C
α	1	A
β	2	B

19. Explain the concepts behind the following

i) Log based Recovery

- Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of failure.
- To do this, the system must keep information about the changes that were applied to data items by the various transactions.
- This information is typically kept in the system log.
- Log is a sequence of records, which maintains the records of actions performed by a transaction.

- It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.
  - The log file is kept on a stable storage media.
  - When a transaction enters the system and starts execution, it writes a log about it as <Tn, Start>.
  - When the transaction modifies an item X, it write logs as follows:

    <Tn,X,V1,V2 >

    It reads Tn has changed the value of X, from V1 to V2

    When the transaction finishes, it logs : <Tn, commit>

- After a <u>system crash has occurred</u>, the <u>system consults the log</u> to determine which transactions need to be redone, and which need to be undone so as to ensure atomicity.
  - Transaction Ti needs to be undone if the log contains the record <Ti,start>, but does not contain either the record <Ti, commit> or the record <Ti,abort>.
  - Transaction Ti needs to be redone if the log contains the record <Ti,start> and either the record <Ti, commit> or the record <Ti,abort>.

ii)Deferred Database Modification

Deferred database modification – All logs are written on to the stable storage and the database is updated when a transaction commits.

The deferred modification techniques do not physically update the database on disk until after a transaction reaches its commit point ; then the updates are recorded in the database.

- A typical deferred update protocol can be stated as follows:

i. A transaction cannot change the database on disk until it reaches its commit point.

ii. A transaction does not reach its commit point until all its REDO- type log entries are recorded in the log and the log buffer is force-written to disk.

- REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the database on disk. In this case, the transaction operations are redone from the log entries during recovery.
- The deferred database modification scheme records all modifications to the log, but defers all the write to after partial commit.

19.b Why recovery is needed in transaction processing

Recovery mechanisms can help roll back or undo the effects of such failed transactions to ensure data consistency

- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that
  - Either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, **or**
  - That the transaction does not have any effect on the database or any other transactions.

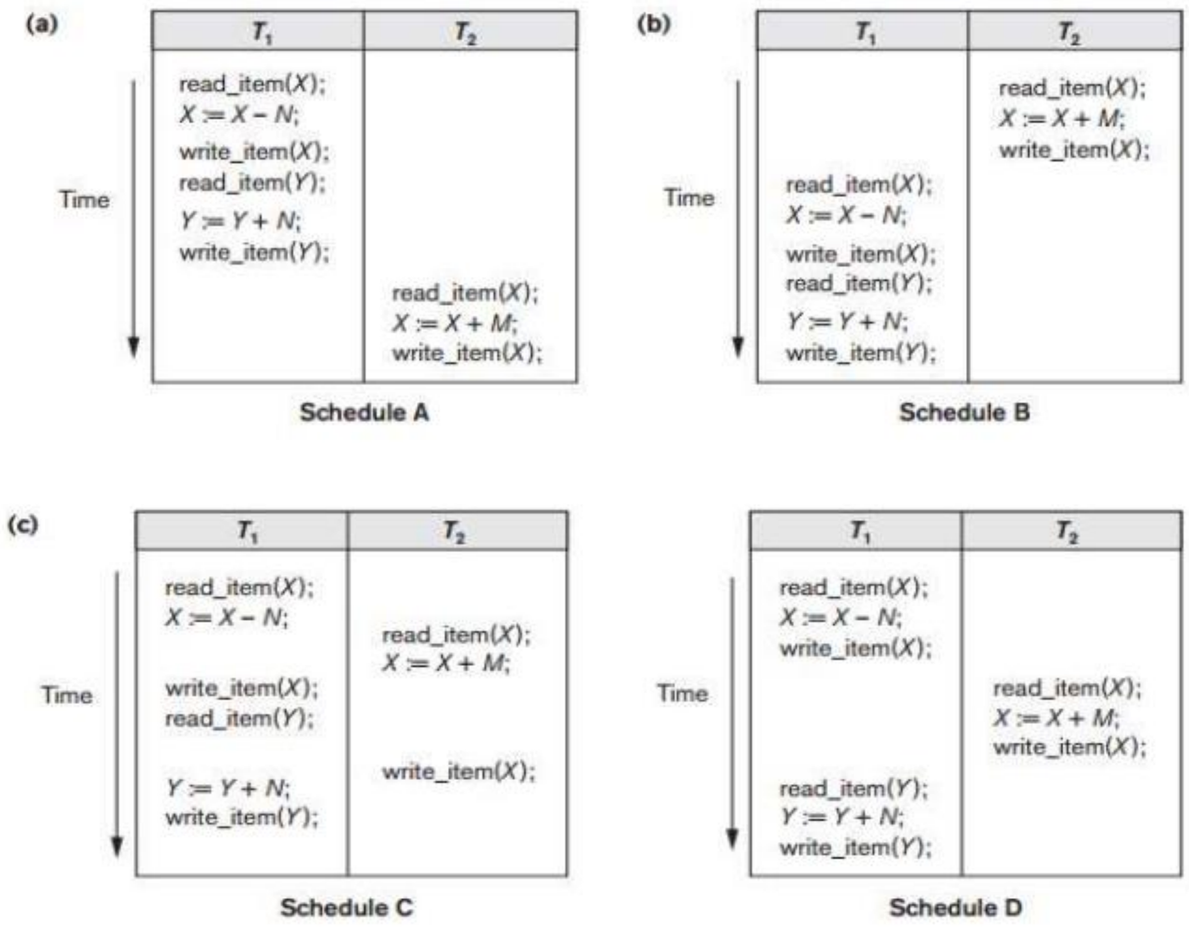
- In the first case, the transaction is said to be **committed**, whereas in the second case, the transaction is **aborted**.
- If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.
-

20.a Differentiate serial and concurrent schedules. Elaborate conflict serializability with suitable example.

Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.
Concurrent Schedule
If interleaving of operations is allowed, then there will be non-serial schedule.
It contains many possible orders in which the system can execute the individual operations of the transactions.

Figure 21.5
Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.


(a)
Time
(b)
Time
(c)
Time
Time

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Schedule A

T_1	T_2

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule B

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

write_item(X);
read_item(Y);	

	write_item(X);
$Y := Y + N$; write_item(Y);	

Schedule C

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

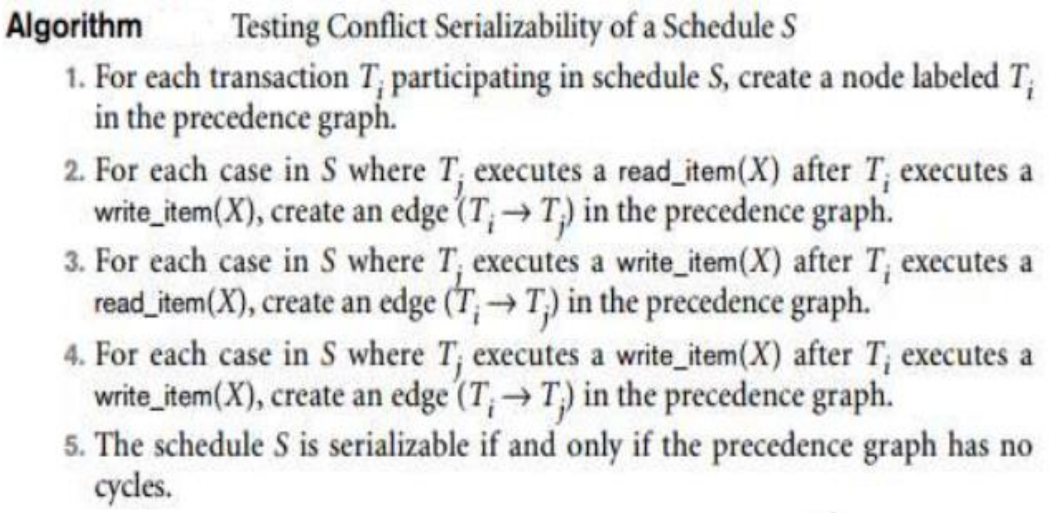
	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);
$Y := Y + N$;
write_item(Y);	

Schedule D

conflict serializable

A schedule S to be **conflict serializable** if it is (conflict) equivalent to some serial schedule S.



Algorithm Testing Conflict Serializability of a Schedule S

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

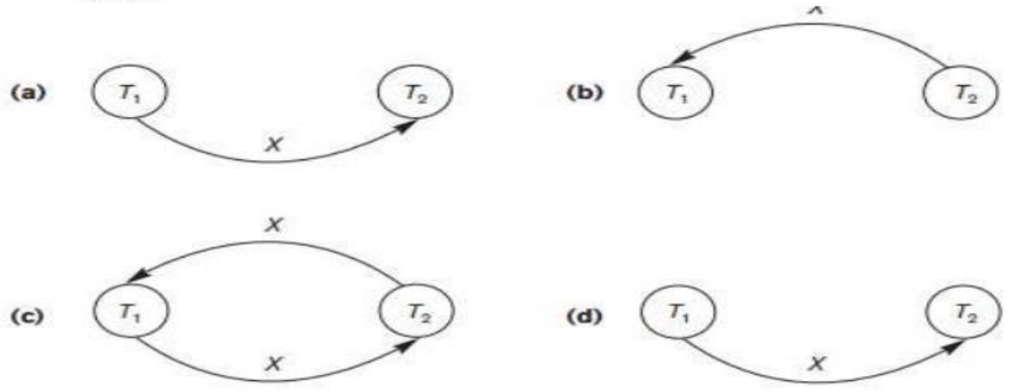


Figure
Constructing the precedence graphs for schedules A to D from Figure 21.5 to test for conflict serializability. (a) Precedence graph for serial schedule A. (b) Precedence graph for serial schedule B. (c) Precedence graph for schedule C (not serializable). (d) Precedence graph for schedule D (serializable, equivalent to schedule A).

20 b. What are the desirable properties of transactions? Explain

- Transactions should possess several properties, often called the ACID properties
- They should be enforced by the concurrency control and recovery methods of the DBMS.
- The following are the ACID properties:
  1. Atomicity
  2. Consistency
  3. Isolation
  4. Durability

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

- All or Nothing

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

Isolation: A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.

- That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

Durability or permanency: The changes applied to the database by a committed transaction must persist in the database.

- These changes must not be lost because of any failure.