**Code No.**

**METHODIST COLLEGE OF ENGINEERING & TECHNOLOGY (An Autonomous Institution)**

**M.B.A III-Semester (Regular) Examination, Feb/March -2023**

**Subject: DATA BASE MANAGEMENT SYSTEM**

**Time: 3 hours Max.Marks:60**

**Note: Missing data, if any, maybe suitably assumed.**

**PART-A**

**Answer All the questions.**

|  |  |
| --- | --- |
| **Q.No** | **Questions** |
| **1. a** | It stands for Data Manipulation Language. It is used to add, retrieve or update the data. It add or update the row of the table. These rows are called as tuple. It is further classified into Procedural and Non-Procedural DML. BASIC command present in DML are UPDATE, INSERT, MERGE etc. While DML uses WHERE clause in its statement. |
| **b** | **Domain Relational Calculus** is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,  { < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn ) }  where, < x1, x2, x3, …, xn > represents resulting domains variables and P (x1, x2, x3, …, xn ) represents the condition or formula equivalent to the Predicate calculus.   **Predicate Calculus Formula:**   1. Set of all comparison operators 2. Set of connectives like and, or, not 3. Set of quantifiers |
| **c** | B-tree in DBMS is an m-way tree which self balances itself. Due to their balanced structure, such trees are frequently used to manage and organise enormous databases and facilitate searches. In a B-tree, each node can have a maximum of n child nodes. In DBMS, B-tree is an example of multilevel indexing. Leaf nodes and internal nodes will both have record references. B Tree is called Balanced stored trees as all the leaf nodes are at same levels.  Following are some of the properties of B-tree in DBMS:  A non-leaf node's number of keys is one less than the number of its children.  The number of keys in the root ranges from one to (m-1) maximum. Therefore, root has a minimum of two and a maximum of m children.  The keys range from min([m/2]-1) to max(m-1) for all nodes (non-leaf nodes) besides the root. Thus, they can have between m and [m/2] children.  The level of each leaf node is the same. |
| **d** | Consistency means that we have to maintain the integrity constraints so that any given database stays consistent both before and after a transaction. If we refer to the example discussed above, then we have to maintain the total amount, both before and after the transaction.  Total after T occurs = 400 + 300 = 700.  Total before T occurs = 500 + 200 = 700.  Thus, the given database is consistent. Here, an inconsistency would occur when T1 completes, but then the T2 fails. As a result, the T would remain incomplete. |
| **e** | The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.  A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.  The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table. |
|  |  |

**PART-B**

**Answer Any Five questions**.

|  |  |  |
| --- | --- | --- |
| **Q.No.** |  | **Questions** |
| **2.** | **a** | In Database Management Systems, integrity constraints are pre-defined set of rules that are applied on the table fields(columns) or relations to ensure that the overall validity, integrity, and consistency of the data present in the database table is maintained. Evaluation of all the conditions or rules mentioned in the integrity constraint is done every time a table insert, update, delete, or alter operation is performed. The data can be inserted, updated, deleted, or altered only if the result of the constraint comes out to be True. Thus, integrity constraints are useful in preventing any accidental damage to the database by an authorized user.  **Types of Integrity Constraints**  There are four types of integrity constraints in DBMS:   1. Domain Constraint 2. Entity Constraint 3. Referential Integrity Constraint 4. Key Constraint   **Domain Constraint**  Domain integrity constraint contains a certain set of rules or conditions to restrict the kind of attributes or values a column can hold in the database table. The data type of a domain can be string, integer, character, DateTime, currency, etc.  **Example:**  Consider a Student's table having Roll No, Name, Age, Class of students.   | **Roll No** | **Name** | **Age** | **Class** | | --- | --- | --- | --- | | 101 | Adam | 14 | 6 | | 102 | Steve | 16 | 8 | | 103 | David | 8 | 4 | | 104 | Bruce | 18 | 12 | | 105 | Tim | 6 | A |   In the above student's table, the value A in the last row last column violates the domain integrity constraint because the Class attribute contains only integer values while A is a character.  **Entity Integrity Constraint**  Entity Integrity Constraint is used to ensure that the primary key cannot be null. A primary key is used to identify individual records in a table and if the primary key has a null value, then we can't identify those records. There can be null values anywhere in the table except the primary key column.  **Example:**  Consider Employees table having Id, Name, and salary of employees   | **ID** | **Name** | **Salary** | | --- | --- | --- | | 1101 | Jackson | 40000 | | 1102 | Harry | 60000 | | 1103 | Steve | 80000 | | 1104 | Ash | 1800000 | |  | James | 36000 |   In the above employee's table, we can see that the ID column is the primary key and contains a null value in the last row which violates the entity integrity constraint.  **Referential Integrity Constraint**  Referential Integrity Constraint ensures that there must always exist a valid relationship between two relational database tables. This valid relationship between the two tables confirms that a foreign key exists in a table. It should always reference a corresponding value or attribute in the other table or be null.  **Example:**  Consider an Employee and a Department table where Dept\_ID acts as a foreign key between the two tables  **Employees Table**   | **ID** | **Name** | **Salary** | **Dept\_ID** | | --- | --- | --- | --- | | 1101 | Jackson | 40000 | 3 | | 1102 | Harry | 60000 | 2 | | 1103 | Steve | 80000 | 4 | | 1104 | Ash | 1800000 | 3 | | 1105 | James | 36000 | 1 |   **Department Table**   | **Dept\_ID** | **Dept\_Name** | | --- | --- | | 1 | Sales | | 2 | HR | | 3 | Technical |   In the above example, Dept\_ID acts as a foreign key in the Employees table and a primary key in the Department table. Row having *DeptID=4* violates the referential integrity constraint since DeptID 4 is not defined as a primary key column in the Departments table.  **Key constraint**  Keys are the set of entities that are used to identify an entity within its entity set uniquely. There could be multiple keys in a single entity set, but out of these multiple keys, only one key will be the primary key. A primary key can only contain unique and not null values in the relational database table.  **Example:**  Consider a student's table   | **Roll No** | **Name** | **Age** | **Class** | | --- | --- | --- | --- | | 101 | Adam | 14 | 6 | | 102 | Steve | 16 | 8 | | 103 | David | 8 | 4 | | 104 | Bruce | 18 | 12 | | 102 | Tim | 6 | 2 |   The last row of the student's table violates the key integrity constraint since Roll No 102 is repeated twice in the primary key column. A primary key must be unique and not null therefore duplicate values are not allowed in the Roll No column of the above student's table. |
| **3.** | **a** | Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output. It performs queries with the help of operators. A binary or unary operator can be used. They take in relations as input and produce relations as output. Recursive relational algebra is applied to a relationship, and intermediate outcomes are also considered relations. Relational Algebra Operations The following are the fundamental operations present in a relational algebra:   * Select Operation * Project Operation * Union Operation * Set Different Operation * Cartesian Product Operation * Rename Operation  Select Operation (or σ) It selects tuples from a relation that satisfy the provided predicate.  **The notation is** − σp(r)  Here **σ** stands for the selection predicate while **r** stands for the relation. p refers to the prepositional logic formula that may use connectors such as **or,** **and,**and **not**. Also, these terms may make use of relational operators such as − =, ≠, ≥, < , >, ≤. Example σsubject = “information”(Novels)  **The output would be** − Selecting tuples from the novels wherever the subject happens to be ‘information’.  σsubject = “information” and cost = “150”(Novels)  **The output would be** − Selecting tuples from the novels wherever the subject happens to be ‘information’ and the ‘price’ is 150.  σsubject = “information” and cost = “150” or year > “2015”(Novels)  **The output would be** − Selecting tuples from the novels wherever the subject happens to be ‘information’ and the ‘price’ is 150 or those novels have been published after 2015. Project Operation (or ∏) It projects those column(s) that satisfy any given predicate.  Here B1, B2 , An refer to the attribute names of the relation **r**.  **The notation is** − ∏B1, B2, Bn (r)  Remember that duplicate rows are eliminated automatically, since relation is a set. Example ∏subject, writer (Novels)  **The output would be** − Selecting and projecting columns named as writer as well as the subject from the relation Novels. Union Operation (or ∪) It would perform binary union between two relations.  **The notation is** − r U s  It is defined as follows:  r ∪ s = { t | t ∈ r or t ∈ s}  Here **r** and **s** either refer to DB relations or the relation result set (or temporary relation).  The given conditions must hold if we want any union operation to be valid:   * **s**, and **r** must contain a similar number of attributes. * The domains of an attribute must be compatible. * The duplicate tuples are eliminated automatically.   ∏ writer (Novels) ∪ ∏ writer (Articles)  **The output would be** − Projecting the names of those writers who might have written either an article or a novel or both. Set Different Operation (or −) Tuples refers to the result of the set difference operation. These are present in just one of the relations but not at all in the second one.  **The notation is** − r − s  Finding all the tuples present in **r** and not present in **s**.  ∏ writer (Novels) − ∏ writer (Articles)  **The output would be** − Providing the writer names who might have written novels but have not written articles. Cartesian Product Operation (or Χ) It helps in combining data and info of two differing relations into a single one.  **The notation is** − r Χ s  Where **s** and **r** refer to the relations. Their outputs would be defined as the follows:  s Χ r = { t ∈ s and q t | q ∈ r}  σwriter = ‘mahesh'(Novels Χ Articles)  **The output would be** − Yielding a relation that shows all the articles and novels written by mahesh. Rename Operation (or ρ) Relations are the results of the relational algebra, but without any name. Thus, the rename operation would allow us to rename the relation output. The ‘rename’ operation is basically denoted by the small Greek letter ρ or **rho**.  **The notation is** − ρ x (E)  Here the result of the E expression is saved with the name of **x**.  The additional operations are as follows:   * Natural join * Assignment * Set intersection |
| **4.** | **a** | Hashing is a DBMS technique for searching for needed data on the disc without utilising an index structure. The hashing method is basically used to index items and retrieve them in a DB since searching for a specific item using a shorter hashed key rather than the original value is faster.  It can be nearly hard to search all index values through all levels of a large database structure and then get to the target data block to obtain the needed data. Hashing is a method for calculating the direct position of an information record on the disk without the use of an index structure.  To generate the actual address of a data record, hash functions containing search keys as parameters are used. Properties of Hashing in DBMS Data is kept in data blocks whose addresses are produced using the hashing function in this technique. Data buckets or data blocks are the memory locations where these records are stored.  In this case, a hash function can produce the address from any column value. The primary key is frequently used by the hash function to generate the data block’s address. To every complex mathematical function, a hash function is a basic mathematical function. The primary key can also be considered as the data block’s address, i.e. each row with the same address as a primary key contained in the data block.  Hashing-in-DBMS  The data block addresses are the same as the primary key value in the picture above. This hash function could alternatively be a simple mathematical function, such as exponential, mod, cos, sin, and so on. Assume we’re using the mod (5) hash function to find the data block’s address. In this scenario, the primary keys are hashed with the mod (5) function, yielding 3, 3, 1, 4, and 2, respectively, and records are saved at those data block locations.  Hashing-in-DBMS-1  **Bucket –**A bucket is a type of storage container. Data is stored in bucket format in a hash file. Typically, a bucket stores one entire disc block, which can then store one or more records.  **Hash Function –**A hash function, abbreviated as h, refers to a mapping function that connects all of the search-keys K to that address in which the actual records are stored. From the search keys to the bucket addresses, it’s a function. Types of Hashing Hashing is of the following types:  Hashing-in-DBMS-2 Static Hashing Whenever a search-key value is given in static hashing, the hash algorithm always returns the same address. If the mod-4 hash function is employed, for example, only 5 values will be generated. For this function, the output address must always be the same. At all times, the total number of buckets available remains constant. Dynamic Hashing The disadvantage of static hashing is that it doesn’t expand or contract dynamically as the database size grows or diminishes. Dynamic hashing is a technology that allows data buckets to be created and withdrawn on the fly. Extended hashing is another name for dynamic hashing.  In dynamic hashing, the hash function is designed to output a huge number of values, but only a few are used at first. |
| **5.** | **a** | **Database systems**, like any other computer system, are subject to failures but the data stored in them must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transaction are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database. There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect commands execution, etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used. Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur during the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.   * The log is kept on disk start\_transaction(T): This log entry records that transaction T starts the execution. * read\_item(T, X): This log entry records that transaction T reads the value of database item X. * write\_item(T, X, old\_value, new\_value): This log entry records that transaction T changes the value of the database item X from old\_value to new\_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X. * commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database. * abort(T): This records that transaction T has been aborted. * checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in a consistent state, and all the transactions were committed.   A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start\_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process.   * **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write\_item(T, x, old\_value, new\_value) and set the value of item x in the database to old-value. There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates. * **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm** * **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.** * **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under the control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified. * **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to the original are updated to refer new replacement page. * **Backward Recovery** – The term “Rollback ” and “UNDO” can also refer to backward recovery. When a backup of the data is not available and previous modifications need to be undone, this technique can be helpful. With the backward recovery method, unused modifications are removed and the database is returned to its prior condition. All adjustments made during the previous traction are reversed during the backward recovery. In another word, it reprocesses valid transactions and undoes the erroneous database updates. * **Forward Recovery** – “Roll forward “and “REDO” refers to forwarding recovery. When a database needs to be updated with all changes verified, this forward recovery technique is helpful. Some failed transactions in this database are applied to the database to roll those modifications forward. In another word, the database is restored using preserved data and valid transactions counted by their past saves.   Some of the backup techniques are as follows :   * **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series. * **Differential backup –** It stores only the data changes that have occurred since the last full database backup. When some data has changed many times since last full database backup, a differential backup stores the most recent version of the changed data. For this first, we need to restore a full database backup. * **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transactions that had happened to the database. Through this, the database can be recovered to a specific point in time. It is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost. |
| **b** |  |
| **6.** | **a** | Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.  The available constraints in SQL are:    * **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more. * **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated. * **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key. * **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key. * **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition. * **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.   **How to specify constraints?**  We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.  **Syntax**:  Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.  CREATE TABLE sample\_table  (  column1 data\_type(size) constraint\_name,  column2 data\_type(size) constraint\_name,  column3 data\_type(size) constraint\_name,  ....  );  **sample\_table**: Name of the table to be created.  **data\_type**: Type of data that can be stored in the field.  **constraint\_name**: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.  **1. NOT NULL –**  If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.  For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.  **2. UNIQUE** **–** This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.  For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID  **3. PRIMARY KEY –**  Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.  A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.  **4. FOREIGN KEY –**  Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables. |
| **b** |  |
| **7.** | **a** | DDL is used as an abbreviation for Data Definition Language. DDL refers to a computer language that is primarily used for creating as well as modifying the structure of the database objects present in a database. Such database objects include indexes, tables, schemas, views, and many more.  DDL is also referred to as a data description language in certain contexts since it describes the records and fields in the DB (database) tables.  The current database industry basically incorporates DDL into the formal language that describes data. However, DDL is often considered a subset of Structured Query Language or SQL. Now, SQL often makes use of imperative verbs, along with normal English (like sentences) in order to implement database modifications. Thus, DDL doesn’t really show up to be a different language in the database of SQL, but it does define all the changes present in the DB schema.  It is mainly used to modify and establish the structure of the objects present in a database by dealing with the database schema descriptions. Unlike DML (data manipulation language) commands that we use for data modification, DDL commands are actually used to alter the structure of the database, like creating new objects or tables along with their attributes (table name, data type, etc.).  CREATE, DROP, ALTER, and TRUNCATE are commonly used in Data Definition Languages in SQL querying. Let’s discuss these in detail. Create The CREATE command builds a new table. It has a predefined syntax, and the CREATE statement syntax goes like this:  CREATE TABLE [name\_of\_table] ([definitions\_of\_column]) [parameters\_of\_table];  For instance,  CREATE TABLE Student (Student Id INTEGER PRIMARY KEY, Name CHAR (50) NULL, Nickname CHAR (75) NOT NULL);  Here, the semi-colon used at the end of the statement is mandatory, and it is used to process every command before it. So, in the example mentioned above, we are trying to specify the data type using the string CHAR. Visit to learn more about the CREATE statement here. Various other data types are NUMBER, INTEGER or DATE. Alter The alter command modifies any existing table in a database. The Alter command can add up an additional column, drop the existing columns and even change the data types of various columns involved in a DB table. Read more on the ALTER command here.  The syntax of an ALTER command is:  ALTER type\_of\_object name\_of\_object parameters;  For instance,  ALTER TABLE Student ADD PRIMARY KEY (student\_pk);  In this example, we have added the unique primary key to this table. It adds a constraint and enforces a unique value. Here, the constraint “student\_pk” is the primary key, and it is on the Student table. Drop The DROP command is used in order to delete objects, like a table, view or index. We cannot rollback the DROP statement. Thus, once a certain object is destroyed, there would be no way at all to recover it.  The syntax of the DROP statement is:  DROP type\_of\_object name\_of\_object;  For instance,  DROP TABLE Student;  We have deleted the Employee table in this example. Truncate Just like the DROP statement, the TRUNCATE statement is mainly used to remove all the records from a table quickly. However, unlike the DROP statement that destroys a table completely, the TRUNCATE statement preserves the full structure so as to be reused later.  The syntax of the TRUNCATE statement is:  TRUNCATE TABLE name\_of\_table;  For instance,  TRUNCATE TABLE Student; |
| **b** | |  |  | | --- | --- | | DDL | DML | | It stands for Data Definition Language. | It stands for Data Manipulation Language. | | It is used to create database schema and can be used to define some constraints as well. | It is used to add, retrieve or update the data. | | It basically defines the column (Attributes) of the table. | It add or update the row of the table. These rows are called as tuple. | | It doesn’t have any further classification. | It is further classified into Procedural and Non-Procedural DML. | | Basic command present in DDL are CREATE, DROP, RENAME, ALTER etc. | BASIC command present in DML are UPDATE, INSERT, MERGE etc. | | DDL does not use WHERE clause in its statement. | While DML uses WHERE clause in its statement. | |
| **8.** | **a** | **Atomicity**. Either all operations of the transaction are reflected properly in the database, or none are. • **Consistency**. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database. • **Isolation**. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj , it appears to Ti that either Tj finished execution before Ti started or Tj started execution after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system. • **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures  These properties are often called the ACID properties; the acronym is derived from the first letter of each of the four properties. As we shall see later, ensuring the isolation property may have a significant adverse effect on system performance. For this reason, some applications compromise on the isolation property. We shall study these compromises after first studying the strict enforcement of the ACID properties. |
| **b** | Several current trends in the field of computing are giving rise to an increase in the amount of concurrency possible. As database systems exploit this concurrency to increase overall system performance, there will necessarily be an increasing number of transactions run concurrently. Early computers had only one processor. Therefore, there was never any real concurrency in the computer. The only concurrency was apparent concurrency created by the operating system as it shared the processor among several distinct tasks or processes. Modern computers are likely to have many processors. These may be truly distinct processors all part of the one computer. However even a single processor may be able to run more than one process at a time by having multiple cores. The Intel Core Duo processor is a well-known example of such a multicore processor. For database systems to take advantage of multiple processors and multiple cores, two approaches are being taken. One is to find parallelism within a single transaction or query. Another is to support a very large number of concurrent transactions. Many service providers now use large collections of computers rather than large mainframe computers to provide their services. They are making this choice based on the lower cost of this approach. A result of this is yet a further increase in the degree of concurrency that can be supported. The bibliographic notes refer to texts that describe these advances in computer architecture and parallel computing. Chapter 18 describes algorithms for building parallel database systems, which exploit multiple processors and multiple cores |
| **9.** | **a** | Suppose that the database administrator decides to revoke the authorization of user U1. Since U4 has authorization from U1, that authorization should be revoked as well. However, U5 was granted authorization by both U1 and U2. Since the database administrator did not revoke update authorization on teaches from U2, U5 retains update authorization on teaches. If U2 eventually revokes authorization from U5, then U5 loses the authorization. A pair of devious users might attempt to defeat the rules for revocation of authorization by granting authorization to each other. For example, if U2 is initially granted an authorization by the database administrator, and U2 further grants it to U3. Suppose U3 now grants the privilege back to U2. If the database administrator revokes authorization from U2, it might appear that U2 retains authorization through U3. However, note that once the administrator revokes authorization from U2, there is no path in the authorization graph from the root to either U2 or to U3. Thus, SQL ensures that the authorization is revoked from both the users. As we just saw, revocation of a privilege from a user/role may cause other users/roles also to lose that privilege. This behavior is called cascading revocation. In most database systems, cascading is the default behavior. However, the revoke statement may specify restrict in order to prevent cascading revocation: revoke select on department from Amit, Satoshi restrict;  In this case, the system returns an error if there are any cascading revocations, and does not carry out the revoke action. The keyword cascade can be used instead of restrict to indicate that revocation should cascade; however, it can be omitted, as we have done in the preceding examples, since it is the default behavior. The following revoke statement revokes only the grant option, rather than the actual select privilege:  *revoke grant option for select on department from Amit;*  Note that some database implementations do not support the above syntax; instead, the privilege itself can be revoked, and then granted again without the grant option. Cascading revocation is inappropriate in many situations. Suppose Satoshi has the role of dean, grants instructor to Amit, and later the role dean is revoked from Satoshi (perhaps because Satoshi leaves the university); Amit continues to be employed on the faculty, and should retain the instructor role. To deal with the above situation, SQL permits a privilege to be granted by a role rather than by a user. SQL has a notion of the current role associated with a session. By default, the current role associated with a session is null (except in some special cases). The current role associated with a session can be set by executing set role role name. The specified role must have been granted to the user, else the set role statement fails. To grant a privilege with the grantor set to the current role associated with a session, we can add the clause:  **granted by current role**  to the grant statement, provided the current role is not null. Suppose the granting of the role instructor (or other privileges) to Amit is done using the granted by current role clause, with the current role set to dean), instead of the grantor being the user Satoshi. Then, revoking of roles/privileges (including the role dean) from Satoshi will not result in revoking of privileges that had the grantor set to the role dean, even if Satoshi was the user who executed the grant; thus, Amit would retain the instructor role even after Satoshi’s privileges are revoked. |
| **b** | The SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations ∪, ∩, and −. We shall now construct queries involving the union, intersect, and except operations over two sets.      **The Union Operation**  To find the set of all courses taught either in Fall 2009 or in Spring 2010, or both, we write:    The union operation automatically eliminates duplicates, unlike the select clause. Thus, using the section relation of Figure 2.6, where two sections of CS-319 are offered in Spring 2010, and a section of CS-101 is offered in the Fall 2009 as well as in the Fall 2010 semester, CS-101 and CS-319 appear only once in the result.  **The Intersect Operation**  To find the set of all courses taught in the Fall 2009 as well as in Spring 2010 we write    The result relation, shown in Figure 3.12, contains only one tuple with CS-101. The intersect operation automatically eliminates duplicates. For example, if it were the case that 4 sections of ECE-101 were taught in the Fall 2009 semester and 2 sections of ECE-101 were taught in the Spring 2010 semester, then there would be only 1 tuple with ECE-101 in the result.  **The Except Operation**  To find all courses taught in the Fall 2009 semester but not in the Spring 2010 semester, we write    The result of this query is shown in Figure 3.13. Note that this is exactly relation c1 of Figure 3.9 except that the tuple for CS-101 does not appear. The except operation7 outputs all tuples from its first input that do not occur in the second input; that is, it performs set difference. The operation automatically eliminates duplicates in the inputs before performing set difference. For example, if 4 sections of ECE-101 were taught in the Fall 2009 semester and 2 sections of ECE-101 were taught in the Spring 2010 semester, the result of the except operation would not have any copy of ECE-101. |

**\*\*\*\*\*\***