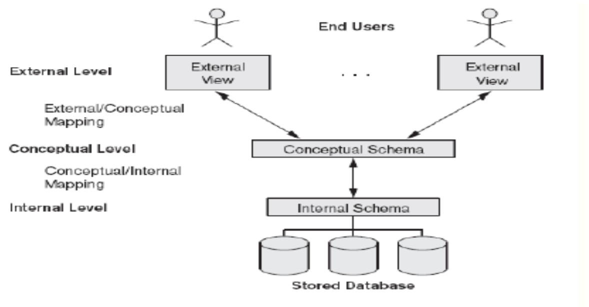
| | |
|---|---|
| **Name:**<br><br>**Enrolment No:** | UPES UNIVERSITY WITH A PURPOSE |

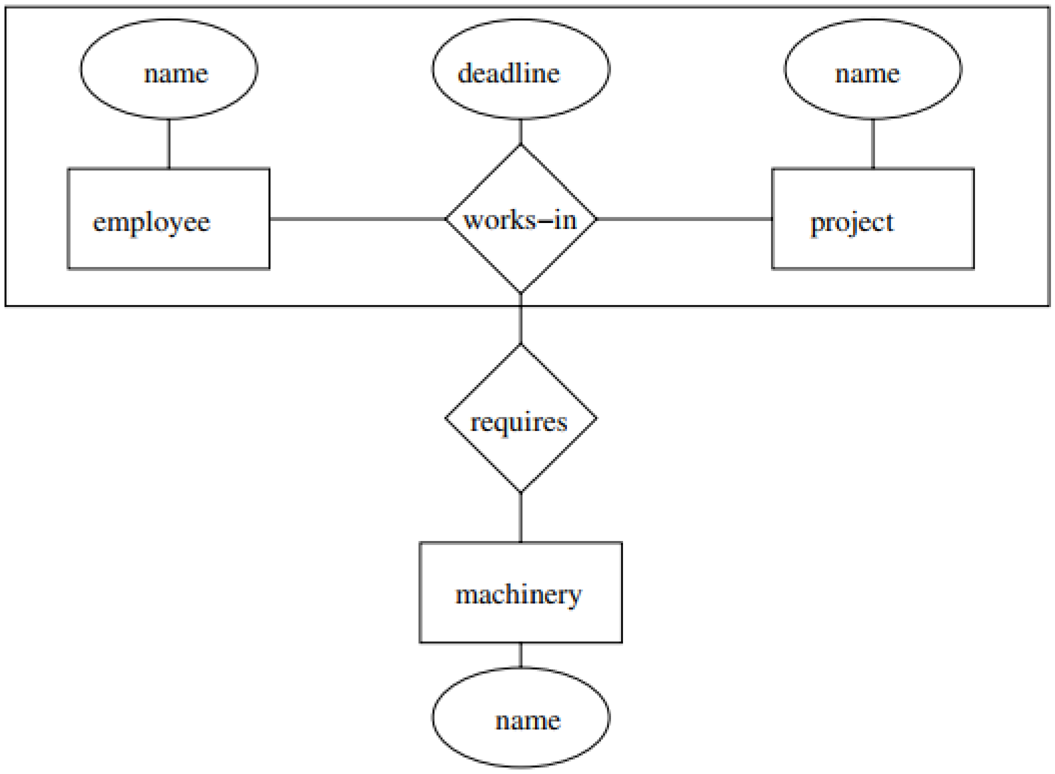
**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**
**End Semester Examination, December 2019**

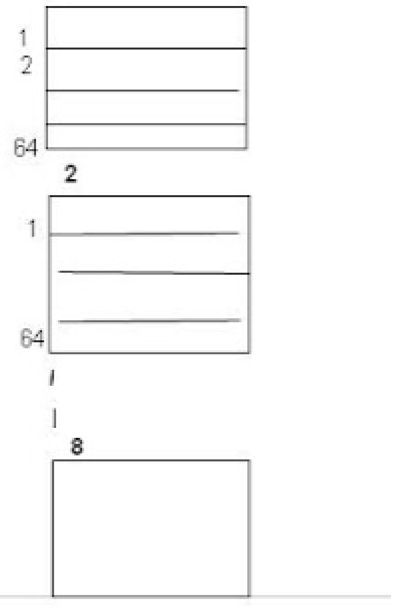
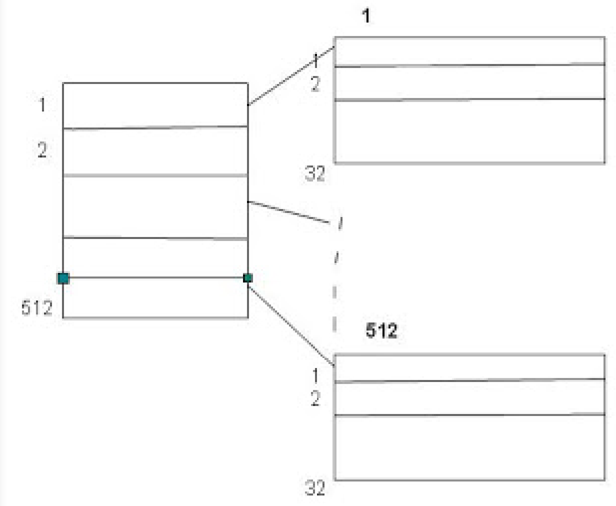
**Course: Advanced Database Management Systems** **Semester : III**
**Program: B.Tech.-CSE with all specialization** **Time : 03 hrs.**
**Course Code: CSEG2017** **Max. Marks : 100**

**Instructions:**

**SECTION A (Attempt all questions)**

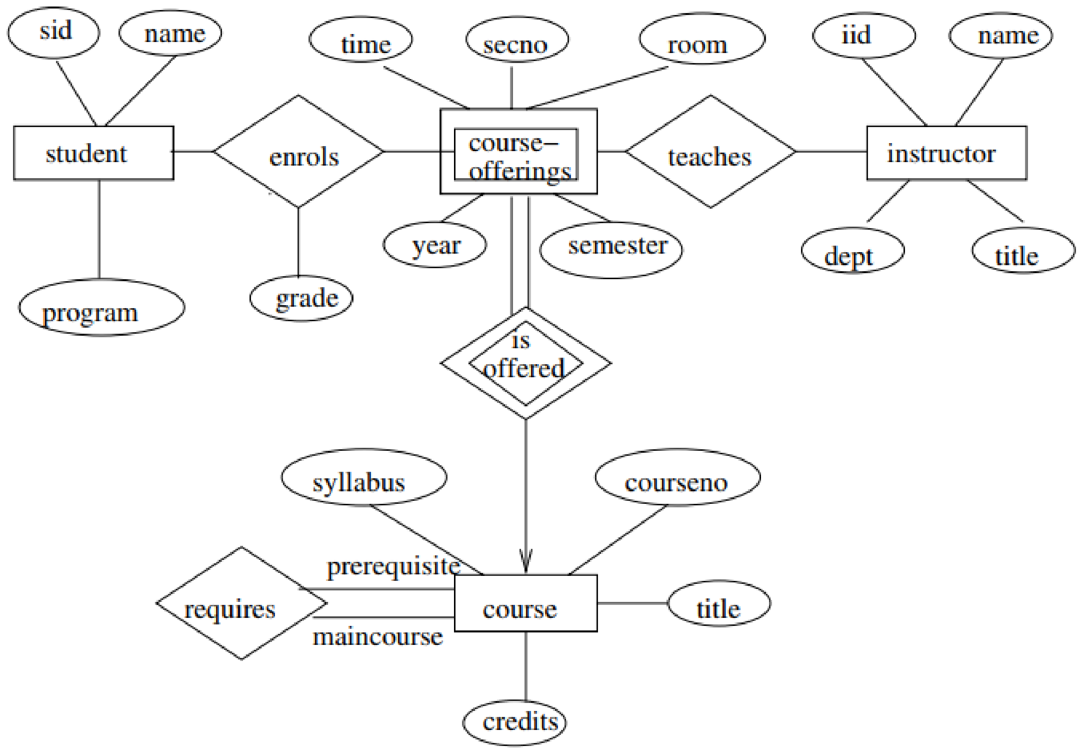
| **S. No.** | **Question** | **Marks** | **CO** |
|---|---|---|---|
| Q 1 | Differentiate between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?<br><br>The goal of three schema database architecture is to achieve **program data independence** (by separating user application and physical database) and providing support for **multiple data views**. In database architecture schemas are described at the following three levels:<br>• Internal level: At this level, schema describes the physical storage structure of the database. Physical data model is used to describe the schema at this level.<br>• Conceptual level: At this level, schema describe the structure of the whole database for a community of users. It hides the detail of physical storage structure and concentrate on describing entities, data types, relationship, user operations and constraints. High level data model or implementation model can be used to describe schema at this level.<br>• External/View level: At this level, schema describes part of the database that a particular user group is interested in and hides the rest of the database from that user group. High level data model or implementation model can be used to describe schema at this level.<br> End Users<br>External Level<br>External View<br>External View<br>External/Conceptual Mapping<br>Conceptual Level<br>Conceptual Schema<br>Conceptual/Internal Mapping<br>Internal Level<br>Internal Schema<br>Stored Database<br>Data independence can be defined as the capacity to change the schema at lower level of database system without changing the schema at the higher level. Only the mapping between the two level is changed. There are two types of data independence: | **4** | **CO1** |

|  |  |  |  |
|---|---|---|---|
|  | • Logical Independence: It is the capacity to change the conceptual schema without changing external schema or application program. We may change the conceptual schema to expand or reduce the database.<br>• Physical data independence: It is the capacity to change the internal schema without changing conceptual schema or external schema. We may change the internal schema to reorganize the physical files. |  |  |
| Q 2 | Through suitable example explain the following terms briefly:<br>• Generalization and Specialization: Specialization is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process while generalization is bottom up process.<br>• Aggregation: Aggregation is an abstraction through which relationships are treated as higher-level entities. Thus the relationship between entities *A* and *B* is treated as if it were an entity *C*. e.g. Employees work for projects. An employee working for a particular project uses various machinery.<br> name<br>deadline<br>name<br>employee<br>works–in<br>project<br>requires<br>machinery<br>name<br>• Disjoint and overlapping constraints: In a disjointness design constraint, an entity can belong to not more than one lower-level entity set. In overlapping generalizations, the same entity may belong to more than one lower-level entity sets. For example, in the employee-workteam example of the book, a manager may participate in more than one work-team<br>• Total and partial constraints: In a total design constraint, each higher-level entity must belong to a lower-level entity set. The same need not be true in a | **4** | **CO2** |

|  |  |  |  |
|---|---|---|---|
|  | partial design constraint. For instance, some employees may belong to no work-team. |  |  |
| Q 3 | Consider ordered data file with following parameters:<br>Number of records = 16348 Record size = 32 bytes<br>Block size = 1024 bytes<br>Index is stored as (key + pointer) pair with the following parameters:<br>Key value = 10 bytes Block pointer = 6 bytes<br>• Find the number of first level and second level blocks required for multilevel index.<br>• Draw the appropriate diagram and justifying your answer.<br><br>Number of records that can be accumulated in block i.e. Blocking factor bfr = 1024/32 = 2^5, so, can have 32 records in a block<br>Number of blocks required for data file = (r/bfr) = 16348/ 32 ~= 511<br><br>Number of blocks in first level of multilevel index will be required to store this much entries where each entry is of 16 bytes(key + pointer size)<br>R' = 16 B = 1024 bfr' = 1024/16 = 2^6=64<br>so number of blocks required for 512 entries would be = r'/bfr' = 511/64 = 2^3 ~= 8<br><br> <br>secondary level index<br> <br>primary level of multilevel index and data file | **4** | **CO3** |
| Q 4 | A file of 4096 blocks is to be sorted with an available buffer space of 64 blocks. How many passes will be needed in the merge phase of the external sort-merge algorithm?<br><br>We first need to compute the number of runs, $N_R$, in the merge phase.<br>$N_R = \lceil b/N_b \rceil$<br>Where b = 4096 (the number of blocks in the file), and $N_b$= 64 (available buffer space)<br>**So, $N_R$= ⌈4096 / 64⌉ = 64 --number of sorted runs** | **4** | **CO4** |

|  |  |  |  |
|---|---|---|---|
|  | In the merge phase, the number of passes is dependent on the degree of merging, $D_M$, where $D_M$ = Min ($N_R$, $N_b$– 1)= Min (64, 64 –1) = 63<br>**Hence, the number of passes is given by ⌈ Log $_{DM}$ ($N_R$) ⌉ = Log⌈63 (64) ⌉ = 2** |  |  |
| Q 5 | Discuss the ACID properties in transaction management.<br>In the context of transaction processing, the acronym *ACID* refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.<br>**Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.<br>**Consistency:** Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.<br>**Isolation:** The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.<br>**Durability:** After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed. | **4** | **CO6** |
| **SECTION B (Attempt all questions)** |  |  |  |
| Q 6 | List out the different reasons of choosing a database system instead of simply storing data in operating system files? When would it make sense not to use a database system?<br><br>• In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. This results in redundancy in defining and storing data and also results in wastage of storage space. It also results in redundant efforts to maintain common up-to-date data. This turn in to inconsistent data. While in the database approach, a single repository maintains data that is defined once and then accessed by various users.<br>• In file systems, each application is free to name data elements independently. While in database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.<br>• Self-Describing Nature of a Database System: Due to presence of metadata in data base system, general purpose DBMS software work well with any number of database applications. While in traditional file processing system, 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. | **8** | **CO1** |

|  |  |  |  |
|---|---|---|---|
|  | • Insulation between Programs and Data: 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.<br>• 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. This facility is not available in file processing system.<br><br>Usage of DBMS is not suggested in the following cases:<br>• If database and application are simple<br>• Not expecting to change<br>• Multiple user access to data is not required |  |  |
| Q 7 | A university registrar's office maintains data about the following entities:<br>a) Courses, including number, title, credits, syllabus, and prerequisites.<br>b) Course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom.<br>c) Students, including student-id, name, and program.<br>d) Instructors including identification number, name, department, and title.<br>Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an E-R diagram for the registrar's office.<br>Document all assumptions that you make about the mapping constraints.<br><br>The assumptions made are:<br>• A class meets only at one particular place and time. This E-R diagram cannot model a class meeting at different places at different times.<br>• There is no guarantee that the database does not have two classes meeting at the same place and time. | **8** | **CO2** |

|  |  |  |  |
|---|---|---|---|
|  |  sid<br>name<br>time<br>secno<br>room<br>iid<br>name<br>student<br>enrols<br>course–offerings<br>teaches<br>instructor<br>year<br>semester<br>dept<br>title<br>program<br>grade<br>is offered<br>syllabus<br>courseno<br>prerequisite<br>requires<br>course<br>title<br>maincourse<br>credits |  |  |
| Q 8 | Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec.<br>a) What is the capacity of a track in bytes?<br>bytes/track = bytes/sector × sectors/track = 512 × 50 = 25K<br>b) What is the capacity of each surface?<br>bytes/surface = bytes/track × tracks/surface = 25K × 2000 = 50, 000K<br>c) What is the capacity of the disk?<br>bytes/disk = bytes/surface× surfaces/disk = 50, 000K × 5 × 2 = 500, 000K<br>d) How many cylinders does the disk have?<br>The number of cylinders is the same as the number of tracks on each platter, which is 2000. | **8** | **CO3** |
| Q 9 | Discuss in detail the Relational Model Constraints and its types.<br>Constraints are conditions that must hold on all valid relation instances. There are three main types of constraints:<br>• Inherent model or Implicit constraints- inherent in the data model. E.g. relation characteristic in relational model. Constraint that a relation cannot have duplicate tuples is an inherent constraint.<br>• Schema based or Explicit constraints-can be directly expressed in schema<br>▪ Domain constraints: Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). Data type, range and formats are used to specify domain constraints.<br>▪ Key constraints: **Keys** are the entity set that is used to identify an entity within its entity set uniquely. An entity set can have multiple **keys**, but out of which one **key** will be the primary **key**. A primary **key** can contain a unique and null value in the relational table. A **key constraint** is a type of | **8** | **CO4, CO5** |

rule in a DBMS that constrains the type of data that can be inserted. A primary **key constraint** ensures that every value in a given field will be unique. A foreign **key constraint** ensures that every value inserted into that field will already exist in another designated field.

- Null constraints: NOT **NULL constraint** makes sure that a column does not hold **NULL** value. When we don't provide value for a particular column while inserting a record into a table, it takes **NULL** value by default. By specifying **NULL constraint**, we can be sure that a particular column(s) cannot have **NULL** values.
- Entity integrity constraints: Entity Integrity: **The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R)**. This is because primary key values are used to identify the individual tuples. t[PK] ≠ null for any tuple t in r(R). Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.
- Referential integrity constraints: Referential integrity constraint involve two relations while key and entity constraint involve only one relation. The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. It is implemented through the concept of foreign key. Tuples in the **referencing relation** $R_1$ have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the **referenced relation** $R_2$. A tuple $t_1$ in $R_1$ is said to reference a tuple $t_2$ in $R_2$ if $t_1$ [FK] = $t_2$[PK]. A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.PK

- Application based or Semantic constraints or business rules- can't be directly expressed in schema, enforced by application program. These are based on application semantics e.g., "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week" or minimum salary must be Rs 1000 Such constraints can be specified and enforced within the application programs that update the database.
- Another type of constraint is the functional dependency constraint which are used as a tool to analyze the quality of relational designs and to "normalize" relations to improve their quality.

**Or**

Consider a relation R with attributes ABCDEFGH and functional dependencies S as follows: S = {A → CD, ACF → G, AD → BEF, BCG → D, CF → AH, CH → G, D → B, H → DEG} Find all keys for R.

We can find keys (candidate keys) for a relation by finding the **closure** of an/set of attributes. Checking each attribute or all subsets of the given set of attributes for a key is time consuming job. Hence, we may employ some of the following heuristics/assumptions in identifying the keys;

- We may start checking all the left hand side attributes of any/all of the given set of functional dependencies.

- If we find the closure of an attribute and that attribute is the candidate key then any superset cannot be the candidate key. For example, if A is a candidate key, then AB is not a candidate key but a super key.

| LHS | Result | Decision |
|---|---|---|
| $A^+$ | = ACD from A → CD<br>= ACDBEF from AD → BEF<br>= ACDBEFG from ACF → G<br>= ACDBEFGH from CF → AH | Result includes all the attributes of relation R. That is, if we know A, then all the attributes of R could be uniquely determined. Hence, A is one candidate key. |
| $ACF^+$ | No need to find the closure of (ACF) because the subset A is already a candidate key. | ACF is a super key but not candidate key. |
| $AD^+$ | No need to find the closure of (AD) because the subset A is already a candidate key. | AD is a super key but not candidate key. |
| $BCG^+$ | = BCGD from BCG → D | Result does not include all the relation R. Hence, (BCG) cannot be a candidate key. |
| $CF^+$ | = CFAH from CF → AH<br>Further, as we know A now, then we can conclude that CF will uniquely determine all the other attributes of A. | Result includes all the attributes of relation R. Hence, (CF) is one candidate key. |
| D+ | = DB from D → B | Result does not include all R. Hence, D cannot be a key. |
| H+ | = HDEG from H → DEG<br>= HDEGB from D → B | Result does not include all R. Hence, H cannot be a key. |

**From the above table, it is clear that only A and CF are the candidate keys.**

Consider a database with objects X and Y and assume that there are two transactions T1 and T2. Transaction T1 reads objects X and Y and then writes object X. Transaction T2 reads objects X and Y and then writes objects X and Y.

a) Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-read conflict.

b) Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a read-write conflict.

**8** **CO6**

Q 10

c) Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-write conflict.

Given that
T1 = r1(X), r1(Y), w1(X)

T2 = r2(X), r2(Y), w2(X), w2(Y)

we design the following schedules which have the asked conflicts.

(a) Write-read conflict

(which necessarily means reading uncommitted data, aka dirty read which results in cycle in the precedence graph):

S1 = r2(X), r2(Y),**w2(X), r1(X)**, r1(Y), w1(X), w2(Y)

The WR conflict is written in bold. Note that, it is not necessary to have the W and R statement immediately after each other to create a WR conflict.

(b) Read-write conflict

(which necessarily means unrepeatable read which causes cycle in the precedence graph):

S2 = r2(X), r2(Y), w2(X), r1(X), **r1(Y), w2(Y)**, w1(X)

The RW conflict is written in bold.

(c) Write-write conflict

(which necessarily means overwriting uncommitted data or blind writes resulting in cycle in the precedence graph):

S3 = r2(X), r2(Y), r1(X), r1(Y), **w2(X), w1(X)**, w2(Y)

The WW conflict is written in bold.

**Or**

Explain the main concept used in object database system.

There is a certain set of basic concepts, supported by each object-oriented database system. These basic concepts are objects and identity, encapsulation, classes and instantiation, inheritance and overloading, overriding and late binding.

**Objects and Identity**

In an object-oriented database, each real-world entity is represented by an object. This object has a state and a behaviour. The combination of the current values of an object's attributes define the object's state. A set of methods, acting on an object's state, define the object's behaviour.

When it comes to identity, each object in the database is defined by a unique object identifier. There are certain differences between keys and object identifiers. A key is an attribute value, or a set of attribute values. An example would be an employee number. As a key is an attribute value, it can be modified, at least theoretically. In this sense, a key is dependent on the object's state. In contrast, object identifiers are independent from an object's state. Two object's have a different object identifier, even if all their attributes' values (their state) is actually the same. Another difference is, that a key is unique in a certain relation only. Object identifiers are unique in the whole database. Finally, object identifiers are generated by the database system. The database user has absolutely no control over these identifiers. In contrast, a key can be changed by the database user by simply performing a modification operation.

Using object identifiers comes with a set of advantages. First of all, the database user need not to concern with defining keys for each relation. Another advantages is the better performance, as the handling of object identifiers is implemented on a low level by the system.

When using object identifiers, we have to distinguish two different kinds of object equality. The first one is identity equality. Identity equality is given, if two objects have the same object identifier. The second one is value equality. Value equality is given, if all the attributes' values of two objects are identical. Identity equal objects are always value equal. The reverse is not true.

**Encapsulation**

Encapsulation is a basic concept for all object-oriented technologies. It was created for making a clear distinction between the specification and the implementation of an operation and in this way for enabling modularity. Software that exceeds a certain size needs some sort of modular architecture for enabling design, implementation and maintenance to be executed by a larger group of programmers.

As far as programming languages are concerned, encapsulation derives from the notion of abstract data types, including the distinction of interfaces and implementation. An interface specifies the operations that can be invoked on an object. This is the only part of the object that is visible from the outside. An implementation is an object's data, representing the object's state, and the implementation of the object's methods. Implementation details are not visible from outside the object.

In principle, the concept of encapsulation in object-oriented databases is the same. The only difference is that it's is not clearly defined, whether the object's data structure is part of the interface. In programming languages, the data structure is certainly part of the implementation.

Methods defined by an object create the object's behaviour. The purpose of such a method can be changing some attributes' values, as well as calculating a certain value in dependency on the object's actual state. Every object may define an arbitrary number of methods. Encapsulation conceptually makes the maintenance of essential routines a task of the data level.

**Classes and Instantiation**

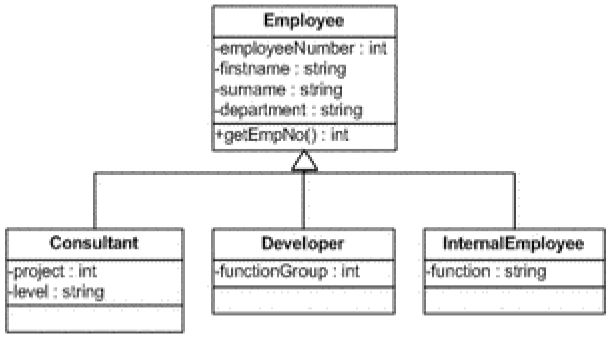
When looking on the concept of classes in object-oriented databases, you have to distinguish the terms class and type. A type is used to describe a set of objects that share the same behaviour. In this sense, an object's type depends on which operations can be invoked on the object. A class is a set of objects that have the exactly same internal structure. In that way, a class defines the implementation of an object and a type describes the way the object can be used.

The term instantiation aims at the fact that a class definition can be used to generate a set of objects that share the same structure and behaviour. A class specifies a structure (a set of attributes), a set of operations and a set of methods, which implement the operations.

A feature, which is very important to the evolution of objects, is that an object can change its class. This means that an object can change is attributes and operations while keeping the same identity. Enabling class changes comes with the need for a mechanism for handling semantic integrity problems that may arise. Applications must deal with exceptions that may arise when an object is referred to as an instance of a class, different to the one expected.

**Inheritance**

Inheritance makes it possible to define a class as a subclass of an already existing one (superclass). The subclass inherits all attributes and methods from the superclass and can additionally define its own attributes and methods. This concept is an important mechanism for supporting reusability. Identical parts of the structure of two different classes may be defined only once in a common superclass. Is this way, less code has to be written. There are some systems that allow a class to be subclass of more than one superclass. This feature is called multiple inheritance (in contrast to single inheritance).



There are certain problems that can arise due to inheritance. And these problems are even more likely to arise if multiple inheritance is used. For example, a subclass may define an attribute with the same name as an attribute already defined by a superclass. The problem gets worse when two superclasses of the same subclass define attributes and methods with the same name. Each system must come of with a mechanism to resolve this problems.

A superclass itself can also have a superclass. And this class in turn can have another superclass. Given this, object-oriented database systems build a hierarchy of their classes.

**Overloading, Overriding and Late Binding**

It is often useful to use the same name for different, but similar, methods. Imaging you want to display a item on your screen. Different items may need different viewers. Maybe you wish to be able to view all items with the method "view". If you call "view" and pass a reference to a picture, a picture viewer is started. When calling "view" and passing a reference to a video, a media player is started. To implement this functionality you first of all have to define the operation "view" in a common superclass "media" of the classes "picture" and "video". Each of the subclasses redefines the "view" operation for its specific needs. This result in different methods that share the same operation name. Using this feature comes with a important advantage. The code can more easily be maintained and the introduction of a new type does nor require any modifications to existing parts of applications. Providing this feature disables the system to bind names of operations to the corresponding methods at compile-time. The delayed binding of operation names to the corresponding methods at run-time is called "late binding"

| SECTION-C(Attempt all questions) | | | |
|---|---|---|---|
| Q 11 | Consider a relation R(A, B, C, D, E, F, G, H, I, J) with functional dependencies: $\{ AB \rightarrow C, A \rightarrow DE, B \rightarrow F, D \rightarrow IJ, F \rightarrow GH \}$ List all the functional dependencies that violate 2NF, 3NF, BCNF. If any, then decompose R accordingly. Also check that normalized form after conversion into BCNF is lossy/lossless and dependency preserving/non dependency preserving?<br><br>Candidate key is: **AB**<br>Functional dependencies $A \rightarrow DE$, $B \rightarrow F$ are violating the conditions of 2NF as there is case of partial dependency. So to achieve 2NF we decompose R in sub relations:<br>$R_1(\underline{A, B}, C)$<br>$R_2(\underline{A}, D, E, I, J)$<br>$R_3(\underline{B}, F, G, H)$<br><br>Now, to check the 3NF for new schema having three sub relations, apply the conditions of 3NF i.e. there should be no transitive dependency.<br>$R_1$ is free from transitive dependency but there is issue of transitive dependency in $R_2$ and $R_3$. So, there is need to further decompose schema $R_2$ and $R_3$.<br>$R_1(\underline{A, B}, C)$<br>$R_{21}(\underline{A}, D, E)$<br>$R_{22}(\underline{D}, I, J)$<br>$R_{31}(\underline{B}, F)$<br>$R_{32}(\underline{F}, G, H)$<br><br>Now, to check the BCNF for new schema having five sub relations, apply the conditions of BCNF i.e. for every $m \rightarrow n$, m must be super key. It is found that its already in BCNF<br>$R_1(\underline{A, B}, C)$<br>$R_{21}(\underline{A}, D, E)$<br>$R_{22}(\underline{D}, I, J)$<br>$R_{31}(\underline{B}, F)$<br>$R_{32}(\underline{F}, G, H)$<br>Here after conversion from 1NF into BCNF, decomposition is lossless and dependency preserving. | **20** | **CO5** |
| Q12 | a) Find the number of block access in case of primary index, secondary index and without index for the following detail:<br>No of records in main file=20000   Record size in main file =150 bytes<br>Block size=2048   Record size in index file=20 bytes<br><br>No of block access **without index**:<br>Blocking factor: Floor(2048/150)=13<br>No of Block required: Floor(20000/13)=1539<br>No of block access required: Ceiling($\log_2$(1539))=**11** | **(10 + 10)** | **CO3, CO4**<br><br>**CO1, CO6** |

No of block access with **primary index**:

Blocking factor: Floor(2048/20)=102
No of Block required: Floor(1539/102)=16
No of block access required: Ceiling($\log_2(16)$)+1=**5**

No of block access with **secondary index**:

Blocking factor: Floor(2048/20)=102
No of Block required: Floor(20000/102)=197
No of block access required: Ceiling($\log_2(197)$)+1=**9**

b) Explain the different way to implement 'Select' operation. Compare them on the basis of number of block access.

- **Linear search** (brute force): Retrieve *every record* in the file, and test whether its attribute values satisfy the selection condition. Average search cost is (b/2) where b is number of blocks in which we have applied the search condition.
- **Binary search**: If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search (which is more efficient than linear search) can be used. (See OP1). Here average search cost is $\log_2(b)$
- **Using a primary index or hash key** to retrieve a single record: If the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key), use the primary index (or the hash key) to retrieve the record. Cost of search in this case will be height of tree to search specific value in the index table (where index table has been implemented using B+tree) +1
- . **Using a primary index** to retrieve multiple records: If the comparison condition is >, ≥, <, or ≤ on a key field with a primary index, use the index to find the record satisfying the corresponding equality condition, then retrieve all subsequent records in the (ordered) file. Search cost in this case is: height of the tree +b, where b represents the number of blocks satisfying the condition
- **Using a clustering index** to retrieve multiple records: If the selection condition involves an equality comparison on a non-key attribute with a clustering index, use the clustering index to retrieve all the records satisfying the selection condition. Cost of search in this case will be height of tree to search specific value in the index table (where index table has been implemented using B+tree) +No of blocks satisfying the specified condition.
- **Using a secondary index**: On an equality comparison, this search method can be used to retrieve a single record if the indexing field has unique values (is a key) or to retrieve multiple records if the indexing field is not a key. If index is on key then cost of search in this case will be height of

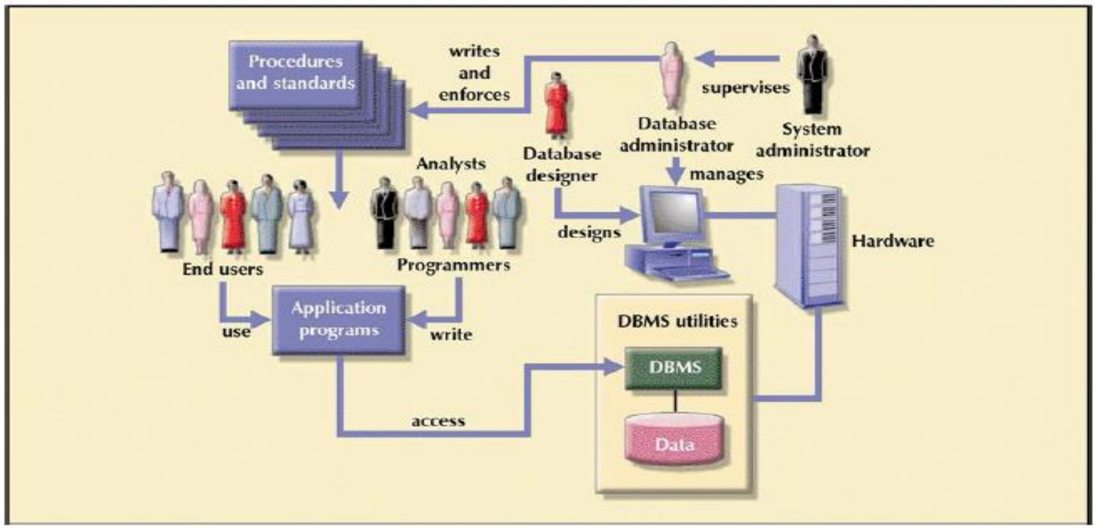
tree to search specific value in the index table (where index table has been implemented using B+tree) +1, same as **primary index.** If index is on non key attribute then Cost of search in this case will be height of tree to search specific value in the index table (where index table has been implemented using B+tree) +No of blocks satisfying the specified condition, same as **clustering index**.

- **Using a secondary index**: In addition, it can be used to retrieve records on conditions involving >,>=, <, or <=. (FOR **RANGE QUERIES**) Cost=No of disk access, since records may be in different blocks
- **Conjunctive selection**: If an attribute involved in any single *simple condition* in the conjunctive condition has an access path that permits the use of one of the methods S2 to S6, use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition.
- **Conjunctive selection using a composite index**: If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined field, we can use the index directly. In this case cost involved is the only cost to search index in in the index table.
- **Conjunctive selection by intersection of record pointers**: This method is possible if secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the conjunctive condition and if the indexes include record pointers (rather than block pointers). Each index can be used to retrieve the *record pointers* that satisfy the individual condition. The *intersection* of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.
- **Disjunctive selection by union of record pointers**: This method is possible if secondary indexes are available on all (or some of) the fields involved in equality comparison conditions in the disjunctive condition and if the indexes include record pointers (rather than block pointers). Each index can be used to retrieve the *record pointers* that satisfy the individual condition. The *union* of these sets of record pointers gives the record pointers that satisfy the disjunctive condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.

**Or**

a) Explain the role and responsibilities of different actors and users of DBMS.

- Database Administrators: **Administering the resources** like database, DBMS and related software is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.
- Database Designer: Database designers are responsible for identifying the data to be stored in the database. He also chooses the appropriate structures to represent and store this data. Database designers develop views of the database that meet the data and processing requirements of different user groups.
- System Analysts: System analysts determine the requirements of end users and develop specifications for standard canned transactions that meet these requirements.
- Application programmers implement the specifications developed by system analysts as programs; then they test, debug, document, and maintain these canned transactions. System analysts and application programmers—commonly referred to as software developers or software engineers—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.
- End User: End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:
    - Casual end users: They occasionally access the database, but they may need different information each time.
    - Naive end users: This group of user make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries. These are called canned transactions.
    - Sophisticated end users: It include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
    - Standalone users: This group of users maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.



b) What is the two-phase locking protocol? How does it guarantee serializability? Show it through an example.

Locking is an operation which secures (a) permission to Read(b) permission to Write a data item for a transaction. e.g. Lock (X). Data item X is locked in behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item.

e.g. Unlock (X): Data item X is made available to all other transactions. Lock and Unlock are Atomic operations.

To provide better concurrency along with isolation, we use different modes of lock.

- Shared: It is denoted by lock-S(q). Transaction can perform read operation. Any other transaction can also obtain same lock on same data item at same time. So it is called shared mode.

- Exclusive: It is denoted by lock-X(q). Transaction can perform both read write operation. Any other transaction can't obtain this lock at the same time.

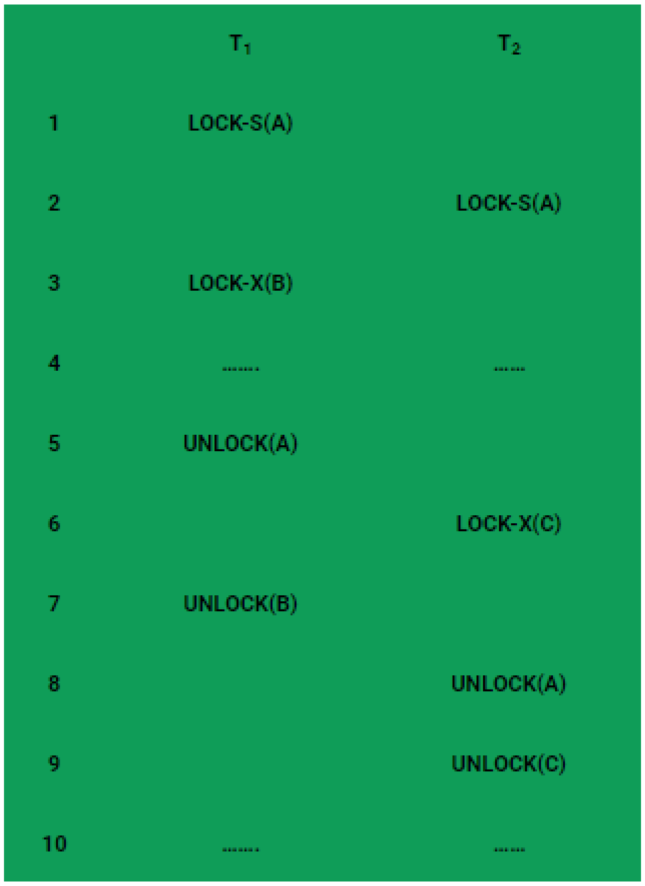
A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.

1. **Growing Phase:** New locks on data items may be acquired but none can be released.
2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

Transaction can perform read or write operation in both phase. If lock conversion is allowed, then upgrading of lock( from S(a) to X(a) ) is allowed in Growing Phase and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

**LOCK POINT:** The Point at which the growing phase ends, i.e., when transaction takes the final lock it needs to carry on its work.

**It ensures conflict serializability and view serializability:**
Serializability is mainly an issue of handling write operation. Because any inconsistency may only be created by write operation. Multiple reads on a database item can happen parallely. 2-Phase Locking protocol restricts this unwanted read/write by applying exclusive lock. Moreover, when there is an exclusive lock on an item it will only be released in shrinking phase. Due to this restriction there is no chance of getting any inconsistent state.



|  | $T_1$ | $T_2$ |
|---|---|---|
| 1 | LOCK-S(A) |  |
| 2 |  | LOCK-S(A) |
| 3 | LOCK-X(B) |  |
| 4 | ……. | …… |
| 5 | UNLOCK(A) |  |
| 6 |  | LOCK-X(C) |
| 7 | UNLOCK(B) |  |
| 8 |  | UNLOCK(A) |
| 9 |  | UNLOCK(C) |
| 10 | ……. | …… |

**Transaction $T_1$:**
- Growing Phase is from steps 1-3.
- Shrinking Phase is from steps 5-7.
- Lock Point at 3

**Transaction $T_2$:**
- Growing Phase is from steps 2-6.
- Shrinking Phase is from steps 8-9.
- Lock Point at 6