Lab 5:

OBJECTIVE: To show the implementation of FIFO Page Replacement Algorithm.

THEORY:

In virtual memory systems, page replacement algorithms are used when a page must be loaded into memory, but all frames are occupied. The algorithm decides which page should be removed.

The FIFO (First-In-First-Out) Page Replacement Algorithm is the simplest method:

- Pages are stored in the order they arrive.
- When a new page must be loaded and all frames are full, the page that has been in memory the longest (the oldest one) is replaced.
- FIFO is easy to implement but may lead to Belady's anomaly, where increasing the number of frames paradoxically increases the number of page faults.

Key terms:

- Page Fault: When a referenced page is not in memory.
- Hit: When a referenced page is already present in memory.

Algorithm:

1. Start.
2. Initiate all frames to empty.
3. For each page in the reference string:
   a. Check if the page is already present in the frame.
      - If yes: It is a hit (no replacement needed).
      - If no: It is a page fault:
        i. Replace the oldest page (using FIFO order).
        ii. Increment the page fault counter.
4. Move the FIFO pointer to the next frame position cyclically.
5. Repeat until all pages are processed.
6. Display the frame status after each page reference.
7. Outline the total number of page faults.
8. Stop.

LAB WORK:

Program: FIFO Page Replacement in C

Source Code:

```
#include <stdio.h>
int main() {
    int i, j, n, f, page[50], frame[10], k = 0,
flag, pageFaults = 0;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    printf("Enter the reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &page[i]);
    printf("Enter the number of frames: ");
    scanf("%d", &f);
    for (i = 0; i < f; i++)
        frame[i] = -1;
    printf("\nPage Replacement Process
using FIFO:\n");
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < f; j++) {
            if (frame[j] == page[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            frame[k] = page[i];
            k = (k + 1) % f;
            pageFaults++;
            for (j = 0; j < f; j++) {
                if (frame[j] != -1)
                    printf("%d ", frame[j]);
                else
                    printf("- ");
            }
            printf("\n");
        }
    }
    printf("\nTotal Page Faults = %d\n",
pageFaults);
    return 0;
}
```

Output:

```
PS D:\os> cd "d:\os\" ; if ($?) { gcc fifo.c -o fifo } ; if ($?) { .\fifo }
Enter the number of pages: 10
Enter the reference string: 0 1 2 3 4 5 6 7 8 9
Enter the number of frames: 3

Page Replacement Process using FIFO:
0 - -
0 1 -
0 1 2
3 1 2
3 4 2
3 4 5
6 4 5
6 7 5
6 7 8
9 7 8

Total Page Faults = 10
PS D:\os>
```

CONCLUSION:

The FIFO Page Replacement Algorithm was successfully implemented. The program correctly replaces the oldest page in memory when a page fault occurs and maintains the frame status throughout execution. The total number of page faults is calculated and displayed, showing the efficiency of the FIFO policy for the given reference string. While FIFO is simple and easy to implement, it may suffer from Belady's anomaly, meaning that increasing the number of frames does not always reduce the number of page faults.

Lab 6:

OBJECTIVE: To show the implementation of LRU Page Replacement Algorithm.

THEORY:

The LRU Page Replacement Algorithm replaces the page that has not been used for the longest time whenever a page fault occurs and memory is full.

- LRU considers the recency of use, unlike FIFO which only considers arrival time.
- It generally performs better than FIFO because it removes pages less likely to be used soon.
- Implementing LRU typically requires keeping track of the order of page accesses.

Key terms:

- Page Fault: When a referenced page is not in memory.
- Hit: When a referenced page is already in memory.

Algorithm:

1. Start.
2. Initialize all frames as empty.
3. For each page in the reference string:
   a. If the page is already in a frame: hit, do nothing.
   b. If the page is not in a frame: page fault occurs:
      i. If there is an empty frame, insert the page.
      ii. If all the frames are full, replace the page that was least recently used.
      iii. Update the recent usage record for all pages.
4. Display the frame status after each page reference.
5. Count and display the total page faults.
6. Stop.

LAB WORK:

Program: LRU Page Replacement in C

Source Code:

```
#include <stdio.h>
int main() {
    int n, f, page[50], frame[10], count[10], i,
j, k, pos, min, pageFaults = 0, flag;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    printf("Enter the reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &page[i]);
    printf("Enter the number of frames: ");
    scanf("%d", &f);
    for (i = 0; i < f; i++) {
        frame[i] = -1;
        count[i] = 0;
    }
    printf("\nPage Replacement Process
using LRU:\n");
    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < f; j++) {
            if (frame[j] == page[i]) {
                flag = 1;
                count[j] = i;
                break;
            }
        }
        if (flag == 0) {
            pos = 0;
            min = count[0];
            for (j = 0; j < f; j++) {
                if (frame[j] == -1) {
                    pos = j;
                    break;
                }
                if (count[j] < min) {
                    min = count[j];
                    pos = j;
                }
            }
            frame[pos] = page[i];
            count[pos] = i;
            pageFaults++;
        }
        for (k = 0; k < f; k++) {
            if (frame[k] != -1)
                printf("%d ", frame[k]);
            else
                printf("- ");
        }
        printf("\n");
    }
    printf("\nTotal Page Faults = %d\n",
pageFaults);
    return 0;
}
```

Output:

```
PS D:\os> cd "d:\os\" ; if ($?) { gcc lru.c -o lru } ; if ($?) { .\lru }
Enter the number of pages: 10
Enter the reference string: 0 1 2 3 4 5 6 7 8 9
Enter the number of frames: 3

Page Replacement Process using LRU:
0 - -
0 1 -
0 1 2
3 1 2
3 4 2
3 4 5
6 4 5
6 7 5
6 7 8
9 7 8

Total Page Faults = 10
PS D:\os>
```

CONCLUSION:

The program successfully implements the LRU (Least Recently Used) Page Replacement Algorithm. It correctly replaces the page that has not been used for the longest time whenever a page fault occurs and updates the memory frames accordingly. The count[] array tracks the last usage of each frame, ensuring the least recently used page is replaced. The total number of page faults is calculated and displayed, demonstrating the algorithm's behavior. While this implementation is straightforward and works for small reference strings, it uses a simple array for tracking recency and may not be the most efficient for very large inputs.

Lab 7:

OBJECTIVE: To show the implementation of FCFS Disk Scheduling Algorithm.

THEORY:

FCFS is the simplest disk scheduling algorithm. It services disk I/O requests in the order they arrive.

- Request Queue: Contains all pending I/O requests.
- Head Movement: The disk arm moves from the current track to the requested track sequentially, based on arrival order.
- Advantages:
  - Simple and easy to implement.
  - Fair; no request is starved.
- **Disadvantages:**
  - May result in high average seek time if requests are scattered across the disk.

Key terms:

- Seek Time: Time taken by the disk arm to move to the desired track.
- Head: Disk read/write arm.
- Request Queue: Sequence of disk I/O requests.

Algorithm:

1. Start.
2. Accept the number of requests and the request queue.
3. Accept the initial position of the disk head.
4. For each request in the queue (in arrival order):
   a. Move the head to the requested track.
   b. Calculate and accumulate the seek time.
5. Display the sequence of head movements and the total/average seek time.
6. Stop.

LAB WORK:

Program: FCFS Disk Scheduling in C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n, i, head, total = 0, distance;
    int request[50];
    printf("Enter the number of requests: ");
    scanf("%d", &n);
    printf("Enter the request sequence: ");
    for (i = 0; i < n; i++)
        scanf("%d", &request[i]);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("\nOrder of servicing requests:\n");
    for (i = 0; i < n; i++) {
        distance = abs(head - request[i]);
        total += distance;
        printf("Move from %d to %d with seek %d\n", head, request[i], distance);
        head = request[i];
    }
    printf("\nTotal Head Movement = %d\n", total);
    printf("Average Seek Time = %.2f\n", (float)total / n);
    return 0;
}
```

Output:

```
PS D:\os> cd "d:\os\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?) { .\fcfs }
Enter the number of requests: 5
Enter the request sequence: 69 56 72 98 110
Enter the initial head position: 50

Order of servicing requests:
Move from 50 to 69 with seek 19
Move from 69 to 56 with seek 13
Move from 56 to 72 with seek 16
Move from 72 to 98 with seek 26
Move from 98 to 110 with seek 12

Total Head Movement = 86
Average Seek Time = 17.20
PS D:\os>
```

CONCLUSION:

The program successfully implements the FCFS Disk Scheduling Algorithm. It services disk requests in the order they arrive, calculates the seek distance for each move, and accumulates the total head movement. The average seek time is also computed, giving insight into the efficiency of the disk scheduling. While FCFS is fair and simple to implement, it may lead to high average seek time if requests are scattered across the disk, as it does not optimize head movement.

Lab 8:

OBJECTIVE: To show the implementation of OPR Algorithm.

THEORY:

The Optimal Page Replacement (OPR) Algorithm, also called the OPT algorithm, is a page replacement strategy used in operating systems to manage memory efficiently. It aims to minimize page faults by replacing the page that will not be used for the longest time in the future.

Key points:

- Decision Basis: OPR requires knowledge of the future reference string to decide which page to replace.
- Page Fault: Occurs when the requested page is not present in memory.
- Optimality: Among all page replacement algorithms, OPR produces the minimum number of page faults for a given reference string.

Algorithm:

1. Start.
2. Initialize memory frames as empty.
3. Read the reference string and number of frames.
4. For each page in the reference string:
   a. Check if the page is already present in a frame:
      - If yes: page hit, do nothing.
      - If no: page fault occurs:
        i. If an empty frame is available, insert the page.
        ii. If all frames are full, replace the page that will not be used for the longest period in the future.
5. Update memory frames and keep track of page faults.
6. Repeat steps 4–5 until all pages in the reference string are processed.
7. Display the sequence of memory frames and total page faults.
8. Stop.

LAB WORK:

Program: Optical Page Replacement in C

Source Code:

```
#include <stdio.h>
int main() {
    int i, j, k, n, framesCount, pages[50],
frames[10], future[10];
    int pageFaults = 0, flag, farthest, pos;

    printf("Enter number of frames: ");
    scanf("%d", &framesCount);
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    for (i = 0; i < framesCount; i++)
        frames[i] = -1;

    for (i = 0; i < n; i++) {
        flag = 0;
        for (j = 0; j < framesCount; j++) {
            if (frames[j] == pages[i]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            int emptyFlag = 0;
            for (j = 0; j < framesCount; j++) {
                if (frames[j] == -1) {
                    frames[j] = pages[i];
                    pageFaults++;
                    emptyFlag = 1;
                    break;
                }
            }
            if (!emptyFlag) {
                for (j = 0; j < framesCount; j++)
{
                    future[j] = -1;
                    for (k = i + 1; k < n; k++) {
                        if (frames[j] == pages[k]) {
                            future[j] = k;
                            break;
                        }
                    }
                }
                farthest = -1;
                pos = -1;
                for (j = 0; j < framesCount; j++)
{
                    if (future[j] == -1) {
                        pos = j;
                        break;
                    }
                    if (future[j] > farthest) {
                        farthest = future[j];
                        pos = j;
                    }
                }
                frames[pos] = pages[i];
                pageFaults++;
            }
        }
        printf("\nFrames after accessing %d: ",
pages[i]);
        for (j = 0; j < framesCount; j++) {
            if (frames[j] != -1)
                printf("%d ", frames[j]);
            else
                printf("- ");
        }
    }
    printf("\n\nTotal Page Faults: %d\n",
pageFaults);
    return 0;
}
```

Output:

```
PS D:\os> cd "d:\os\" ; if ($?) { gcc opr.c -o opr } ; if ($?) { .\opr }
Enter number of frames: 3
Enter number of pages: 10
Enter the reference string: 0 1 2 3 4 5 6 7 8 9

Frames after accessing 0: 0 - -
Frames after accessing 1: 0 1 -
Frames after accessing 2: 0 1 2
Frames after accessing 3: 3 1 2
Frames after accessing 4: 4 1 2
Frames after accessing 5: 5 1 2
Frames after accessing 6: 6 1 2
Frames after accessing 7: 7 1 2
Frames after accessing 8: 8 1 2
Frames after accessing 9: 9 1 2

Total Page Faults: 10
PS D:\os>
```

CONCLUSION:

The program implements the Optimal Page Replacement (OPR) Algorithm, replacing the page that will not be used for the longest time in the future. It efficiently minimizes page faults and provides the best possible performance for a given reference string, though it requires knowledge of future page references.

Lab 9:

OBJECTIVE: To show the avoidance of deadlock using Banker's Algorithm.

THEORY:

The Banker's Algorithm, proposed by Edsger Dijkstra, is a deadlock avoidance algorithm used in operating systems to allocate resources safely. It ensures that the system never enters an unsafe state where deadlocks could occur.

Key Concepts:

- Safe State: A state where there exists a sequence of processes such that each can obtain its maximum resource needs and complete execution.
- Unsafe State: A state where such a sequence does not exist; entering it may lead to deadlock.
- Resource Allocation: The algorithm checks before granting any resource request if the system will remain in a safe state.

Algorithm:

1. Start.
2. Initialize: Read the number of processes, resources, allocation, maximum need, and available resources.
3. Calculate Need Matrix: Need[i][j] = Max[i][j] - Allocation[i][j]
4. For each resource request by a process:
    a. If Request[i] ≤ Need[i] and Request[i] ≤ Available:
        i. Temporarily allocate the resources.
        ii. Perform a safety check to see if the system remains in a safe state.
        iii. If safe: grant resources.
        iv. If unsafe: deny request; process waits.
    b. b. If request exceeds Need or Available: process waits.
5. Repeat until all processes are executed safely.
6. Stop.

LAB WORK:

Program: Banker's algorithm in C

Source Code:

```
#include <stdio.h>
int main() {
   int n, m, i, j, k;
   int alloc[10][10], max[10][10],
avail[10];
   int need[10][10], finish[10], safeSeq[10];
   int work[10], count = 0, found;

   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter number of resource types:
");
   scanf("%d", &m);

   printf("Enter Allocation matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &alloc[i][j]);

   printf("Enter Max matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &max[i][j]);

   printf("Enter Available resources:\n");
   for (i = 0; i < m; i++)
      scanf("%d", &avail[i]);

   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];

   for (i = 0; i < n; i++)
      finish[i] = 0;
   for (i = 0; i < m; i++)
      work[i] = avail[i];

   while (count < n) {
      found = 0;
      for (i = 0; i < n; i++) {
         if (!finish[i]) {
            int canAllocate = 1;
            for (j = 0; j < m; j++) {
               if (need[i][j] > work[j]) {
                  canAllocate = 0;
                  break;
               }
            }
            if (canAllocate) {
               for (k = 0; k < m; k++)
                  work[k] += alloc[i][k];
               safeSeq[count++] = i;
               finish[i] = 1;
               found = 1;
            }
         }
      }
      if (!found) {
         printf("System is in UNSAFE state.
Deadlock may occur.\n");
         return 0;
      }
   }

   printf("System is in SAFE state.\nSafe
Sequence: ");
   for (i = 0; i < n; i++)
      printf("P%d ", safeSeq[i]);
   printf("\n");
   return 0;
}
```

Output:

```
PS D:\os> cd "d:\os\" ; if ($?) { gcc banker.c -o banker } ; if ($?) { .\banker }
Enter number of processes: 3
Enter number of resource types: 3
Enter Allocation matrix:
0 0 1
0 2 1
3 1 6
Enter Max matrix:
1 0 0
2 2 2
4 0 1
Enter Available resources:
3 3 3
System is in SAFE state.
Safe Sequence: P0 P1 P2
PS D:\os>
```

CONCLUSION:

The program demonstrates deadlock avoidance using the Banker's Algorithm. It checks whether the system is in a safe or unsafe state before allocating resources. By following this approach, the system ensures that all processes can complete without causing deadlock, maintaining safe execution.

Lab 10:

OBJECTIVE: To show the implementation of Dining Philosopher Problem.

THEORY:

The Dining Philosopher Problem is a classic synchronization problem in operating systems, used to illustrate issues like deadlock, starvation, and resource sharing.

- Scenario:
  Five philosophers sit around a table. Each philosopher alternates between thinking and eating. There is a single fork between each pair of philosophers. To eat, a philosopher needs both forks on either side.
- Problem:
  If all philosophers pick up the left fork at the same time, no one can eat, causing a deadlock. Similarly, some philosophers may never get a chance to eat (starvation) if resources are not properly synchronized.
- Solution:
  Use semaphores or other synchronization mechanisms to control access to forks, ensuring:
  1. No two neighboring philosophers eat at the same time.
  2. Deadlock is avoided.
  3. Each philosopher eventually gets a chance to eat.

Algorithm:

1. Start.
2. Initialize semaphore for each chopstick with value 1.
3. Repeat for each philosopher:
   a. Think for some time.
   b. Wait (down operation) on left chopstick.
   c. Wait (down operation) on right chopstick.
   d. Eat for some time.
   e. Signal (up operation) left chopstick.
   f. Signal (up operation) right chopstick.
4. Repeat forever of for a given time.
5. Stop.

LAB WORK:

Program: Dining Philosopher Problem in C

Source Code:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define N 5

sem_t chopstick[N];
pthread_t philosopher[N];
int phil_num[N] = {0, 1, 2, 3, 4};

void *dine(void *num) {
   int id = *(int *)num;

   printf("Philosopher %d is thinking.\n",
id);
   sleep(1);
   sem_wait(&chopstick[id]);
   sem_wait(&chopstick[(id + 1) % N]);
   printf("Philosopher %d is eating.\n", id);
   sleep(2);
   sem_post(&chopstick[id]);
   sem_post(&chopstick[(id + 1) % N]);
   printf("Philosopher %d finished eating
and is thinking again.\n", id);
   return NULL;
}

int main() {
   int i;
   for (i = 0; i < N; i++)
      sem_init(&chopstick[i], 0, 1);
   for (i = 0; i < N; i++)
      pthread_create(&philosopher[i],
NULL, dine, &phil_num[i]);
   for (i = 0; i < N; i++)
      pthread_join(philosopher[i], NULL);
   for (i = 0; i < N; i++)
      sem_destroy(&chopstick[i]);
   return 0;
}
```

Output:

```
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is eating.
Philosopher 3 is eating.
Philosopher 1 finished eating and is thinking again.
Philosopher 0 is eating.
Philosopher 2 is eating.
Philosopher 3 finished eating and is thinking again.
Philosopher 0 finished eating and is thinking again.
Philosopher 2 finished eating and is thinking again.
Philosopher 4 is eating.
Philosopher 4 finished eating and is thinking again.


=== Code Execution Successful ===
```

CONCLUSION:

The program simulates the Dining Philosopher Problem using semaphores for fork synchronization. It ensures that philosophers can think and eat alternately without causing deadlock, demonstrating proper resource sharing and process synchronization.

S.N	Name of Experiment	Date of Experiment	Date of Submission	Signature
5.	To show the implementation of FIFO page replacement algorithm	2082-05-01	2082-05- 12	
6.	To show the implementation of LRU page replacement algorithm	2082-05-01	2082-05- 12	
7.	To show the implementation of FCFS disk scheduling algorithm.	2082-05-01	2082-05- 12	
8.	To show the implementation of OPR page replacement algorithm.	2082-05-01	2082-05- 12	
9.	To show the avoidance of deadlock using banker's algorithm.	2082-05-01	2082-05- 12	
10.	To show the implementation of dining philosopher problem.	2082-05-01	2082-05- 12	

Lab 3

1. Title: To study different data types in MySQL.

2. Theory: In MySQL, data types define the kind of values a column can store in a database table. Choosing the correct data type is very important because it affects storage requirements, performance, and data accuracy. MySQL provides several categories of data types:

1. Numeric Data Types

Used for storing numbers (integers and decimals).

- Integer Types
  - TINYINT → very small integers (-128 to 127)
  - SMALLINT → small integers (-32,768 to 32,767)
  - MEDIUMINT → medium integers (-8 million to 8 million approx.)
  - INT / INTEGER → standard integer (-2 billion to 2 billion)
  - BIGINT → very large integers
- Decimal & Floating Types
  - DECIMAL(M, D) or NUMERIC(M, D) → exact values (good for money)
  - FLOAT → approximate single-precision floating-point numbers
  - DOUBLE / REAL → approximate double-precision floating-point numbers

2. Date and Time Data Types

Used to store dates, times, and timestamps.

- DATE → stores date in format YYYY-MM-DD
- TIME → stores time in format HH:MM:SS
- DATETIME → stores both date and time
- TIMESTAMP → stores date & time (with automatic updates for current time)
- YEAR → stores year in 2-digit or 4-digit format

3. String (Character) Data Types

Used to store text, characters, or binary data.

- Fixed-length strings
  - CHAR(n) → stores fixed-length strings (always uses n characters)
- Variable-length strings
  - VARCHAR(n) → stores variable-length strings (saves space if text length varies)
- Text types (for large text storage)
  - TINYTEXT → up to 255 characters
  - TEXT → up to 65,535 characters
  - MEDIUMTEXT → up to 16 million characters
  - LONGTEXT → up to 4 billion characters
- Binary types (for images, files, etc.)
  - BLOB (Binary Large Object) with variations:

    - TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- Others
    - ENUM('val1', 'val2', …) → predefined set of values (choose one)
    - SET('val1', 'val2', …) → predefined set of values (choose multiple)

4. Boolean Data Type

- MySQL doesn't have a direct BOOLEAN type.
- Instead, BOOLEAN is treated as a synonym for TINYINT(1), where 0 = FALSE and 1 = TRUE

Key Points:

- Always select the most efficient data type (saves memory & improves performance).
- Use DECIMAL for exact precision (e.g., financial data) instead of FLOAT.
- For strings, use CHAR when values have a fixed length and VARCHAR when lengths vary.
- ENUM and SET help restrict values to predefined options.
- Date and time types are best for chronological data handling instead of storing as strings.

3. Queries:

Database with a table named student with multiple datatypes:

```
+-----------------+-----------------------------------------+------+-----+---------------------+----------------+
| Field           | Type                                    | Null | Key | Default             | Extra          |
+-----------------+-----------------------------------------+------+-----+---------------------+----------------+
| student_id      | int(11)                                 | NO   | PRI | NULL                | auto_increment |
| roll_no         | tinyint(3) unsigned                     | YES  |     | NULL                |                |
| marks           | decimal(5,2)                            | YES  |     | NULL                |                |
| grade           | char(1)                                 | YES  |     | NULL                |                |
| name            | varchar(100)                            | YES  |     | NULL                |                |
| address         | text                                    | YES  |     | NULL                |                |
| gender          | enum('Male','Female','Other')           | YES  |     | NULL                |                |
| hobbies         | set('Sports','Music','Reading','Coding') | YES  |     | NULL                |                |
| dob             | date                                    | YES  |     | NULL                |                |
| admission_time  | timestamp                               | NO   |     | current_timestamp() |                |
| year_of_passout | year(4)                                 | YES  |     | NULL                |                |
| active          | tinyint(1)                              | YES  |     | NULL                |                |
+-----------------+-----------------------------------------+------+-----+---------------------+----------------+
```

Insert into table student:

```
MariaDB [users]> select * from student;
+------------+---------+-------+-------+--------------+-------------------------------+--------+---------------+------------+---------------------+--------------+--------+
| student_id | roll_no | marks | grade | name         | address                       | gender | hobbies       | dob        | admission_time      | year_of_passout | active |
+------------+---------+-------+-------+--------------+-------------------------------+--------+---------------+------------+---------------------+--------------+--------+
|          1 |       1 | 95.50 | A     | Rahul Sharma | Pokhara-12, Kaski, Nepal      | Male   | Sports,Music  | 2004-11-23 | 2025-08-20 08:05:48 | 2022         |      1 |
|          2 |       2 | 88.75 | B     | Sita Adhikari| Lalitpur-3, Kathmandu, Nepal  | Female | Reading,Coding| 2005-05-14 | 2025-08-20 08:05:48 | 2023         |      1 |
|          3 |       3 | 72.25 | C     | Aman Banstola| Butwal-5, Rupandehi, Nepal    | Male   | Music         | 2004-09-09 | 2025-08-20 08:05:48 | 2021         |      0 |
|          4 |       4 | 64.00 | D     | Rita Koirala | Dharan-7, Sunsari, Nepal      | Female | Sports,Reading| 2003-01-25 | 2025-08-20 08:05:48 | 2021         |      1 |
|          5 |       5 | 91.10 | A     | Kiran Shrestha| Bhaktapur-10, Bhaktapur, Nepal| Other  | Coding        | 2005-12-05 | 2025-08-20 08:05:48 | 2024         |      1 |
+------------+---------+-------+-------+--------------+-------------------------------+--------+---------------+------------+---------------------+--------------+--------+
5 rows in set (0.005 sec)
```

4. Conclusion: Understanding MySQL data types is fundamental to effective database design. Each category—numeric, string, date/time, and special—serves a distinct purpose and offers specific advantages. Selecting the appropriate data type ensures efficient storage, accurate data representation, and optimal performance. This lab provides a foundational overview that supports deeper exploration into schema design, normalization, and query optimization.

Lab 4

1. Title: To study and implement the alter command in MYSQL.

2. Theory:

The ALTER command in MySQL is **a** Data Definition Language (DDL) command.
It is used to modify the structure of an existing table without dropping and recreating it.

With ALTER TABLE, you can:

- Add new columns
- Modify existing columns (datatype, size, constraints)
- Rename a column
- Drop (delete) a column
- Rename the table itself
- Add or drop constraints (like PRIMARY KEY, FOREIGN KEY, UNIQUE)
- Change default values

Common Uses of ALTER TABLE

Add a New Column

ALTER TABLE student ADD age INT;

Adds a new column age of type INT.

Add Multiple Columns

ALTER TABLE student
ADD address VARCHAR(100),
ADD phone VARCHAR(15);

Modify a Column (datatype/size)

ALTER TABLE student MODIFY name VARCHAR(150);

Changes the size of name column.

Rename a Column

ALTER TABLE student CHANGE old_name new_name VARCHAR(100);

CHANGE lets you rename a column and change its datatype.

Drop a Column

ALTER TABLE student DROP COLUMN phone;

Rename a Table

ALTER TABLE student RENAME TO learners;

Add a Constraint

ALTER TABLE student ADD CONSTRAINT pk_student PRIMARY KEY (id);

Drop a Constraint

ALTER TABLE student DROP PRIMARY KEY;

Perform Multiple Actions

MySQL allows multiple alterations in a single ALTER TABLE statement, making schema updates efficient and
atomic.

Syntax:

ALTER TABLE table_name ADD COLUMN column1 datatype, MODIFY COLUMN column2 new_datatype,
DROP COLUMN column3;
Example:
ALTER TABLE students ADD COLUMN email VARCHAR(100), MODIFY COLUMN student_id INT NOT
NULL, DROP COLUMN age;

3. Queries:

Create database:

```
MariaDB [users]> describe student;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| name    | varchar(20)          | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
4 rows in set (0.020 sec)
```

Add column:

```
MariaDB [users]> alter table student add column phone varchar(30);
Query OK, 0 rows affected (0.026 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe student;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| name    | varchar(20)          | YES  |     | NULL    |                |
| phone   | varchar(30)          | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
5 rows in set (0.018 sec)
```

Drop Column:

```
MariaDB [users]> alter table student drop phone;
Query OK, 0 rows affected (0.022 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe student;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| name    | varchar(20)          | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
4 rows in set (0.024 sec)
```

Modify a column:

```
MariaDB [users]> alter table student modify column name varchar(100);
Query OK, 0 rows affected (0.023 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe student;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| name    | varchar(100)         | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
4 rows in set (0.019 sec)
```

Change table name:

```
MariaDB [users]> alter table student rename to students;
Query OK, 0 rows affected (0.025 sec)

MariaDB [users]> describe students;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| name    | varchar(100)         | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
4 rows in set (0.008 sec)
```

Change column name and properties:

```
MariaDB [users]> alter table students change name sname varchar(30);
Query OK, 0 rows affected (0.047 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe students;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| sid     | int(11)              | NO   | PRI | NULL    | auto_increment |
| address | text                 | YES  |     | NULL    |                |
| gender  | enum('Male','Female') | YES  |     | NULL    |                |
| sname   | varchar(30)          | YES  |     | NULL    |                |
+---------+----------------------+------+-----+---------+----------------+
4 rows in set (0.021 sec)
```

Add or remove primary key:

```
MariaDB [users]> alter table students add primary key(sid);
Query OK, 0 rows affected (0.043 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe students;
+---------+----------------------+------+-----+---------+-------+
| Field   | Type                 | Null | Key | Default | Extra |
+---------+----------------------+------+-----+---------+-------+
| sid     | int(11)              | NO   | PRI | NULL    |       |
| address | text                 | YES  |     | NULL    |       |
| gender  | enum('Male','Female') | YES  |     | NULL    |       |
| sname   | varchar(30)          | YES  |     | NULL    |       |
+---------+----------------------+------+-----+---------+-------+
4 rows in set (0.007 sec)
```

```
MariaDB [users]> alter table students modify sid int;
Query OK, 0 rows affected (0.017 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> alter table students drop primary key;
Query OK, 0 rows affected (0.045 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe students;
+---------+----------------------+------+-----+---------+-------+
| Field   | Type                 | Null | Key | Default | Extra |
+---------+----------------------+------+-----+---------+-------+
| sid     | int(11)              | NO   |     | NULL    |       |
| address | text                 | YES  |     | NULL    |       |
| gender  | enum('Male','Female') | YES  |     | NULL    |       |
| sname   | varchar(30)          | YES  |     | NULL    |       |
+---------+----------------------+------+-----+---------+-------+
4 rows in set (0.021 sec)
```

Perform multiple actions:

```
MariaDB [users]> alter table students
    -> add column phone varchar(10),
    -> drop column gender;
Query OK, 0 rows affected (0.020 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> describe students;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| sid     | int(11)     | NO   | PRI | NULL    |       |
| address | text        | YES  |     | NULL    |       |
| sname   | varchar(30) | YES  |     | NULL    |       |
| phone   | varchar(10) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.011 sec)
```

4. Conclusion: Hence, the ALTER command in MySQL provides a flexible way to modify table structures without losing existing data. It enables adding, modifying, or dropping columns, renaming tables or columns, and managing primary keys, making it essential for efficient and organized database management.

Lab 5

1. Title: To study and implement the DROP command in MYSQL.

2. Theory: The DROP command in MySQL is a powerful SQL statement used to permanently delete database objects, including databases, tables, columns, indexes, and constraints. Unlike commands such as DELETE or TRUNCATE, which only remove data while keeping the table or structure intact, the DROP command completely removes both the object and all the data it contains, leaving no trace behind. This means that once executed, the operation is irreversible, and the deleted objects cannot be recovered unless a prior backup exists. The DROP command is widely used in database management for cleaning up obsolete or unused objects, restructuring databases, or freeing up storage space. Still, its destructive nature makes it extremely sensitive. Therefore, it requires careful planning, proper authorization, and caution before execution to prevent accidental loss of valuable data. It is an essential tool for database administrators, but it must always be used responsibly to maintain the integrity and stability of the database system.

Precautions:

1. The DROP command is irreversible; once executed, data cannot be retrieved unless a backup exists.

2. It helps in database maintenance, cleaning up unused objects, and redesigning structures.

3. Proper authorization and caution are required since accidental usage can lead to data loss.

4. Always ensure backups exist before performing drop operations, especially on important databases or tables.

Key Uses of the DROP Command:

• Drop a Database: Removes an entire database including all tables, views, triggers, and procedures inside it.

DROP DATABASE database_name;

• Drop a Table: Deletes an entire table along with all its rows and structure.

DROP TABLE table_name;

• Drop a Column: Removes a column from an existing table. ALTER

TABLE table_name DROP COLUMN column_name;

• Drop an Index or Constraint: Deletes an index or a foreign key constraint from a table.
DROP INDEX index_name ON table_name;

3. Queries:

```
MariaDB [users]> select * from students;
+-----+-----------+--------------+-------------+
| sid | address   | sname        | phone       |
+-----+-----------+--------------+-------------+
|   1 | Pokhara   | Aman         | 9801234567  |
|   2 | Kathmandu | Jayaswor     | 9812345678  |
|   3 | Butwal    | Anish        | 9823456789  |
|   4 | Chitwan   | Rita         | 9845678901  |
|   5 | Dharan    | Sushma       | 9851234567  |
|   6 | Biratnagar| Ruby         | 9862345678  |
|   7 | Lalitpur  | Sita         | 9873456789  |
+-----+-----------+--------------+-------------+
7 rows in set (0.001 sec)

MariaDB [users]> alter table students drop column phone;
Query OK, 0 rows affected (0.021 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> select * from students;
+-----+-----------+--------------+
| sid | address   | sname        |
+-----+-----------+--------------+
|   1 | Pokhara   | Aman         |
|   2 | Kathmandu | Jayaswor     |
|   3 | Butwal    | Anish        |
|   4 | Chitwan   | Rita         |
|   5 | Dharan    | Sushma       |
|   6 | Biratnagar| Ruby         |
|   7 | Lalitpur  | Sita         |
+-----+-----------+--------------+
7 rows in set (0.001 sec)

MariaDB [users]> drop table students;
Query OK, 0 rows affected (0.014 sec)

MariaDB [users]> drop database users;
Query OK, 0 rows affected (0.010 sec)
```

4. Conclusion: Hence, the DROP command in MySQL provides a powerful way to permanently remove databases, tables, or columns. It allows for efficient database maintenance and restructuring, but must be used carefully, as the deleted data and structures cannot be recovered without a backup.

Lab 6

1. Title: To query the given relational database in order to retrieve the required information by using the join operations.

2. Theory: In relational databases, data is often distributed across multiple tables to reduce redundancy, maintain consistency, and improve data organization. To retrieve meaningful information that spans multiple tables, JOIN operations are used. JOINs allow combining rows from two or more tables based on a relationship between columns, enabling comprehensive and useful datasets.

Types of joins:

Inner join:

• Retrieves only the rows that have matching values in both tables.

• Ensures that the result includes only related data, ignoring unmatched rows.

• Useful when you need information that exists in both tables, such as students with enrolled courses.

Left join (left outer join):

• Returns all rows from the left table, along with the matching rows from the right table.

• If there is no match in the right table, the result will contain NULL values for the right table's columns.

• Useful for situations where you want to retain all data from the primary table, such as listing all students and showing course information where available.

Right join (right outer join):

• Returns all rows from the right table, along with the matching rows from the left table.

• Unmatched rows from the left table appear as NULL.

• Useful when the focus is on the secondary table, such as listing all courses and showing enrolled students if any.

Full outer join:

• Combines all rows from both tables. If a row has no match in the other table, NULL values are shown in the unmatched columns.

• Provides a complete dataset that includes both matched and unmatched data from all tables.

• Useful for comprehensive reports, such as showing all students and all courses, even if some students are not enrolled or some courses have no students.

Natural join:

• Automatically joins tables based on columns that share the same name and compatible data

types.

• Reduces the need to explicitly specify join conditions, simplifying queries when tables have standardized column names.

• Useful for quickly combining related tables without manually identifying the common columns.

Theta join:

• A generalized form of join where any comparison operator can be used (e.g., >, <, >=, <=, <>), not just equality.

• Allows flexible conditions for combining tables based on a specific relationship between columns.

• Useful for complex queries, such as finding students with marks greater than the average of another table's values.

3. Queries:

Create database:

```
MariaDB [users]> create table student(
    -> sid int primary key auto_increment not null,
    -> name varchar(30),
    -> address text,
    -> contact varchar(10));
Query OK, 0 rows affected (0.012 sec)

MariaDB [users]> INSERT INTO student (name, address, contact)
    -> VALUES
    -> ('Ram Sharma', 'Kathmandu', '9812345678'),
    -> ('Sita Rai', 'Pokhara', '9801122334'),
    -> ('Aman Banstola', 'Chitwan', '9845671234'),
    -> ('Krishna Thapa', 'Butwal', '9867543210'),
    -> ('Rita Gurung', 'Lalitpur', '9823456789');
Query OK, 5 rows affected (0.002 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [users]> create table book(
    -> bid int primary key auto_increment not null,
    -> title varchar(30),
    -> author varchar(30));
Query OK, 0 rows affected (0.025 sec)

MariaDB [users]> INSERT INTO book (title, author)
    -> VALUES
    -> ('Introduction to Databases', 'C. J. Date'),
    -> ('Learn SQL in 7 Days', 'John Smith'),
    -> ('Operating System Concepts', 'Silberschatz'),
    -> ('Database System Concepts', 'Henry Korth'),
    -> ('Clean Code', 'Robert C. Martin');
Query OK, 5 rows affected (0.013 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
MariaDB [users]> CREATE TABLE borrow (
    ->     borrowid INT PRIMARY KEY AUTO_INCREMENT,
    ->     sid INT,
    ->     bid INT,
    ->     FOREIGN KEY (sid) REFERENCES student(sid),
    ->     FOREIGN KEY (bid) REFERENCES book(bid)
    -> );
Query OK, 0 rows affected (0.010 sec)

MariaDB [users]> INSERT INTO borrow (sid, bid)
    -> VALUES
    -> (1, 3),
    -> (2, 5),
    -> (3, 1),
    -> (4, 2),
    -> (5, 4);
Query OK, 5 rows affected (0.013 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

All information from inner join borrow:

```
MariaDB [users]> select distinct name from student inner join borrow on student.sid=borrow.sid;
+---------------+
| name          |
+---------------+
| Ram Sharma    |
| Sita Rai      |
| Aman Banstola |
| Krishna Thapa |
| Rita Gurung   |
+---------------+
5 rows in set (0.003 sec)
```

All information from student left join borrow:

```
MariaDB [users]> select * from student inner join borrow on student.sid=borrow.sid;
+-----+---------------+-----------+------------+----------+-----+-----+
| sid | name          | address   | contact    | borrowid | sid | bid |
+-----+---------------+-----------+------------+----------+-----+-----+
|   1 | Ram Sharma    | Kathmandu | 9812345678 |        1 |   1 |   3 |
|   2 | Sita Rai      | Pokhara   | 9801122334 |        2 |   2 |   5 |
|   3 | Aman Banstola | Chitwan   | 9845671234 |        3 |   3 |   1 |
|   4 | Krishna Thapa | Butwal    | 9867543210 |        4 |   4 |   2 |
|   5 | Rita Gurung   | Lalitpur  | 9823456789 |        5 |   5 |   4 |
+-----+---------------+-----------+------------+----------+-----+-----+
5 rows in set (0.012 sec)
```

All information from student right join borrow:

```
MariaDB [users]> select * from student left join borrow on student.sid=borrow.sid;
+-----+---------------+-----------+------------+----------+-----+-----+
| sid | name          | address   | contact    | borrowid | sid | bid |
+-----+---------------+-----------+------------+----------+-----+-----+
|   1 | Ram Sharma    | Kathmandu | 9812345678 |        1 |   1 |   3 |
|   2 | Sita Rai      | Pokhara   | 9801122334 |        2 |   2 |   5 |
|   3 | Aman Banstola | Chitwan   | 9845671234 |        3 |   3 |   1 |
|   4 | Krishna Thapa | Butwal    | 9867543210 |        4 |   4 |   2 |
|   5 | Rita Gurung   | Lalitpur  | 9823456789 |        5 |   5 |   4 |
+-----+---------------+-----------+------------+----------+-----+-----+
5 rows in set (0.003 sec)
```

Name of students borrowing at least one book:

```
MariaDB [users]> select * from student right join borrow on student.sid=borrow.sid;
+-----+--------------+-----------+------------+----------+-----+-----+
| sid | name         | address   | contact    | borrowid | sid | bid |
+-----+--------------+-----------+------------+----------+-----+-----+
|   1 | Ram Sharma   | Kathmandu | 9812345678 |        1 |   1 |   3 |
|   2 | Sita Rai     | Pokhara   | 9801122334 |        2 |   2 |   5 |
|   3 | Aman Banstola | Chitwan  | 9845671234 |        3 |   3 |   1 |
|   4 | Krishna Thapa | Butwal   | 9867543210 |        4 |   4 |   2 |
|   5 | Rita Gurung  | Lalitpur  | 9823456789 |        5 |   5 |   4 |
+-----+--------------+-----------+------------+----------+-----+-----+
5 rows in set (0.003 sec)
```

Title of all books that are borrowed:

```
MariaDB [users]> select distinct title from book inner join borrow on book.bid=borrow.bid;
+---------------------------+
| title                     |
+---------------------------+
| Introduction to Databases |
| Learn SQL in 7 Days       |
| Operating System Concepts |
| Database System Concepts  |
| Clean Code                |
+---------------------------+
5 rows in set (0.003 sec)
```

Name of students borrowing no books:

```
MariaDB [users]> select name from student inner join borrow on student.sid=borrow.sid where bid is null;
Empty set (0.015 sec)

MariaDB [users]>
```

Title of all books that are not borrowed yet:

```
MariaDB [users]> select title from book inner join borrow on book.bid=borrow.bid where sid is null;
Empty set (0.002 sec)

MariaDB [users]>
```

4. Conclusion: Hence, JOIN operations are essential for querying relational databases, allowing data from multiple related tables to be combined and retrieved efficiently. By using INNER, LEFT, RIGHT, FULL OUTER, NATURAL, and THETA joins, meaningful and accurate information can be extracted, supporting effective data analysis, reporting, and decision-making. In our sample data, there were instances where every student borrowed at least one book.Therefore, we got to see an empty set as an output table.

Lab 7

1. Title: Create the following database schema and run SQL queries to retrieve the information as instructed.
Schema: employee(eid, name, post, salary, dob, gender, address, contact)

2. Theory: A database is an organized collection of data that allows easy storage, retrieval, and management. Ithelps in maintaining large amounts of information in a structured format, making it easier to update,manipulate, and analyze. A Relational Database Management System (RDBMS) stores data in tables(relations) where relationships among tables are defined through keys. The schema defines the structureof the database, including tables, columns, data types, constraints, and relationships. For instance, theschema for an employee table may be:

employee(eid, name, post, salary, dob, gender, address, contact)

This table stores essential details about employees within an organization

Importance of SQL:

• Widely used in HR systems, banking, schools, inventory management, and business analytics.

• Ensures efficient data management, integrity, and security.

• Essential for roles like database developers, data analysts, and backend developers.

SQL (Structured Query Language) is the standard language used to manage and interact with relational databases. SQL enables the creation, modification, deletion, and querying of data. Its main categories include:

Data Definition Language (DDL): Used to define and alter the database structure.

CREATE – To create new tables or databases.

ALTER – To modify existing tables. DROP

– To delete tables or databases. Example:

CREATE TABLE student (.....);

Data Manipulation Language (DML): Used to manage the data stored in tables.

INSERT – To add new records.

UPDATE – To modify existing records.

DELETE – To remove records.

Example: INSERT INTO student (name, class, …...) VALUES (….);

Data Query Language (DQL): Used to retrieve data from tables. SELECT –

Retrieves information based on specified conditions. Example: SELECT name,

class FROM student;

Use of Aliases:

Column and table aliases allow temporary renaming for better readability. Example:

SELECT s.name AS StudentName, s.post AS grade FROM student s; Here, s is the table alias, and StudentName and grade are column aliases.

Common SQL Functions and Clauses:

DISTINCT – Returns unique values only.

ORDER BY – Sorts results in ascending or descending order. BETWEEN –

Filters values within a specified range.

NOT BETWEEN – Filters values outside a specified range.

Arithmetic operations (e.g., salary * 0.15) – Can be used to calculate taxes or bonuses.

3. Queries:

Creating Database:

```
MariaDB [users]> create table employee(
    -> eid int primary key auto_increment not null,
    -> name varchar(30),
    -> post varchar(30),
    -> salary DECIMAL(10,2),
    -> dob date,
    -> gender enum("male","female"),
    -> address varchar(30),
    -> contact varchar(15));
Query OK, 0 rows affected (0.028 sec)

MariaDB [users]> insert into employee(name, post, salary, dob, gender, address, contact) values
    -> ('John Doe', 'Manager', 75000.00, '1985-05-15', 'male', '123 Main St', '555-1234'),
    -> ('Jane Smith', 'Analyst', 60000.50, '1990-11-22', 'female', '456 Oak Ave', '555-5678'),
    -> ('Peter Jones', 'Developer', 80000.75, '1988-02-10', 'male', '789 Pine Ln', '555-9012'),
    -> ('Mary Brown', 'Designer', 65000.25, '1992-07-30', 'female', '101 Maple Rd', '555-3456'),
    -> ('David Lee', 'Intern', 40000.00, '1998-09-05', 'male', '202 Birch Blvd', '555-7890');
Query OK, 5 rows affected (0.015 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Name, post, and salary of the employee using column alias and table alias:

```
MariaDB [users]> select e.name as name, e.post as post,e.salary as salary from employee e;
+-------------+-----------+----------+
| name        | post      | salary   |
+-------------+-----------+----------+
| John Doe    | Manager   | 75000.00 |
| Jane Smith  | Analyst   | 60000.50 |
| Peter Jones | Developer | 80000.75 |
| Mary Brown  | Designer  | 65000.25 |
| David Lee   | Intern    | 40000.00 |
+-------------+-----------+----------+
5 rows in set (0.013 sec)
```

List of different posts:

```
MariaDB [users]> select distinct post as post from employee;
+------------+
| post       |
+------------+
| Manager    |
| Analyst    |
| Developer  |
| Designer   |
| Intern     |
+------------+
5 rows in set (0.001 sec)
```

List out name, salary and tax amount (15% of salary).

```
MariaDB [users]> select name as name, salary as salary, salary * 0.15 as taxamount from employee;
+-------------+----------+------------+
| name        | salary   | taxamount  |
+-------------+----------+------------+
| John Doe    | 75000.00 | 11250.0000 |
| Jane Smith  | 60000.50 |  9000.0750 |
| Peter Jones | 80000.75 | 12000.1125 |
| Mary Brown  | 65000.25 |  9750.0375 |
| David Lee   | 40000.00 |  6000.0000 |
+-------------+----------+------------+
5 rows in set (0.014 sec)
```

Name of all employees in ascending order:

```
MariaDB [users]> select name as name from employee order by name asc;
+-------------+
| name        |
+-------------+
| David Lee   |
| Jane Smith  |
| John Doe    |
| Mary Brown  |
| Peter Jones |
+-------------+
5 rows in set (0.013 sec)
```

Name of employees having salary between 40,000 and 50,000:

```
MariaDB [users]> select name as name from employee where salary between 40000 and 50000;
+-----------+
| name      |
+-----------+
| David Lee |
+-----------+
1 row in set (0.003 sec)
```

Name of employee having salary not between 40,000 and 50,000:

```
MariaDB [users]> select name as name from employee where salary not between 40000 and 50000;
+-------------+
| name        |
+-------------+
| John Doe    |
| Jane Smith  |
| Peter Jones |
| Mary Brown  |
+-------------+
4 rows in set (0.001 sec)
```

4. Conclusion: This lab report effectively demonstrated the fundamental concepts of relational databases, schema definition, and the use of SQL for data manipulation and retrieval. By creating an employee table and running various queries, we were able to perform essential database operations such as inserting data, filtering records, sorting results, and calculating derived values. The exercises reinforced the importance of SQL as a powerful tool for managing and interacting with structured data, highlighting its role in ensuring data integrity, security, and efficient analysis.

Lab 8

1. Title: Create the following database schema and run the SQL queries to retrieve the required information.
Schema: employee(eid, name, post, salary, gender, address, contact)

2. Theory: A database schema defines the structure of a database, including tables, columns, data types, and constraints. Creating a schema involves designing tables to store related data efficiently and accurately. For example, an employee table may include columns like eid, name, post, salary, gender, address, and contact. This table stores employee details such as ID, name, post, salary, gender, address, and contact.

SQL (Structured Query Language) is used to manage and query data in relational databases. Key operations include:

• Creating tables: Using the CREATE TABLE statement to define columns and constraints (e.g., primary key, not null).

• Inserting data: Using INSERT INTO to add records.

• Retrieving data: Using SELECT statements to query information, apply calculations like COUNT, AVG, and find specific records (e.g., maximum salary using subqueries).

• Aggregating data: Using functions like COUNT(), SUM(), and AVG() to summarize information for analysis.

• Filtering and grouping: Using WHERE and GROUP BY clauses to analyze data by specific conditions, such as gender.

This exercise focuses on aggregate functions and subqueries to provide essential business insights:

• COUNT(): Counts the number of rows (e.g., total employees).

• AVG(): Calculates the average value (e.g., average salary).

• SUM(): Adds values (e.g., total monthly salary).

• GROUP BY: Groups rows based on a column (e.g., gender) to perform aggregate calculations per group.

• Subqueries: Retrieve data based on results of another query (e.g., employee(s) with maximum salary).

These queries help analyze workforce size, salary distribution by gender, identify top earners, and calculate total salary expenses, supporting informed decision-making and payroll management.

3. Queries:

Count the total number of employees in the ‘employee’ table.

```
MariaDB [users]> CREATE TABLE employee (
    -> eid INT PRIMARY KEY AUTO_INCREMENT,
    -> name VARCHAR(50) NOT NULL,
    -> post VARCHAR(50),
    -> salary DECIMAL(10,2),
    -> gender ENUM('Male','Female'),
    -> address VARCHAR(100),
    -> contact VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.019 sec)

MariaDB [users]> INSERT INTO employee (name, post, salary, gender, address, contact) VALUES
    -> ('Aman Banstola', 'Manager', 60000.00, 'Male', 'Pokhara', '9801234567'),
    -> ('Sita Sharma', 'Accountant', 45000.00, 'Female', 'Kathmandu', '9812345678'),
    -> ('Ramesh Koirala', 'Developer', 50000.00, 'Male', 'Lalitpur', '9823456789'),
    -> ('Priya Adhikari', 'HR Officer', 40000.00, 'Female', 'Biratnagar', '9834567890'),
    -> ('Kiran Thapa', 'Designer', 42000.00, 'Male', 'Chitwan', '9845678901');
Query OK, 5 rows affected (0.014 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [users]> select count(*) as total_employees from employee;
+----------------+
| total_employees|
+----------------+
| 5              |
+----------------+
1 row in set (0.013 sec)
```

Find the average salary of all employees.

```
MariaDB [users]> select avg(salary)  as average_salary from employee;
+-----------------+
| average_salary |
+-----------------+
|   47400.000000 |
+-----------------+
1 row in set (0.001 sec)
```

Find the average salary of male and female employees.

```
MariaDB [users]> select gender, avg(salary) as average_salary from employee group by gender;
+--------+-----------------+
| gender | average_salary |
+--------+-----------------+
| Male   |   50666.666667 |
| Female |   42500.000000 |
+--------+-----------------+
2 rows in set (0.014 sec)
```

Find the total number of male and female employees and their average salary.

```
MariaDB [users]> select gender, count(*) as total_employee, avg(salary) as average_salary from employee group by gender;
+--------+----------------+-----------------+
| gender | total_employee | average_salary |
+--------+----------------+-----------------+
| Male   |              3 |   50666.666667 |
| Female |              2 |   42500.000000 |
+--------+----------------+-----------------+
2 rows in set (0.002 sec)

MariaDB [users]>
```

Find the name post and salary of the employee having maximum salary (use subquery)

```
MariaDB [users]> select name, post from employee where salary=(select max(salary) from employee);
+--------------+---------+
| name         | post    |
+--------------+---------+
| Aman Banstola | Manager |
+--------------+---------+
1 row in set (0.001 sec)
```

Find the total amount the company must pay as salary per month.

```
MariaDB [users]> select sum(salary) as total_salary from employee;
+---------------+
| total_salary |
+---------------+
|    237000.00 |
+---------------+
1 row in set (0.001 sec)
```

How much does the company pay to all the females as salary?

```
MariaDB [users]> select sum(salary) as female_salary from employee where gender='female';
+---------------+
| female_salary |
+---------------+
|      85000.00 |
+---------------+
1 row in set (0.001 sec)
```

Find the name of an employee having a salary less than average.

```
MariaDB [users]> select name from employee where salary<(select avg(salary) from employee);
+----------------+
| name           |
+----------------+
| Sita Sharma    |
| Priya Adhikari |
| Kiran Thapa    |
+----------------+
3 rows in set (0.002 sec)
```

4. Conclusion: In this lab, we created and manipulated a database schema using MySQL. We practiced essential operations such as ALTER (to modify the structure of a table), UPDATE (to modify existing records), and DELETE (to remove records). We also learned to: Add new columns to existing tables, Update records based on conditions and subqueries, use aggregate functions (AVG, SUM, COUNT) for data analysis, apply string manipulation (CONCAT) in updates, and Change table and column names for better schema management. This lab reinforced the practical use of SQL in managing databases effectively, ensuring flexibility in modifying structures, maintaining data accuracy, and supporting organizational decision-making.

Lab 9

1. Title: Create the following database schema and run the different UPDATE, DELETE, and ALTER queries using MySQL.

2. Theory: In relational database management systems (RDBMS) like MySQL, it is often necessary not only to create and retrieve data but also to modify, update, or restructure existing tables. SQL provides powerful commands such as UPDATE, DELETE, and ALTER to perform these tasks efficiently. These operations ensure that the database remains accurate, flexible, and adaptable to new requirements.

UPDATE Command

The UPDATE command is used to modify existing records in a table. It allows users to change one or more column values based on specific conditions.

General Syntax:

- UPDATE table_name
- SET column1 = value1, column2 = value2
- WHERE condition;
- **Example:** Increasing all employees' salaries by 10%:
- UPDATE employee
- SET salary = salary * 1.10;

The WHERE clause is optional, but without it, the update applies to **all rows** in the table.

DELETE Command

The DELETE command removes rows from a table based on a specified condition. It helps maintain data accuracy by eliminating unnecessary or invalid records.

General Syntax:

- DELETE FROM table_name
- WHERE condition;
- **Example:** Deleting all male employees from Pokhara:
- DELETE FROM employee
- WHERE city = 'Pokhara' AND gender = 'Male';

If the WHERE clause is omitted, all rows will be deleted, leaving the table structure intact.

ALTER Command

The ALTER command modifies the structure of an existing table without deleting its data. It can be used to add, delete, or modify columns, as well as rename columns or tables.

- **Adding a column:**
- ALTER TABLE employee
- ADD COLUMN postal_code VARCHAR(10) AFTER city;
- **Renaming a column:**
- ALTER TABLE employee
- CHANGE COLUMN name full_name VARCHAR(30);
- **Renaming a table:**
- RENAME TABLE employee TO emp;

Thus, ALTER provides schema flexibility as organizational needs evolve.

Practical Applications of These Queries

- **Updating Salaries:** Companies frequently need to revise salaries, bonuses, or designations. Using UPDATE, we can apply global increments (e.g., all employees get a 10% raise) or targeted changes (e.g., only managers get a raise).
- **Maintaining Accuracy:** With DELETE, invalid or outdated records can be removed. For example, removing all male employees from Pokhara ensures the database reflects current staff data.
- **Adapting Schema:** Businesses often need to modify their data model. Adding a postal_code column allows more precise employee location tracking, while renaming name to full_name improves clarity. Similarly, renaming tables helps align with new naming standards.

3. Queries:

Creating database:

```
MariaDB [users]> CREATE TABLE employee (
    -> eid INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    -> name VARCHAR(30),
    -> post VARCHAR(30),
    -> salary DECIMAL(10,2),
    -> gender ENUM('Male','Female'),
    -> city VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.041 sec)

MariaDB [users]> INSERT INTO employee (name, post, salary, gender, city) VALUES
    -> ('Aman Banstola','Manager',55000.00,'Male','Pokhara'),
    -> ('Sita Sharma','Accountant',42000.00,'Female','Kathmandu'),
    -> ('Ramesh Koirala','Developer',48000.00,'Male','Lalitpur'),
    -> ('Priya Adhikari','HR Officer',40000.00,'Female','Biratnagar'),
    -> ('Kiran Thapa','Designer',45000.00,'Male','Chitwan');
Query OK, 5 rows affected (0.013 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Increase salary of all employees by 10%:

```
MariaDB [users]> update employee set salary=(salary * 1.1);
Query OK, 5 rows affected (0.024 sec)
Rows matched: 5  Changed: 5  Warnings: 0

MariaDB [users]> select * from employee;
+-----+---------------+-----------+----------+--------+-----------+
| eid | name          | post      | salary   | gender | city      |
+-----+---------------+-----------+----------+--------+-----------+
|   1 | Aman Banstola | Manager   | 60500.00 | Male   | Pokhara   |
|   2 | Sita Sharma   | Accountant| 46200.00 | Female | Kathmandu |
|   3 | Ramesh Koirala| Developer | 52800.00 | Male   | Lalitpur  |
|   4 | Priya Adhikari| HR Officer| 44000.00 | Female | Biratnagar|
|   5 | Kiran Thapa   | Designer  | 49500.00 | Male   | Chitwan   |
+-----+---------------+-----------+----------+--------+-----------+
5 rows in set (0.013 sec)
```

Add a new column postal code after city:

```
MariaDB [users]> ALTER TABLE employee ADD COLUMN postal_code INT AFTER city;
Query OK, 0 rows affected (0.028 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> SELECT * FROM employee;
+-----+---------------+------------+----------+--------+-----------+-------------+
| eid | name          | post       | salary   | gender | city      | postal_code |
+-----+---------------+------------+----------+--------+-----------+-------------+
|   1 | Aman Banstola | Manager    | 60500.00 | Male   | Pokhara   | NULL        |
|   2 | Sita Sharma   | Accountant | 46200.00 | Female | Kathmandu | NULL        |
|   3 | Ramesh Koirala| Developer  | 52800.00 | Male   | Lalitpur  | NULL        |
|   4 | Priya Adhikari| HR Officer | 44000.00 | Female | Biratnagar| NULL        |
|   5 | Kiran Thapa   | Designer   | 49500.00 | Male   | Chitwan   | NULL        |
+-----+---------------+------------+----------+--------+-----------+-------------+
5 rows in set (0.002 sec)
```

List out different cities in the table using select query and update the postal code for each city.

```
MariaDB [users]> select distinct city from employee;
+------------+
| city       |
+------------+
| Pokhara    |
| Kathmandu  |
| Lalitpur   |
| Biratnagar |
| Chitwan    |
+------------+
5 rows in set (0.005 sec)
```

```
MariaDB [users]> UPDATE employee SET postal_code=33600 WHERE employee.city="Pokhara";
Query OK, 1 row affected (0.022 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [users]> SELECT * FROM employee;
+-----+--------------+------------+----------+--------+-----------+-------------+
| eid | name         | post       | salary   | gender | city      | postal_code |
+-----+--------------+------------+----------+--------+-----------+-------------+
|   1 | Aman Banstola | Manager    | 60500.00 | Male   | Pokhara   | 33600       |
|   2 | Sita Sharma   | Accountant | 46200.00 | Female | Kathmandu | NULL        |
|   3 | Ramesh Koirala| Developer  | 52800.00 | Male   | Lalitpur  | NULL        |
|   4 | Priya Adhikari| HR Officer | 44000.00 | Female | Biratnagar| NULL        |
|   5 | Kiran Thapa   | Designer   | 49500.00 | Male   | Chitwan   | NULL        |
+-----+--------------+------------+----------+--------+-----------+-------------+
5 rows in set (0.001 sec)
```

Add a * sign after the post of those employees who have salary more than average.

```
MariaDB [users]> UPDATE employee SET post=CONCAT(post,'*') WHERE salary>(SELECT avg(salary) FROM employee);
Query OK, 2 rows affected (0.027 sec)
Rows matched: 2  Changed: 2  Warnings: 0

MariaDB [users]> SELECT * FROM employee;
+-----+--------------+------------+----------+--------+-----------+-------------+
| eid | name         | post       | salary   | gender | city      | postal_code |
+-----+--------------+------------+----------+--------+-----------+-------------+
|   1 | Aman Banstola | Manager*   | 60500.00 | Male   | Pokhara   | 33600       |
|   2 | Sita Sharma   | Accountant | 46200.00 | Female | Kathmandu | NULL        |
|   3 | Ramesh Koirala| Developer* | 52800.00 | Male   | Lalitpur  | NULL        |
|   4 | Priya Adhikari| HR Officer | 44000.00 | Female | Biratnagar| NULL        |
|   5 | Kiran Thapa   | Designer   | 49500.00 | Male   | Chitwan   | NULL        |
+-----+--------------+------------+----------+--------+-----------+-------------+
5 rows in set (0.001 sec)
```

Delete all those records of those employees who are from Pokhara and are male.

```
MariaDB [users]> delete from employee where city='pokhara' and gender='male';
Query OK, 1 row affected (0.005 sec)

MariaDB [users]> select * from employee;
+-----+----------------+------------+----------+--------+------------+-------------+
| eid | name           | post       | salary   | gender | city       | postal_code |
+-----+----------------+------------+----------+--------+------------+-------------+
|   2 | Sita Sharma    | Accountant | 46200.00 | Female | Kathmandu  |        NULL |
|   3 | Ramesh Koirala | Developer* | 52800.00 | Male   | Lalitpur   |        NULL |
|   4 | Priya Adhikari | HR Officer | 44000.00 | Female | Biratnagar |        NULL |
|   5 | Kiran Thapa    | Designer   | 49500.00 | Male   | Chitwan    |        NULL |
+-----+----------------+------------+----------+--------+------------+-------------+
4 rows in set (0.001 sec)
```

Change the table name to ‘emp’

```
MariaDB [users]> alter table employee rename to emp;
Query OK, 0 rows affected (0.030 sec)

MariaDB [users]> select * from emp;
+-----+----------------+------------+----------+--------+------------+-------------+
| eid | name           | post       | salary   | gender | city       | postal_code |
+-----+----------------+------------+----------+--------+------------+-------------+
|   2 | Sita Sharma    | Accountant | 46200.00 | Female | Kathmandu  |        NULL |
|   3 | Ramesh Koirala | Developer* | 52800.00 | Male   | Lalitpur   |        NULL |
|   4 | Priya Adhikari | HR Officer | 44000.00 | Female | Biratnagar |        NULL |
|   5 | Kiran Thapa    | Designer   | 49500.00 | Male   | Chitwan    |        NULL |
+-----+----------------+------------+----------+--------+------------+-------------+
4 rows in set (0.002 sec)
```

Change the name of the column ‘name’ to ‘full_name’

```
MariaDB [users]> alter table emp change full_name fullName varchar(30) not null;
Query OK, 0 rows affected (0.016 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [users]> select fullName from emp;
+----------------+
| fullName       |
+----------------+
| Sita Sharma    |
| Ramesh Koirala |
| Priya Adhikari |
| Kiran Thapa    |
+----------------+
4 rows in set (0.001 sec)
```

4. Conclusion:

In real-world database management, simply creating tables is not enough — ongoing updates, deletions, and alterations are essential for accuracy, relevance, and adaptability.UPDATE ensures that existing records remain current. DELETE keeps the database clean and consistent. ALTER allows structural changes without losing data. Together, these commands empower administrators to maintain both the data and the structure of the database dynamically, ensuring it remains aligned with evolving organizational requirements.

Lab 10

1. Title: To define views in MYSQL

2. Theory: In MySQL, a view is a virtual table that is based on the result of a SQL query. Unlike a physical table, a view does not store data itself but provides a way to represent data from one or more tables. Views are useful for simplifying complex queries, providing data security, and customizing the way users interact with data.

- Definition:
- CREATE VIEW view_name AS
- SELECT columns
- FROM table_name
- WHERE condition;
- Key Features of Views:
    - Simplification: Complex joins or queries can be stored as views and accessed like a table.
    - Security: Views can restrict access to certain columns or rows, showing only relevant data.
    - Data Independence: Users can work with views without needing to know the underlying schema design.
    - Reusability: Views can be reused multiple times in queries without rewriting SQL code.

3. Queries:

creating database:

```
MariaDB [users]> CREATE TABLE employee (
    -> eid INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    -> name VARCHAR(30),
    -> post VARCHAR(30),
    -> salary DECIMAL(10,2),
    -> gender ENUM('Male','Female'),
    -> city VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.041 sec)

MariaDB [users]> INSERT INTO employee (name, post, salary, gender, city) VALUES
    -> ('Aman Banstola', 'Manager', 55000.00, 'Male', 'Pokhara'),
    -> ('Sita Sharma', 'Accountant', 42000.00, 'Female', 'Kathmandu'),
    -> ('Ramesh Koirala', 'Developer', 48000.00, 'Male', 'Lalitpur'),
    -> ('Priya Adhikari', 'HR Officer', 40000.00, 'Female', 'Biratnagar'),
    -> ('Kiran Thapa', 'Designer', 45000.00, 'Male', 'Chitwan');
Query OK, 5 rows affected (0.013 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

View to display employee names and posts only

```
MariaDB [company]> CREATE VIEW emp_basic AS
    -> SELECT name, post
    -> FROM employee;
Query OK, 0 rows affected (0.009 sec)

MariaDB [company]> select * from emp_basic;
+---------------+-------------+
| name          | post        |
+---------------+-------------+
| Aman Banstola | Manager     |
| Sita Sharma   | Accountant  |
| Ramesh Koirala| Developer   |
| Priya Adhikari| HR Officer  |
| Kiran Thapa   | Designer    |
+---------------+-------------+
5 rows in set (0.003 sec)
```

View to display employees with salary greater than 45,000

```
MariaDB [company]> CREATE VIEW high_salary AS
    -> SELECT name, post, salary
    -> FROM employee
    -> WHERE salary > 45000;
Query OK, 0 rows affected (0.016 sec)

MariaDB [company]> select * from high_salary;
+---------------+-----------+----------+
| name          | post      | salary   |
+---------------+-----------+----------+
| Aman Banstola | Manager   | 60000.00 |
| Ramesh Koirala| Developer | 50000.00 |
+---------------+-----------+----------+
2 rows in set (0.002 sec)
```

View to count the number of employees in each city

```
MariaDB [company]> CREATE VIEW city_wise_count AS
    -> SELECT city, COUNT(*) AS total_employees
    -> FROM employee
    -> GROUP BY city;
Query OK, 0 rows affected (0.013 sec)

MariaDB [company]> select * from city_wise_count;
+------------+-----------------+
| city       | total_employees |
+------------+-----------------+
| Biratnagar |               1 |
| Chitwan    |               1 |
| Kathmandu  |               1 |
| Lalitpur   |               1 |
| Pokhara    |               1 |
+------------+-----------------+
5 rows in set (0.002 sec)
```

4. Conclusion: In this lab, we learned how to create and use views in MySQL. Views simplify complex queries, provide controlled access to data, and improve data management. By creating different views (basic details, high salary employees, and city-wise distribution), we saw how views can be applied for reporting and security purposes. This demonstrates the importance of views in making databases more efficient and user-friendly.

S.N	Name of Experiment	Date of Experiment	Date of Submission	Signature
3.	To study and implement different datatypes in MYSQL	2082-04-25	2082-05-12	
4.	To implement the ALTER command in MYSQL.	2082-04-25		
5.	To implement the DROP command in MYSQL.	2082-04-25		
6.	To Query the relational database to retrieve information by using join operators.	2082-04-28		
7.	To create database schema and run SQL queries to retrieve information as instructed. Schema: employee(eid, name, post, salary, dob, gender, address, contact)	2082-04-28		
8.	To create the following database schema and run the SQL queries to retrieve the required information. Schema: employee(eid, name, post, salary, gender, address, contact)	2082-04-29		
9.	To Create the following database schema and run the different UPDATE, DELETE and ALTER queries using MySQL.	2082-04-29		
10.	To define views in MYSQL.	2082-04-29		