Explain system catalog and Discuss in detail the information stored in system catalog.

A relational DBMS maintain information about every table and index that it contains. the descriptive information is itself stored in a collection of special tables called the catalog tables.

The catalog tables are also called the data dictionary, system catalog or simply the catalog.

Information in system catalog are as follows:-

The system catalog has system wide information, such as the size of the buffer pool and the page size, and following information about individual tables, indexes and views.

* For each table:

- Its table name, file name and the file structure of the file in which it is stored.

- The attribute name and type of each of its attributes.

- The index name of each index on the table.

- The integrity constraints on the table.

* For each Index:

- The index name and the structure of the index.

- The search key attributes.

* For each view:-

→ Its view name and definition.

- In addition, statistics about tables and indexes are stored in the system catalogs and updated periodically.

- The following information is commonly stored:

- Cardinality :→ The number of tuples Ntuple(R) for each table R.

- size :→ The number of pages Npages(R) for table R.

- Index Cardinality :→ The number of distinct key values Nkeys(I) for each index I.

- Index size :→ The number of pages (I) for each Index I.

- Index height :→ The number of non leaf levels for each tree index.

- Index Range :→ The minimum present key values Ilow(I) and maximum Present key value Ihigh(I) for each index I.

⇒ The catalog also contain information about users such as accounting information and authorization information.

Explain Three techniques commonly used in algorithms to evaluate relational operator.

A few simple techniques are used to develop algorithm for each operator.

① Indexing

If a selection or join condition is specified, use an index to examine just the tuples that satisfy the condition.

② Iteration

Examine all tuples in an input table one after the other. If we need only a few fields from each tuple and there is an index whose key contains all these fields, instead of examining data tuples, we can scan all index data entries.

## ⑧ Partitioning

By Partitioning tuples on a sort key, we can often decompose an operation into a less expensive collection of operations on partitions sorting and hasing are two commonly used Partitioning techniques.

Discuss Access path and selectivity of Access path with reference to the role of Indexing in algorithm development

An access path is a way of retriving tuples from a table and consists of either

① a file scan

② an index plus a matching selection condition.

- Consider a simple selection that is a conjuction of conditions of the form attr op value, where op is one of the comparison operators.

$<, \leq, =, \neq, \geq, or >$.

- such selection are said to be conjuctive normal form (CNF) and each condition is called a conjuct.

⇒ The following example illustrate access Path.

If we have a hash index H on the search key (bid, sid) and we also have B+ tree index on day. the selection condition day 8/9/2002 ∧ bid = 5 ∧ sid = 3 offers us a choice. Both indexes match the selection condition. and we can use either to retrieve Reserves tuples. Whichever index we use the conjucts in the selection condition that are not match by the index must be check for each retrieved tuples.

Explain Query optimization with Diagram.

Query optimization is one of the most important tasks of a relational DBMS. One of the strength of relational Query language is the wide variety of ways in which a user can express and thus the system can evaluate a query.

- Although this flexibility makes it easy to write queries, good performance relies greatly on the quality of the query optimizer.

- Queries are passed and then presented to a query optimizer which is responsible for identifying an efficient execution plan.

- The optimizer generates alternative plans and choose the plan with the least estimated cost.

- Estimating the cost of each enumerated plan and choosing the plan with the lowest estimated cost.

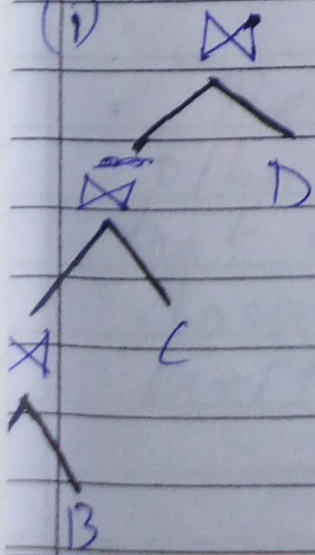
Discuss Evaluation plans with example and how the alternative plans considered.

A query evaluation plan consists of an extended relational algebra tree, with additional annotations at each node, indicating the access methods to use for each table and the implementation method to use for each relational operator

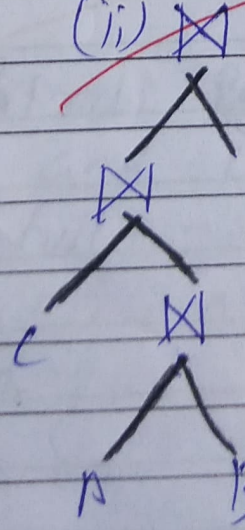
- Consider the following SQL query.

```
SELECT  S.sname
FROM    Reserves R, Sailors S.
WHERE   R.sid = S.sid AND
        R.bid = 100 AND
        S.Rating > 5.
```

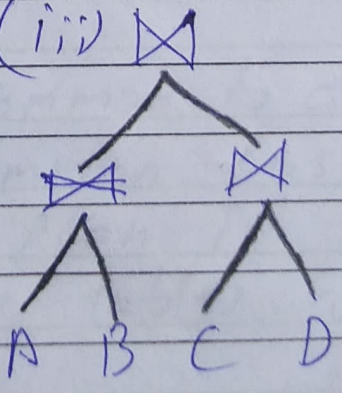
(i)




(ii)




(iii)



Explain the cost estimation of a plan.

The cost of a plan is the sum of costs for the operations it contains. The cost of individual operators in the plan is estimated using information obtained from the system catalog. about properties of their input tables.

- If we focus on the metric of I/O costs the cost of a plan can be broken down into three parts.

(1) reading the input tables.

(2) writing intermediate tables.

(3) costing the final result.

- The third part is common to all plans and in the common case that a fully-pipelined plan is chosen, no intermediate tables are written.

- The cost for fully-pipelined plan is dominated by part.

(i) This cost depend, greatly on the access paths used to read input tables: of course, access paths that are used repeatedly to retrive matching tuples in a join algorithm are especially important.

- This query can be expressed in relational algebra as follows:

$$\pi_{sname}(\sigma_{bid=100 \wedge rating>5}(Reserves \bowtie_{sid=sid} Sailor))$$

⇒ Alternative plans considered:

* Left-Deep plans.

- Consider a query of the form $A \bowtie B \bowtie C \bowtie D$ that is, the natural join of four tables.

- Three relational algebra operator trees that are equivalent to this query shown in Figure.

- By convention, the left child of a join node is the outer table and the right child is the inner table.

- The number of tuples in the result of a projection is the same as the input, assuming that duplicates are not eliminated.

- The result size for a join can be estimated by multiplying the maximum result size, which is the product of the input table sizes by the reduction factor of the join condition.

- Thus, the number of tuples that have the same value in column A as a give value in column in R1 key(R).

How System catalogs are stored?

An elegant aspect of a relation DBMS is that is that the system catalog is itself a collection of tables. For example, we might store information about the attributes of tables in a catalog table called Attribute_cat:

Attribute_Cat (attr_name : string,
rel_name : String,
type : string, position : integer)

Suppose that the database contains the two tables that we introduced at the begining of this chapter.

Sailors (sid : integer, sname : string,
rating : integer, age : real)

Reserves (sid : integer, bid : integer,
day : dates, name : string)

| attrname | rel | type | |
|---|---|---|---|
| attr_name | Attribute_Cat | string | 1 |
| rel_name | " | " | 2 |
| type | " | " | 3 |
| position | " | integer | 4 |
| sid | Sailors | " | 1 |
| sname | " | string | 2 |
| rating | " | integer | 3 |
| age | " | real | 4 |
| sid | Reserves | integer | 1 |
| bid | " | " | 2 |
| day | " | dates | 3 |
| rname | " | string | 4 |

The fact that the system catalog is also a collection of tables is very useful. For example, catalog tables can be queried just like any other table, using the query language of the DBMS! Further, all the techniques available for implementing and managing tables apply directly to catalog tables. The choice of catalog tables and their schemas is

not unique and is made by the implementor of the DBMS. Real systems vary in their catalog schema design, but the catalog is always implemented as a collection of tables, and it essentially describes all the data stored in the database.

1. Difference of Dense Index vs Spare Index.

| Dense Index | Spare Index |
|---|---|
| - An Index records appears for every search key value in File. | - An index record are created only for some of the records. |
| - Indicies are Faster | - Indicies are slower |
| - Indicies require more space | - Indicies require less space. |
| - Indicies impose more maintainance for insertion and deletions | - Indicies impose less maintaince for insertion and deletion. |

e.g.?

Discuss Iteration and partitioning technique with reference to σ π ⋈.

## Iteration :

Examine all tuples in an input table, one after the other. If we need only a few fields from each tuple and there is an index whose key contains all these fields, instead of examining data tuples, we can scan all index data entries.

## Partitioning :

By partitioning tuples on a sort key, we can often decompose an operation into a less expensive collection of operations on partitions. Sorting and hashing are two commonly used partitioning techniques.

## Reference with Selection

Consider a selection of the form rname < 'C%' on the Reserves table.

Assuming that names are uniformly distributed with respect to the initial letter, for simplicity, we estimate that roughly 1/26 of Reserves tuples are in the Result. This is a total of 10000 tuples, or 100 pages. If we have a clustered B+ tree index on the sname field of Reserves, we can retrieve the qualifying tuples with 100 I/Os. However, if the index is unclustered, we could have up to 10000 I/Os in the worst case. Since each tuple could cause us to read a page.

What is pipelined Evaluation? What its benefits?

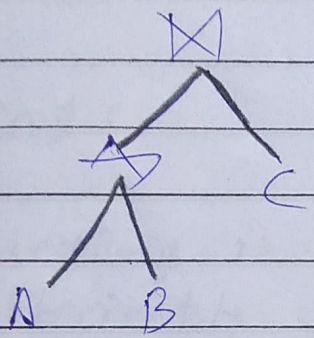
→ When a query is composed of several operators, the result of one operator is sometimes pipelines to another operator without creating temporary table to hold the intermediate result.

→ Pipelined the output of operator into the reset operator saves the cost of sorting out the intermediate result and reading it back in.

→ If the output of an operator saved in a temporary, we say that the tuple are materialized.

→ pipelined evaluation has overhead costs than materialized and is chosen the algorithm for the operator calculation permit it.

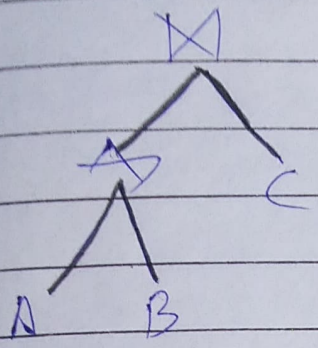
→ Example: Consider a join of the form $(A \bowtie B) \bowtie C$, shown in figure as tree of join operation




Result of type of First join Pipelined into join with C

→ A query tree, illustrating pipelining Both joins can be evaluated as version of a nested loop join.

→ Conceptually the evaluation is initiated from the root, and the node joining A and B produces tuples of as and when they are really by it Parent node, when they root node gets a page of tuples from its left child all matching inner tuples is disclosed.




Result of type of First join Pipelined into join with C

→ A query tree, illustrating pipelining Both joins can be evaluated as a version of a nested loop join.

→ Conceptually the evaluation is initiated from the root, and the node joining A and B produces tuples of as and when they are ready by it Parent node, when they root node gets a page of tuples from its left child all matching inner tuples is disclosed.

Define

i) Indexing :

- Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

- Indexing in database system is similar what we see in books.

- Indexing is defined based on its indexing attributes.

- Indexing can be of the following types.

* Primary index

Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation

## * Secandary Index

Secandary Index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

## * clustering Index

clustering index is defined an ordered data file. The data file is ordered on a non key field.

⇒ Ordered Indexing is of two types:-

(1) Dense Index
(2) Sparse Index

- **Dense Index** -

An index record appears for every search key in the file.

- **Sparse Index** -

An index record appears for only some of the search key value

- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

④ CNF :-

Conjunctive normal form (CNF) is an approach to boolean login that express formulas as conjustions of clauses with an AND or OR.

- Each clause connected by a conjunction, or AND must be either a literal or contain a disjunction or OR. Operator. CNF is usertul for avtomated theorem proving.

- In conjunctive normal form, statemed in Boolean logic are conjuctions of clauses with clauses of disjunctions.

- In other words, a statement is a series of ORs connected by ANDs.

## ⑥ Materialization

- It is easiest to understand intuitively how to evaluate an expression by looking at a pictorial ~~representation~~ representation of the expression in an operator tree.

- If we apply the materialization approach, we start from the lowest level operations in the expression

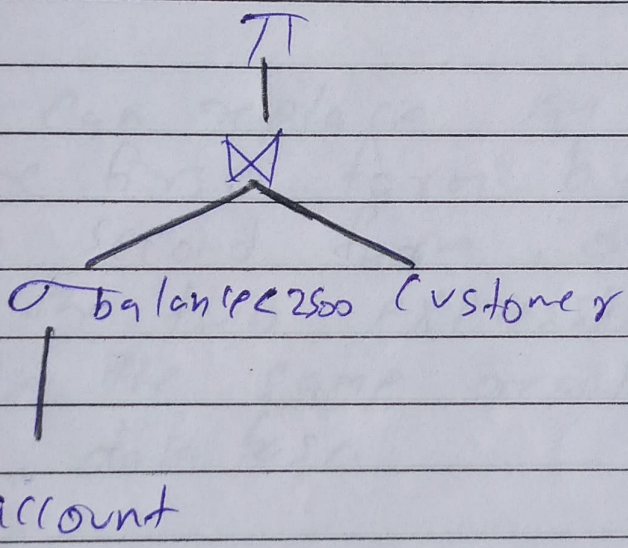
- There is only one such operation: the selection operation on account.

- The inputs to the lowest level operations are relations in the database. We stores the result in the temporary results.

- In our example, the inputs to the join are customer relation and the temporary relation created by the selection on account.

- The join can now be evaluated, creating another temporary relation.

- By repeating the process, we will eventually evaluate the operation at the root of the tree, giving the final result of the expression.




- In our Example, we get the final result by executing the projection operation at the root of the tree, using as input the temporary relation created by the join.

- Evaluation as just described is called materialized evaluation, since the result of each intermediate

operation are created and then are used for evaluation of next level operation.

## ② Equivalance Rule

- An Equivalance rule says that expressions of two forms are equivalent.

- We can replace an expression of the first form by an expression of the second form, or vice versa. Since the two expressions would generate the same result on any valid data base.

- We now list the number of general equivalence rules on relational algebra expression.

①. Conjuctive selection operations can be deconstructed into a sequence of individual selections. This information is referred to as a cascade of σ.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

(2) selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

(3) Only the final operations in a sequence of projection operations are needed the others can be omitted. This transaction can also be referred to as a cascade $\Pi$

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E)$$

(4) selection can be combined with Cartesian products and theta joins.

a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

This Expression is just the definition of the theta join.

b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

(5) Theta Join operations are commutative

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

(6) Natural join operations are associative

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

(7) The selection operation distributes over the theta-join operation under the following two conditions:

(a) It distributes when all the attributes in selection condition $\theta_0$ involve only the attributes of one of the expressions being joined.

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

(b) It distributes when selection condition $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$

$$\sigma_{\theta_1 \cap \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

(3) The Projection operation distributes over the theta join operation under the following condition.

(a) $\Pi_{L_1, \theta L_2}(E_1 \bowtie E_2) = (\Pi_{\theta L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$

(b) $\Pi_{L_1, \theta L_4}(E_1 \bowtie_\theta E_2) = \Pi_{L_1, \theta L_3}((\Pi_{L_1, \theta L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \theta L_4}(E_2)))$

(4) The set operations union and intersection are commutative.

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

Set difference is not commutative.

(10) Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$

$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

(11) The selection operation distributes over the union intersection and set difference operations.

$$\sigma_p (E_1 - E_2) = \sigma_p(E_1) - \sigma_p (E_2)$$

- similarly the preceding equivalence with replaced with either '$\cup$' or '$\cap$', also holds further,

$$\sigma_p (E_1 - E_2) = \sigma_p (E_1) - E_2$$

The proceding equivalence, with replaced by $\cap$, also holds, but does not hold if - is replaced by '$\cup$'.

(12) The projection operation distributes over the union operation

$$\Pi_L (E_1 \cup E_2) = (\Pi_L (E_1)) \cup (\Pi_L (E_2))$$

Q14 Discuss selection operation/algorithm for calculating I/o cost in terms of no. of Pages Per I/o.

ns

If One or more indexes on R match the selection, we can use the index to retrieve matching tuples, and apply any remaining selection condition to further restrict the result set.

As an example, consider a selection of the form rname < 'c%' on the Reserves table. Assuming that names are uniformly distributed with respect to the initial letter, for simplicity, we estimate that roughly 10% of Reserves tuples are the result. This is total of 10000 tuples, or 100 Pages. If we have a clustered B+ tree index on the rname field of Reserves, we can retrieve the qualifying tuples with 100 I/os. however if the index is unclustered, we could have up to 10000 I/O's in the worst case. Since each tuple could cause us to

read a page.

As a rule of thumb, it is probably cheaper to simply scan the entire table if over 5% of the tuples are to be retrieved.

Discuss algorithm for computing the join of relations.

Joins are expensive operations and very common. Therefore, they have been widely studied and systems typically support several algorithms to carry out joins.

Consider the join of Reserves and Sailors, with the join condition Reserves.sid = Sailors.sid. Suppose that one of the tables, say Sailors, has an index on the sid column. We can scan Reserves and, for each tuple, use the index to go to the Sailors

for matching tuples. This approach is called index nested loops Join.

suppose that we have a hash-based index using Alternative on the sid attribute of sailors and that it takes about 1.2 I/Os on average to retrieve the appropriate page of the index. Since sid is a key for sailors, we have at most one matching tuple, Indeed, sid in Reserves is a foreign key referring to sailors, and therefore we have exactly one matching sailors tuple for each Reserves tuple; let us consider the cost of scanning Reserves and using the index to retrieve the matching Sailors tuple for each Reserves tuple, The cost of scanning Reserves is 1000. There are 100*1000 tuples in Reserves. For each of these tuples retrieving the index page containing the rid of the matching sailors tuple costs 1.2 I/Os; in addition, we have to retrieve the sailors page containing the qualifying tuple, Therefore we have 100000*(1+1.2) I/Os to

retrieve matching sailors tuples.
The total cost is 221000 I/Os.

If we do not have an index that matches the join condition on either table, we cannot use index nested loops. In this case, we can sort both tables on the join column and then scan them to find matches. This is called sort-merge join.

Q16 Discuss the step in Query processing with diagram?

Ans Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database language into expressions that can be used at the physical level of the file system, a variety of query optimizing transformation and actual evaluation of queries.

- The steps involved in processing a query appear in the figure.

- The basic step are:

(1) Parsing and translation

(2) Optimization

(3) Evaluation

- Before query processing can begin, the system must translate the ~~query~~ query into a usable form.

- Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output.

Q17 Name the two scan algorithm to implement selection operation.

① A1 (Linear search)

- In a linear search, the system scans each file block and tests all records to see whether they satisfy the selection condition.

- An initial seek is required to access the first block of the file. In case blocks of the file are not stored contiguously, extra seeks may be required, but we ignore this effect for simplicity.

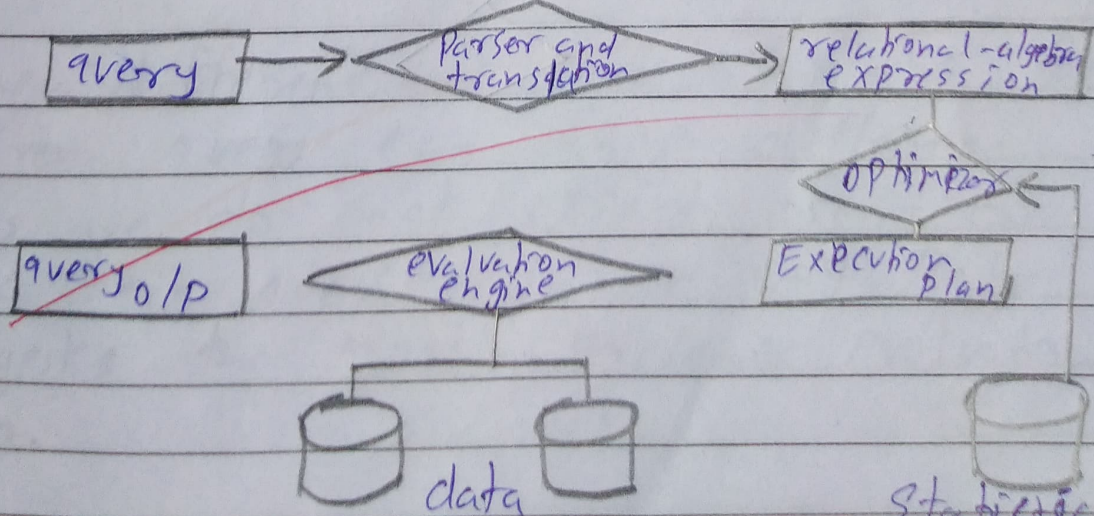
- The cost of linear search, in terms of number of disk operations, is one seek plus denotes the number of blocks in the file, or equivalently

- Thus the first action the system must take in query processing is to translate a given query into it's internal form.

- This translation process is similar to work performed by the parser of a compiler

- In generating the internal form of the query the parser checks of user's query verifies that the relation names appearing in the query are names of the relations in the database and so on.

- The system constructs a parse-tree representation of the query, which it then translate into a relational algebra expression.



- To specify fully how to evaluate a query, we need not only to provide the relational algebra expression, but also to annotate it with instructions specifying how to evaluate each operation.

- Annotations may state the algorithm to be used for a specific operation or the particular index or indices to use.

- A relational algebra operation annotated with instruction on how to evaluate is called an evaluation primitive.

- A sequence of primitive operation that can be used to evaluate a query is a query execution plan or query-evaluation plan.

- The different evaluation plans for a given query can have different costs. We do not expect users to write their queries in a way that suggests the most efficient evaluation plan.

the time cost is $t_S + b_r * t_T$

- For a selection on a key attribute have an average transfer cost of $b_r/2$, but still have a worst case cost of $b_r$ block transfer in addition to one seek.

- Although it may be slower than other algorithm for implementing selection, the linear search algorithm can be applied to any file, regardless of the ordering of the file, or the availability of indices, or the nature of the selection operation.

→ A2 (binary search):

- If the file is ordered on an attribute, and the selection condition is equality comparison on the attribute we can use a binary search to locate records that satisfy the selection. The system perform the binary search on the blocks of the file.

- In the worst case; the number of blocks that need to be examined to find a block containing the required records is $\lceil \log_2(b_r) \rceil$ where $b_r$ denotes the number of blocks in the file.

- Each of these block accesses require a disk seek in addition to a block transfer and the time cost is thus $\lceil \log_2(b_r) \rceil * (t_r + t_s)$

- If the selection is on a nonkey attributes, more than one block may contain required records, and the cost of reading the extra blocks has to be added to the cost estimate we can estimate this number by estimating the size of the selection result and dividing it by the average number of records that are stored contigously, so we only pay a transfer cost $t_T$ per extra block.

Q18 Define Transaction what is a transaction state and what are the different states a transaction could be in? Explain with diagram.

Ans Transaction :-

- Collections of operations that form a single logical unit of work are called transaction

⇒ Transaction state :-

- Transaction State in DBMS are the states through which a transaction goes throughout its lifetime.

⇒ Different states of transaction :-

- Active :-

In this state, the transaction is being executed. This is the Initial state of every transaction.

- Partially committed :-

When a transaction executes its final operation, it is said to be in a partially committed state.

- Failed :-

A transaction is said to be in a failed state if any of the checks made by the database recovery system fails.

- A fail transaction can no longer proceed further.

- Aborted :-

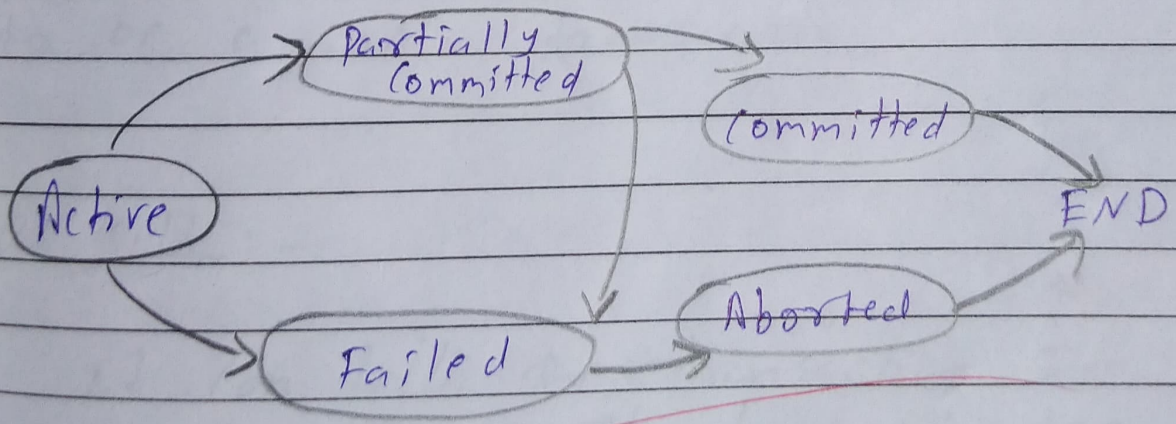
If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operation on the database back to its original state when it was prior to the execution of the transaction. Transaction in this state are called

aborted. The database recovery module can select one of the two operations after a transaction aborts.

- Re start the transaction
- kill the transaction.

- Committed :-

If a transaction executes all its operation successfully it is said to be committed. All its effects are now permanently established on the database system.



Q19 What are two options a transaction has after entering abort?

Ans

When the transaction enters aborted state the system has two options.

① Restart :-

It can restart the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restored transaction is considered to be a new transaction.

② Kill :-

It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program or because the input was bad, or

because the desired data was not found in the database.

Discuss is anomalies due to interleaved transactions.

(i) Reading Uncommitted Data (WR Conflicts)

- The first source of anomalies is that a transaction T2 could read a database object A that has been modified by another transaction T1, which has not yet committed. Such a read is called a dirty read. A simple example illustrates how such a schedule could lead to an inconsistent database state. Consider two transaction T1 and T2 each of which, run alone, preserves database consistency:

T1 transfers $100 from A to B and T2 increments both A and B by 6%.

Suppose that the actions are interleaved so that

(1) The account transfer program T1 deducts $100 from account A, then

(2) The interest deposit Program T2 reads the current values of accounts A and B and adds 6% interest to each, and then

(3) The account transfer program credits $100 to account B.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| | R(B) |
| | W(B) |
| | Commit |
| R(B) | |
| W(B) | |
| Commit | |

## => Unrepeatable Reads (RW conflicts)

- The second way in which anomalous behavior could result is that a transaction T2 could change the value of an object A that has been read by a transaction T1, while T1 is still in progress.

If T1 tries to read the value of A again, it will get a different result, even though it has not modified A in the meantime. This situation could not arise in a serial execution of two transactions; it is called an unrepeatable read.

This situation can never arise in a serial execution of T1 and T2; the second transaction would read A and see 0 and therefore not proceed with the order.

Overwriting Uncommitted Data
(ww conflicts).

The third source of anomalous behavior is that a transaction T2 could overwrite the value of an object A, which has already been modified by a transaction T1, while T1 is still in progress. Even if T2 does not read the value of A written by T1, a potential problem exists as the following example illustrates.

Suppose that Harry and Larry are two employees, and their salaries must be kept equal. Transaction T1 sets their salaries to $2000 and transaction T2 sets their salaries to $1000. If we execute these in the serial order T1 followed by T2, both receive the salary $1000. the serial order T2 followed by T1 gives each the salary $2000. Either of these is acceptable from a consistency standpoint. Note that

neither transaction reads a salary value before writing it such a write is called a blind write, for obvious reasons.

The problem is that we have a lost update. The first transaction to commit, T2 overwrote Larry's salary as set by T1. In the serial order T2 followed by T1, Larry's salary should reflect T1's update rather than T2's but T1's update is lost.

2) Explain schedules involving aborted transactions & Recoverable schedules.

Ans A Serializable schedule over a set S of transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete Serial schedule over the set of Committed transaction in S.

- The definition of serializability relies on the actions of aborted

transactions being undone completely, which may be impossible in some situations.

- Suppose that,

(1) An account transfer program T1 deducts $100 from account A, then

(2) An interest deposit program T2 reads the current values of accounts A and B and add 6% interest to each, then commits and then

(3) T1 is aborted.

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| | R(B) |
| | W(B) |
| | Commit |
| Abort | |

- Now, $T_2$ has read a value for A that should never have been there. If $T_2$ had not yet committed, we could deal with the situation by cascading the abort of $T_1$ and also aborting $T_2$, this process recursively aborts any transaction that read data written by $T_2$ and so on. But $T_2$ has already commited and so we cannot undo its actions we say that such a schedule is unrecoverable.

- In a recoverable schedule, transaction commit only after all transaction whose changes they read commit.

- If transaction read only the changes of commited transaction, not only is the schedule recoverable but also aborting a transaction can be accomplished without cascading the abort to other transaction such a schedule is said to avoid cascading abort.

Q22 Compare Index Sequential File and B+ tree Index files.

ns

| Index Sequential File | B+ tree Index files |
|---|---|
| - Performance degrads as file grows since many overflow blocks get created. | - Automatically reorganised itself with small, local changes in the face of insertion and deletions. |
| - Periodic Reorganisation of file is required | - Reorganization of file is not required to maintain performance |
| - Less insertion and deletion overhead, space overhead. | - Extra insertion & deletion overhead, space overhead. |

# Explain cascading aborts

Even if a schedule is recoverable to recover correctly from the failure of a ~~transaction~~ transaction $T_i$, we may have to roll back several transactions. Such transaction occur when transaction have read data written by $T_i$

| $T_{10}$ | $T_{11}$ | |
|---|---|---|
| read (A) | | |
| read (B) | | |
| write (A) | | |
| | read (A) | |
| | write (A) | |
| | | read (A) |

- Transaction $T_{10}$ ~~writes values~~ of A that is read by transaction $T_{11}$. transaction $T_{11}$ writes value of A which is again Read by $T_{10}$.

- suppose at this point $T_{10}$ fails $T_{10}$ must roll back. since $T_{11}$ is dependent on $T_{10}$. $T_{11}$ must be roll back. since $T_{12}$ is dependent on $T_{11}$, $T_{12}$ must roll back

- This phenomen in which a single transaction failure leads to a series of transaction roll back / failure is called cascading roll back / cascading aborted.

- Cascading roll back is undesirable. since leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading roll back cannot occur.

12/8/2020

# Assignment 2

1) Discuss Thomas Write Rule in detail.

Ans In optimistic concurrency control, a timestamp ordering is imposed on transactions, and validation checks that all conflicting actions occured in the same order.

Timestamp can also be used in another way: each transaction can be assigned a timestamp at startup and we can ensure, at execution time that if action ai of transaction Ti conflicts with action aj of transaction Tj, ai occurs before aj if $TS(Ti) < TS(Tj)$. If an action violates this ordering the transaction is aborted and restarted.

To implement this concurrency control scheme, every database object O is given a "read timestamp RTS(O)" and "write timestamp WTS(O).

Let us consider what happens when transaction T wants to write object O.

1. If $TS(T) < RTS(O)$, the write action conflicts with the most recent read action of O, and T is therefor

aborted and restarted.

2. If TS(T) < WTS(O), a naive approach would be to abort T because its write action conflicts with the most recent write of O and is out of timestamp order. It turns out that we can safely ignore such writes and continue ignoring outdated write is called the Thomas write Rule.

3. Otherwise, T write O and WTS(O) is set to TS(T)

* The Thomas write Rule.

- We now consider the justification for the thomas write Rule. If TS(T) < WTS(O), the current write action has, in effect, been made absolute by the most recent write of O, which follows the current write according to the timestamp ordering on transactions.

We can think of T's write action as if it had occured immediately before the most recent write of O and was never read by anyone.

If the Thomas Write Rule is not used, that is T, is aborted in case (2) above, the time stamp Protocol like 2PL allows only Conflict Serializable schedules are Permitted that are not Conflict Serializable.

| T1 | T2 |
|---|---|
| R(A) | |
| | W(A) |
| | Commit |
| W(A) | |
| Commit | |

A Serializable schedule that is not Conflict serializable.

T1's Read are precedes T1's write of the same object, this schedule is not Conflict Serializable. The Thomas Write Rule relies on the observation that T2's write is never seen by any transaction and the schedule is therefore equivalent to the serializable schedule obtained by deleting this write action, which is shown in figure.

| $T_1$ | $T_2$ |
| --- | --- |
| R(A) | commit |
| W(A) | |
| commit | |

A conflict serializable schedule.

2) Discuss three Phase of ARIES Recovery algorithm.

Ans

ARIES is a recovery algorithm that is designed to work with a steal, no force approach. When the recovery manager is invoked after a crash, restart proceeds in three Phases.

1) Analysis :-

Identifies dirty pages in the buffer Pool and active transactions at the time of the crash.

2) Redo :-

Reports all actions, starting from an appropriate point in the log, and restores the database state to what it was at the time of the crash.

3) Undo :-

Undoes the actions of transaction then did not commit so that database reflects only the actions of commited transactions.

3) Short note on CLR.

Ans A compensation log record (CLR) is written just before the change recorded in an update log record is undone.

A compensation log record (also contains a field called Undo Next LSN, which is the LSN of the next log record that is to be undone for the transaction that wrote update record U. This field in C is set to the value of prevLSN in U.

Unlike an update log record, a CLR describes an action that will never be ~~described an action that will never be~~ undone, that is we never undo an undo action. The reason is simple an update log record describes changes made by a transaction during normal execution and the transaction may subsequently be aborted an action taken to roll back a transaction for which the decision to abort has already been made

Thus, the transaction action must be rollback and the undo action describe by the CLR is definetly required.

This observation is very useful because it bounds the amount of space needed for the log during restart from a crash. The number of CLRs that can be written during undo is no more than the number of update log records for active transaction at the time of the crash.

It may well happen that CLR is written to stable storage but that the undo action that it describe is not yet written to disk when the system crashes again.

4) Define transaction with references to MySQL.

Ans A transaction is a set of one or more SQL statements that are logically grouped together and that must be either applied to the database in their entirely or not applied at all.

Consider the commonly cited example of a funds transfer from one account to another. In its most simple form, this transfer will involve two update statements one to reduce the account balance in the "from" account, and another to increase the account balance in "to" account. Suppose that "from" account has been updated but then the change to the "to" account cannot be completed. We must be sure to undo that first update; or the money that was to be transferred will have in effect "disappeared".

We expect database transactions to conform to the ACID principle which means the transaction should be,

Atomic :- The transaction is indivisible either all the statements in the

transaction are applied to the database or none are,

Consistent :- The database remains in a consistent state before and after transaction execution.

Isolated :- while multiple transaction can be executed by one or more users simultaneously, one transaction should not see the effects of other concurrent transactions.

Durable :- once a transaction is saved to the database its changes are expected to persist. Even if the user turns off her computer or the database server goes down the changes will be saved. This is usually means that the result of the transaction must be written to a non volatile form of storage such a harddisk. Stored program provide an excellent mechanism for defining encapsulating and managing transactions. Without these features available in Stored programs the calling provide the logic to control locking and handle transaction failure.

With MySQL stored program support we can now encapsulate the multiple interdependent SQL statement of the transaction into single stored program.

There are two most popular storage engines used with MySQL are MyISAM and InnoDB.

5) Discuss four properties of transaction to ensure integrity of data (ACID).

Ans ① Atomic:

- User should be able to regard the execution of each transaction as atomic, either all actions are carried out or non are user should not have to worry about the effect of incomplete transaction.

② Consistency:

- Each transaction, run by itself with no concurrent execution of other transactions must preserve the consistency of the database. This property is called consistency and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

③ Isolation:

User should be able to understand a transaction without considering the effect of other concurrency executing transaction, even if the DBMS

interleaves the actions of several transaction for performance reasons. This property is sometimes referred to as isolation. Transaction are isolated or protected from the effects of concurrency scheduling other transactions.

④ Durability:

once the DBMS informs the user that a transaction has been successfully completed its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability

⇒ The acronym ACID is sometimes used to refer to the four properties of transactions.

Page No.
Date :

6) Discuss the role of DBA in security with reference to Database.

Ans The database administrator (DBA) plays an important role in enforcing the security-related aspects of Database design.

In conjunction with the owners of the data, the DBA will probably also contribute to developing a security policy. The DBA has a special account, which we will call the system account and is responsible for the overall security of the system.

→ In particular the DBA deals with the following.

① Creating new accounts:

Each new user or group must be assigned an authorization id and a password. Note that application program that access the database have the same authorization id as the user execution the program.

② Mandatory control issues:

If the DBMS supports mandatory

control some customized systems for applications with very high security requirements provide such the DBA must assign security classes to each database object and assign security clearances to each authorization id accordance with the chosen security policy.

The DBA is also responsible for maintaining the audit trail, which is essentially the log of updates with the authorization id added to each log entry. This log is just a minor extension of the log mechanism used to recover from crashes.

Additionally, the DBA may choose to maintain a log of all actions including reads, performed by a user, Analyzing such histories of how the DBMS is accessed can help prevent security violations by identifying suspicious patterns before an intruder finally succeeds in breaking in or it can help track down an intruder after a violation has been detected.

7) Discuss two important assumptions with respect to transactions.

Ans

① Transactions interact with each other only via database read and write operations. For example, they are not allowed to exchange messages.

② A database is a field collection of independent objects. When objects are added to or deleted from a database or there are relationships between database objects what we want to exploit for performance, some additional issue arise.

/**********************************************************************************************************

Name : Pradip S Karmakar

Class : M.C.A 2

Roll_No : 10

Subject : RDBMS

**********************************************************************************************************

Question : Creare a procedure for emp which will fetch all employees details and display on screen.

**********************************************************************************************************/

// Procedure For Creating Tables

```
DELIMITER //
CREATE PROCEDURE create_table()
BEGIN
   create table emp(
      emp_id int primary key auto_increment,
      empname varchar(50),
      position varchar(50),
      salary decimal(8,2)
   );
END //
```

// Procedure For Inserting Data in Tables

```
CREATE PROCEDURE insert_data()
BEGIN
   insert into emp values(1,'Pradip','CEO_APPLE',80000),
   (2,'Ajinkya','CEO_GOOGLE','60000'),
   (3,'Nirav','CEO_MIRCOSOFT','50000'),
   (4,'Milind','CEO_SAMSUNG','60000'),
   (5,'Lakshya','CEO_FACEBOOK','90000');
END //
```

// Procedure With CURSORS for Display Data on the Screen

```
CREATE PROCEDURE cur_pro()
BEGIN
   DECLARE emp_id int;
   DECLARE emp_name varchar(50);
   DECLARE position varchar(50);
   DECLARE salary decimal(8,2);
   DECLARE c_finish integer DEFAULT 0;
   DECLARE curs cursor for select * from emp;
   DECLARE CONTINUE HANDLER for NOT FOUND set c_finish = 1;
   OPEN curs;
   get_line : LOOP
      FETCH curs into emp_id,emp_name,position,salary;
      IF c_finish = 1 THEN
         LEAVE get_line;
      END IF;
      SELECT CONCAT(emp_id,CONCAT(' | ',CONCAT(emp_name,CONCAT(' |
',CONCAT(position,CONCAT(' | ',salary))))))  as Employee_Data;
   END LOOP get_line;
   CLOSE curs;
```

```
END //

DELIMITER ;

/*************************************************************************************
************************

OUTPUT :

MariaDB [test]> call create_table();
Query OK, 0 rows affected (0.042 sec)

MariaDB [test]> call insert_data();
Query OK, 5 rows affected (0.022 sec)

MariaDB [test]> call cur_pro;
+----------------------------------+
| Employee_Data                    |
+----------------------------------+
| 1 | Pradip | CEO_APPLE | 80000.00 |
+----------------------------------+
1 row in set (0.001 sec)

+----------------------------------+
| Employee_Data                    |
+----------------------------------+
| 2 | Ajinkya | CEO_GOOGLE | 60000.00 |
+----------------------------------+
```

1 row in set (0.003 sec)

+--------------------------------------+

| Employee_Data |

+--------------------------------------+

| 3 | Nirav | CEO_MIRCOSOFT | 50000.00 |

+--------------------------------------+

1 row in set (0.005 sec)

+--------------------------------------+

| Employee_Data |

+--------------------------------------+

| 4 | Milind | CEO_SAMSUNG | 60000.00 |

+--------------------------------------+

1 row in set (0.007 sec)

+---------------------------------------+

| Employee_Data |

+---------------------------------------+

| 5 | Lakshya | CEO_FACEBOOK | 90000.00 |

+---------------------------------------+

1 row in set (0.011 sec)

Query OK, 0 rows affected (0.013 sec)

**********
**********/

==========

Question 2 : Create a cursor to find list of all employees jn a deoartment passed as an argument from

the employee table.

==========

```
DELIMITER //

MariaDB [test]> CREATE PROCEDURE create_tables()
    ->    BEGIN
    ->        create table department(
    ->            dept_id int primary key,
    ->            deptname varchar(30)
    ->        );
    ->
    ->        create table employee(
    ->            emp_id int primary key auto_increment,
    ->            empname varchar(30),
    ->            dept_id int,
    ->            designation varchar(20),
    ->            salary decimal(10,2),
    ->            FOREIGN KEY (dept_id) REFERENCES department(dept_id)
    ->        );
    ->
    ->    END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> call create_tables() //
```

```
Query OK, 0 rows affected (0.125 sec)

**********************************************************************************
*****

MariaDB [test]> CREATE PROCEDURE insert_data()
    ->    BEGIN
    ->        insert into department values(1,'Accounts'),
    ->        (2,'Production'),
    ->        (3,'Marketing');
    ->
    ->        insert into employee values(1,'Pradip',1,'Manager',80000),
    ->        (2,'Ajinkya',2,'Clerk',60000),
    ->        (3,'Nirav',3,'Staff',50000),
    ->        (4,'Milind',2,'Manager',60000),
    ->        (5,'Lakshya',3,'Staff',90000);
    ->    END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> call insert_data() //
Query OK, 8 rows affected (0.032 sec)

MariaDB [test]> select * from employee //
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 60000.00 |
|      3 | Nirav   |       3 | Staff       | 50000.00 |
|      4 | Milind  |       2 | Manager     | 60000.00 |
|      5 | Lakshya |       3 | Staff       | 90000.00 |
```

```
+--------+---------+---------+-------------+----------+
5 rows in set (0.000 sec)

MariaDB [test]> select * from department //
+---------+------------+
| dept_id | deptname   |
+---------+------------+
|       1 | Accounts   |
|       2 | Production |
|       3 | Marketing  |
+---------+------------+
3 rows in set (0.000 sec)

**********************************************************************************
*****

MariaDB [test]> set @dname = 'Production' //
Query OK, 0 rows affected (0.000 sec)

MariaDB [test]> set @list = '' //
Query OK, 0 rows affected (0.000 sec)

MariaDB [test]> CREATE PROCEDURE cur_pro(IN dept_name varchar(100), INOUT list varchar(100))
    ->    BEGIN
    ->    DECLARE emp_name varchar(50);
    ->    DECLARE c_finish integer DEFAULT 0;
    ->    DECLARE curs cursor for select empname from employee where dept_id = (select dept_id
from department where deptname = dept_name );
    ->    DECLARE CONTINUE HANDLER for NOT FOUND set c_finish = 1;
    ->    OPEN curs;
    ->    get_line : LOOP
    ->        FETCH curs into emp_name;
```

```
    ->       IF c_finish = 1 THEN
    ->          LEAVE get_line;
    ->       END IF;
    ->       set list = CONCAT(list,CONCAT(emp_name, " | "));
    ->    END LOOP get_line;
    ->    CLOSE curs;
    ->    END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call cur_pro(@dname,@list);
Query OK, 0 rows affected (0.000 sec)

MariaDB [test]> select @list as LISTS;
+---------------------+
| LISTS               |
+---------------------+
| Ajinkya | Milind |  |
+---------------------+
1 row in set (0.000 sec)
```

==========

Qusetion 3 : Create a cursor to increment the salary based on the designation

==========

```
DELIMITER //
```

```
MariaDB [test]> set @increment = 1000 //
Query OK, 0 rows affected (0.000 sec)

MariaDB [test]> set @designation = 'Staff' //
Query OK, 0 rows affected (0.000 sec)


MariaDB [test]> CREATE PROCEDURE salary_increment(IN desig varchar(30) , IN incre decimal(10,2) )
    -> BEGIN
    ->     DECLARE empid int;
    ->     DECLARE saly decimal(10,2);
    ->     DECLARE c_finish integer DEFAULT 0;
    ->     DECLARE curs cursor for select emp_id,salary from employee where designation = desig;
    ->     DECLARE CONTINUE HANDLER for NOT FOUND set c_finish = 1;
    ->     OPEN curs;
    ->     get_line : LOOP
    ->         FETCH curs into empid,saly;
    ->         IF c_finish = 1 THEN
    ->             LEAVE get_line;
    ->         END IF;
    ->         UPDATE employee set salary = (saly + incre) where emp_id = empid;
    ->     END LOOP get_line;
    ->     CLOSE curs;
    -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> call salary_increment(@designation,@increment);
Query OK, 2 rows affected (0.022 sec)
```

```
MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 60000.00 |
|      3 | Nirav   |       3 | Staff       | 51000.00 |
|      4 | Milind  |       2 | Manager     | 60000.00 |
|      5 | Lakshya |       3 | Staff       | 91000.00 |
+--------+---------+---------+-------------+----------+
5 rows in set (0.000 sec)
```

==========

/**********
**********

Name : Pradip S Karmakar

Class : M.C.A 2

Roll_No : 10

Subject : RDBMS

**********
**********/

==========

GENERAL PL/SQL BLOCKS

==========

Question 1 : Input two numbers and find out all arthmetic operations( +, -, x, / ).

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> create procedure question1( IN a int,IN b int )
    -> BEGIN
    -> DECLARE c INT;
    ->
    -> set c = a+b;
    -> select c as Addition;
    ->
    -> set c = a-b;
    -> select c as Subtraction;
    ->
```

```
    -> set c = a*b;
    -> select c as Multiplication;
    ->
    -> set c = a/b;
    -> select c as Division;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> delimiter ;
MariaDB [test]> call artmetic(10,5);
+----------+
| Addition |
+----------+
|       15 |
+----------+
1 row in set (0.000 sec)

+-------------+
| Subtraction |
+-------------+
|           5 |
+-------------+
1 row in set (0.006 sec)

+----------------+
| Multiplication |
+----------------+
|             50 |
+----------------+
1 row in set (0.008 sec)
```

```
+----------+
| Division |
+----------+
|        2 |
+----------+
1 row in set (0.012 sec)


Query OK, 0 rows affected (0.015 sec)
```

==========

Question 2 : Enter rollno and three subject marks. Find out Total, percentage, result & Grade.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> CREATE PROCEDURE question2( IN r_no int, IN marks1 int, IN marks2 int, IN marks3 int )
    -> BEGIN
    -> DECLARE total INT;
    -> DECLARE percentage FLOAT;
    -> DECLARE Grade VARCHAR(15);
    -> DECLARE result VARCHAR(4);
    ->
    -> set total = marks1 + marks2 + marks3;
    -> set percentage = (total * 100) / 300;
    ->
    -> IF percentage > 80 THEN
```

```
    -> set Grade = "DISTINCTION";
    -> set result = "PASS";
    -> ELSEIF percentage > 70 THEN
    -> set Grade = "FIRST CLASS";
    -> set result = "PASS";
    -> ELSEIF percentage > 60 THEN
    -> set Grade = "SECOND CLASS";
    -> set result = "PASS";
    -> ELSEIF percentage > 50 THEN
    -> set Grade = "THIRD CLASS";
    -> set result = "PASS";
    -> ELSEIF percentage > 35 THEN
    -> set Grade = "PASS";
    -> set result = "PASS";
    -> ELSE
    -> set Grade = "FAIL";
    -> set result = "FAIL";
    -> END IF;
    ->
    -> SELECT r_no,marks1,marks2,marks3,total,percentage,Grade,result as RESULT;
    ->
    -> END //
Query OK, 0 rows affected (0.026 sec)

MariaDB [test]>
MariaDB [test]> DELIMITER ;
MariaDB [test]> call question2(10,78,95,71);
+------+--------+--------+--------+-------+------------+-------------+--------+
| r_no | marks1 | marks2 | marks3 | total | percentage | Grade       | RESULT |
+------+--------+--------+--------+-------+------------+-------------+--------+
|   10 |     78 |     95 |     71 |   244 |    81.3333 | DISTINCTION | PASS   |
```

```
+------+--------+--------+--------+-------+------------+-------------+--------+
```

1 row in set (0.001 sec)

Query OK, 0 rows affected (0.005 sec)

==========

Question 3 : Print First 10 Odd Number unsing Loops.

==========

```
MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE PROCEDURE question3()
    -> BEGIN
    -> DECLARE odd varchar(50);
    -> DECLARE cnt INT;
    -> DECLARE num INT;
    -> SET odd = '';
    -> SET num = 1;
    -> SET cnt = 1;
    -> loop_odd: LOOP
    ->     IF cnt > 10 THEN
    ->         LEAVE loop_odd;
    ->     END IF;
    ->     IF (num mod 2) THEN
    ->         SET odd = CONCAT(odd,num,"  ");
    ->         SET cnt= cnt + 1;
    ->         SET num = num + 1;
    ->     ELSE
```

```
    ->         SET num = num + 1;
    ->     END IF;
    -> END LOOP;
    ->
    -> select odd as FIRST_10_ODD_NUMBERS;
    ->
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call question3;
+-------------------------------------+
| FIRST_10_ODD_NUMBERS                |
+-------------------------------------+
| 1  3  5  7  9  11  13  15  17  19   |
+-------------------------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.006 sec)
```

==========

Question 4 : Print Prime Number Upto 10 using While Loops.

==========

```
MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE PROCEDURE question4()
```

```
   -> BEGIN
   -> DECLARE prime varchar(50);
   -> DECLARE cnt INT;
   -> DECLARE i INT;
   -> DECLARE num INT;
   -> SET prime = '';
   -> SET i = 1;
   ->
   -> WHILE i <= 10 DO
   ->    SET cnt = 0;
   ->    SET num = 1;
   ->    WHILE num <= (i/2) DO
   ->       IF (i mod num = 0) THEN
   ->          SET cnt = cnt + 1;
   ->       END IF;
   ->       SET num = num + 1;
   ->    END WHILE;
   ->    IF cnt = 1 THEN
   ->       SET prime = CONCAT(prime,i,"   ");
   ->    END IF;
   ->    SET i = i + 1;
   -> END WHILE;
   ->
   -> select prime as PRIME_NUMBER_UPTO_10;
   ->
   -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call question4;
+----------------------+
```

```
| PRIME_NUMBER_UPTO_10 |
+----------------------+
| 2  3  5  7           |
+----------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.005 sec)
```

==========

Question 5 : Print MAX & MIN number from 3 numbers.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> CREATE PROCEDURE question5(IN num1 int, IN num2 int, IN num3 int)
    -> BEGIN
    -> DECLARE Minimum INT;
    -> DECLARE Maximum INT;
    -> set Maximum = 0;
    -> set Minimum = 0;
    -> IF (num1 > num2) AND (num1 > num3) THEN
    ->     set Maximum = num1;
    -> ELSEIF (num2 > num1) AND (num2 > num3) THEN
    ->     set Maximum = num2;
    -> ELSE
    ->     set Maximum = num3;
    -> END IF;
    -> IF (num1 < num2) AND (num1 < num3) THEN
    ->     set Minimum = num1;
```

```
   -> ELSEIF (num2 < num1) AND (num2 < num3) THEN
   ->     set Minimum = num2;
   -> ELSE
   ->     set Minimum = num3;
   -> END IF;
   ->
   -> select Maximum,Minimum;
   ->
   -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call question5(10,12,15);
+---------+---------+
| Maximum | Minimum |
+---------+---------+
|      15 |      10 |
+---------+---------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.005 sec)
```

==========

Question 6 : Get Input From user as empid and check whether that empid is exist, if

Not then Show appropriate Message else show empname and salary.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> CREATE PROCEDURE question6(IN empid int)
   -> BEGIN
   -> IF (select emp_id from employee where emp_id = empid) = empid THEN
   ->    select empname,salary from employee where emp_id = empid;
   -> ELSE
   ->    select "NO SUCH EMPLOYEE ID EXIST" as MESSAGE;
   -> END IF;
   -> END //
Query OK, 0 rows affected (0.019 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call question6(1);
+---------+----------+
| empname | salary   |
+---------+----------+
| Pradip  | 80000.00 |
+---------+----------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.007 sec)

MariaDB [test]> call question6(4);
+---------+----------+
| empname | salary   |
+---------+----------+
| Milind  | 60000.00 |
+---------+----------+
1 row in set (0.000 sec)
```

```
Query OK, 0 rows affected (0.007 sec)

MariaDB [test]> call question6(8);
+--------------------------+
| MESSAGE                  |
+--------------------------+
| NO SUCH EMPLOYEE ID EXIST |
+--------------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.007 sec)
```

==========

Question 7 : Get Input From user as empid and check whether that empid is exist, if Not then Show appropriate Message else show empname and salary.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> CREATE PROCEDURE create_table()
   -> BEGIN
   ->    create table customer(
   ->       cust_id int primary key auto_increment,
   ->       cust_name varchar(15),
   ->       address varchar(150),
   ->       city varchar(25)
   ->    );
   -> END //
Query OK, 0 rows affected (0.018 sec)
```

```
MariaDB [test]> call create_table //
Query OK, 0 rows affected (0.040 sec)

MariaDB [test]> CREATE PROCEDURE insert_data()
    -> BEGIN
    ->     insert into customer values(1,'Pradip','P-block','Navsari'),
    ->     (2,'Ajinkya','E-block','Gandhidham'),
    ->     (3,'Nirav','C-block','Mundra'),
    ->     (4,'Milind','F-block','Navranpura'),
    ->     (5,'Lakshya','G-block','Gandhidham');
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> call insert_data //
Query OK, 5 rows affected (1.609 sec)

MariaDB [test]> CREATE PROCEDURE question7(IN custid int,IN custname varchar(15), IN
cust_address varchar(150), IN cust_city varchar(25))
    -> BEGIN
    -> IF (select cust_id from customer where cust_id = custid) = custid THEN
    ->     select "CUSTOMER ID ALREADY EXIST" as MESSAGE;
    -> ELSE
    ->     insert into customer values(custid,custname,cust_address,cust_city);
    -> END IF;
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call question7(6,'sudip','Kolkata','S-block');
Query OK, 1 row affected (0.007 sec)
```

```
MariaDB [test]> select * from customer;
+---------+-----------+---------+------------+
| cust_id | cust_name | address | city       |
+---------+-----------+---------+------------+
|       1 | Pradip    | P-block | Navsari    |
|       2 | Ajinkya   | E-block | Gandhidham |
|       3 | Nirav     | C-block | Mundra     |
|       4 | Milind    | F-block | Navranpura |
|       5 | Lakshya   | G-block | Gandhidham |
|       6 | sudip     | Kolkata | S-block    |
+---------+-----------+---------+------------+
6 rows in set (0.000 sec)

MariaDB [test]> call question7(3,'kamal','Kolkata','k-block');
+--------------------------+
| MESSAGE                  |
+--------------------------+
| CUSTOMER ID ALREADY EXIST |
+--------------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.007 sec)
```

==========

Functions

==========

Question 1 : Input name and count the length of the name.

==========

```
MariaDB [test]> CREATE FUNCTION fun_question1(name varchar(20))
    -> RETURNS INT
    ->
    -> BEGIN
    -> DECLARE len INT DEFAULT 0;
    ->
    -> set len = LENGTH(name);
    ->
    -> Return len;
    -> END //
Query OK, 0 rows affected (1.578 sec)
```

```
MariaDB [test]> CREATE PROCEDURE Q1(IN name varchar(20))
    -> BEGIN
    -> DECLARE len INT DEFAULT 0;
    -> set len = fun_question1(name);
    -> select len;
    -> END //
Query OK, 0 rows affected (0.025 sec)
```

```
MariaDB [test]> delimiter ;
MariaDB [test]> call Q1("pradip");
+------+
| len  |
+------+
|    6 |
+------+
1 row in set (0.001 sec)

Query OK, 0 rows affected (0.005 sec)

MariaDB [test]> call Q1("pradip karmakar");
+------+
| len  |
+------+
|   15 |
+------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.007 sec)
```

==========

Question 2 : WAF which accepts one number and return TRUE if no is prime and return FALSE if No. is not prime.

==========

```
MariaDB [test]> DELIMITER //
```

```
MariaDB [test]> CREATE FUNCTION Prime(n INT)
    -> RETURNS BOOL
    -> BEGIN
    ->     DECLARE i INT DEFAULT 0;
    ->     DECLARE FLAG INT DEFAULT 0;
    ->     IF n = 1 THEN
    ->         RETURN FALSE;
    ->     ELSE
    ->         SET i = 2;
    ->         MYLOOP : WHILE i <= (n/2) DO
    ->             IF(n mod i = 0) THEN
    ->                 SET FLAG = 1;
    ->                 LEAVE MYLOOP;
    ->             END IF;
    ->             SET i = i + 1;
    ->         END WHILE;
    ->         IF FLAG = 1 THEN
    ->             RETURN FALSE;
    ->         ELSE
    ->             RETURN TRUE;
    ->         END IF;
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> select  Prime(1);
+----------+
| Prime(1) |
+----------+
|        0 |
```

```
+----------+
1 row in set (0.000 sec)

MariaDB [test]> select  Prime(2);
+----------+
| Prime(2) |
+----------+
|        1 |
+----------+
1 row in set (0.000 sec)

MariaDB [test]> select  Prime(3);
+----------+
| Prime(3) |
+----------+
|        1 |
+----------+
1 row in set (0.000 sec)

MariaDB [test]> select  Prime(4);
+----------+
| Prime(4) |
+----------+
|        0 |
+----------+
1 row in set (0.000 sec)
```

==========

Question 3 : Write a function which accepts the department no and returns maximum salary of that

Department. Handle the error if deptno does not exist or select statement return more than one row.

==========

```
MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE FUNCTION get_max(dept_no INT)
    -> RETURNS int
    -> BEGIN
    -> DECLARE get_salary DECIMAL(8,2) DEFAULT 0;
    -> DECLARE row INT default 0;
    -> SELECT COUNT(*) INTO row FROM employee WHERE dept_id = dept_no;
    ->     IF (row > 0) THEN
    ->         SELECT MAX(salary) INTO get_salary FROM employee WHERE dept_id = dept_no GROUP BY dept_id;
    ->         RETURN get_salary;
    ->     ELSE
    ->         RETURN -404;
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 60000.00 |
```

```
|     3 | Nirav   |       3 | Staff       | 51000.00 |
|     4 | Milind  |       2 | Manager     | 60000.00 |
|     5 | Lakshya |       3 | Staff       | 91000.00 |
+--------+---------+---------+-------------+----------+
5 rows in set (0.000 sec)


MariaDB [test]> select get_max(3);
+------------+
| get_max(3) |
+------------+
|     91000 |
+------------+
1 row in set (0.001 sec)


MariaDB [test]> select get_max(9);
+------------+
| get_max(9) |
+------------+
|      -404 |
+------------+
1 row in set (0.000 sec)
```

==========

Question 4 : Write a function to display whether the entered (User Input) employee no exists or not.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE FUNCTION isexists(emp_no INT)
    -> RETURNS VARCHAR(25)
    -> BEGIN
    ->     DECLARE row INT DEFAULT 0;
    ->     SELECT COUNT(*) INTO row FROM employee WHERE emp_id = emp_no;
    ->     IF row > 0 THEN
    ->         RETURN "EMPLOYEE EXIST";
    ->     ELSE
    ->         RETURN "EMPLOYEE DOES NOT EXIST";
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]>
MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 60000.00 |
|      3 | Nirav   |       3 | Staff       | 51000.00 |
|      4 | Milind  |       2 | Manager     | 60000.00 |
|      5 | Lakshya |       3 | Staff       | 91000.00 |
+--------+---------+---------+-------------+----------+
5 rows in set (0.000 sec)

MariaDB [test]> select isexists(4);
+----------------+
| isexists(4)    |
```

```
+----------------+
| EMPLOYEE EXIST |
+----------------+
1 row in set (0.003 sec)

MariaDB [test]> select isexists(9);
+-------------------------+
| isexists(9)             |
+-------------------------+
| EMPLOYEE DOES NOT EXIST |
+-------------------------+
1 row in set (0.000 sec)
```

==========

Question 5 : WAF which accepts one no and returns that no+100. Use INOUT mode.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE FUNCTION summation(num INT)
    -> RETURNS INT
    -> BEGIN
    -> SET num = num + 100;
    -> RETURN num;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
```

```
MariaDB [test]> SELECT summation(95);
+---------------+
| summation(95) |
+---------------+
|           195 |
+---------------+
1 row in set (0.000 sec)



MariaDB [test]> SELECT summation(-45);
+----------------+
| summation(-45) |
+----------------+
|             55 |
+----------------+
1 row in set (0.000 sec)
```

==========

Question 6 : WAF which accepts the empno.

If salary<10000 than give raise by 30%.

If salary<20000 and salary>=10000 than give raise by 20%.

If salary>20000 than give raise by 10%. Handle the error if any.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE FUNCTION salary_raise(emp_no INT)
    -> RETURNS VARCHAR(30)
    -> BEGIN
```

```
   -> DECLARE get_sal DECIMAL(8,2) DEFAULT 0;
   -> DECLARE row INT DEFAULT 0;
   -> SELECT COUNT(*) INTO row FROM employee WHERE emp_id = emp_no;
   ->    IF(row > 0) THEN
   ->       SELECT salary INTO get_sal FROM employee WHERE emp_id = emp_no;
   ->       IF get_sal > 20000 THEN
   ->          SET get_sal = get_sal + (get_sal*10)/100;
   ->          update employee set salary = get_sal WHERE emp_id = emp_no;
   ->       ELSEIF get_sal > 10000 THEN
   ->          SET get_sal = get_sal + (get_sal*20)/100;
   ->          update employee set salary = get_sal WHERE emp_id = emp_no;
   ->       ELSE
   ->          SET get_sal = get_sal + (get_sal*30)/100;
   ->          update employee set salary = get_sal WHERE emp_id = emp_no;
   ->       END IF;
   ->       RETURN CONCAT('Salary Raised To : ',get_sal);
   ->    ELSE
   ->       RETURN CONCAT('No Such Employee ID Exits');
   ->    END IF;
   -> END //
Query OK, 0 rows affected (1.785 sec)

MariaDB [test]> DELIMITER ;



MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 60000.00 |
```

```
|     3 | Nirav   |       3 | Staff       | 15000.00 |
|     4 | Milind  |       2 | Manager     | 60000.00 |
|     5 | Lakshya |       3 | Staff       |  9000.00 |
+--------+---------+---------+-------------+----------+
5 rows in set (0.000 sec)


MariaDB [test]> SELECT salary_raise(3);
+----------------------------+
| salary_raise(3)            |
+----------------------------+
| Salary Raised To : 18000.00 |
+----------------------------+
1 row in set (0.004 sec)


MariaDB [test]> SELECT salary_raise(5);
+----------------------------+
| salary_raise(5)            |
+----------------------------+
| Salary Raised To : 11700.00 |
+----------------------------+
1 row in set (0.027 sec)


MariaDB [test]> SELECT salary_raise(2);
+----------------------------+
| salary_raise(2)            |
+----------------------------+
| Salary Raised To : 66000.00 |
+----------------------------+
```

1 row in set (0.004 sec)

MariaDB [test]> SELECT salary_raise(6);

```
+--------------------------+
| salary_raise(6)          |
+--------------------------+
| No Such Employee ID Exits |
+--------------------------+
```

1 row in set (0.000 sec)

MariaDB [test]> select * from employee;

```
+--------+---------+---------+-------------+----------+
| emp_id | empname | dept_id | designation | salary   |
+--------+---------+---------+-------------+----------+
|      1 | Pradip  |       1 | Manager     | 80000.00 |
|      2 | Ajinkya |       2 | Clerk       | 66000.00 |
|      3 | Nirav   |       3 | Staff       | 18000.00 |
|      4 | Milind  |       2 | Manager     | 60000.00 |
|      5 | Lakshya |       3 | Staff       | 11700.00 |
+--------+---------+---------+-------------+----------+
```

5 rows in set (0.000 sec)

==========

Question 7 : WAF which accepts the empno and returns the experience in years. Handle the error if empno does not exist.

EMP(Empno, Empname, DOJ);

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE FUNCTION exp_in_year(emp_no INT)
    -> RETURNS VARCHAR(30)
    -> BEGIN
    ->     DECLARE row INT DEFAULT 0;
    ->     DECLARE experience INT DEFAULT 0;
    ->     SELECT COUNT(*) INTO row FROM employee WHERE emp_id = emp_no;
    ->     IF ( row > 0 ) THEN
    ->         SELECT YEAR(CURDATE())-YEAR(date_of_join) INTO experience FROM employee WHERE
emp_id = emp_no;
    ->         RETURN CONCAT('Experience : ',experience,' years');
    ->     ELSE
    ->         RETURN CONCAT('No Such Employee Id Exists.');
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.024 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> SELECT * FROM employee;
+--------+---------+---------+-------------+----------+--------------+
| emp_id | empname | dept_id | designation | salary   | date_of_join |
+--------+---------+---------+-------------+----------+--------------+
|      1 | Pradip  |       1 | Manager     | 80000.00 | 2013-02-03   |
|      2 | Ajinkya |       2 | Clerk       | 66000.00 | 2015-08-13   |
|      3 | Nirav   |       3 | Staff       | 18000.00 | 2003-11-09   |
|      4 | Milind  |       2 | Manager     | 60000.00 | 2019-02-22   |
|      5 | Lakshya |       3 | Staff       | 11700.00 | 2010-10-10   |
+--------+---------+---------+-------------+----------+--------------+
5 rows in set (0.000 sec)
```

```
MariaDB [test]> select exp_in_year(1);
+----------------------+
| exp_in_year(1)       |
+----------------------+
| Experience : 7 years |
+----------------------+
1 row in set (0.006 sec)

MariaDB [test]> select exp_in_year(4);
+----------------------+
| exp_in_year(4)       |
+----------------------+
| Experience : 1 years |
+----------------------+
1 row in set (0.000 sec)

MariaDB [test]> select exp_in_year(7);
+-----------------------------+
| exp_in_year(7)              |
+-----------------------------+
| No Such Employee Id Exists. |
+-----------------------------+
1 row in set (0.000 sec)
```

==========

CURSORS

==========

Question 1 : Create a cursor for the emp table. Produce the output in following format:

{empname} employee working in department {deptno} earns Rs. {salary}.

EMP(empno, empname, salary, deptno);

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> drop procedure cursor_get_detail //
Query OK, 0 rows affected (0.013 sec)

MariaDB [test]> CREATE PROCEDURE cursor_get_detail()
   -> BEGIN
   ->    DECLARE name VARCHAR(20);
   ->    DECLARE deptid INT;
   ->    DECLARE emp_salary DECIMAL(8,2);
   ->    DECLARE stats VARCHAR(100);
   ->    DECLARE FINISHED INT DEFAULT 0;
   ->    DECLARE C1 CURSOR FOR SELECT empname,dept_id,salary FROM employee;
   ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
   ->    OPEN C1;
   ->       data :LOOP
   ->          IF (FINISHED = 1) THEN
   ->             LEAVE data;
   ->          END IF;
   ->          FETCH C1 INTO name,deptid,emp_salary;
```

```
   ->          SET stats = '';
   ->          SET stats = CONCAT(stats,name,' EMPLOYEE WORKING IN DEPARTMENT ',deptid,' EARNS
RS. ',emp_salary);
   ->          SELECT stats as EMP_DETAIL;
   ->       END LOOP;
   ->    CLOSE C1;
   -> END //
Query OK, 0 rows affected (0.014 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call cursor_get_detail;
+-----------------------------------------------------------+
| EMP_DETAIL                                                |
+-----------------------------------------------------------+
| Pradip EMPLOYEE WORKING IN DEPARTMENT 1 EARNS RS. 80000.00 |
+-----------------------------------------------------------+
1 row in set (0.000 sec)

+-----------------------------------------------------------+
| EMP_DETAIL                                                |
+-----------------------------------------------------------+
| Ajinkya EMPLOYEE WORKING IN DEPARTMENT 2 EARNS RS. 66000.00 |
+-----------------------------------------------------------+
1 row in set (0.004 sec)

+-----------------------------------------------------------+
| EMP_DETAIL                                                |
+-----------------------------------------------------------+
| Nirav EMPLOYEE WORKING IN DEPARTMENT 3 EARNS RS. 18000.00 |
+-----------------------------------------------------------+
1 row in set (0.009 sec)
```

```
+-----------------------------------------------------------+
| EMP_DETAIL                                                |
+-----------------------------------------------------------+
| Milind EMPLOYEE WORKING IN DEPARTMENT 2 EARNS RS. 60000.00 |
+-----------------------------------------------------------+
1 row in set (0.012 sec)

+-----------------------------------------------------------+
| EMP_DETAIL                                                |
+-----------------------------------------------------------+
| Lakshya EMPLOYEE WORKING IN DEPARTMENT 3 EARNS RS. 11700.00 |
+-----------------------------------------------------------+
1 row in set (0.015 sec)

Query OK, 0 rows affected (0.023 sec)
```

==========

Question 2 : Create a cursor for updating the salary of emp working in deptno 10 by 20%.

If any rows are affected than display the no of rows affected.

Use implicit cursor.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE cursor_upate_implicit()
   -> BEGIN
   ->    DECLARE row INT DEFAULT -1;
   ->    DECLARE empid INT;
   ->    DECLARE FINISHED INT DEFAULT 0;
```

```
    ->    DECLARE C1 CURSOR FOR SELECT emp_id FROM employee WHERE dept_id = 10;
    ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
    ->    OPEN C1;
    ->       data : LOOP
    ->       IF FINISHED = 1 THEN
    ->          LEAVE data;
    ->       END IF;
    ->       FETCH C1 INTO empid;
    ->          UPDATE employee SET salary = salary + (salary * 20)/100 WHERE emp_id = empid;
    ->          SET row = row+1;
    ->       END LOOP;
    ->    CLOSE C1;
    ->    IF row > 0 THEN
    ->       SELECT CONCAT('Row Affected : ', row) as Message;
    ->    ELSE
    ->       SELECT 'No Row Effected' as Message;
    ->    END IF;
    -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+--------------+
| emp_id | empname | dept_id | designation | salary   | date_of_join |
+--------+---------+---------+-------------+----------+--------------+
|      1 | Pradip  |       1 | Manager     | 80000.00 | 2013-02-03   |
|      2 | Ajinkya |       2 | Clerk       | 66000.00 | 2015-08-13   |
|      3 | Nirav   |      10 | Staff       | 18000.00 | 2003-11-09   |
|      4 | Milind  |       2 | Manager     | 60000.00 | 2019-02-22   |
|      5 | Lakshya |      10 | Staff       | 11700.00 | 2010-10-10   |
```

```
+--------+---------+---------+-------------+----------+--------------+
```

5 rows in set (0.000 sec)

```
MariaDB [test]> CALL cursor_upate_implicit;
+------------------+
| Message          |
+------------------+
| Row Affected : 2 |
+------------------+
```

1 row in set (0.023 sec)

```
MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+--------------+
| emp_id | empname | dept_id | designation | salary   | date_of_join |
+--------+---------+---------+-------------+----------+--------------+
|      1 | Pradip  |       1 | Manager     | 80000.00 | 2013-02-03   |
|      2 | Ajinkya |       2 | Clerk       | 66000.00 | 2015-08-13   |
|      3 | Nirav   |      10 | Staff       | 21600.00 | 2003-11-09   |
|      4 | Milind  |       2 | Manager     | 60000.00 | 2019-02-22   |
|      5 | Lakshya |      10 | Staff       | 14040.00 | 2010-10-10   |
+--------+---------+---------+-------------+----------+--------------+
```

5 rows in set (0.000 sec)

Query OK, 2 rows affected (0.026 sec)

==========

Question 3 : Create a cursor for updating the salary of emp working in deptno 10 by 20%.

If any rows are affected than display the no of rows affected.

Use EXPLICIT cursor.

```
================================================================================
===============
```

```
MariaDB [test]> CREATE PROCEDURE cursor_upate_explicit()
    -> BEGIN
    ->     DECLARE empid INT;
    ->     DECLARE i INT DEFAULT 0;
    ->     DECLARE FINISHED INT DEFAULT 0;
    ->     DECLARE C1 CURSOR FOR SELECT emp_id FROM employee WHERE dept_id = 10;
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
    ->     OPEN C1;
    ->         data : LOOP
    ->         IF FINISHED = 1 THEN
    ->             LEAVE data;
    ->             set i = i + 1;
    ->             SELECT i as LEAVING;
    ->         ELSE
    ->         FETCH C1 INTO empid;
    ->             UPDATE employee SET salary = salary + (salary * 20)/100 WHERE emp_id = empid;
    ->             set i = i + 1;
    ->             SELECT i as FETCHING;
    ->         END IF;
    ->         END LOOP;
    ->     CLOSE C1;
    -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> call cursor_upate_explicit;
```

```
Query OK, 2 rows affected, 1 warning (0.025 sec)

MariaDB [test]> select * from employee;
+--------+---------+---------+-------------+----------+--------------+
| emp_id | empname | dept_id | designation | salary   | date_of_join |
+--------+---------+---------+-------------+----------+--------------+
|      1 | Pradip  |       1 | Manager     | 80000.00 | 2013-02-03   |
|      2 | Ajinkya |       2 | Clerk       | 66000.00 | 2015-08-13   |
|      3 | Nirav   |      10 | Staff       | 25920.00 | 2003-11-09   |
|      4 | Milind  |       2 | Manager     | 60000.00 | 2019-02-22   |
|      5 | Lakshya |      10 | Staff       | 20217.60 | 2010-10-10   |
+--------+---------+---------+-------------+----------+--------------+
5 rows in set (0.000 sec)
```

==========

Question 4 : WAP that will display the name, department and salary of the first 10 employees getting the highest salary.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> drop procedure top_10_salary //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> CREATE PROCEDURE top_10_salary()
    -> BEGIN
    ->     DECLARE name VARCHAR(20);
    ->     DECLARE deptid INT;
    ->     DECLARE emp_salary FLOAT;
```

```
   ->    DECLARE FINISHED INTEGER DEFAULT 0;
   ->    DECLARE C1 CURSOR FOR SELECT empname,dept_id,salary FROM employee ORDER BY salary
DESC LIMIT 10;
   ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
   ->    OPEN C1;
   ->       data :LOOP
   ->          IF FINISHED = 1 THEN
   ->             LEAVE data;
   ->          END IF;
   ->          FETCH C1 INTO name,deptid,emp_salary;
   ->             select CONCAT( name,' | ',deptid,' | ',emp_salary) as Employee_Data;
   ->       END LOOP;
   ->    CLOSE C1;
   -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> CALL top_10_salary;
+--------------------+
| Employee_Data      |
+--------------------+
| Pradip | 1 | 80000 |
+--------------------+
1 row in set (0.000 sec)

+---------------------+
| Employee_Data       |
+---------------------+
| Ajinkya | 2 | 66000 |
+---------------------+
1 row in set (0.006 sec)
```

```
+------------------+
| Employee_Data    |
+------------------+
| Neel | 4 | 62000 |
+------------------+
1 row in set (0.008 sec)

+------------------------+
| Employee_Data          |
+------------------------+
| Lakshya | 10 | 60369.4 |
+------------------------+
1 row in set (0.014 sec)

+--------------------+
| Employee_Data      |
+--------------------+
| Milind | 2 | 60000 |
+--------------------+
1 row in set (0.017 sec)

+--------------------+
| Employee_Data      |
+--------------------+
| Pratik | 8 | 56000 |
+--------------------+
1 row in set (0.021 sec)

+---------------------+
| Employee_Data       |
```

+--------------------+

| Shubham | 3 | 49000 |

+--------------------+

1 row in set (0.024 sec)

+----------------------+

| Employee_Data |

+----------------------+

| Nirav | 10 | 44789.8 |

+----------------------+

1 row in set (0.026 sec)

+--------------------+

| Employee_Data |

+--------------------+

| Dhaval | 5 | 35000 |

+--------------------+

1 row in set (0.030 sec)

+--------------------+

| Employee_Data |

+--------------------+

| Hemang | 6 | 32000 |

+--------------------+

1 row in set (0.035 sec)

Query OK, 0 rows affected (0.042 sec)

==========

Question 5 : WAP using parameterized cursor to display all the information of employee living in specified city. Ask the city from user.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE search_city(user_city VARCHAR(20))
   -> BEGIN
   ->    DECLARE custid INT;
   ->    DECLARE custname VARCHAR(15);
   ->    DECLARE addr VARCHAR(30);
   ->    DECLARE FINISHED INTEGER DEFAULT 0;
   ->    DECLARE C1 CURSOR FOR SELECT cust_id,cust_name,address FROM customer WHERE city =
user_city;
   ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
   ->    OPEN C1;
   ->       data :LOOP
   ->       IF (FINISHED =1) THEN
   ->          LEAVE data;
   ->       END IF;
   ->       FETCH C1 INTO custid,custname,addr;
   ->          SELECT CONCAT(custid,' | ',custname,' | ',addr,' | ',user_city) as User_Detail;
   ->       END LOOP;
   ->    CLOSE C1;
   -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> CALL search_city('Gandhidham');
+-----------------------------------+
```

| User_Detail |

+----------+

| 2 | Ajinkya | E-block | Gandhidham |

+----------+

1 row in set (0.001 sec)

+----------+

| User_Detail |

+----------+

| 5 | Lakshya | G-block | Gandhidham |

+----------+

1 row in set (0.005 sec)

Query OK, 0 rows affected (0.014 sec)

==========

Question 6 : WAP which display the sum of salary department wise.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE salary_dept()
    -> BEGIN
    ->     DECLARE emp_salary DECIMAL(8,2);
    ->     DECLARE deptid INT;
    ->     DECLARE stats VARCHAR(100) DEFAULT ' ';
    ->     DECLARE FINISHED INTEGER DEFAULT 0;
    ->     DECLARE C1 CURSOR FOR SELECT dept_id FROM department;
```

```
    ->     DECLARE C2 CURSOR FOR SELECT SUM(salary) FROM employee WHERE dept_id = deptid
GROUP BY dept_id;
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
    ->     OPEN C1;
    ->       data :LOOP
    ->         FETCH C1 INTO deptid;
    ->         IF FINISHED = 1 THEN
    ->           LEAVE data;
    ->         END IF;
    ->         OPEN C2;
    ->           data2 :LOOP
    ->             FETCH C2 INTO emp_salary;
    ->             IF FINISHED = 1 THEN
    ->               LEAVE data2;
    ->             END IF;
    ->             SET stats = '';
    ->             SET stats = CONCAT(stats,deptid,' | ',emp_salary);
    ->           END LOOP data2;
    ->         CLOSE C2;
    ->         SET FINISHED = 0;
    ->         SELECT stats;
    ->       END LOOP data;
    ->     CLOSE C1;
    -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]>
MariaDB [test]> DELIMITER ;
MariaDB [test]> CALL salary_dept;
+--------------+
| stats        |
```

```
+--------------+
| 1 | 80000.00 |
+--------------+
1 row in set (0.002 sec)

+---------------+
| stats         |
+---------------+
| 2 | 126000.00 |
+---------------+
1 row in set (0.008 sec)

+--------------+
| stats        |
+--------------+
| 3 | 49000.00 |
+--------------+
1 row in set (0.010 sec)

+--------------+
| stats        |
+--------------+
| 4 | 62000.00 |
+--------------+
1 row in set (0.015 sec)

+--------------+
| stats        |
+--------------+
| 5 | 35000.00 |
+--------------+
```

1 row in set (0.016 sec)

+--------------+

| stats |

+--------------+

| 6 | 32000.00 |

+--------------+

1 row in set (0.021 sec)

+--------------+

| stats |

+--------------+

| 8 | 56000.00 |

+--------------+

1 row in set (0.026 sec)

+----------------+

| stats |

+----------------+

| 10 | 105159.18 |

+----------------+

1 row in set (0.033 sec)

Query OK, 0 rows affected (0.036 sec)

==========

Question 7 : Create a cursor to generate defferent two tables from one master table.

Students(Rno, Name, Std, B_date, Sex);

Girl_Table(Rno, Name, Std, B_date);

Boy_Table(Rno, Name, Std, B_date);

First fetch the row from Student table. If sex is ‘M’ then insert that row in

Boy_Table and if ‘F’ then insert that row in Girl_Table.

In both table Rollno entry must be in Sequence(Using create sequence command).

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE stud_gender()
   -> BEGIN
   ->    DECLARE row INT;
   ->    DECLARE s_rno INT;
   ->    DECLARE s_name VARCHAR(20);
   ->    DECLARE s_std INT;
   ->    DECLARE s_bday DATE;
   ->    DECLARE s_sex VARCHAR(1);
   ->    DECLARE FINISHED INTEGER DEFAULT 0;
   ->    DECLARE C1 CURSOR FOR SELECT * FROM students;
   ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET FINISHED = 1;
   ->    OPEN C1;
   ->       data :LOOP
   ->          FETCH C1 INTO s_rno,s_name,s_std,s_bday,s_sex;
   ->          IF (FINISHED =1) THEN
   ->             LEAVE data;
   ->          END IF;
   ->          IF (s_sex = 'F') THEN
```

```
   ->             SELECT COUNT(*) INTO row FROM information_schema.tables WHERE table_schema =
'test' AND table_name = 'girl';
   ->             IF row = 0 THEN
   ->                CREATE TABLE girl(Rno INT AUTO_INCREMENT PRIMARY KEY, Name VARCHAR(20),
Std INT, B_date DATE);
   ->                INSERT INTO girl(Name,Std,B_date) VALUES(s_name,s_std,s_bday);
   ->             ELSE
   ->                INSERT INTO girl(Name,Std,B_date) VALUES(s_name,s_std,s_bday);
   ->             END IF;
   ->          END IF;
   ->          IF (s_sex = 'M') THEN
   ->             SELECT COUNT(*) INTO row FROM information_schema.tables WHERE table_schema =
'test'AND table_name = 'boy';
   ->             IF row = 0 THEN
   ->                CREATE TABLE boy(Rno INT AUTO_INCREMENT PRIMARY KEY, Name VARCHAR(20),
Std INT, B_date DATE);
   ->                INSERT INTO boy (Name,Std,B_date) VALUES(s_name,s_std,s_bday);
   ->             ELSE
   ->                INSERT INTO boy (Name,Std,B_date) VALUES(s_name,s_std,s_bday);
   ->             END IF;
   ->          END IF;
   ->       END LOOP data;
   ->    CLOSE C1;
   -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> CALL stud_gender;
Query OK, 14 rows affected (1.786 sec)
```

MariaDB [test]> SELECT * FROM GIRL;

+-----+---------+------+------------+

| Rno | Name    | Std  | B_date     |

+-----+---------+------+------------+

|   1 | Kanchan |    8 | 2000-06-01 |

|   2 | Shivani |    8 | 1999-03-26 |

|   3 | Riddhi  |    8 | 1998-08-17 |

+-----+---------+------+------------+

3 rows in set (0.000 sec)

MariaDB [test]> SELECT * FROM BOY;

+-----+---------+------+------------+

| Rno | Name    | Std  | B_date     |

+-----+---------+------+------------+

|   1 | Pradip  |    8 | 1998-04-25 |

|   2 | Monil   |    8 | 1999-12-19 |

|   3 | Piyush  |    8 | 1998-02-21 |

|   4 | Anubhav |    8 | 1997-07-22 |

+-----+---------+------+------------+

4 rows in set (0.000 sec)

MariaDB [test]> SELECT * FROM STUDENTS;

+-----+---------+-----+------------+-----+

| Rno | Name    | Std | B_date     | Sex |

+-----+---------+-----+------------+-----+

|   1 | Pradip  |   8 | 1998-04-25 | M   |

|   2 | Monil   |   8 | 1999-12-19 | M   |

|   3 | Kanchan |   8 | 2000-06-01 | F   |

|   4 | Piyush  |   8 | 1998-02-21 | M   |

| 5 | Shivani | 8 | 1999-03-26 | F |

| 6 | Riddhi | 8 | 1998-08-17 | F |

| 7 | Anubhav | 8 | 1997-07-22 | M |

+-----+---------+-----+------------+-----+

7 rows in set (0.000 sec)

==========

Procedure

==========

Question 1 : Write a procedure which accepts the empno and returns the associated empname.

If empno does not exist than give proper error message.

EMP(Empno, Empname).

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE emp_call(IN EMP_NO VARCHAR(20))
   -> BEGIN
   ->    DECLARE row INT;
   ->    SELECT COUNT(*) INTO row FROM emp1 WHERE Empno = EMP_NO;
   ->    IF row > 0 THEN
   ->       SELECT Empname FROM emp1 as Name WHERE Empno = EMP_NO;
   ->    ELSE
   ->       SELECT "EMPLOYEE DOSE NOT EXIST." as MESSAGE;
   ->    END IF;
   -> END //
```

```
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> CALL emp_call(1);
+---------+
| Empname |
+---------+
| Pradip  |
+---------+
1 row in set (0.002 sec)

Query OK, 1 row affected (0.005 sec)

MariaDB [test]> CALL emp_call(5);
+---------+
| Empname |
+---------+
| Lakshya |
+---------+
1 row in set (0.000 sec)

Query OK, 1 row affected (0.005 sec)

MariaDB [test]> CALL emp_call(8);
+--------------------------+
| MESSAGE                  |
+--------------------------+
| EMPLOYEE DOSE NOT EXIST. |
+--------------------------+
1 row in set (0.000 sec)
```

Query OK, 1 row affected (0.006 sec)

==========

Question 2 : WAP which accepts the student rollno and returns the name,city and marks of all the subjects of that student.

STUDENT (Stud_ID, Stud_name, m1, m2, m3).

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE std_data(IN R_NO INT)
   -> BEGIN
   -> DECLARE row INT DEFAULT 0;
   ->    SELECT COUNT(*) INTO row FROM student1 WHERE Stud_ID = R_NO;
   ->    IF row > 0 THEN
   ->       SELECT Stud_name,m1,m2,m3 from student1 where Stud_ID = R_NO;
   ->    ELSE
   ->       SELECT "NO DETAIL FOUND" as MESSAGE;
   ->    END IF;
   -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call std_data(1);
+-----------+------+------+------+
| Stud_name | m1   | m2   | m3   |
+-----------+------+------+------+
| Pradip    |   45 |   76 |   66 |
```

```
+-----------+------+------+------+
```

1 row in set (0.000 sec)

Query OK, 1 row affected (0.002 sec)

```
MariaDB [test]> call std_data(2);
+-----------+------+------+------+
| Stud_name | m1   | m2   | m3   |
+-----------+------+------+------+
| Nirav     |   96 |   97 |   99 |
+-----------+------+------+------+
```

1 row in set (0.000 sec)

Query OK, 1 row affected (0.002 sec)

```
MariaDB [test]> call std_data(3);
+-----------------+
| MESSAGE         |
+-----------------+
| NO DETAIL FOUND |
+-----------------+
```

1 row in set (0.000 sec)

Query OK, 1 row affected (0.006 sec)

==========

Question 3 : WAP which accepts the name from the user. Return UPPER if name is in uppercase,

LOWER if name is in lowercase, MIXCASE if name is entered using both the case.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE case_check(IN user_input VARCHAR(20))
   -> BEGIN
   ->    DECLARE i INT DEFAULT 1;
   ->    DECLARE up INT DEFAULT 0;
   ->    DECLARE low INT DEFAULT 0;
   ->    DECLARE len INT;
   ->    DECLARE ch int;
   ->    SET LEN = LENGTH(user_input);
   ->    WHILE i <= len DO
   ->       SET ch = ASCII(SUBSTR(user_input,i,1));
   ->       IF ch >= 65 AND ch <= 90 THEN
   ->          SET up = up + 1;
   ->       ELSE
   ->          SET low = low + 1;
   ->       END IF;
   ->       SET i = i + 1;
   ->    END WHILE;
   ->    IF ( len = up) THEN
   ->       SELECT "STRING IS IN UPPERCASE FORM." as MESSAGE;
   ->    ELSEIF(len = low) THEN
   ->       SELECT "STRING IS IN LOWERCASE FORM." as MESSAGE;
   ->    ELSE
   ->       SELECT "STRING IS IN MIXCASE FORM" as MESSAGE;
```

```
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call case_check('pradip');
+------------------------------+
| MESSAGE                      |
+------------------------------+
| STRING IS IN LOWERCASE FORM. |
+------------------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.003 sec)

MariaDB [test]> call case_check('PRADIP');
+------------------------------+
| MESSAGE                      |
+------------------------------+
| STRING IS IN UPPERCASE FORM. |
+------------------------------+
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.004 sec)

MariaDB [test]> call case_check('PrAdip');
+--------------------------+
| MESSAGE                  |
+--------------------------+
| STRING IS IN MIXCASE FORM |
+--------------------------+
```

1 row in set (0.000 sec)

Query OK, 0 rows affected (0.003 sec)

==========

Question 4 : WAP which accepts the student rollno and returns the highest percent and name of that student to the calling block.

STUDENT(Stud_ID,Stud_name,percent);

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE std_percent(IN R_NO INT)
   -> BEGIN
   -> DECLARE row INT DEFAULT 0;
   -> DECLARE name varchar(30);
   -> DECLARE total FLOAT DEFAULT 0;
   -> DECLARE percent FLOAT DEFAULT 0;
   ->    SELECT COUNT(*) INTO row FROM student1 WHERE Stud_ID = R_NO;
   ->    IF row > 0 THEN
   ->       Select Stud_name INTO name from student1 where Stud_ID = R_NO;
   ->       select m1+m2+m3 INTO total from student1 where Stud_ID = R_NO;
   ->       set percent = (total*100)/300;
   ->       SELECT CONCAT('Highest Percent of ', percent ,' Obtain By ',name) as STUDENT_DATA;
   ->    ELSE
   ->       SELECT "NO DETAIL FOUND" as MESSAGE;
   ->    END IF;
   -> END //
Query OK, 0 rows affected (0.021 sec)
```

```
MariaDB [test]> DELIMITER ;
MariaDB [test]> call std_percent(1);
+-----------------------------------------+
| STUDENT_DATA                            |
+-----------------------------------------+
| Highest Percent of 62.3333 Obtain By Pradip |
+-----------------------------------------+
1 row in set (0.001 sec)

Query OK, 3 rows affected (0.006 sec)

MariaDB [test]> call std_percent(2);
+-----------------------------------------+
| STUDENT_DATA                            |
+-----------------------------------------+
| Highest Percent of 97.3333 Obtain By Nirav |
+-----------------------------------------+
1 row in set (0.000 sec)

Query OK, 3 rows affected (0.006 sec)

MariaDB [test]> call std_percent(3);
+-----------------+
| MESSAGE         |
+-----------------+
| NO DETAIL FOUND |
+-----------------+
1 row in set (0.000 sec)

Query OK, 1 row affected (0.005 sec)
```

==========

Question 5 : WAP which accepts the date of joining for specific employee and returns the years of experience along with its name. Accept the Employee no from user.

EMP (empno, empname, DOJ);

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE exp_name(IN empno INT)
   -> BEGIN
   ->    DECLARE row INT;
   ->    DECLARE experience INT;
   ->    DECLARE name VARCHAR(20);
   ->    SELECT COUNT(*) INTO row FROM employee WHERE emp_id = empno;
   ->    IF row > 0 THEN
   ->       SELECT YEAR(CURDATE())-YEAR(date_of_join) INTO experience FROM employee WHERE emp_id = empno;
   ->       SELECT empname INTO name FROM employee WHERE emp_id = empno;
   ->       SELECT name as NAME,experience as Experience;
   ->    ELSE
   ->       SELECT 'NO such Employee ID Found' as MESSAGE;
   ->    END IF;
   -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> CALL exp_name(1);
+--------+------------+
```

```
| NAME   | Experience |
+--------+------------+
| Pradip |          7 |
+--------+------------+
1 row in set (0.001 sec)

Query OK, 3 rows affected (0.006 sec)

MariaDB [test]> CALL exp_name(3);
+-------+------------+
| NAME  | Experience |
+-------+------------+
| Nirav |         17 |
+-------+------------+
1 row in set (0.000 sec)

Query OK, 3 rows affected (0.005 sec)

MariaDB [test]> CALL exp_name(19);
+--------------------------+
| MESSAGE                  |
+--------------------------+
| NO such Employee ID Found |
+--------------------------+
1 row in set (0.000 sec)

Query OK, 1 row affected (0.006 sec)
```

==========

Question 6 : WAP which accepts the student roll no and returns the result (in the form of class: first class, second class, third class or fail).

STUDENT (Stud_ID, Stud_name,m1, m2, m3).

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> drop procedure std_result //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> CREATE PROCEDURE std_result(IN R_NO INT)
    -> BEGIN
    ->    DECLARE row INT DEFAULT 0;
    ->    DECLARE name varchar(30);
    ->    DECLARE total FLOAT DEFAULT 0;
    ->    DECLARE percent FLOAT DEFAULT 0;
    ->       SELECT COUNT(*) INTO row FROM student1 WHERE Stud_ID = R_NO;
    ->       IF row > 0 THEN
    ->          Select Stud_name INTO name from student1 where Stud_ID = R_NO;
    ->          select m1+m2+m3 INTO total from student1 where Stud_ID = R_NO;
    ->          set percent = (total*100)/300;
    ->          IF percent > 80 THEN
    ->             SELECT name as Name, "DISTINCTION" as RESULT;
    ->          ELSEIF percent > 70 THEN
    ->             SELECT name as Name, "FIRST CLASS" as RESULT;
    ->          ELSEIF percent > 60 THEN
    ->             SELECT name as Name, "SECOND CLASS" as RESULT;
    ->          ELSEIF percent > 50 THEN
    ->             SELECT name as Name, "THIRD CLASS" as RESULT;
```

```
    ->         ELSEIF percent > 35 THEN
    ->            SELECT name as Name, "PASS CLASS" as RESULT;
    ->         ELSE
    ->            SELECT name as Name, "FAIL" as RESULT;
    ->         END IF;
    ->      ELSE
    ->         SELECT "NO DETAIL FOUND" as MESSAGE;
    ->      END IF;
    -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call std_result(1);
+--------+--------------+
| Name   | RESULT       |
+--------+--------------+
| Pradip | SECOND CLASS |
+--------+--------------+
1 row in set (0.001 sec)

Query OK, 3 rows affected (0.006 sec)

MariaDB [test]> call std_result(2);
+-------+-------------+
| Name  | RESULT      |
+-------+-------------+
| Nirav | DISTINCTION |
+-------+-------------+
1 row in set (0.000 sec)

Query OK, 3 rows affected (0.006 sec)
```

```
MariaDB [test]> call std_result(3);
+-----------------+
| MESSAGE         |
+-----------------+
| NO DETAIL FOUND |
+-----------------+
1 row in set (0.000 sec)

Query OK, 1 row affected (0.005 sec)



E.O.F

********************************************************************************
**************/
```

**********
**********

Name : Pradip S Karmakar

Class : M.C.A 2

Roll_No : 10

Subject : RDBMS

**********
**********/

==========
==========

TOPIC : Triggers

==========
==========

1. Q(example 11.2) : This example is divided in three categories : Insert, Update and Delete

   a. Insert : Write a trigger which updates the sale value if customer already exists else create new entry of customer.
   b. Update : If the customer is updating , WAT to update the sales value by incrementing the Sale_vale field.
   c. Delete : If the customer is deleting , WAT to update the sales value by decrementing the Sale_vale field.

==========
==========

INSERT

==========
==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER sales_bi_trg BEFORE INSERT ON sales
   -> FOR EACH ROW
   -> BEGIN
   -> DECLARE row_count INTEGER;
   ->    SELECT COUNT(*)
   ->    INTO row_count
   ->    FROM customer_sales_total
   ->    WHERE cust_id=NEW.cust_id;
   ->
   ->    IF row_count > 0 THEN
   ->       UPDATE customer_sales_total
   ->       SET sale_value=sale_value+NEW.sale_value
   ->       WHERE cust_id=NEW.cust_id;
   ->    ELSE
   ->       INSERT INTO customer_sales_total
   ->       (cust_id,sale_value)
   ->       VALUES(NEW.cust_id,NEW.sale_value);
   ->    END IF;
   -> END//
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;


MariaDB [test]> insert into sales(cust_id,product_name,sale_value) values(1,'printer',3500);
Query OK, 1 row affected (0.016 sec)


MariaDB [test]> select * from customer_sales_total;
```

```
+---------+------------+
| cust_id | sale_value |
+---------+------------+
|       1 |    3500.00 |
+---------+------------+
1 row in set (0.000 sec)
```

```
MariaDB [test]> select * from sales;
+----------+---------+--------------+------------+
| sales_id | cust_id | product_name | sale_value |
+----------+---------+--------------+------------+
|        1 |       1 | printer      |    3500.00 |
+----------+---------+--------------+------------+
1 row in set (0.000 sec)
```

```
MariaDB [test]> insert into sales(cust_id,product_name,sale_value) values(1,'Page Bundle',400);
Query OK, 1 row affected (0.008 sec)
```

```
MariaDB [test]> select * from customer_sales_total;
+---------+------------+
| cust_id | sale_value |
+---------+------------+
|       1 |    3900.00 |
+---------+------------+
1 row in set (0.000 sec)
```

```
MariaDB [test]> select * from sales;
```

```
+----------+---------+--------------+------------+
| sales_id | cust_id | product_name | sale_value |
+----------+---------+--------------+------------+
|        1 |       1 | printer      |    3500.00 |
|        2 |       1 | Page Bundle  |     400.00 |
+----------+---------+--------------+------------+
2 rows in set (0.000 sec)

MariaDB [test]> insert into sales(cust_id,product_name,sale_value) values(2,'mouse',870);
Query OK, 1 row affected (0.007 sec)

MariaDB [test]> select * from customer_sales_total;
+---------+------------+
| cust_id | sale_value |
+---------+------------+
|       1 |    3900.00 |
|       2 |     870.00 |
+---------+------------+
2 rows in set (0.000 sec)

MariaDB [test]> select * from sales;
+----------+---------+--------------+------------+
| sales_id | cust_id | product_name | sale_value |
+----------+---------+--------------+------------+
|        1 |       1 | printer      |    3500.00 |
|        2 |       1 | Page Bundle  |     400.00 |
|        3 |       2 | mouse        |     870.00 |
+----------+---------+--------------+------------+
3 rows in set (0.000 sec)
```

```
================================================================================
=============

                              UPDATE

================================================================================
=============

MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER sales_bu_trg BEFORE UPDATE ON sales FOR EACH ROW
   -> BEGIN
   ->    UPDATE customer_sales_total
   ->    SET sale_value=sale_value+(NEW.sale_value-OLD.sale_value)
   ->    WHERE cust_id=NEW.cust_id;
   -> END //
Query OK, 0 rows affected (0.019 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> update sales set sale_value = 550 where sales_id = 2;
Query OK, 1 row affected (0.007 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [test]> select * from customer_sales_total;
+---------+------------+
| cust_id | sale_value |
+---------+------------+
|       1 |    4050.00 |
|       2 |     870.00 |
+---------+------------+
2 row in set (0.000 sec)

MariaDB [test]> select * from sales;
+----------+---------+--------------+------------+
```

| sales_id | cust_id | product_name | sale_value |

+----------+---------+--------------+------------+

|       1 |       1 | printer      |    3500.00 |

|       2 |       1 | Page Bundle  |     550.00 |

|       3 |       2 | mouse        |     870.00 |

+----------+---------+--------------+------------+

3 rows in set (0.000 sec)

==========

DELETE

==========

MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE  TRIGGER sales_bd_trg BEFORE DELETE ON sales FOR EACH ROW

   -> BEGIN

   ->    UPDATE customer_sales_total

   ->    SET sale_value=sale_value-OLD.sale_value

   ->    WHERE cust_id=OLD.cust_id;

   -> END //

Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> delete from sales where sales_id = 3;

Query OK, 1 row affected (0.008 sec)

MariaDB [test]> select * from sales;

+----------+---------+--------------+------------+

| sales_id | cust_id | product_name | sale_value |

```
+----------+---------+--------------+------------+
|       1 |       1 | printer      |    3500.00 |
|       2 |       1 | Page Bundle  |     550.00 |
+----------+---------+--------------+------------+
2 rows in set (0.000 sec)

MariaDB [test]> select * from customer_sales_total;
+---------+------------+
| cust_id | sale_value |
+---------+------------+
|       1 |    4050.00 |
|       2 |       0.00 |
+---------+------------+
```

**********

==========

2. Q(example 11.4) Wirte a program to create trigger signal to restrict entering negative value in balance.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER account_balance_bu BEFORE UPDATE ON account_balance
    -> FOR EACH ROW
    -> BEGIN
```

```
    ->     IF (NEW.balance < 0) THEN
    ->         SIGNAL SQLSTATE '80000'
    ->         SET MESSAGE_TEXT='Account balance cannot be less than 0';
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.028 sec)

MariaDB [test]> DELIMITER ;



MariaDB [test]> insert into account_balance(balance) values(10000)
    -> ,(23000),
    -> (45000);
Query OK, 3 rows affected (0.005 sec)
Records: 3  Duplicates: 0  Warnings: 0



MariaDB [test]> select * from account_balance;
+--------+----------+
| acc_id | balance  |
+--------+----------+
|      1 | 10000.00 |
|      2 | 23000.00 |
|      3 | 45000.00 |
+--------+----------+
3 rows in set (0.000 sec)



MariaDB [test]> update account_balance set balance = -2000 where acc_id = 2;
ERROR 1644 (80000): Account balance cannot be less than 0
```

**********

==========

3. Q(example 11.5) Write a trigger to perform data validation using select statement.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER account_balance_bu  BEFORE UPDATE ON account_balance FOR EACH ROW
   -> BEGIN
   ->    DECLARE dummy INT;
   ->    IF NEW.balance<0 THEN
   ->       SELECT `Account balance cannot be less than 0` INTO dummy
   ->       FROM account_balance WHERE acc_id=NEW.acc_id;
   ->    END IF;
   -> END //
Query OK, 0 rows affected (0.024 sec)

MariaDB [test]>
MariaDB [test]> DELIMITER ;
MariaDB [test]> update account_balance set balance = -6000 where acc_id = 3;
ERROR 1054 (42S22): Unknown column 'Account balance cannot be less than 0' in 'field list'
```

**********

==========

4. Q(figure 2.17) :write a example to create a sales table whichprovides free shipping on orders above 500

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER sales_bi_trg1 BEFORE INSERT ON sales1
   -> FOR EACH ROW
   -> BEGIN
   ->    IF NEW.sale_value>500 THEN
   ->       SET NEW.free_shipping='Y';
   ->    ELSE
   ->       SET NEW.free_shipping='N';
   ->    END IF;
   ->       IF NEW.sale_value>1000 THEN
   ->          SET NEW.discount=NEW.sale_value*0.5;
   ->       ELSE
   ->          SET NEW.discount=0;
   ->    END IF;
   -> END //
Query OK, 0 rows affected (0.025 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> INSERT INTO sales1(customer_id,sale_value,free_shipping,discount)
VALUES(201,20000,'N',0);
Query OK, 1 row affected (0.008 sec)
```

MariaDB [test]> select * from sales1;

+----------+-------------+------------+---------------+----------+

| sales_id | customer_id | sale_value | free_shipping | discount |

+----------+-------------+------------+---------------+----------+

| 1 | 201 | 20000 | Y | 10000 |

+----------+-------------+------------+---------------+----------+

1 row in set (0.000 sec)

==========

TOPIC : Transaction

==========

5. Q(example 8.1) : Create a procedure to commence a transaction using auto commit.

==========

MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE PROCEDURE transfer_funds (from_account int, to_account int,transfer_amount decimal(10,2))

-> BEGIN

-> SET autocommit=0;

-> UPDATE ACCOUNTS SET amount_balance = amount_balance - transfer_amount WHERE acc_id=from_account;

-> UPDATE ACCOUNTS SET amount_balance = amount_balance + transfer_amount WHERE acc_id=to_account;

-> COMMIT;

-> END //

Query OK, 0 rows affected (1.759 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> insert into accounts(branch_name,amount_balance)
values('Navsari',43900),('Surat',23090),('Ahmedabad',60897);

Query OK, 3 rows affected (0.008 sec)

Records: 3  Duplicates: 0  Warnings: 0

MariaDB [test]> select * from accounts;

+--------+-------------+----------------+

| acc_id | branch_name | amount_balance |

+--------+-------------+----------------+

|  1 | Navsari  |  43900.00 |

|  2 | Surat  |  23090.00 |

|  3 | Ahmedabad  |  60897.00 |

+--------+-------------+----------------+

3 rows in set (0.000 sec)

MariaDB [test]> call transfer_funds(3,1,4500);

Query OK, 2 rows affected (0.007 sec)

MariaDB [test]> select * from accounts;

+--------+-------------+----------------+

| acc_id | branch_name | amount_balance |

+--------+-------------+----------------+

|  1 | Navsari  |  48400.00 |

|  2 | Surat  |  23090.00 |

|  3 | Ahmedabad  |  56397.00 |

+--------+-------------+----------------+

3 rows in set (0.000 sec)

==========

6. Q(example 8.2) : Create a procedure to commence a transaction using start transaction.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE trans_tfer_funds(from_account int, to_account
int,tfer_amount decimal(10,2))
   -> BEGIN
   ->    START TRANSACTION;
   ->       UPDATE ACCOUNTS SET amount_balance =amount_balance - tfer_amount WHERE
acc_id=from_account;
   ->       UPDATE ACCOUNTS SET amount_balance =amount_balance + tfer_amount WHERE
acc_id=to_account;
   ->    COMMIT;
   -> END //
Query OK, 0 rows affected (0.021 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> select * from accounts;
+--------+-------------+----------------+
| acc_id | branch_name | amount_balance |
+--------+-------------+----------------+
|      1 | Navsari     |       48400.00 |
|      2 | Surat       |       23090.00 |
|      3 | Ahmedabad   |       56397.00 |
```

```
+--------+-------------+----------------+
3 rows in set (0.000 sec)


MariaDB [test]> call transfer_funds(2,3,3000);
Query OK, 2 rows affected (0.007 sec)


MariaDB [test]> select * from accounts;
+--------+-------------+----------------+
| acc_id | branch_name | amount_balance |
+--------+-------------+----------------+
|      1 | Navsari     |       48400.00 |
|      2 | Surat       |       20090.00 |
|      3 | Ahmedabad   |       59397.00 |
+--------+-------------+----------------+
3 rows in set (0.000 sec)
```

==========

7. Q(example 8.3) : create a procedure which displays use of Savepoint with a transaction

==========

```
DELIMITER //
create procedure creating_table()
BEGIN
   create table location(location varchar(20),address1 varchar(20),address2 varchar(20),zipcode int);
   create table AUDIT_LOG (audit_message varchar(20));
   create table departments(department_name varchar(20),location varchar(20),manager_id int);
```

```
END //


MariaDB [test]> CREATE PROCEDURE savepoint_example(in_department_name
VARCHAR(30),in_location VARCHAR(30),in_address1 VARCHAR(30),in_address2
VARCHAR(30),in_zipcode VARCHAR(10), in_manager_id INT)
    ->    BEGIN
    ->    DECLARE location_exists INT DEFAULT 0;
    ->    DECLARE duplicate_dept INT DEFAULT 0;
    ->    START TRANSACTION;
    ->       SELECT COUNT(*) INTO location_exists FROM location WHERE location=in_location;
    ->       IF location_exists=0 THEN
    ->          INSERT INTO AUDIT_LOG (audit_message) VALUES (CONCAT('Creating new location
',in_location));
    ->          INSERT INTO location (location,address1,address2,zipcode) VALUES
(in_location,in_address1,in_address2,in_zipcode);
    ->       ELSE
    ->          UPDATE location set address1=in_address1, address2=in_address2, zipcode=in_zipcode
WHERE location=in_location;
    ->       END IF;
    ->    SAVEPOINT savepoint_location_exists;
    ->    BEGIN
    ->    DECLARE DUPLICATE_KEY CONDITION FOR 1062;
    ->    DECLARE CONTINUE HANDLER FOR DUPLICATE_KEY /*Duplicate key value*/
    ->    BEGIN
    ->    SET duplicate_dept=1;
    ->    ROLLBACK TO SAVEPOINT savepoint_location_exists;
    ->    END;
    ->    INSERT INTO AUDIT_LOG (audit_message) VALUES (CONCAT('Creating new
department',in_department_name));
    ->    INSERT INTO DEPARTMENTS (department_name,location,manager_id) VALUES
(in_department_name,in_location, in_manager_id);
    ->       IF duplicate_dept=1 THEN
    ->          UPDATE departments SET location=in_location,manager_id=in_manager_id WHERE
department_name=in_department_name;
```

```
    ->         END IF;
    ->         END;
    ->     COMMIT;
    -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]>  CALL savepoint_example('Designing','Navsari','Grid road','Bhagvati Sankul
Society',396445,1);
Query OK, 5 rows affected (0.007 sec)

MariaDB [test]> select * from location;
+----------+-----------+----------------------+---------+
| location | address1  | address2             | zipcode |
+----------+-----------+----------------------+---------+
| Navsari  | Grid road | Bhagvati Sankul Soci |  396445 |
+----------+-----------+----------------------+---------+
1 row in set (0.000 sec)

MariaDB [test]> select * from AUDIT_LOG;
+----------------------------------+
| audit_message                    |
+----------------------------------+
| Creating new location Navsari    |
| Creating new department Designing |
+----------------------------------+
2 rows in set (0.002 sec)

MariaDB [test]> select * from departments;
+-----------------+----------+------------+
```

| department_name | location | manager_id |

+-----------------+----------+------------+

| Designing | Navsari | 1 |

+-----------------+----------+------------+

1 row in set (0.000 sec)

==========

TOPIC : Triggers (DO IT YOURSELF)

==========

1. Write a Trigger that stores the old data table of student table in student_backup while updating the student table.

Student_backup (Stud_ID, Stud_name, Address, Contact_no, Branch, Operation_date)

Student (Stud_ID, Stud_name, Address, Contact_no, Branch)

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE PROCEDURE creating_table1()
   -> BEGIN
   ->     CREATE TABLE Student(Stud_ID INT PRIMARY KEY, Stud_name VARCHAR(20), Address VARCHAR(30), Contact_no INT(11), Branch VARCHAR(60));
   ->     CREATE TABLE Student_backup(Stud_ID INT PRIMARY KEY, Stud_name VARCHAR(20), Address VARCHAR(30), Contact_no INT(11), Branch VARCHAR(60),Operation_date date);
   -> END //
Query OK, 0 rows affected (0.009 sec)
```

```
MariaDB [test]> call creating_table1() //
Query OK, 0 rows affected (0.291 sec)

MariaDB [test]> CREATE TRIGGER stud_backup BEFORE UPDATE ON student FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO Student_backup
values(OLD.Stud_ID,OLD.Stud_name,OLD.Address,OLD.Contact_no,OLD.Branch,curdate());
    -> END //
Query OK, 0 rows affected (0.028 sec)

MariaDB [test]> CREATE PROCEDURE insert_table1()
    -> BEGIN
    ->     insert into Student(Stud_name,Address,Contact_no,Branch)
values('Pradip','Navsari',8882228888,'Kabilpore'),
    ->     ('Ajinkya','Kutch',8881118888,'Gandhidham'),
    ->     ('Milind','Ahmedabad',8268228888,'Kalupur'),
    ->     ('Lakshya','Kutch',9888221358,'Gandhidham'),
    ->     ('Nirav','Kutch',8892220088,'Gandhidham');
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> call insert_table1();
Query OK, 5 rows affected, 5 warnings (0.007 sec)

MariaDB [test]> UPDATE Student SET Contact_no = 8238118848 WHERE Stud_ID=1;
Query OK, 0 rows affected, 1 warning (0.010 sec)
Rows matched: 1  Changed: 0  Warnings: 1
```

```
MariaDB [test]> select * from students;
ERROR 1146 (42S02): Table 'test.students' doesn't exist
MariaDB [test]> select * from student;
+---------+-----------+-----------+------------+------------+
| Stud_ID | Stud_name | Address   | Contact_no | Branch     |
+---------+-----------+-----------+------------+------------+
|       1 | Pradip    | Navsari   | 2147483647 | Kabilpore  |
|       2 | Ajinkya   | Kutch     | 2147483647 | Gandhidham |
|       3 | Milind    | Ahmedabad | 2147483647 | Kalupur    |
|       4 | Lakshya   | Kutch     | 2147483647 | Gandhidham |
|       5 | Nirav     | Kutch     | 2147483647 | Gandhidham |
+---------+-----------+-----------+------------+------------+
5 rows in set (0.000 sec)


MariaDB [test]> select * from student_backup;
+---------+-----------+---------+------------+-----------+----------------+
| Stud_ID | Stud_name | Address | Contact_no | Branch    | Operation_date |
+---------+-----------+---------+------------+-----------+----------------+
|       1 | Pradip    | Navsari | 2147483647 | Kabilpore | 2020-05-06     |
+---------+-----------+---------+------------+-----------+----------------+
1 row in set (0.000 sec)
```

==========

2. Write a trigger, that ensures the empno of emp table is in a format ‘E00001’ (empno must start with ‘E’ and must be 6 characters long). If not, than complete empno with this format before inserting into the employee table.

```
================================================================================
=============

MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER emp_format BEFORE INSERT ON emp_tr1 FOR EACH ROW
    -> BEGIN
    ->     DECLARE I INT DEFAULT 1;
    ->     DECLARE CH INT;
    ->     DECLARE LEN INT;
    ->     DECLARE FLAG INT DEFAULT 0;
    ->     DECLARE EMP_ID VARCHAR(10);
    ->     SET EMP_ID=NEW.empid;
    ->     SET LEN=LENGTH(NEW.empid);
    ->     IF (LEN<6) THEN
    ->        SIGNAL SQLSTATE '80000'
    ->        SET MESSAGE_TEXT='EMPLOYEE ID MUST BE 6 CHARACTER LONG';
    ->     ELSEIF (LEN>6) THEN
    ->        SIGNAL SQLSTATE '80001'
    ->        SET MESSAGE_TEXT='EMPLOYEE ID MUST BE 6 CHARACTER LONG';
    ->     ELSE
    ->        SET CH=ASCII(SUBSTR(EMP_ID,I,1));
    ->        IF (CH=69) THEN
    ->           SET I=I+1;
    ->           MYLOOP : WHILE (I<LEN) DO
    ->              SET CH=ASCII(SUBSTR(EMP_ID,I,1));
    ->              IF (CH>=48 AND CH<=57) THEN
    ->                 SET I=I+1;
    ->              ELSE
    ->                 SET FLAG=1;
    ->                 LEAVE MYLOOP;
```

```
    ->             END IF;
    ->           END WHILE;
    ->       ELSE
    ->           SIGNAL SQLSTATE '80002'
    ->           SET MESSAGE_TEXT='EMPLOYEE ID MUST BE LIKE E00001';
    ->       END IF ;
    ->    END IF;
    ->    IF (FLAG=1) THEN
    ->       SIGNAL SQLSTATE '80003'
    ->       SET MESSAGE_TEXT='EMPLOYEE ID MUST BE LIKE E00001';
    ->    END IF;
    -> END //
Query OK, 0 rows affected (0.032 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> insert into emp_tr1 values(1,'pradip');
ERROR 1644 (80000): EMPLOYEE ID MUST BE 6 CHARACTER LONG

MariaDB [test]> insert into emp_tr1 values(123456,'pradip');
ERROR 1644 (80002): EMPLOYEE ID MUST BE LIKE E00001

MariaDB [test]> insert into emp_tr1 values('E00001','pradip');
Query OK, 1 row affected (0.007 sec)

MariaDB [test]> select * from emp_tr1;
+--------+--------+
| empid  | name   |
+--------+--------+
| E00001 | pradip |
+--------+--------+
```

1 row in set (0.000 sec)

==========

3. Write a trigger which checks the age of employee while inserting the record in emp table.

If it is negative than generate the error and display proper message.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]>
MariaDB [test]> CREATE TRIGGER check_age BEFORE INSERT ON emp_tr1
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE AGE INT;
    ->     SET AGE=YEAR(CURDATE())-YEAR(NEW.birth_day);
    ->     IF AGE<0 THEN
    ->         SIGNAL SQLSTATE '80005'
    ->         SET MESSAGE_TEXT='Please Enter Valid BirthDay';
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> DELIMITER ;

MariaDB [test]> insert into emp_tr1 values('E00002','Nirav','2021-06-23');
ERROR 1644 (80005): Please Enter Valid BirthDay

MariaDB [test]> insert into emp_tr1 values('E00002','Nirav','1999-06-23');
```

Query OK, 1 row affected (0.007 sec)

```
MariaDB [test]> select * from emp_tr1;
+--------+--------+------------+
| empid  | name   | birth_day  |
+--------+--------+------------+
| E00001 | pradip | 1998-04-25 |
| E00002 | Nirav  | 1999-06-23 |
+--------+--------+------------+
2 rows in set (0.000 sec)
```

==========

4. Write a trigger which converts the employee name in upper case if it is inserted in any other case. Change should be done before the insertion only.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER uppercase_name BEFORE INSERT ON emp_tr1 FOR EACH ROW
    ->    BEGIN
    ->    DECLARE I INT DEFAULT 1;
    ->    DECLARE NAME VARCHAR(20) default ' ';
    ->    DECLARE STRING VARCHAR(20) DEFAULT " ";
    ->    DECLARE RES VARCHAR(2) DEFAULT "";
    ->    DECLARE CH INT;
    ->    DECLARE LEN INT;
    ->    SET NAME=NEW.name;
    ->    SET LEN=LENGTH(NAME);
```

```
    ->     WHILE (I<=LEN) do
    ->       SET CH=ASCII(SUBSTR(NAME,I,1));
    ->       IF (CH>=97 AND CH<=122) THEN
    ->          SET CH=CH-32;
    ->       END IF;
    ->          SET RES=CHAR(CH);
    ->          SET STRING=CONCAT(STRING,RES);
    ->          SET I=I+1;
    ->     END WHILE;
    ->     set new.name=string;
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> select * from emp_tr1;
+--------+--------+------------+
| empid  | name   | birth_day  |
+--------+--------+------------+
| E00001 | pradip | 1998-04-25 |
| E00002 | Nirav  | 1999-06-23 |
+--------+--------+------------+
2 rows in set (0.001 sec)

MariaDB [test]> insert into emp_tr1 values('E00002','ajinkya','1999-01-26');
Query OK, 1 row affected (0.011 sec)

MariaDB [test]> select * from emp_tr1;
+--------+----------+------------+
| empid  | name     | birth_day  |
+--------+----------+------------+
| E00001 | pradip   | 1998-04-25 |
```

| E00002 | Nirav    | 1999-06-23 |

| E00002 |  AJINKYA | 1999-01-26 |

+--------+----------+------------+

3 rows in set (0.000 sec)

==========

5. WAT that stores the data of emp table in emp_backup table for every delete operation and store the old data for every update operation.

EMP(Empno, Empname, salary);

Emp_Backup(Empno,Empname,Date_of_operation,Type_of_operation (i.e.update or delete));

==========

MariaDB [test]> DELIMITER //

MariaDB [test]> CREATE TRIGGER emp_bu BEFORE UPDATE ON EMP1  FOR EACH ROW

   -> BEGIN

   ->     INSERT INTO Emp_Backup(Empno,Empname,Date_of_operation,Type_of_operation) values (NEW.Empno,NEW.Empname,CURDATE(),'Update');

   -> END //

Query OK, 0 rows affected (0.020 sec)

MariaDB [test]> CREATE TRIGGER emp_bd BEFORE DELETE ON EMP1  FOR EACH ROW

   -> BEGIN

   ->     INSERT INTO Emp_Backup(Empno,Empname,Date_of_operation,Type_of_operation) values (old.Empno,old.Empname,CURDATE(),'Delete');

   -> END //

```
Query OK, 0 rows affected (0.022 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]> insert into EMP1 values(1,'Pradip',50000);
Query OK, 1 row affected (0.008 sec)

MariaDB [test]> select * from EMP1;
+-------+---------+--------+
| Empno | Empname | salary |
+-------+---------+--------+
|     1 | Pradip  |  50000 |
+-------+---------+--------+
1 row in set (0.002 sec)

MariaDB [test]> update EMP1 set salary = 60000 where Empno = 1;
Query OK, 1 row affected (0.008 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [test]> select * from EMP1;
+-------+---------+--------+
| Empno | Empname | salary |
+-------+---------+--------+
|     1 | Pradip  |  60000 |
+-------+---------+--------+
1 row in set (0.000 sec)

MariaDB [test]> delete from EMP1 where Empno = 1;
Query OK, 1 row affected (0.007 sec)

MariaDB [test]> select * from emp_backup;
+-------+---------+-------------------+-------------------+
```

| Empno | Empname | Date_of_operation | Type_of_operation |

+-------+---------+-------------------+-------------------+

| 1 | Pradip | 2020-05-06 | Update |

| 1 | Pradip | 2020-05-06 | Delete |

+-------+---------+-------------------+-------------------+

2 rows in set (0.000 sec)

==========

6. WAT which display the message ‘Updating’,’Deleting’ or ’Inserting’ when Update, Delete or Insert operation is performed on the emp table respectively.

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER emp_insert BEFORE INSERT ON emp FOR EACH ROW
   -> BEGIN
   ->    SIGNAL SQLSTATE '80000'
   ->    SET MESSAGE_TEXT='Inserting An EMP.';
   -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [test]>
MariaDB [test]> INSERT INTO emp(empname,position,salary)
VALUES('Shubham','CEO_TENCENT',70000) //
ERROR 1644 (80000): Inserting An EMP.
MariaDB [test]>
MariaDB [test]> CREATE TRIGGER emp_update BEFORE UPDATE ON emp  FOR EACH ROW
   -> BEGIN
```

```
    ->    SIGNAL SQLSTATE '80001'
    ->    SET MESSAGE_TEXT='Updating An EMP.';
    ->
    -> END //
Query OK, 0 rows affected (0.045 sec)

MariaDB [test]>
MariaDB [test]> UPDATE emp SET salary = '75000' WHERE emp_id = 6 //
ERROR 1644 (80001): Updating An EMP.
MariaDB [test]>
MariaDB [test]> CREATE TRIGGER emp_delete BEFORE DELETE ON emp  FOR EACH ROW
    -> BEGIN
    ->    SIGNAL SQLSTATE '80002'
    ->    SET MESSAGE_TEXT='Deleting An EMP.';
    -> END //
Query OK, 0 rows affected (0.019 sec)

MariaDB [test]> DELETE from emp where empname = 'Lakshya' //
ERROR 1644 (80002): Deleting An EMP.
MariaDB [test]>
MariaDB [test]> DELIMITER ;
```

==========

7. WAT which generate an error if any user try to delete from product_master table on weekends

==========

```
MariaDB [test]> DELIMITER //
```

```
MariaDB [test]> CREATE TRIGGER emp_day_bd BEFORE DELETE ON product_master
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE DAY VARCHAR(20);
    ->    SET DAY=DAYNAME(curdate());
    ->    IF DAY='Thursday' THEN
    ->       SIGNAL SQLSTATE '80007'
    ->       SET MESSAGE_TEXT='Deletion is not possible on Thursday.';
    ->    END IF;
    -> END //
Query OK, 0 rows affected (0.291 sec)

MariaDB [test]> select * from product_master;
+------------+--------------+
| product_id | product_name |
+------------+--------------+
|          1 | TV           |
|          2 | LAPTOP       |
|          3 | FRIDGE       |
+------------+--------------+
3 rows in set (0.000 sec)

MariaDB [test]> delete from product_master where product_id='3';
ERROR 1644 (80007): Deletion is not possible on Thursday.
```

==========

8. We have two tables student_mast and stu_log. student_mast have three columns

STUDENT_ID, NAME, ST_CLASS. stu_log table has two columns user_id and description.

WAT which inserts the student details in stu_log table as soon as we promote the students in student master table( e.g. when a student is promoted from sem 2 to 3, auto entry in log table)

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> CREATE TRIGGER stu_log BEFORE UPDATE ON student_mast
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE DES VARCHAR(100) DEFAULT ' ';
    ->     DECLARE SID INT;
    ->     DECLARE SEM_NEW INT;
    ->     DECLARE SEM_OLD INT;
    ->     SET SEM_OLD=OLD.CLASS;
    ->     SET SEM_NEW =SEM_OLD +1;
    ->     SET DES= CONCAT('Student is promoted from semister ',SEM_OLD,' to ',SEM_NEW,DES);
    ->     SET SID=OLD.student_id;
    ->     INSERT INTO stu_log VALUES(SID,DES);
    -> END //
Query OK, 0 rows affected (0.026 sec)

MariaDB [test]> DELIMITER ;


MariaDB [test]> insert into student_mast(name,class) value('pradip',10),
    -> ('Ajinkya',9);
Query OK, 2 rows affected (0.004 sec)
```

```
Records: 2  Duplicates: 0  Warnings: 0


MariaDB [test]> SELECT * FROM student_mast;
+------------+---------+-------+
| student_id | name    | class |
+------------+---------+-------+
|          1 | pradip  |    10 |
|          2 | Ajinkya |     9 |
+------------+---------+-------+
2 rows in set (0.000 sec)


MariaDB [test]> UPDATE student_mast SET class=class + 1 WHERE student_id = 1;
Query OK, 1 row affected (0.007 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [test]> SELECT * FROM student_mast;
+------------+---------+-------+
| student_id | name    | class |
+------------+---------+-------+
|          1 | pradip  |    11 |
|          2 | Ajinkya |     9 |
+------------+---------+-------+
2 rows in set (0.000 sec)

MariaDB [test]> SELECT * FROM stu_log;
+---------+-----------------------------------------+
| user_id | description                             |
+---------+-----------------------------------------+
|       1 | Student is promoted from semister 10 to 11 |
```

```
+---------+-----------------------------------------+
1 row in set (0.000 sec)
```

==========

9. WAT to calculate the Income Tax amount and insert it in emp table. EMP(emp_no,emp_name, emp_income, income_tax);

If emp_income <100000 and >=50000 then incometax = 10%

If emp_income <200000 and >=100000 then incometax = 15%

If emp_income <300000 and >=200000 then incometax = 20%

==========

```
MariaDB [test]> DELIMITER //
MariaDB [test]> drop trigger income_tax_decide //
Query OK, 0 rows affected (0.000 sec)

MariaDB [test]> CREATE TRIGGER income_tax_decide BEFORE INSERT ON emp3
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE tax FLOAT;
    ->    IF (NEW.emp_income >= 50000 AND NEW.emp_income < 100000) THEN
    ->       set tax = (NEW.emp_income*10)/100;
    ->       set NEW.income_tax = tax;
    ->    ELSEIF (NEW.emp_income >= 100000 AND NEW.emp_income < 200000) THEN
    ->       set tax = (NEW.emp_income*15)/100;
    ->       set NEW.income_tax = tax;
    ->    ELSEIF (NEW.emp_income >= 200000 AND NEW.emp_income < 300000) THEN
```

```
    ->       set tax = (NEW.emp_income*20)/100;
    ->       set NEW.income_tax = tax;
    ->    END IF;
    -> END //
Query OK, 0 rows affected (0.019 sec)

MariaDB [test]> DELIMITER ;
MariaDB [test]>
MariaDB [test]> insert into emp3(emp_name,emp_income,income_tax) values('pradip',80000,0);
Query OK, 1 row affected (0.006 sec)

MariaDB [test]>
MariaDB [test]>
MariaDB [test]> SELECT * FROM EMP3;
+--------+----------+------------+------------+
| emp_no | emp_name | emp_income | income_tax |
+--------+----------+------------+------------+
|      8 | pradip   |      80000 |          0 |
|      9 | pradip   |      80000 |          0 |
|     10 | pradip   |      80000 |          0 |
|     11 | pradip   |      80000 |          0 |
|     12 | pradip   |      15000 |          0 |
|     13 | pradip   |      80000 |       8000 |
+--------+----------+------------+------------+
6 rows in set (0.000 sec)

MariaDB [test]> insert into emp3(emp_name,emp_income,income_tax) values('pradip',80000,0);
Query OK, 1 row affected (0.004 sec)

MariaDB [test]> SELECT * FROM EMP3;
+--------+----------+------------+------------+
```

| emp_no | emp_name | emp_income | income_tax |

+--------+----------+------------+------------+

|  1 | pradip  |  80000 |  8000 |

+--------+----------+------------+------------+

1 row in set (0.000 sec)

MariaDB [test]> insert into emp3(emp_name,emp_income,income_tax) values('Ajinkya',190000,0);

Query OK, 1 row affected (0.004 sec)

MariaDB [test]> SELECT * FROM EMP3;

+--------+----------+------------+------------+

| emp_no | emp_name | emp_income | income_tax |

+--------+----------+------------+------------+

|  1 | pradip  |  80000 |  8000 |

|  2 | Ajinkya  |  190000 |  28500 |

+--------+----------+------------+------------+

2 rows in set (0.000 sec)