

**S.Y.B.Sc. I.T.**

**DBMS Solution Set, October 2018**

|      |  |
|------|--|
| 1 a  | What is the role of a DBMS, and what are its advantages? What are its disadvantages?   |
| Ans. | <p><i>A database management system (DBMS) is a collection of programs that manage the database structure and controls access to the data stored in the database”.</i></p> <p>The DBMS serves as the intermediate between the user and the database. The database structure is stored as a collection of files. These data can be accessed in those files through the DBMS. The DBMS hides much of the database’s internal complexity from the application programs and users.</p> <p><b>Advantages of DBMS</b></p> <p>The database management system has promising potential advantages, which are explained below:</p> <p><b>1. Controlling Redundancy:</b> In file system, each application has its own private files, which cannot be shared between multiple applications. This can often lead to considerable redundancy in the stored data, which results in wastage of storage space. By having centralized database most of this can be avoided. It is not possible that all redundancy should be eliminated. Sometimes there are sound business and technical reasons for maintaining multiple copies of the same data. In a database system, however this redundancy can be controlled.</p> <p><b>2. Integrity can be enforced:</b> Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database. A DBMS should provide capabilities for defining and enforcing the constraints.</p> <p><b>3. Inconsistency can be avoided :</b> When the same data is duplicated and changes are made at one site, which is not propagated to the other site, it gives rise to inconsistency and the two entries regarding the same data will not agree. At such times the data is said to be inconsistent. So, if the redundancy is removed chances of having inconsistent data is also removed.</p> <p><b>4. Data can be shared:</b> As explained earlier, the data about Name, Class, Father __name etc. of General_Office is shared by multiple applications in centralized DBMS as compared to file system so now applications can be developed to operate against the same stored data. The applications may be developed without having to create any new stored files.</p> <p><b>5. Standards can be enforced :</b> Since DBMS is a central system, so standard can be enforced easily may be at Company level, Department level, National level or International level. The standardized data is very helpful during migration or</p> |

interchanging of data. The file system is an independent system so standard cannot be easily enforced on multiple independent applications.

**6. Restricting unauthorized access:** When multiple users share a database, it is likely that some users will not be authorized to access all information in the database. For example, account office data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas other are allowed both to retrieve and to update. Hence, the type of access operation retrieval or update must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically.

**7. Solving Enterprise Requirement than Individual Requirement:** Since many types of users with varying level of technical knowledge use a database, a DBMS should provide a variety of user interface. The overall requirements of the enterprise are more important than the individual user requirements. So, the DBA can structure the database system to provide an overall service that is "best for the enterprise".

For example: A representation can be chosen for the data in storage that gives fast access for the most important application at the cost of poor performance in some other application. But, the file system favors the individual requirements than the enterprise requirements

**8. Providing Backup and Recovery:** A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the .database is restored to the state it was in before the program started executing.

**9. Cost of developing and maintaining system is lower:** It is much easier to respond to unanticipated requests when data is centralized in a database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, but the cost of developing and maintaining application programs to be far lower than for similar service using conventional systems. The productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.

**10. Data Model can be developed :** The centralized system is able to represent the complex data and interfile relationships, which results better data modeling properties. The data madding properties of relational model is based on Entity and their Relationship, which is discussed in detail in chapter 4 of the book.

**11. Concurrency Control :** DBMS systems provide mechanisms to provide concurrent access of data to multiple users.

### **Disadvantages of DBMS**

The disadvantages of the database approach are summarized as follows:

|  |  |
|--|--|
|  | <p><b>1. Complexity :</b> The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.</p> <p><b>2. Size :</b> The complexity and breadth of functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.</p> <p><b>3. Performance:</b> Typically, a File Based system is written for a specific application, such as invoicing. As result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.</p> <p><b>4. Higher impact of a failure:</b> The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the ~vailability of the DBMS, the failure of any component can bring operations to a halt.</p> <p><b>5. Cost of DBMS:</b> The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recurrent annual maintenance cost.</p> <p><b>6. Additional Hardware costs:</b> The disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.</p> <p><b>7. Cost of Conversion:</b> In some situations, the cost of ttle DBMS and extra hardware may be insignificant compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to modern database technology.</p> |
|--|--|

|      |  |
|------|--|
| 1 b  | Explain Storage system and query processor components of database structure.   |
| Ans. | <p><b>Storage Manager</b></p> <p>The <i>storage manager</i> is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands.</p> |

|  |   |
|--|---|
|  | <p>Thus, the storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include:</p> <ul style="list-style-type: none"> <li>• Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.</li> <li>• <b>Transaction manager</b>, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.</li> <li>• <b>File manager</b>, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.</li> <li>• <b>Buffer manager</b>, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.</li> </ul> <p>The storage manager implements several data structures as part of the physical system implementation:</p> <ul style="list-style-type: none"> <li>• <b>Data files</b>, which store the database itself.</li> <li>• <b>Data dictionary</b>, which stores metadata about the structure of the database, in particular the schema of the database.</li> <li>• <b>Indices</b>, which can provide fast access to data items.</li> </ul> <p><b>The Query Processor</b></p> <p>The query processor components include:</p> <ul style="list-style-type: none"> <li>• <b>DDL interpreter</b>, which interprets DDL statements and records the definitions in the data dictionary.</li> <li>• <b>DML compiler</b>, which translates DML statements in a query language into an Evaluation plan consisting of low-level instructions that the query evaluation Engine understands. A query can usually be translated into any of a number of alternative Evaluation plans that all give the same result. The DML compiler also performs <b>Query optimization</b>; that is, it picks the lowest cost evaluation plan from Among the alternatives.</li> <li>• <b>Query evaluation engine</b>, which executes low-level instructions generated by the DML compiler.</li> </ul> |
|--|---|

|      |   |
|------|---|
| 1 c  | What is a business rule, and what is its purpose in data modeling?  |
| Ans. | <p>When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they might start by gaining thorough understanding of what types of data exist in an organization, how the data is used, and in what time frames it is used. But such data and information do not, by themselves, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when it reflects properly defined <i>business rules</i>. A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business</p> |

rules are misnamed: they apply to *any* organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information. Business rules derived from a detailed description of an organization’s operations help to create and enforce actions within that organization’s environment. Business rules must be rendered in writing and updated to reflect any change in the organization’s operational environment.

Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as “an agent can serve many customers, and each customer can be served by only one agent,” business rules area work. To be effective, business rules must be easy to understand and widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data *as viewed by the company*. Examples of business rules are as follows:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities (CUSTOMER and INVOICE) and a 1: M relationship between those two entities. The third business rule establishes a constraint (no fewer than 10 people and no more than 30 people), two entities (EMPLOYEE and TRAINING), and also implies a relationship between EMPLOYEE And TRAINING.

When database designers selecting / determining the entities, Attributes, relationships etc. That are used to build a Data model with a thorough understands.

What types of data are used in an organization.

How the data are used.

What reports that they need after processing?

From the database point of view, the collection of data becomes meaningful only when it reflects properly defined business rules.

Thus, a business rule is a brief, precise and unambiguous description of a Policy, Procedure, or Principle within a specific organization. Business rules, thus, derived from a detailed description of an Organization’s operations, that help to create and enforce actions within that organization’s environment. To be effective, Business rules must be in written, properly defined, easy to understand to every person in the Organization., shares a common interpretation of the rules. Some of the Example of Business rules are,

- i) A customer may generate many invoices
- ii) An invoice is generated by only one customer

|  |   |
|--|---|
|  | <p>iii) A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees., etc.</p> <p><b>Use of Business Rules:</b></p> <p>The business rules are essential to Database designer for several reasons, such as</p> <ul style="list-style-type: none"> <li>a) They help standardize the Company's view of data</li> <li>b) They can be a communication tool in between users and designers</li> <li>c) They allow the designer to understand the nature, role and scope of Data</li> <li>d) They allow the designer to understand business processes</li> </ul> <p>They allow the DB designer to understand to develop appropriate relationship participation rules and constraints to create an accurate data model....so on.</p> |
|--|---|

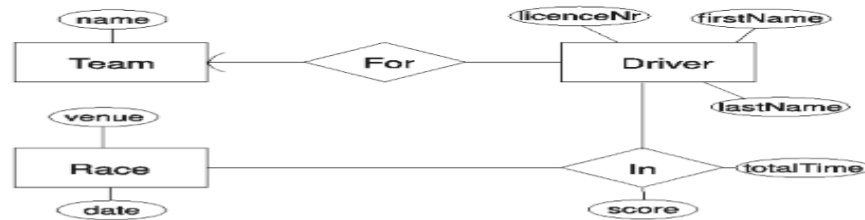
|      |  |
|------|--|
| 1 d  | Give comparison between hierarchical, network & relational model.  |
| Ans. | <p>The hierarchical model is not like the other two models you asked about. In the hierarchical model, data is stored in a defined hierarchy. For instance, a company is made of departments and each department has employees. So a tree structure is created with the company at the root of the tree. To get to all employees in the company, one would have to traverse the entire tree. Click here for more information on <a href="#">the hierarchical model</a>. The hierarchical model does exist today, but typically in legacy applications.</p> <p>The network model was the first attempt to address the inefficiencies of the hierarchical model. In the network model, you could create a network showing how data related to each other. The network model never caught on, and was eventually replaced by the relational model. Click here for more information on <a href="#">the network model</a>.</p> <p>The relational model has proven to be the most efficient and most flexible database model in use today. There are many advantages of the relational model over the other models, which is why the most popular databases in use today employ this methodology. Click here for more information on <a href="#">the relational model</a>.</p> |

|     |  |  |  |
|-----|--|--|--|
| 1 e | List and explain Codd's rule in detail.        |  |  |
| Ans | <b>DR. CODD'S 12 RELATIONAL DATABASE RULES</b> |  |  |
|     | <b>RULE</b>                                    | <b>RULE NAME</b>   | <b>DESCRIPTION</b>   |
|     | 1  | Information  | All information in a relational database must be logically represented column values in rows within tables.  |
|     | 2  | Guaranteed access  | Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.   |
|     | 3  | Systematic treatment of nulls  | Nulls must be represented and treated in a systematic way, independent of data type.   |
|     | 4  | Dynamic online catalog based on the relational model   | The metadata must be stored and managed as ordinary data—that is, tables within the database; such data must be available to authorized users using the standard database relational language.   |
|     | 5  | Comprehensive data sublanguage   | The relational database may support many languages; however, it must support one well-defined, declarative language as well as data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback). |
|     | 6  | View updating  | Any view that is theoretically updatable must be updatable through the system.   |
|     | 7  | High-level insert, update, and delete  | The database must support set-level inserts, updates, and deletes.   |
|     | 8  | Physical data independence   | Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.  |
|     | 9  | Logical data independence  | Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).  |
|     | 10   | Integrity independence   | All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.  |
|     | 11   | Distribution independence  | The end users and application programs are unaware of and unaffected by the data location (distributed vs. local databases).   |
| 12  | Nonsubversion                                  | If the system supports low-level access to the data, users must not be allowed to bypass the integrity rules of the database.                                |  |
| 13  | Rule zero                                      | All preceding rules are based on the notion that to be considered relational, a database must use its relational facilities exclusively for data management. |  |

|      |   |
|------|---|
| 1 f  | <p>Explain ER diagram and its components. Give the distinction between disjoint, overlapping, total and partial constraints. Draw E-R diagram for the following situations that correctly models this domain and its constraints.</p> <p>A small racing league want a database to keep track of teams, drivers, races and scores in the league. The league is run for teams, which are identified by their names. Each team has one or more drivers signed up, and each driver is registered with the league and has a unique league licence number. First and last names of the drivers should also be included. A driver may only participate for a single team throughout the season. Races are identified simply by the dates when they are run. For each race, the league also wants to store the venue where it took place. Drivers participate in races, and for each participating driver the database should store the total race time for that driver, and the league score they got from that race.</p>  |
| Ans. | <p>An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear—qualities that may well account in large part for the widespread use of the E-R model.</p> <p><b>An E-R diagram consists of the following major components:</b></p> <p><b>Rectangles</b> divided into two parts represent entity sets. The first part, which in this textbook is shaded blue, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.</p> <ul style="list-style-type: none"> <li>• <b>Diamonds</b> represent relationship sets.</li> <li>• <b>Undivided rectangles</b> represent the attribute so far relationship set. Attributes that are part of the primary key are underlined.</li> <li>• <b>Lines</b> link entity sets to relationship sets.</li> <li>• <b>Dashed lines</b> link attributes of a relationship set to the relationship set.</li> <li>• <b>Double lines</b> indicate total participation of an entity in a relationship set.</li> <li>• <b>Double diamonds</b> represent identifying relationship sets linked to weak entity sets.</li> </ul> <p><b>Disjoint.</b> A <i>disjointness constraint</i> requires that an entity belong to no more than one lower-level entity set. In our example, <i>student</i> entity can satisfy only one condition for the <i>student type</i> attribute; an entity can be either a graduate student or an undergraduate student, but cannot be both.</p> <p>• <b>Overlapping.</b> In <i>overlapping generalizations</i>, the same entity may belong to more than one lower-level entity set within a single generalization. For an e.g., consider the employee work-team example, and assume that certain employees participate in more than one work team. Given employee May therefore appear in more than one of the team entity sets that are lower level Entity sets of <i>employee</i>. Thus, the generalization is overlapping.</p> <p>Final constraint, the <b>completeness constraint</b> on a generalization or specialization, Specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization.</p> <p><b>This constraint may be one of the following:</b></p> <ul style="list-style-type: none"> <li>• <b>Total generalization or specialization.</b> Each higher-level entity must belong to a lower-level entity set.</li> </ul> |




• **Partial generalization or specialization.** Some higher-level entities may not belong to any lower-level entity set.



|      |  |
|------|--|
| 2 a  | Why are entity integrity and referential integrity important in a database? Explain in detail.   |
| Ans. | Referential and entity integrity are crucial to preserving valid relationships between tables and data within a database. SQL queries will begin to fail if the data keys that connect the dots between their relationships do not match. If an entity or table is relying on the keys in another entity or table, then relationships between the two can be lost if bad data is entered into one location. For instance, referential integrity can be used to ensure foreign key values are valid. For instance, a database table listing all the parts installed on a specific aircraft should have referential integrity connecting the part numbers to a table listing valid part numbers for that aircraft so that in the event of a bad part number being “fat-fingered” into the database, the RBDMS will return an error concerning the bad data . |

| INTEGRITY RULES              |  |
|------------------------------|--|
| ENTITY INTEGRITY             | DESCRIPTION  |
| Requirement                  | All primary key entries are unique, and no part of a primary key   |
| Purpose                      | Each row will have a unique identity, and foreign key values can't be primary key values.  |
| Example                      | No invoice can have a duplicate number, nor can it be null; in short, each invoice is uniquely identified by their invoice number.   |
| REFERENTIAL INTEGRITY        | DESCRIPTION  |
| Requirement                  | A foreign key may have either a null entry, as long as it is not a primary key, or an entry that matches the primary key value in the table it is related; (every non-null foreign key value <i>must</i> reference an existing primary key value).   |
| Purpose                      | It is possible for an attribute <i>not</i> to have a corresponding value, but it is impossible to have an invalid entry; the enforcement of the referential integrity makes it impossible to delete a row in one table whose primary key has matching foreign key values in another table. |
| Example                      | A customer might not yet have an assigned sales representative, but it would be impossible to have an invalid sales representative (number).   |
| ENTITY INTEGRITY DESCRIPTION |  |

|      |  |
|------|--|
| 2 b  | <p>Explain why normalization is necessary in database system &amp; also explain database anomalies in detail.</p> <p>You are given the following set of functional dependencies for a relation <math>R(A,B,C,D,E,F)</math>,</p> $F = \{ AB \rightarrow C, DC \rightarrow AE, E \rightarrow F \}$ <p>a. What are the keys of this relation?</p> <p>b. Is this relation in BCNF? If not, explain why by showing one violation.</p> <p>c. Is the decomposition <math>(A, B, C, D) (B, C, D, E, F)</math> a dependency preserving decomposition? If not, explain briefly.</p>  |
| Ans. | <p><b>Database normalization</b> is the process of organizing the attributes and tables of a relational <b>database</b> to minimize data redundancy.</p> <p>If a database design is not perfect it may contain anomalies, which are like a bad dream for database itself. Managing a database with anomalies is next to impossible. Normalization is a method to remove all these anomalies and bring database to consistent state and free from any kinds of anomalies.</p> <p>At a basic level, normalization is the simplification of any bulk quantity to an optimum value. In the digital world, normalization usually refers to database normalization which is the process of organizing the columns (attributes) and tables (relations) of a relational database to minimize data repetition. In the process of database creation, normalization involves organizing data into optimal tables in such a way that the results obtained are always unambiguous and clear in concept.</p> |

|  |  |
|--|--|
|  | <p>Though database normalization can have the effect of duplication of data, it completely removes data redundancy. This process can be considered as a refinement process after the initial identification of data objects that are to be included in the database. It involves identification of the relationship between the data objects and defining the tables required and the columns to be added within each table.</p> <p>If a database design is not done properly, it may cause several anomalies to occur in it. Normalization is essential for removing various anomalies like:</p> <p><b>Anomalies in Database</b></p> <p>1) Update Anomalies: When several instances of the same data are scattered across the database without proper relationship/link, it could cause strange conditions where a few of the instances will get updated with new values whereas some of them will not. This leaves the database in an inconsistent state.</p> <p>2) Deletion Anomalies: Incomplete deletion of a particular data section which leaves some residual instances. The database creator remains unaware of such unwanted data as it is present at a different location.</p> <p>3) Insertion Anomalies: This occurs when an attempt to insert data into a non-existent record.</p> <p>Paying attention to these anomalies can help to maintain a consistent database.</p> <p><b>ANSWER:</b></p> <p>a. What are the keys of this relation?<br/> <math>\{A, B, D\}</math> and <math>\{B, C, D\}</math>.</p> <p>b. Is this relation in BCNF? If not, explain why by showing one violation.<br/> No, all functional dependencies are actually violating this. No dependency contains a superkey on its left side.</p> <p>c. Is the decomposition <math>(A, B, C, D)</math> <math>(B, C, D, E, F)</math> a dependency preserving decomposition? If not, explain briefly. <br/> Yes, <math>AB \rightarrow C</math> and <math>DC \rightarrow A</math> are preserved in the first relation. <math>DC \rightarrow E</math> and <math>E \rightarrow F</math> are preserved in the second relation.</p> |
|--|--|

|      |  |
|------|--|
| 2 c  | Write short note on Cartesian product with its syntax and example.   |
| Ans. | The <b>Cartesian-product</b> operation, denoted by a cross ( $\times$ ), allows us to combine information from any two relations. We write the Cartesian product of relations $r_1$ and $r_2$ as $r_1 \times r_2$ . A relation is by definition a subset of a Cartesian product of a set of domains. From that definition, we should already have an intuition about the definition of the Cartesian-product operation. However, since the same attribute name may appear in both $r_1$ and $r_2$ , we need to devise a naming schema to distinguish between these |

|  |  |
|--|--|
|  | <p>attributes. We do so here by attaching to an attribute the name of the relation from which the attribute originally came.</p> <p>For example, the relation schema for <math>r = instructor \times teaches</math> is:<br/> <i>(instructor.ID, instructor.name, instructor.dept name, instructor.salary teaches.ID, teaches.course id, teaches.sec id, teaches.semester, teaches.year)</i></p> <p>With this schema, we can distinguish <i>instructor.ID</i> from <i>teaches.ID</i>. For those attributes that appear in only one of the two schemas, we shall usually drop the relation-name prefix. This simplification does not lead to any ambiguity. We can then write the relation schema for <math>r</math> as:<br/> <i>(instructor.ID, name, dept name, salary teaches.ID, course id, sec id, semester, year)</i></p> <p>This naming convention <i>requires</i> that the relations that are the arguments of the Cartesian-product operation have distinct names.</p> <p><b>SYNTAX:--</b> (QUERY1) * (QUERY2)<br/> Let there be relation A(A1,A2) and relation B(B1,B2)<br/> <b>THE CARTESIAN PRODUCT C</b> of A and B which is <math>A * B</math> is<br/> <math>C = A * B</math><br/> <math>C = (A1B1, A1B2, A2B1, A2B2)</math></p> |
|--|--|

|      |   |
|------|---|
| 2 d  | Explain SET operators in details along with example.  |
| Ans. | <p>Set operators allows combine results from two or more SELECT statement. SQL set operators combine rows from different queries with strong preconditions-all involved SELECTS must:--</p> <ol style="list-style-type: none"> <li>i. Retrieve the same number of columns and</li> <li>ii. The datatypes of corresponding columns in each involved SELECT must be compatible.</li> <li>iii. Set operators are:-- UNION, INTERSECT, DIFFERENCE OR MINUS</li> </ol> <p><b>Union Operation (U)</b></p> <p>It performs binary union between two given relations and is defined as –</p> $r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$ <p><b>Notation</b> – <math>r \cup s</math></p> <p>Where <b>r</b> and <b>s</b> are either database relations or relation result set (temporary relation).</p> <p>For a union operation to be valid, the following conditions must hold –</p> |

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{\text{author}}(\text{Books}) \cup \prod_{\text{author}}(\text{Articles})$

**Intersect operator ( $\cap$ ):**

The intersect operators is denoted by the symbol ( $\cap$ ). The sql intersect operator is used to combine two select statement but retrurns rows only from the first select statement that are identical to a row in the second select statement. This means INTERSECT returns only common rows returned by the two select statement.

**Notation –  $r \cap s$**

$\prod_{\text{author}}(\text{Books}) \cap \prod_{\text{author}}(\text{Articles})$

**Set Difference ( $-$ )**

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

**Notation –  $r - s$**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{\text{author}}(\text{Books}) - \prod_{\text{author}}(\text{Articles})$

|     |  |
|-----|--|
| 2 e | Explain formal definitions with safety of expressions of tuples relational calculus.   |
| Ans | <p><b>Formal Definition :</b></p> <p>We are now ready for a formal definition. A tuple-relational-calculus expression is of the form:</p> $\{t \mid P(t)\}$ <p>where <math>P</math> is a <i>formula</i>. Several tuple variables may appear in a formula. A tuple variable is said to be a <i>free variable</i> unless it is quantified by a <math>\exists</math> or <math>\forall</math>. Thus, in:</p> $t \in instructor \wedge \exists s \in department(t[dept\_name] = s[dept\_name])$ <p><math>t</math> is a free variable. Tuple variable <math>s</math> is said to be a <i>bound</i> variable.</p> <p>A tuple-relational-calculus formula is built up out of <i>atoms</i>. An atom has one of the following forms:</p> <ul style="list-style-type: none"> <li>• <math>s \in r</math>, where <math>s</math> is a tuple variable and <math>r</math> is a relation (we do not allow use of the <math>\notin</math> operator).</li> <li>• <math>s[x] \Theta u[y]</math>, where <math>s</math> and <math>u</math> are tuple variables, <math>x</math> is an attribute on which <math>s</math> is defined, <math>y</math> is an attribute on which <math>u</math> is defined, and <math>\Theta</math> is a comparison operator (<math>&lt;, \leq, =, \neq, &gt;, \geq</math>); we require that attributes <math>x</math> and <math>y</math> have domains whose members can be compared by <math>\Theta</math>.</li> <li>• <math>s[x] \Theta c</math>, where <math>s</math> is a tuple variable, <math>x</math> is an attribute on which <math>s</math> is defined, <math>\Theta</math> is a comparison operator, and <math>c</math> is a constant in the domain of attribute <math>x</math>.</li> </ul> <p>We build up formulae from atoms by using the following rules:</p> <ul style="list-style-type: none"> <li>• An atom is a formula.</li> <li>• If <math>P_1</math> is a formula, then so are <math>\neg P_1</math> and <math>(P_1)</math>.</li> <li>• If <math>P_1</math> and <math>P_2</math> are formulae, then so are <math>P_1 \vee P_2</math>, <math>P_1 \wedge P_2</math>, and <math>P_1 \Rightarrow P_2</math>.</li> <li>• If <math>P_1(s)</math> is a formula containing a free tuple variable <math>s</math>, and <math>r</math> is a relation, then</li> </ul> $\exists s \in r (P_1(s)) \text{ and } \forall s \in r (P_1(s))$ |

are also formulae.

As we could for the relational algebra, we can write equivalent expressions that are not identical in appearance. In the tuple relational calculus, these equivalences include the following three rules:

1.  $P_1 \wedge P_2$  is equivalent to  $\neg(\neg(P_1) \vee \neg(P_2))$ .
2.  $\forall t \in r (P_1(t))$  is equivalent to  $\neg \exists t \in r (\neg P_1(t))$ .
3.  $P_1 \Rightarrow P_2$  is equivalent to  $\neg(P_1) \vee P_2$ .

### Safety of Expressions :

There is one final issue to be addressed. A tuple-relational-calculus expression may generate an infinite relation. Suppose that we write the expression:

$$\{t \mid \neg(t \in instructor)\}$$

There are infinitely many tuples that are not in *instructor*. Most of these tuples contain values that do not even appear in the database! Clearly, we do not wish to allow such expressions.

To help us define a restriction of the tuple relational calculus, we introduce the concept of the **domain** of a tuple relational formula,  $P$ . Intuitively, the domain of  $P$ , denoted  $dom(P)$ , is the set of all values referenced by  $P$ . They include values mentioned in  $P$  itself, as well as values that appear in a tuple of a relation mentioned in  $P$ . Thus, the domain of  $P$  is the set of all values that appear explicitly in  $P$  or that appear in one or more relations whose names appear in  $P$ . For example,  $dom(t \in instructor \wedge t[salary] > 80000)$  is the set containing 80000 as well as the set of all values appearing in any attribute of any tuple in the *instructor* relation. Similarly,  $dom(\neg(t \in instructor))$  is also the set of all values appearing in *instructor*, since the relation *instructor* is mentioned in the expression.

We say that an expression  $\{t \mid P(t)\}$  is *safe* if all values that appear in the result are values from  $dom(P)$ . The expression  $\{t \mid \neg(t \in instructor)\}$  is not safe. Note that  $dom(\neg(t \in instructor))$  is the set of all values appearing in *instructor*. However, it is possible to have a tuple  $t$  not in *instructor* that contains values that do not appear in *instructor*. The other examples of tuple-relational-calculus expressions that we have written in this section are safe.

---

The number of tuples that satisfy an unsafe expression, such as  $\{t \mid \neg(t \in instructor)\}$ , could be infinite, whereas safe expressions are guaranteed to have finite results. The class of tuple-relational-calculus expressions that are allowed is therefore restricted to those that are safe.

| <b>BASIS FOR COMPARISON</b> | <b>RELATIONAL ALGEBRA</b>  | <b>RELATIONAL CALCULUS</b>                                    |
|-----------------------------|--|---|
| Basic                       | Relational Algebra is a Procedural language.                                     | Relational calculus is Declarative language.                  |
| States                      | Relational Algebra states how to obtain the result.                              | Relational Calculus states what result we have to obtain.     |
| Order                       | Relational Algebra describes the order in which operations have to be performed. | Relational Calculus does not specify the order of operations. |
| Domain                      | Relational Algebra is not domain dependent.                                      | Relation Calculus can be domain dependent.                    |
| Related                     | It is close to a programming language.   | It is close to the natural language.                          |

|      |   |
|------|---|
| 3 a  | What are constraints? What are the different types of constraints? Explain. |
| Ans. | SQL constraints are used to specify rules for the data in a table.          |



Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value.

E.g. CREATE TABLE Persons (  
ID int NOT NULL,  
LastName varchar (255) NOT NULL,  
FirstName varchar (255) NOT NULL,  
Age int  
);

- **UNIQUE** - Ensures that all values in a column are different.

E.g. CREATE TABLE Persons (  
ID int NOT NULL UNIQUE,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Age int  
);

- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

E.g. CREATE TABLE Persons (  
ID int NOT NULL,  
LastName varchar (255) NOT NULL,  
FirstName varchar (255),  
Age int,  
PRIMARY KEY (ID)  
);

- **FOREIGN KEY** - Uniquely identifies a row/record in another table.

E.g. CREATE TABLE Orders (  
OrderID int NOT NULL,  
OrderNumber int NOT NULL,  
PersonID int,  
PRIMARY KEY (OrderID),

|  |  |
|--|--|
|  | <p style="text-align: center;">FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)<br/>);</p> <ul style="list-style-type: none"> <li>• <b><u>CHECK</u></b> - Ensures that all values in a column satisfies a specific condition.</li> <li>• <b><u>DEFAULT</u></b> - Sets a default value for a column when no value is specified.</li> </ul> <p>Explanation with example excepted.</p> |
|--|--|

|                 |  |           |  |                 |                                   |               |   |
|-----------------|--|-----------|--|-----------------|-----------------------------------|---------------|---|
| 3 b             | <p>When can a view be updated? Explain the syntax of updating a view. And also state the difference between views and table.</p>   |           |  |                 |                                   |               |   |
| Ans.            | <p><b>A view be updated: ---</b></p> <ol style="list-style-type: none"> <li>1. The view is defined based on one and only one table.</li> <li>2. The view must include the PRIMARY KEY of the table based upon which the view has been created.</li> <li>3. The view should not have any field made out of aggregate functions.</li> <li>4. The view must not have any DISTINCT clause in its definition.</li> <li>5. The view must not have any GROUP BY or HAVING clause in its definition.</li> <li>6. The view must not have any SUBQUERIES in its definitions.</li> <li>7. If the view you want to update is based upon another view, the later should be updatable.</li> <li>8. Any of the selected output fields (of the view) must not use constants, strings or value expressions.</li> </ol> <p><b>Syntax:</b></p> <pre>UPDATE &lt; view_name &gt; SET&lt;column1&gt;=&lt;value1&gt;,&lt;column2&gt;=&lt;value2&gt;,,..... WHERE &lt;condition&gt;;</pre> <table style="width: 100%; margin-top: 10px;"> <tr> <td style="padding: 5px;">view_name</td> <td style="padding: 5px;">Name of the virtual table or view where data will be modified.</td> </tr> <tr> <td style="padding: 5px;">column1,column2</td> <td style="padding: 5px;">Name of the columns of the table.</td> </tr> <tr> <td style="padding: 5px;">value1,value2</td> <td style="padding: 5px;">Values for the columns which are going to be updated.</td> </tr> </table> | view_name | Name of the virtual table or view where data will be modified. | column1,column2 | Name of the columns of the table. | value1,value2 | Values for the columns which are going to be updated. |
| view_name       | Name of the virtual table or view where data will be modified.   |           |  |                 |                                   |               |   |
| column1,column2 | Name of the columns of the table.  |           |  |                 |                                   |               |   |
| value1,value2   | Values for the columns which are going to be updated.  |           |  |                 |                                   |               |   |

|  | <p>condition                      Condition or criteria.</p> <p><b>Parameters:</b></p> <p>Updating a view must affect the rows in the corresponding table.</p> <p>E.g. create view vw_emp</p> <p>As</p> <p>Select * from emp</p> <p>Where deptno=10;</p>  |       |        |  |   |   |  |   |                                  |  |  |
|--|---|-------|--------|--|---|---|--|---|----------------------------------|--|--|
|  | <table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%;">VIEWS</th> <th style="width: 50%;">TABLES</th> </tr> </thead> <tbody> <tr> <td>1. There is one type of relation which is not a part of the physical database.</td> <td>1. A base relation is a relation that is a part of the physical database.</td> </tr> <tr> <td>2. It has no direct or physical relation with the database.</td> <td>2. While it can manipulate the contents of the relations stored in the data.</td> </tr> <tr> <td>3. Views can be used to provide security mechanism.</td> <td>3. It does not provide security.</td> </tr> <tr> <td>4. Modification through a view (e.g. insert, update, delete) not permitted).</td> <td>4. Modification may be done with a view.</td> </tr> </tbody> </table> | VIEWS | TABLES | 1. There is one type of relation which is not a part of the physical database. | 1. A base relation is a relation that is a part of the physical database. | 2. It has no direct or physical relation with the database. | 2. While it can manipulate the contents of the relations stored in the data. | 3. Views can be used to provide security mechanism. | 3. It does not provide security. | 4. Modification through a view (e.g. insert, update, delete) not permitted). | 4. Modification may be done with a view. |
| VIEWS  | TABLES  |       |        |  |   |   |  |   |                                  |  |  |
| 1. There is one type of relation which is not a part of the physical database. | 1. A base relation is a relation that is a part of the physical database.   |       |        |  |   |   |  |   |                                  |  |  |
| 2. It has no direct or physical relation with the database.                    | 2. While it can manipulate the contents of the relations stored in the data.  |       |        |  |   |   |  |   |                                  |  |  |
| 3. Views can be used to provide security mechanism.                            | 3. It does not provide security.  |       |        |  |   |   |  |   |                                  |  |  |
| 4. Modification through a view (e.g. insert, update, delete) not permitted).   | 4. Modification may be done with a view.  |       |        |  |   |   |  |   |                                  |  |  |

|     |  |
|-----|--|
| 3 c | <p>Consider the relations :</p> <p>Worker<br/>(WORKER_ID,FIRST_NAME,LAST_NAME,SALARY,JOINING_DATE,DEPARTMENT)</p> <p>Write the SQL queries for the following:</p> <ol style="list-style-type: none"> <li>a. Write An SQL Query To Print The FIRST_NAME And LAST_NAME From Worker Table Into A Single Column COMPLETE_NAME. A Space Char Should Separate Them.</li> <li>b. Write An SQL Query That Fetches The Unique Values Of DEPARTMENT From Worker Table And Prints Its Length.</li> <li>c. Write An SQL Query To Print First Three Characters Of FIRST_NAME From Worker Table.</li> <li>d. Write An SQL Query To Fetch Worker Names With Salaries &gt;= 50000 And &lt;= 100000.</li> </ol> |
|-----|--|

|     |  |
|-----|--|
|     | e. Write An SQL Query To Fetch The No. Of Workers for Each Department in the Descending Order.   |
| Ans | <p>a. Select CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'COMPLETE_NAME' from Worker;</p> <p>b. Select distinct length(DEPARTMENT) from Worker;</p> <p>c. Select substring(FIRST_NAME,1,3) from Worker;</p> <p>d. SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) As Worker_Name, Salary FROM worker WHERE WORKER_ID IN (SELECT WORKER_ID FROM worker WHERE Salary BETWEEN 50000 AND 100000);</p> <p>e. SELECT DEPARTMENT, count(WORKER_ID) No_Of_Workers FROM worker GROUP BY DEPARTMENT ORDER BY No_Of_Workers DESC;</p> |

|      |   |
|------|---|
| 3 d  | Write in brief about SQL with its advantages and also explain NULL value concept. How NULL values are different from EMPTY values.  |
| Ans. | <p><b>SQL is Structured Query Language</b>, which is a computer language for storing, manipulating and retrieving data stored in a relational database.</p> <p>SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. <b>SQL is widely popular because it offers the following advantages –</b></p> <ul style="list-style-type: none"> <li>• Allows users to access data in the relational database management systems.</li> <li>• Allows users to describe the data.</li> <li>• Allows users to define the data in a database and manipulate that data.</li> <li>• Allows to embed within other languages using SQL modules, libraries &amp; pre-compilers.</li> <li>• Allows users to create and drop databases and tables.</li> <li>• Allows users to create view, stored procedure, functions in a database.</li> <li>• Allows users to set permissions on tables, procedures and views.</li> </ul> |

|  |  |
|--|--|
|  | <p><b>NULL value concept:--</b></p> <p>The NULL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.</p> <p>A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.</p> <p>Syntax</p> <pre>SQL&gt; CREATE TABLE CUSTOMERS(   ID INT NOT NULL,   NAME VARCHAR (20) NOT NULL,   AGE INT NOT NULL,   ADDRESS CHAR (25) ,   SALARY DECIMAL (18, 2),   PRIMARY KEY (ID) );</pre> <p>Here, <b>NOT NULL</b> signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.</p> <p>A field with a NULL value is the one that has been left blank during the record creation.</p> <p><b>NULL values are different from EMPTY values:--</b></p> <p>An <b>empty</b> string is a <b>value</b>, but is just <b>empty</b>. <b>NULL</b> is special to a database. <b>NULL</b> has no bounds, it can be used for string , integer , date , etc. fields <b>in a</b> database. <b>NULL</b> isn't allocated any memory, the string with <b>NULL value</b> is just a pointer which is pointing to nowhere in memory. A NULL value represents the absence of a value for a record in a field (other software calls it a missing value). An empty value is a "field-formatted" value with no significant data in it. Null has no bounds, it can be used for string, integer, date, etc. fields in a database. Empty string is just regarding a string. If you have no value for a field, use null, not an empty string.</p> |
|--|--|

|     |   |
|-----|---|
| 3 e | <p>Define Join and List its type and explain any two in details. Consider the following relation and solve the below query:</p> <p><b>Sample table:</b> departments<br/>( DEPARTMENT_ID,DEPARTMENT_NAME ,MANAGER_ID , LOCATION_ID )</p> <p><b>Sample table:</b> employees</p> |
|-----|---|

|     |   |
|-----|---|
|     | <p>(EMPLOYEE_ID , FIRST_NAME , LAST_NAME , EMAIL , PHONE_NUMBER ,HIRE_DATE , JOB_ID , SALARY , COMMISSION_PCT , MANAGER_ID , DEPARTMENT_ID )</p> <p>i) Write a query in SQL to display the first name, last name, department number, and department name for each employee.</p>   |
| ANS | <p>An SQL <b>join</b> clause combines columns from one or more tables in a relational database. It creates a set that can be saved as a table or used as it is. A JOIN is a means for combining columns from one (self-join) or more tables by using values common to each.</p> <p>Oracle proprietary joins are:</p> <ul style="list-style-type: none"> <li>• Equijoin</li> <li>• Non-equijoin</li> <li>• Outer join</li> <li>• Self join</li> </ul> <p>ANSI-standard SQL specifies five types of JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS. As a special case, a table (base table, view, or joined table) can JOIN to itself in a <i>self-join</i>.</p> <p><b>Defining Joins</b></p> <p>When data from more than one table in the database is required, a <i>join</i> condition is used. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns, that is, usually primary and foreign key columns.</p> <p>To display data from two or more related tables, write a simple join condition in the WHERE clause.</p> <p><b>SELECT</b>     <i>table1.column, table2.column</i><br/> <b>FROM</b>       <i>table1, table2</i><br/> <b>WHERE</b>      <i>table1.column1 = table2.column2;</i></p> <p>In the syntax:</p> <p>          <i>table1.column</i>           denotes the table and column from which data is retrieved</p> <p>          <i>table1.column1 = table2.column2</i>    is the condition that joins (or relates) the tables together</p> <ul style="list-style-type: none"> <li>• When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.</li> <li>• If the same column name appears in more than one table, the column name must be prefixed with the table name.</li> <li>• To join <i>n</i> tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.</li> </ul> <p>Explanation of each type of join expected in brief.</p> <p>SELECT E.first_name , E.last_name ,<br/>           E.department_id , D.department_name</p> |

|  |   |
|--|---|
|  | FROM employees E<br>JOIN departments D<br>ON E.department_id = D.department_id; |
|--|---|

3 f Differentiate between ANY and ALL operators with example & also explain hierarchical query.

|          |  |   |
|----------|--|---|
| ANS<br>: | <b>ANY</b>   | <b>ALL</b>  |
|          | 1. The ANY operator returns TRUE if any of the subquery values meet the condition. | The ALL operator returns TRUE if all of the subquery values meet the condition. |
|          | 2. Evaluates to FALSE if the query returns no rows.                                | 2. Evaluates to TRUE if the query returns no rows.                              |

ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row. ANY must be preceded by comparison operators. Suppose using greater than ( > ) with ANY means greater than at least one value.

**Syntax:**

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE expression2 comparison_operator { ALL | ANY | SOME } ( subquery )
```

**Parameters:**

| Name        | Description   |
|-------------|---|
| column_name | Name of the column of the table.  |
| expression1 | Expression made up of a single constant, variable, scalar function can also be the pieces of a SQL query that compare values against perform arithmetic calculations. |
| table_name  | Name of the table.  |

|                      |  |
|----------------------|--|
| WHERE<br>expression2 | Compares a scalar expression until a match is found for ANY operator. One of the rows must match the expression to return a Boolean TRUE value for the ANY operator. |
| comparison_operator  | Compares the expression to the subquery. The comparison must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).   |

ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list or results from a query. The ALL must be preceded by the comparison operators and evaluates to TRUE if the query returns no rows. For example, ALL means greater than every value, means greater than the maximum value. Suppose ALL (1, 2, 3) means greater than 3.

**Syntax:**

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE expression2 comparison_operator { ALL | ANY | SOME } ( subquery )
```

**HIERARCHICAL QUERIES:--**

A hierarchical query is the one that works on tree like relationship. It loops through a result set and returns rows in a hierarchical sequence. It is used when a parent has a child element and each child element may have one or more child elements.

**Syntax:--**

```
SELECT column_list
FROM tablename
[WHERE Search_condition]
START WITH row_specification
CONNECT BY PRIOR connect_expression
[ORDER BY list]
STARTWITH specifies the root rows of the hierarchy.
```



|  |  |
|--|--|
|  | <p>CONNECTBY specifies the relationship between parent rows and child rows of the hierarchy.</p> <p>e.g. SELECT empid,ename,mgr</p> <p>FROM emp</p> <p>Where empid&lt;7500</p> <p>CONNECT BY PRIOR empid=mgr</p> |
|  |  |

|     |  |
|-----|--|
| 4 a | List the ACID properties. Explain the usefulness of each.  |
| ANS | <ul style="list-style-type: none"> <li>• <b>Atomicity.</b> Either all operations of the transaction are reflected properly in the database, or none are.</li> <li>• <b>Consistency.</b> Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.</li> <li>• <b>Isolation.</b> Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions <math>T_i</math> and <math>T_j</math>, it appears to <math>T_i</math> that either <math>T_j</math> finished execution before <math>T_i</math> started or <math>T_j</math> started execution after <math>T_i</math> finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.</li> <li>• <b>Durability.</b> After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.</li> </ul> |

|     |  |
|-----|--|
| 4 b | Explain the concept of serializability and explain in detail view serializability.   |
| ANS | <p><b>Serializability.</b> <b>Serializability</b> 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</p> <p><b>View Serializability</b></p> <ul style="list-style-type: none"> <li>○ A schedule will view serializable if it is view equivalent to a serial schedule.</li> <li>○ If a schedule is conflict serializable, then it will be view serializable.</li> <li>○ The view serializable which does not conflict serializable contains blind writes.</li> </ul> |

## View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

### 1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

| T1      | T2       |
|---------|----------|
| Read(A) | Write(A) |

**Schedule S1**

| T1      | T2       |
|---------|----------|
| Read(A) | Write(A) |

**Schedule S2**

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

### 2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

| T1       | T2       | T3      |
|----------|----------|---------|
| Write(A) | Write(A) | Read(A) |

**Schedule S1**

| T1       | T2       | T3          |
|----------|----------|-------------|
| Write(A) | Write(A) | <u>Read</u> |

**Schedule S2**

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

### 3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

| T1       | T2      | T3       |
|----------|---------|----------|
| Write(A) | Read(A) | Write(A) |

**Schedule S1**

| T1       | T2      | T3       |
|----------|---------|----------|
| Write(A) | Read(A) | Write(A) |

**Schedule S2**

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

### Example:

| T1       | T2       | T3       |
|----------|----------|----------|
| Read(A)  | Write(A) | Write(A) |
| Write(A) |          |          |

### Schedule S

With 3 transactions, the total number of possible schedule

= 3! = 6

S1 = <T1 T2 T3>

S2 = <T1 T3 T2>

S3 = <T2 T3 T1>

S4 = <T2 T1 T3>

S5 = <T3 T1 T2>

S6 = <T3 T2 T1>

**Taking first schedule S1:**

| <b>T1</b>           | <b>T2</b> | <b>T3</b> |
|---------------------|-----------|-----------|
| Read(A)<br>Write(A) | Write(A)  | Write(A)  |

**Schedule S1**

**Step 1:** final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

**Step 2:** Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

**Step 3:** Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

**Hence, view equivalent serial schedule is:**

T1 → T2 → T3

|     |  |
|-----|--|
| 4 c | What are concurrent transaction? Explain in details the main features of concurrent execution.   |
| ANS | <p>The database system must control the interaction among the concurrent transaction to prevent them from destroying the consistency of the database is termed as concurrent control scheme.</p> <p>The main features of the concurrent execution are given below:-</p> <ol style="list-style-type: none"> <li>1.Improved throughput and resources utilization</li> <li>2.Reducing waiting time</li> </ol> <p>Example &amp; explanation in detail is excepted.</p> |

|     |   |
|-----|---|
| 4 d | What are some disadvantages of time stamping methods for concurrency control? And also explain timestamp ordering protocol in detail.   |
| ANS | <p>The locking protocols that we have described thus far determine the order between every pair of conflicting transactions at execution time by the first lock that both members of the pair request that involves incompatible modes. Another method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering scheme.</p> <p>15.4.1 Timestamps With each transaction <math>T_i</math> in the system, we associate a unique fixed timestamp, denoted by <math>TS(T_i)</math>. This timestamp is assigned by the database system before the transaction <math>T_i</math> starts execution. If a transaction <math>T_i</math> has been assigned timestamp <math>TS(T_i)</math>, and a new transaction <math>T_j</math> enters the system, then <math>TS(T_i) &lt; TS(T_j)</math>. There are two simple methods for implementing this scheme:</p> <ol style="list-style-type: none"> <li>1. Use the value of the system clock as the timestamp; that is, a transaction's timestamp is equal to the value of the clock when the transaction enters the system.</li> <li>2. Use a logical counter that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter when the transaction enters the system.</li> </ol> <p>The timestamps of the transactions determine the serializability order. Thus, if <math>TS(T_i) &lt; TS(T_j)</math>, then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction <math>T_i</math> appears before transaction <math>T_j</math>. To implement this scheme, we associate with each data item <math>Q</math> two timestamp values:</p> <ul style="list-style-type: none"> <li>• W-timestamp(<math>Q</math>) denotes the largest timestamp of any transaction that executed write(<math>Q</math>) successfully.</li> <li>• R-timestamp(<math>Q</math>) denotes the largest timestamp of any transaction that executed read(<math>Q</math>) successfully.</li> </ul> <p>These timestamps are updated whenever a new read(<math>Q</math>) or write(<math>Q</math>) instruction is executed.</p> |

15.4.2 The Timestamp-Ordering Protocol The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

1. Suppose that transaction  $T_i$  issues  $\text{read}(Q)$ . a. If  $\text{TS}(T_i) < \text{W-timestamp}(Q)$ , then  $T_i$  needs to read a value of  $Q$  that was already overwritten. Hence, the read operation is rejected, and  $T_i$  is rolled back. b. If  $\text{TS}(T_i) \geq \text{W-timestamp}(Q)$ , then the read operation is executed, and  $\text{R-timestamp}(Q)$  is set to the maximum of  $\text{R-timestamp}(Q)$  and  $\text{TS}(T_i)$ . 2. Suppose that transaction  $T_i$  issues  $\text{write}(Q)$ . a. If  $\text{TS}(T_i) < \text{R-timestamp}(Q)$ , then the value of  $Q$  that  $T_i$  is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls  $T_i$  back. b. If  $\text{TS}(T_i) < \text{W-timestamp}(Q)$ , then  $T_i$  is attempting to write an obsolete value of  $Q$ . Hence, the system rejects this write operation and rolls  $T_i$  back. c. Otherwise, the system executes the write operation and sets  $\text{W-timestamp}(Q)$  to  $\text{TS}(T_i)$ .

If a transaction  $T_i$  is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it. To illustrate this protocol, we consider transactions  $T_{25}$  and  $T_{26}$ .

Transaction  $T_{25}$  displays the contents of accounts  $A$  and  $B$ :

$T_{25}$ :  $\text{read}(B)$ ;  $\text{read}(A)$ ;  $\text{display}(A + B)$ .

Transaction  $T_{26}$  transfers \$50 from account  $B$  to account  $A$ , and then displays the contents of both:

$T_{26}$ :  $\text{read}(B)$ ;  $B := B - 50$ ;  $\text{write}(B)$ ;  $\text{read}(A)$ ;  $A := A + 50$ ;  $\text{write}(A)$ ;  $\text{display}(A + B)$ .

In presenting schedules under the timestamp protocol, we shall assume that a transaction is assigned a timestamp immediately before its first instruction. Thus, in schedule 3 of Figure 15.17,  $\text{TS}(T_{25}) < \text{TS}(T_{26})$ , and the schedule is possible under the timestamp protocol. We note that the preceding execution can also be produced by the two-phase locking protocol. There are, however, schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and vice versa (see Exercise 15.29). The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order. The protocol ensures freedom from deadlock, since no transaction ever waits.

However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction. If a transaction is suffering from repeated restarts, conflicting transactions need to be temporarily blocked to enable the transaction to finish.

$T_{25} \ T_{26} \ \text{read}(B) \ \text{read}(B) \ B := B - 50 \ \text{write}(B) \ \text{read}(A) \ \text{read}(A) \ \text{display}(A+B) \ A := A + 50 \ \text{write}(A) \ \text{display}(A+B)$ .

The protocol can generate schedules that are not recoverable. However, it can be extended to make the schedules recoverable, in one of several ways:

- Recoverability and cascadelessness can be ensured by performing all writes together at the end of the transaction. The writes must be atomic in the following sense: While the writes are in progress, no transaction is permitted to access any of the data items that have been written. •

Recoverability and cascadelessness can also be guaranteed by using a limited form of

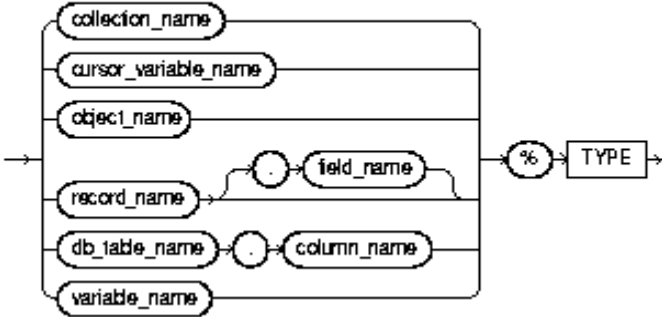
|  |  |
|--|--|
|  | <p>locking, whereby reads of uncommitted items are postponed until the transaction that updated the item commits</p> <ul style="list-style-type: none"> <li>• Recoverability alone can be ensured by tracking uncommitted writes, and allowing a transaction <math>T_i</math> to commit only after the commit of any transaction that wrote a value that <math>T_i</math> read.</li> </ul> |
|--|--|

|     |   |
|-----|---|
| 4 e | <p>What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking?</p>   |
| ANS | <p>Rigorous two-phase locking has the advantages of strict 2PL. In addition it has the property that for two conflicting transactions, their commit order is their serializability order. In some systems users might expect this behavior. Also, its implementation is easier than strict 2PL.</p> <p>In rigorous two-phase locking protocol a transaction is not allowed to release any lock ( shared and exclusive until it commit ). This means that until the transaction commits , other transactions might acquire a shared lock on a data items , on which the un-committed transaction has shared lock but can not acquire any lock on an data items , on which the uncommitted transaction has an exclusive lock.</p> <p>Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.</p> <p>It ensures conflict serializable schedules.</p> <p>If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.</p> <p><b>This protocol can be divided into two phases,</b></p> <ol style="list-style-type: none"> <li><b>1. In Growing Phase,</b> a transaction obtains locks, but may not release any lock.</li> <li><b>2. In Shrinking Phase,</b> a transaction may release locks, but may not obtain any lock.</li> </ol> <p>Two-Phase Locking does not ensure freedom from deadlocks.</p> <p><b>Types of Two – Phase Locking Protocol</b></p> <p><b>Following are the types of two – phase locking protocol:</b></p> <ol style="list-style-type: none"> <li>1. Strict Two – Phase Locking Protocol</li> <li>2. Rigorous Two – Phase Locking Protocol</li> <li>3. Conservative Two – Phase Locking Protocol</li> </ol> <p><b>1. Strict Two-Phase Locking Protocol</b></p> |

|  |  |
|--|--|
|  | <p>Strict Two-Phase Locking Protocol avoids cascaded rollbacks.</p> <p>This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.</p> <p>It is not deadlock free.</p> <p>It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.</p> <p>Most of the database systems implement rigorous two – phase locking protocol.</p> <p><b>2. Rigorous Two-Phase Locking</b></p> <p>Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.</p> <p>This protocol requires that all the share and exclusive locks to be held until the transaction commits.</p> <p><b>3. Conservative Two-Phase Locking Protocol</b></p> <p>Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.</p> <p>This protocol is almost free from deadlocks as all required items are listed in advanced.</p> <p>It requires locking of all data items to access before the transaction starts.</p> |
|--|--|

|     |   |
|-----|---|
| 4 f | <p>If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain your answer.</p>   |
| ANS | <p>A transaction may become the victim of deadlock-prevention rollback arbitrarily many times, thus creating a potential starvation situation.</p> <p>Explanation in details expected</p> |



|      |   |
|------|---|
| 5 a  | What is the use of % TYPE attributes and how it is beneficial while declaring the variable?   |
| Ans. | <p>The %TYPE attribute let's use the data type of a field, record, nested table, database column, or variable in your own declarations, instead of hardcoding the type names. You can use the %TYPE attribute as a datatype specified when declaring constants, variables, fields, and parameters. If the types that you reference change, your declarations are automatically updated. This technique saves you from making code changes when, for example, the length of a VARCHAR2 column is increased.</p> <p><b>Syntax</b></p> <p>type_attribute</p>  <p><b>Keyword and Parameter Description</b></p> <p><b>collection_name</b><br/>A nested table, index-by table, or varray previously declared within the current scope.</p> <p><b>cursor_variable_name</b><br/>A PL/SQL cursor variable previously declared within the current scope. Only the value of another cursor variable can be assigned to a cursor variable.</p> <p><b>db_table_name.column_name</b><br/>A table and column that must be accessible when the declaration is elaborated.</p> <p><b>object_name</b><br/>An instance of an object type, previously declared within the current scope.</p> <p><b>record_name</b><br/>A user-defined or %ROWTYPE record, previously declared within the current scope.</p> <p><b>record_name.field_name</b><br/>A field in a user-defined or %ROWTYPE record, previously declared within the current scope.</p> <p><b>variable_name</b><br/>A variable, previously declared in the same scope.</p> |

|      |  |
|------|--|
|      | <p>The %TYPE attribute is particularly useful when declaring variables, fields, and parameters that refer to database columns. Your code can keep working even when the lengths or types of the columns change. The NOT NULL column constraint is not inherited by items declared using %TYPE. <b>Examples</b></p> <pre> DECLARE EMPLOYEEID EMP.EMPID%TYPE; BEGIN SELECT EMPID INTO EMPLOYEEID FROM EMP WHERE ENAME = 'KING'; DBMS_OUTPUT.PUT_LINE(EMPLOYEEID); END; / </pre>  |
| 5 b  | Illustrate the attributes of Implicit cursor with examples.  |
| Ans. | <p>Implicit cursors are automatically created by Oracle whenever an SQL statement is Executed, when there is no explicit cursor for the statement. Programmers cannot control<br/>The implicit cursors and the information in it.<br/>In PL/SQL, you can refer to the most recent implicit cursor as the <b>SQL cursor</b>, which Always has attributes such as <b>%FOUND</b>, <b>%ISOPEN</b>, <b>%NOTFOUND</b>, and <b>%ROWCOUNT</b>. The SQL cursor has additional attributes, <b>%BULK_ROWCOUNT</b> and <b>%BULK_EXCEPTIONS</b>, designed for use with the <b>FORALL</b> statement. The following<br/>Table provides the description of the most used attributes:</p> <p><b>Attribute</b></p> <p><b>%FOUND:</b> -- Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.</p> <p><b>%NOTFOUND:</b>--The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.</p> <p><b>%ISOPEN:</b>--Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.</p> <p><b>%ROWCOUNT:</b>--Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.<br/>Any SQL cursor attribute will be accessed as <b>sql%attribute_name</b> as shown below in the example.<br/>The following program will update the table and increase the salary of each customer by</p> |

|      |  |
|------|--|
|      | <p>500 and use the <b>SQL%ROWCOUNT</b> attribute to determine the number of rows affected:</p> <pre> DECLARE Total_rows number (2); BEGIN UPDATE customers SET salary = salary + 500; IF sql%notfound THEN dbms_output.put_line('no customers selected'); ELSIF sql%found THEN total_rows:= sql%rowcount; dbms_output.put_line( total_rows    ' customers selected '); END IF; END; / </pre>   |
| 5 c  | <p>Explain the function <code>Raise_Application_Error ()</code> with example.</p>  |
| Ans. | <p>The <i>raise_application_error</i> is actually a procedure defined by Oracle that allows the developer to raise an exception and associate an error number and message with the procedure. This allows the application to raise application errors rather than just Oracle errors. Error numbers are defined between -20,000 and -20,999.</p> <p><b>RAISE_APPLICATION_ERROR ( )</b></p> <p><b>RAISE_APPLICATION_ERROR</b> is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999. Whenever a message is displayed using <b>RAISE_APPLICATION_ERROR</b>, all previous transactions which are not committed within the PL/SQL Block are rolled back automatically (i.e. change due to INSERT, UPDATE, or DELETE statements).</p> <p><b>RAISE_APPLICATION_ERROR</b> raises an exception but does not handle it. <b>RAISE_APPLICATION_ERROR</b> is used for the following reasons,</p> <ul style="list-style-type: none"> <li>a) to create a unique id for an user-defined exception.</li> <li>b) To make the user-defined exception look like an Oracle error.</li> </ul> <p>The General Syntax to use this procedure is:</p> <p><b>RAISE_APPLICATION_ERROR (error_number, error_message);</b></p> |

- The Error number must be between -20000 and -20999
- The Error\_message is the message you want to display when the error occurs.

Steps to be followed to use RAISE\_APPLICATION\_ERROR procedure:

1. Declare a user-defined exception in the declaration section.
2. Raise the user-defined exception based on a specific business rule in the execution section.
3. Finally, catch the exception and link the exception to a user-defined error number in RAISE\_APPLICATION\_ERROR.

*DECLARE*

*Huge\_quantity EXCEPTION;*

*CURSOR product\_quantity is*

*SELECT p.product\_name as name, sum (o.total\_units) as units*

*FROM order\_tems o, product p*

*WHERE o.product\_id = p.product\_id;*

*Quantity order\_tems.total\_units%type;*

*Up\_limit CONSTANT order\_tems.total\_units%type: = 20;*

*Message VARCHAR2 (50);*

*BEGIN*

*FOR product\_rec in product\_quantity LOOP*

*Quantity: = product\_rec. Units;*

*IF quantity > up\_limit THEN*

*RAISE huge\_quantity;*

*ELSIF quantity < up\_limit THEN*

*v\_message:= 'The number of unit is below the discount limit.'*

*END IF;*

*Dbms\_output.put\_line (message);*

*END LOOP;*

*EXCEPTION*

*WHEN huge\_quantity THEN*

*raise\_application\_error (-2100, 'The number of unit is above the discount*

*limit.');*

*END;*

*/*

|     |  |
|-----|--|
| 5 d | List & explain the various features of PL/SQL & also differentiate between Anonymous blocks and Subprograms. |
|-----|--|

|   |  |
|---|--|
| Ans.  | <p><b>High performance:</b> PL/SQL is high performance transaction processing language.</p> <ul style="list-style-type: none"> <li>• <b>Code Re-usability:</b> It support code re-usability, it means no need to write code again and again same like SQL.</li> <li>• <b>Error Handling:</b> when any error are occurred then it return user friendly error message.</li> <li>• <b>Procedural Language Capability:</b> It consists of procedural language constructs such as conditional statements (if...else) and loops statements (for loop)</li> <li>• <b>Block Statement:</b> It consists of blocks of code, which can be nested within each other. Each and every block forms a unit of a task or a logical module.</li> <li>• <b>Better performance:</b> oracle engine processes multiple SQL statements simultaneously as a single block, due to this reducing network traffic.</li> <li>• <b>Declare variable:</b> It give you control to declare variable and access them within the block.</li> <li>• It support looping and conditional statements</li> <li>• <b>Reduce Network Traffic:</b> oracle engine processes multiple SQL statements simultaneously as a single block, due to this reducing network traffic.</li> <li>• <b>Portable application:</b> Applications are written in PL/SQL are portable in any available Operating system.</li> </ul> |
| <b>Differences between Anonymous Blocks and Subprograms</b> |  |
| <b>Anonymous blocks</b>                                     | <b>Subprograms</b>   |
| Unnamed PL/SQL blocks                                       | Named PL/SQL blocks  |
| Compiled every time   | Compiled only once   |
| Does not store in database                                  | Stores in database   |
| Cannot be invoked by other applications                     | These are named and therefore can be invoked by other applications   |
| Do not return values  | Subprogram called functions must return values   |
| Cannot take parameters                                      | Can take parameters  |

|     |   |
|-----|---|
| 5 e | What are packages in PL/SQL? List and explain the various advantages of packages. Create a package to display the employee name and salary. |
|-----|---|

|      |  |
|------|--|
| Ans. | <p>Packages bundle related PL/SQL types, items, and subprograms into one container. For example, a Human Resources package can contain hiring and firing procedures, commission and bonus functions, and tax exemption variables.</p> <p>A package usually has a specification and a body, stored separately in the database. The specification is the interface to your applications. It declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The package specification may also include PRAGMAs, which are directives to the compiler.</p> <p>The body fully defines cursors and subprograms, and so implements the specification. The package itself cannot be called, parameterized, or nested. Still, the format of a package is similar to that of a subprogram. Once written and compiled, the contents can be shared by many applications.</p> <p>When you call a packaged PL/SQL construct for the first time, the whole package is loaded into memory. Thus, later calls to constructs in the same package require no disk input/output (I/O).</p> <p><b>Advantages of package:--</b><br/>-----</p> <p>A. Modularity:--Encapsulate related constructs.</p> <p>B. Easier Application Design: -- Code and compile specification and body separately.</p> <p>C. Hiding Information: -- Only the declarations in the package specification are visible and accessible to application. Private constructs in the package body are hidden and inaccessible.<br/>- All coding is hidden in the package body.</p> <p>D. Added Functionality: - Persistency of variables and cursors.</p> <p>E. Better Performance:--The entire package is loaded into memory when the package is first referenced. There is only one copy in memory for all users. The dependency hierarchy is simplified.</p> <p>F. Overloading:--Multiple subprograms of the same name.</p> <p><b>Example to create package to display name and salary must be given.</b></p> |
|------|--|

|     |   |
|-----|---|
| 5 f | <p>What are triggers? Explain the syntax for creating a trigger in PL/SQL. List the benefits of creating trigger in PL/SQL.</p> |
|-----|---|

Ans.

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

#### Creating Triggers

The syntax for creating a trigger is –

```
CREATE [OR REPLACE ] TRIGGER trigger_name
```

```
{ BEFORE | AFTER | INSTEAD OF }
```

```
{ INSERT [OR] | UPDATE [OR] | DELETE }
```

```
[OF col_name]
```

```
ON table_name
```

```
[REFERENCING OLD AS o NEW AS n]
```

```
[FOR EACH ROW]
```

```
WHEN (condition)
```

```
DECLARE
```

```
Declaration-statements
```

```
BEGIN
```

```
Executable-statements
```

```
EXCEPTION
```

```
Exception-handling-statements
```

```
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger\_name – Creates or replaces an existing trigger with the *trigger\_name*.
- {BEFORE | AFTER | INSTEAD OF} – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} – This specifies the DML operation.
- [OF col\_name] – This specifies the column name that will be updated.
- [ON table\_name] – This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] – This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] – This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

### **Benefits of Triggers**

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions
- Serializability is a concurrency scheme where the concurrent transaction is equivalent to one that executes the transactions serially.
- A schedule is a list of transactions.
- Serial schedule defines each transaction is executed consecutively without any interference from other transactions.



- |  |  |
|--|--|
|  | <ul style="list-style-type: none"><li>• Non-serial schedule defines the operations from a group of concurrent transactions that are interleaved.</li><li>• In non-serial schedule, if the schedule is not proper, then the problems can arise like multiple update, uncommitted dependency and incorrect analysis.</li><li>• The main objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interference and produce a database state that could be produced by a serial execution.</li></ul> |
|--|--|