14CS IT 504

Hall Ticket Number:

III/IV B.Tech (Regular) DEGREE EXAMINATION

November, 2016 **Common to CSE & IT**

Fifth Semester **Database Management Systems**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)

a What is a relational model?

b Write any three advantages of using the DBMS.

c What is the use of Structural Constraints?

d What is the difference between a database schema and a database state?

e What are the Unary Relational Operations?

f What is meant by correlated queries?

g What is an entity type? What is an entity set?

h What is meant by a recursive relationship type?

i What is Normalization? Define 1NF .

j List the desirable properties.

k Define Serializability?

l What is meant by Granularity?

UNIT – I

2.a Discuss the main characteristics of the Database Approach and how it differs from traditional file systems. 6M

2.b Construct ER Diagram for a Banking Enterprise. Identify entities, roles, weak entity sets if any, IS A relationship if any 6M

(OR)

3.a Explain Conceptual database design with ER Model. 6M

3.b Describe in detail about 3-Schema Architecture. 6M

UNIT – II

4 Discuss about SELECT & JOIN, JOIN & DIVISION with examples 12M

(OR)

5 Consider the following schema: 12M

Suppliers(*sid:* integer, *sname:* string, *address:* string) Parts(pid: integer, *pname:* string, *color:* string) Catalog(*sid:* integer, *pid:* integer, *cost:* real)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers.

Write the following queries in tuple relational calculus.

i. Find the *names* of suppliers who supply some red part
ii. Find the *sids* of suppliers who supply some red or green part.
iii. Find the *sids* of suppliers who supply some red part or are at 221 Packer Ave
iv. Find the *sids* of suppliers who supply some red part and some green part
v. Find the *sids* of suppliers who supply every red or green part

UNIT – III

6 Describe about Types of Indexes and how they use B-Trees & B+ Trees 12M

(OR)

7.a Differentiate Ordered and Unordered records in files. 6M

7.b Analyze about Normalization and describe briefly about 1NF, 2NF, 3NF. 6M

UNIT – IV

8 Demonstrate Two-phase locking techniques in concurrency control and how time stamp ordering will be used in 2PL. 12M

(OR)

9.a How do you characterize schedules based on Serializability. 8M

9.b Explain about Multiple Granularity Locking mechanism. 4M

14CS IT 504

Hall Ticket Number:

III/IV B.Tech (Regular) DEGREE EXAMINATION

November, 2016 **Common to CSE & IT**

Fifth Semester **Database Management Systems**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)

a **What is a relational model?**

The relational model uses a collection of tables to represent both data and the relationships among those data. The relational model is an example of a record based model.

b **Write any three advantages of using the DBMS.**

The advantages of using a DBMS are **[For Any 3 points 1 mark awarded]**

a) Controlling redundancy
b) Restricting unauthorized access
c) Providing multiple user interfaces
d) Enforcing integrity constraints.
e) Providing back up and recovery

c **What is the use of Structural Constraints?**

Structural Constraints are applicable for binary relationships.

d **What is the difference between a database schema and a database state?**

The data in the database at a particular moment in time is called a **database State**. The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.

e **What are the Unary Relational Operations?**

(i) Select (ii) Project (iii) Rename

f **What is meant by correlated queries?**

correlated subquery (also known as a synchronized subquery) is a subquery (a query nested inside another query) that uses values from the outer query. Because the subquery is evaluated once for each row processed by the outer query .This causes correlated subqueries to be less efficient than other subqueries.

g **What is an entity type? What is an entity set?**

Entity type: An entity type defines a collection of entities that have the same attributes.
Entity set: The set of all entities of the same type is termed as an entity set.

h **What is meant by a recursive relationship type?**

If the same entity type participate more than once in a relationship type in different roles then such relationship types are called recursive relationship.

OR

A recursive relationship is a relationship between an entity and itself.

i **What is Normalization? Define 1NF** .

Normalization: It is a process of analyzing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties

- Minimizing redundancy
- Minimizing insertion, deletion and updating anomalies.

1NF: The domain of attribute must include only atomic (simple, indivisible) values.

j **List the desirable properties**.

The properties of transactions are:

- Atomicity
- Consistency
- Isolation
- Durability
-

k **Define Serializability?**

Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operations.

l **What is meant by Granularity?**

The size of data items is often called the data item granularity. Fine granularity refers to small item sizes, whereas coarse granularity refers to large item sizes.

UNIT – I

2.a Discuss the main characteristics of the Database Approach and how it differs from traditional file systems. 6M

The main characteristics of the database approach versus the file-processing approach are the following:

■ **Self-describing nature of a database system**
■ **Insulation between programs and data, and data abstraction**
■ **Support of multiple views of the data**
■ **Sharing of data and multiuser transaction processing**

Listing the characteristics ---2 M
Writing any relevant points ---4 M

Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

Insulation between Programs and Data and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence. In some types of database systems, such as object-oriented and object-relational systems , users can define operations on data as part of the database definitions. An operation is specified in two parts. The interface of an operation includes the operation name and the data types of its arguments. The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

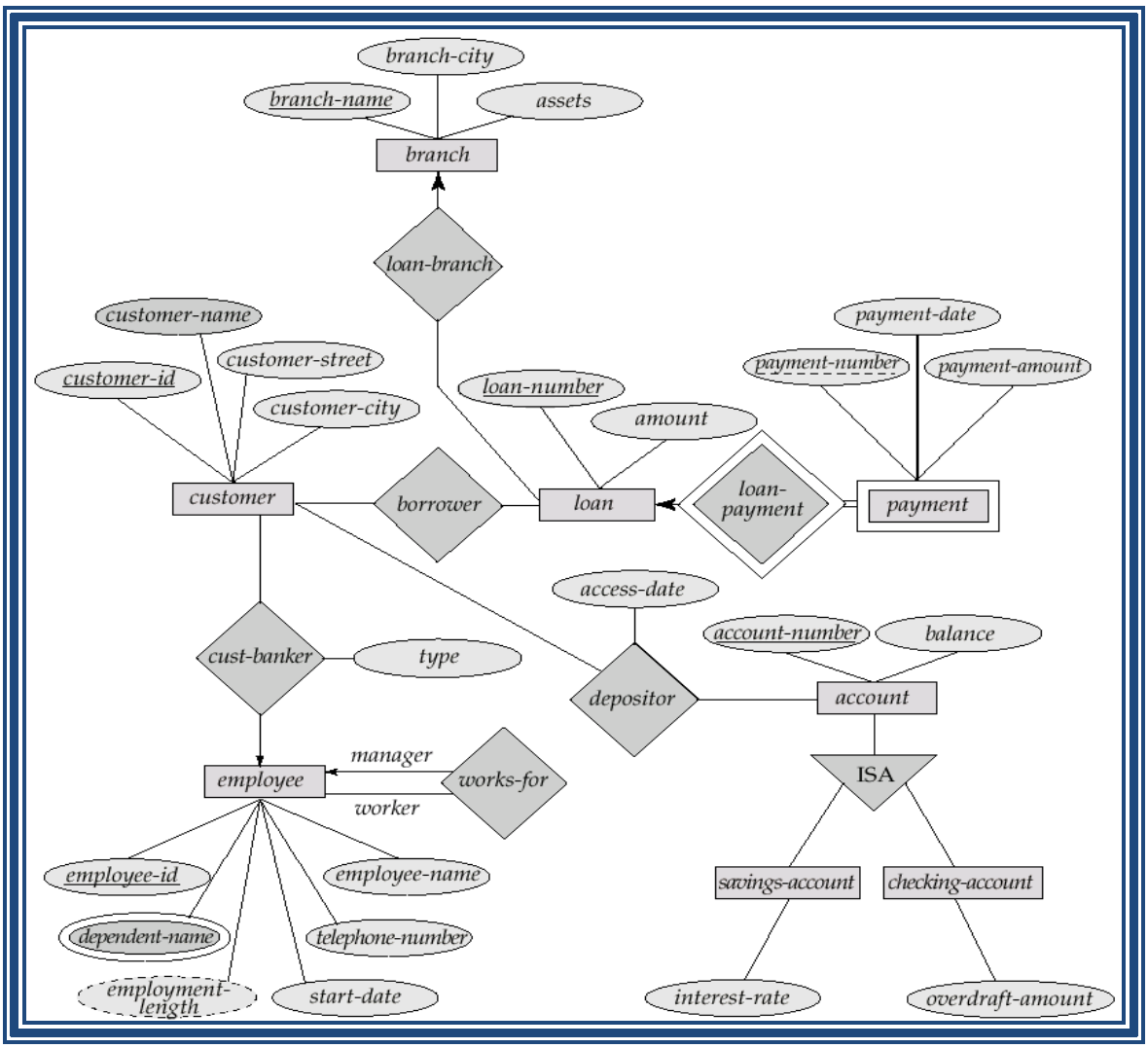
Support of Multiple Views of the Data

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

Sharing of Data and Multiuser Transaction Processing

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

2.b Construct ER Diagram for a Banking Enterprise. Identify entities, roles, weak entity sets if any, IS A relationship if any 6M



STEP 1: *Identifying the entities*

ENTITIES

- ***Branch***
- ***Customer***
- ***Employee***
- ***Account***
- ***Loan***
- ***Payment (it is a weak entity as it depends on loan)***

STEP 2: *Find the relationships*

RELATIONSHIPS:

- ***A branch can have many accounts so cardinality will be 1: n.***

- Customers will be having account in different branches Cardinality will be n: 1.

- Customers are allowed to take loan from the bank. Cardinality will be 1: n.

- Loan is paid back with the help of payment number. Cardinality will be 1: n.

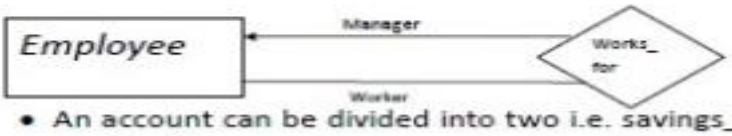
- A customer can be an employee of the bank.

  Cardinality will be n: m.

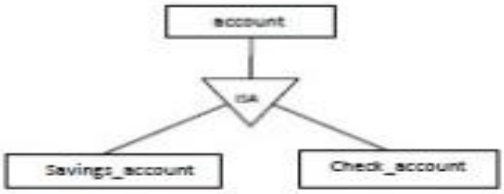
- Branches of bank give loans to the customers.

  Cardinality will be 1: n

- Employee can be a manager or a worker of that particular branch



- An account can be divided into two i.e. savings_account and check_account.



STEP3: ***Identify the primary key***

Entity	Primary key
Branch	Branch_name
Customer	Customer_id
Loan	Loan_number
Employee	Employee_id
Account	Account_id
Payment	Payment_number

(OR)

3.a Explain Conceptual database design with ER Model. 6M

[Explanation---4M]

The first step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the userdefined **operations** (or **transactions**) that will be applied to

the database, including both retrievals and updates. In software design, it is common to use *data flow diagrams, sequence diagrams, scenarios*, and other techniques to specify functional requirements. We will not discuss any of these techniques here; they are usually described in detail in software engineering texts.

Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS.

The last step is the physical design phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high level transaction specifications

[Diagram--2M]

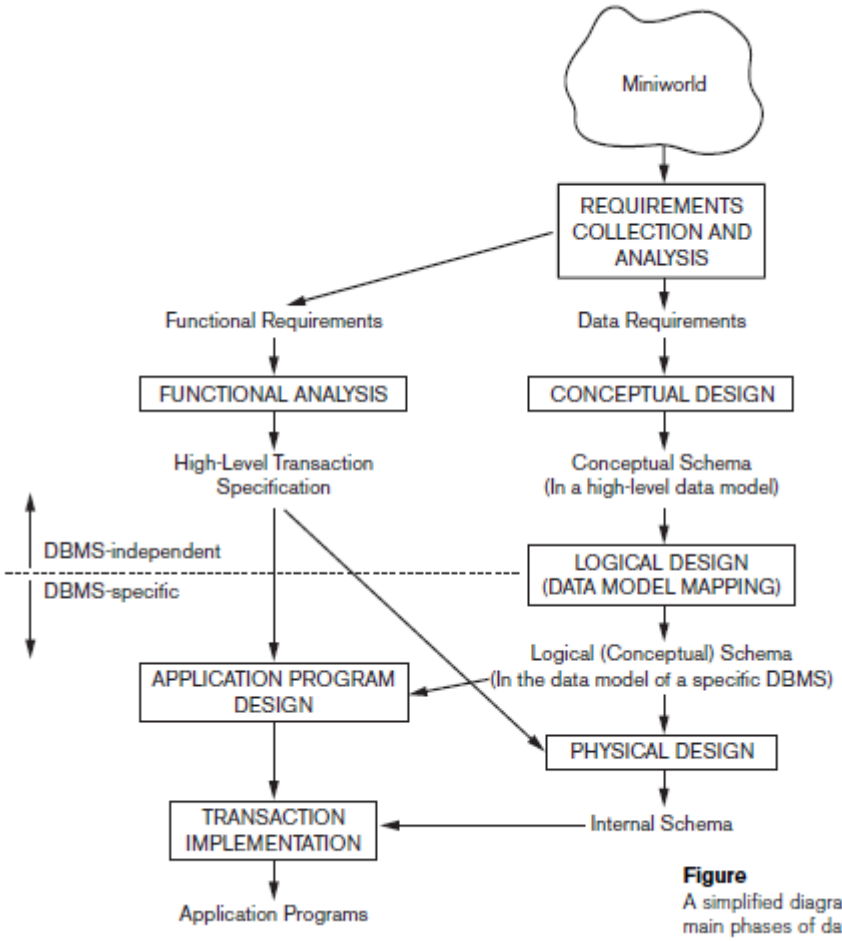


Figure
A simplified diagram to illustrate the main phases of database design.

3.b Describe in detail about 3-Schema Architecture. 6M

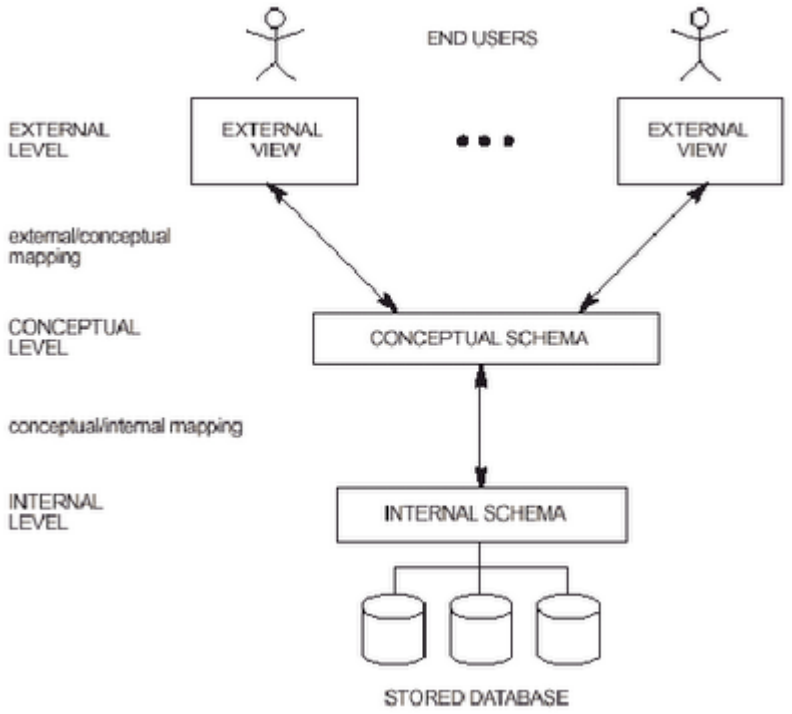
[Explanation of 3 Levels--3M]

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at 3 levels :

1. **Internal level or Internal schema** : Describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. **Conceptual level or Conceptual schema** : Describes the structure of the whole database for a community of users. It hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

Implementation data model can be used at this level.
3. **External level or External schema :** It includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user is interested in and hides the rest of the database from user. Implementation data model can be used at this level.

[Diagram--3M]



UNIT – II

4 Discuss about SELECT , JOIN & DIVISION with examples 12M

Any relevant example can be considered

The SELECT Operation **[Explanation of Select operation with example--4M]**

The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.3 One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to restrict the tuples in a relation to only those tuples that satisfy the condition. The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

$\sigma_{Dno=4}(\text{EMPLOYEE})$
$\sigma_{Salary>30000}(\text{EMPLOYEE})$

In general, the SELECT operation is denoted by

$\sigma_{<\text{selection condition}>}(R)$

where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation *R*. Notice that *R* is generally a *relational algebra expression* whose result is a relation—the simplest such expression is just the name of a database relation. The relation resulting from the SELECT operation has the *same attributes* as *R*.

The JOIN Operation [Explanation of Join operation with example--4M]

The JOIN operation, denoted by , is used to combine related tuples from two relations into single "longer" tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations. To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple.We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows:

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$$
$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$$

The first operation is illustrated in Figure 6.6. Note that Mgr_ssn is a foreign key of the DEPARTMENT relation that references Ssn, the primary key of the EMPLOYEE relation. This referential integrity constraint plays a role in having matching tuples in the referenced relation EMPLOYEE

The DIVISION Operation [Explanation of Division operation with example--4M]

The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications. An example is Retrieve the names of employees who work on all the projects that 'John Smith' works on. To express this query using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

Any relevant example can be considered

$$\text{SMITH} \leftarrow \sigma_{\text{Fname='John' AND Lname='Smith'}}(\text{EMPLOYEE})$$
$$\text{SMITH\_PNOS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} \bowtie_{\text{Essn=Ssn}} \text{SMITH})$$

Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

$$\text{SSN\_PNOS} \leftarrow \pi_{\text{Essn, Pno}}(\text{WORKS\_ON})$$

Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

$$\text{SSNS(Ssn)} \leftarrow \text{SSN\_PNOS} \div \text{SMITH\_PNOS}$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname}}(\text{SSNS} * \text{EMPLOYEE})$$

The preceding operations are shown in Figure 6.8(a).

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

(OR)

5 Consider the following schema: 12M

Suppliers(*sid:* integer, *sname:* string, *address:* string)
Parts(pid: integer, *pname:* string, *color:* string)
Catalog(*sid:* integer, *pid:* integer, *cost:* real)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers.

Write the following queries in tuple relational calculus.

vi. Find the *names* of suppliers who supply some red part
{s.sname|suppliers(s) AND ((∃c)(∃p)(catalog(c) AND parts(p) AND p.color='red' AND s.sid=c.sid AND c.pid=p.pid))}

vii. Find the *sids* of suppliers who supply some red or green part.
{s.sid|suppliers(s) AND ((∃c)(∃p)(catalog(c) AND parts(p) AND p.color='red' OR p.color='green' AND s.sid=c.sid AND c.pid=p.pid))}

viii. Find the *sids* of suppliers who supply some red part or are at 221 Packer Ave
{s.sid|suppliers(s) AND ((∃c)(∃p)(catalog(c) AND parts(p) AND p.color='red' AND s.sid=c.sid AND c.pid=p.pid)) OR ((∃c)(∃p)(catalog(c) AND parts(p) AND p.pname=' 221 Packer Ave ' AND s.sid=c.sid AND c.pid=p.pid))}

ix. Find the *sids* of suppliers who supply some red part and some green part
{s.sid|suppliers(s) AND ((∃c)(∃p)(catalog(c) AND parts(p) AND p.color='red' AND p.color='green' AND s.sid=c.sid AND c.pid=p.pid))}

x. Find the *sids* of suppliers who supply every red or green part
{s.sid|suppliers(s) AND ((∃c)(∀ p)(catalog(c) AND parts(p) AND p.color='red' OR p.color='green' AND s.sid=c.sid AND c.pid=p.pid))}

UNIT – III

6 Describe about Types of Indexes and how they use B-Trees & B+ Trees 12M

[Types of Indexes--6M
B+ Trees---6M]

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

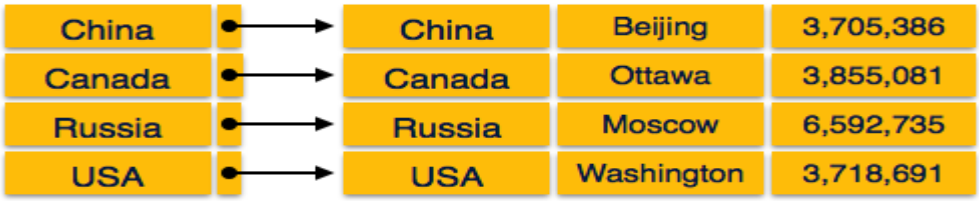
- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

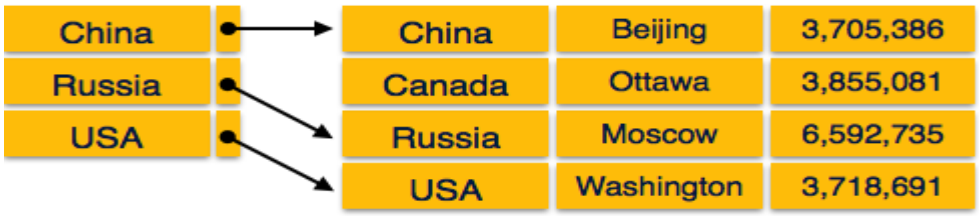
Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



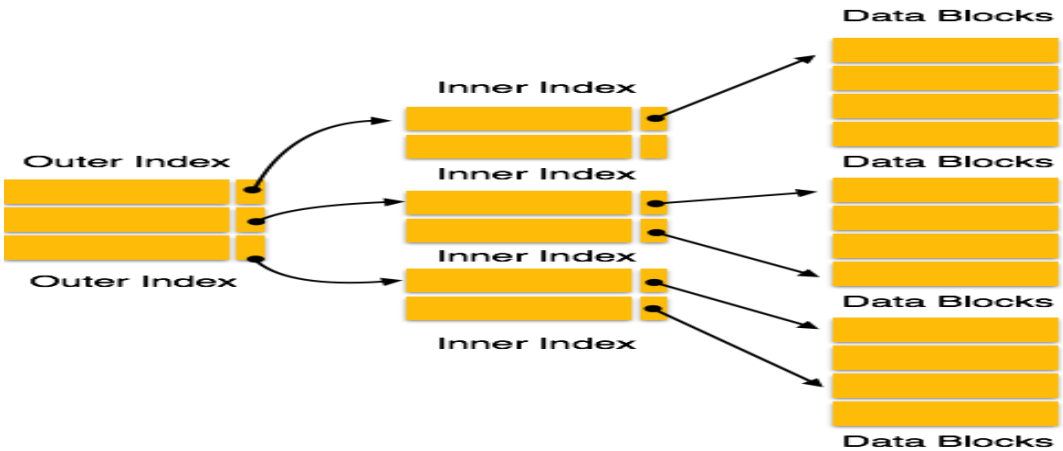
Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



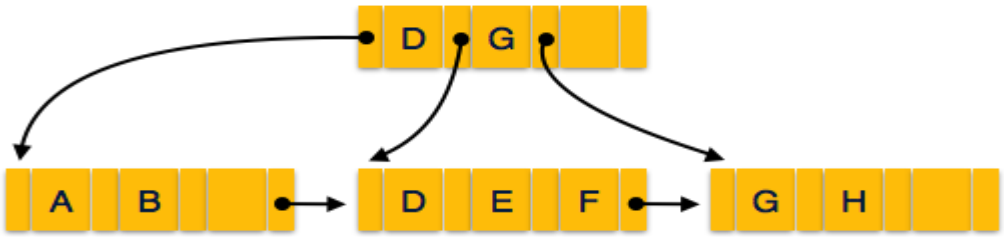
Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B^+ Tree

A B^+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B^+ tree denote actual data pointers. B^+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

Every leaf node is at equal distance from the root node. A B^+ tree is of the order **n** where **n** is fixed for every B^+ tree.



Internal nodes −

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes −

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.

B^+ Tree Insertion

- B^+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
    - Split node into two parts.
    - Partition at $\mathbf{i = \lfloor (m+1)_{/2} \rfloor}$.
    - First **i** entries are stored in one node.
    - Rest of the entries (i+1 onwards) are moved to a new node.
    - $\mathbf{i^{th}}$ key is duplicated at the parent of the leaf.
- If a non-leaf node overflows −
    - Split node into two parts.
    - Partition the node at $\mathbf{i = \lceil (m+1)_{/2} \rceil}$.
    - Entries up to **i** are kept in one node.
    - Rest of the entries are moved to a new node.

B^+ Tree Deletion

- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
    - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
    - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
    - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
    - Merge the node with left and right to it.

(OR)

7.a Differentiate Ordered and Unordered records in files. 6M

Writing any relevant points can be considered

In an ordered file, the records are sequenced on some field in the record (like StudentId or StudentName etc). In an unordered file, the records are not in any particular order.

Ordered File	Unordered File
RecordA	RecordM
RecordB	RecordH
RecordG	RecordB
RecordH	RecordN
RecordK	RecordA
RecordM	RecordK
RecordN	RecordG

Records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the Such an organization is called a heap or pile file. This organization is often used with additional access paths, such as the secondary indexes. Inserting a new record is very efficient. The last disk block of the file is copied into a buffer, the new record is added, and the block is then rewritten back to disk. The address of the last file block is kept in the file header. However, searching for a record using any search condition involves a linear search through the file block by block an expensive procedure.

To delete a record, a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally rewrite the block back to the disk. This leaves unused space in the disk block. Deleting a large number of records in this way results in wasted storage space.

An ordered Sequential file, is a file ordered upon some key field. The ordering of the records in the file makes it possible to process an ordered file in ways that are not available to us with unordered files. While it is not really possible to apply batch updates or deletes to an unordered file these are possible with ordered files. Ordered records have some advantages over unordered files. First, reading the records in order of the ordering key values becomes extremely efficient because no sorting is required. Second, finding the next record from the current one in order of the ordering key usually requires no additional block accesses because the next record is in the same block as the current one

7.b Analyze about Normalization and describe briefly about 1NF, 2NF, 3NF. 6M

Normalization--1 1/2 M
Normal forms for each----1 1/2 M

Normalization is the process of efficiently organizing data in a database with two goals in mind
First goal: eliminate redundant data for example, storing the same data in more than one table
Second Goal: ensure data dependencies make sense for example, only storing related data in a table .

Advantages of Normal forms
Less storage space
Quicker updates
Less data inconsistency
Clearer data relationships
Easier to add data
Flexible Structure

First Normal Form

'A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only.'

Second Normal Form

'A relation R is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully functionally dependent on the primary key.

Third Normal Form

'A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

OR

A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X->A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

UNIT – IV

8 Demonstrate Two-phase locking techniques in concurrency control and how time stamp ordering will be used in 2PL. 12M

Two-phase locking techniques---6M
Time stamp ordering---6M

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

- Lock based protocols
- Time stamp based protocols

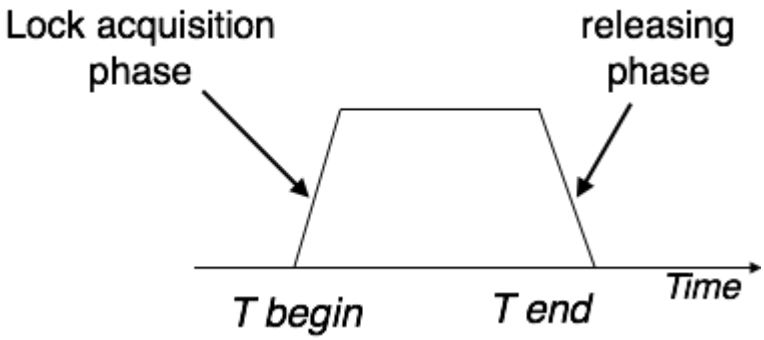
Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Two-Phase Locking 2PL

- This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

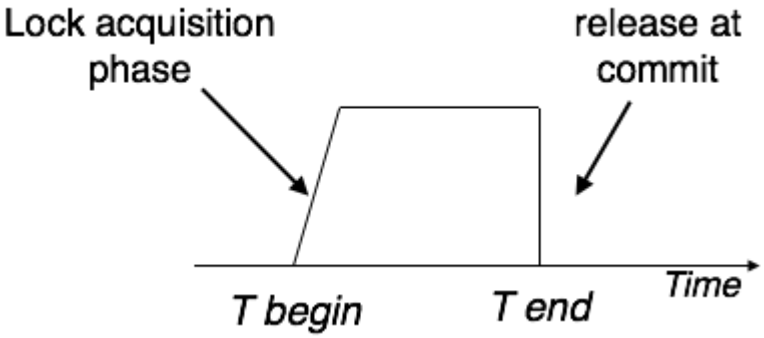


- Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is **shrinking,** where the locks held by the transaction are being released.

- To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

- The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

- The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

- Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

- Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

- In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction Ti is denoted as TS(Ti).
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

- **If a transaction Ti issues a read(X) operation −**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  - All data-item timestamps updated.
- **If a transaction Ti issues a write(X) operation −**
  - If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

Thomas' Write Rule

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.Time-stamp ordering rules can be modified to make the schedule view serializable. Instead of making Ti rolled back, the 'write' operation itself is ignored.

(OR)

9a **How do you characterize schedules based on Serializability.** **8M**

We characterize the types of schedules that are always considered to be correct when concurrent transactions are executing. Such schedules are known as serializable schedules. Suppose that two users—for example, two airline reservations agents—submit to the DBMS transactions T1 and T2 at approximately the same time. If no interleaving of operations is permitted, there are only two possible outcomes:

1. Execute all the operations of transaction T1 (in sequence) followed by all the operations of transaction T2 (in sequence).
2. Execute all the operations of transaction T2 (in sequence) followed by all the operations of transaction T1 (in sequence).
These two schedules—called serial schedules.If interleaving of operations is allowed, there will be many possible orders in which the system can execute the individual operations of the transactions. The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules.

Any relevant example can be considered

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)

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);

Time

Schedule A

(b)

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);	

Time

Schedule B

(c)

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);	

Time

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);	

Time

Schedule D

Schedules A and B are called serial because the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. Schedules C and D are called nonserial because each sequence interleaves operations from the two transactions.

- **Serializable schedule**: A schedule S is **serializable** if it is **equivalent** to some serial schedule of the same n transactions.
- **Result equivalent**: Two schedules are called result equivalent if they produce the same final state of the database.
- **Conflict equivalent**: Two schedules are conflict equivalent if the relative order of *any two conflicting operations* is the same in both schedules.
- Two operations are **conflicting** if:
  - They access the same data item X
  - They are from two different transactions
  - At least one is a write operation
- Read-Write conflict example: r1(X) and w2(X)
- Write-write conflict example: w1(Y) and w2(Y)

  Changing the order of conflicting operations generally *causes a different outcome*
- **Example:** changing r1(X); w2(X) to w2(X); r1(X) means that T1 will read *a different value for X*
- **Example:** changing w1(Y); w2(Y) to w2(Y); w1(Y) means that the final value for Y in the

database can be different

- Note that read operations are **not conflicting**; changing r1(Z); r2(Z) to r2(Z); r1(Z) does not change the outcome

Conflict equivalence of schedules is used to determine which schedules are correct in general (serializable). A schedule S is said to be **serializable** if it is conflict equivalent to some serial schedule S'.

Testing Conflict Serializability of a Schedule S

1. For each transaction Ti participating in schedule S, create a node labeled Ti in the precedence graph.

2. For each case in S where Tj executes a read_item(X) after Ti executes a write_item(X), create an edge (Ti→ Tj) in the precedence graph.

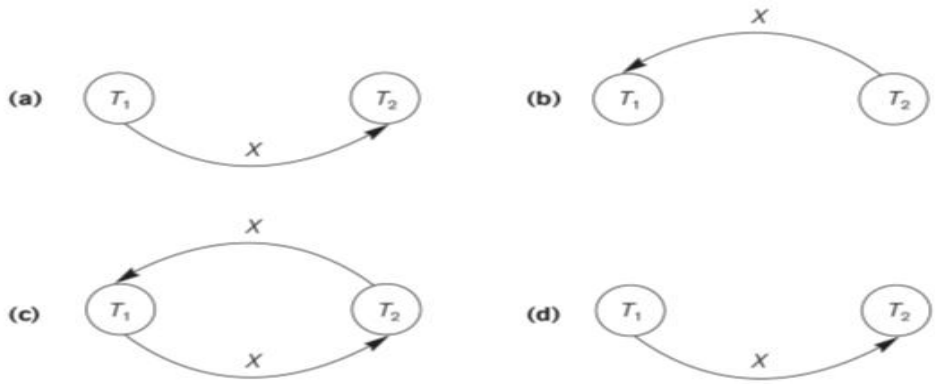
3. For each case in S where Tj executes a write_item(X) after Ti executes a read_item(X), create an edge (Ti→Tj) in the precedence graph.

4. For each case in S where Tj executes a write_item(X) after Ti executes a write_item(X), create an edge (Ti→ Tj) in the precedence graph.

5. The schedule S is serializable if and only if the precedence graph has no cycles.

The precedence graph is constructed as described in Algorithm 21.1. If there is a cycle in the precedence graph, schedule S is not (conflict) serializable; if there is no cycle, S is serializable. A cycle in a directed graph is a sequence of edges C = ((Tj→ Tk), (Tk → Tp), ..., (Ti → Tj)) with the property that the starting node of each edge—except the first edge—is the same as the ending node of the previous edge, and the starting node of the first edge is the same as the ending node of the last edge.

Any relevant example can be considered



Constructing the precedence graphs for schedules A to D from Figure 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).

- **View equivalence**: A less restrictive definition of equivalence of schedules than conflict serializability *when blind writes are allowed*

- **View serializability:** definition of serializability based on view equivalence. A schedule is *view serializable* if it is *view equivalent* to a serial schedule.

Two schedules are said to be **view equivalent** if the following three conditions hold:

- The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
- For any operation Ri(X) of Ti in S, if the value of X read was written by an operation Wj(X) of Tj (or if it is the original value of X before the schedule started), the same condition must hold for the value of X read by operation Ri(X) of Ti in S'.
- If the operation Wk(Y) of Tk is the last operation to write item Y in S, then Wk(Y) of Tk must also be the last operation to write item Y in S'.

9b Explain about Multiple Granularity Locking mechanism. 9M

Writing any relevant points can be considered

Multiple granularity locks allow us to lock at different granularities (database, tables, pages, tuples). This is useful because we can choose different granularities for different transactions.it is just alocking protocol like binary locking and two phase locking protocol.

Why

Specifying granularities is important because we often do not need to lock at the highest granularity and therefore, free up objects that would otherwise still be locked.

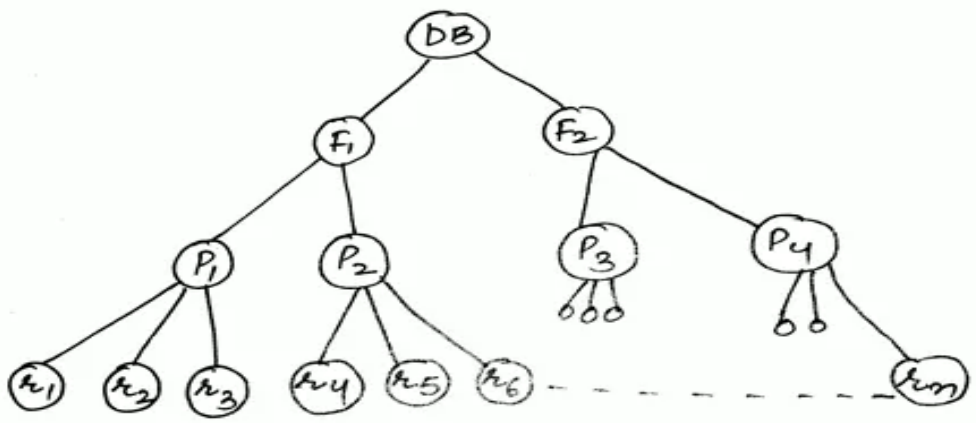
Strategy

We introduce a concept called? **intention?** locks which give the transaction the right to request locks at finer granularities

- IS - intention to get an S lock at finer granularity
- IX - intention to get an X lock at finer granularity
- SIX mode - like a S and IX lock

	IS	IX	SIX	S	X
IS	√	√	√	√	-
IX	√	√	-	-	-
SIX	√	-	-	-	-
S	√	-	-	√	-
X	-	-	-	-	-

Suppose a database is divided into files; files are divided into pages; pages are divided into records.



If there is a need to lock a record, then a transaction can easily lock it. But if there is a need to lock a file, the transaction have to lock firstly all the records one after another, then pages in that file and finally the file. So, there is a need to provide a mechanism for locking the files also which is provided by multiple granularity.