Task One

DDL create table script for DutyRosterHistory table is as follows

```
CREATE TABLE IF NOT EXISTS `DutyRosterHistory` (
  `EmployeeID` CHAR(8) NOT NULL,
  `BranchID` INT NOT NULL,
  `WorkingShiftID` INT NOT NULL,
  `WeekStarting` Date,
  PRIMARY KEY (`EmployeeID`, `WorkingShiftID`, `BranchID`, `WeekStarting`),
  CONSTRAINT `fk_DutyRosterHistory_Employee`
    FOREIGN KEY (`EmployeeID`)
    REFERENCES `Employee` (`EmployeeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_DutyRosterHistory_Branch1`
    FOREIGN KEY (`BranchID`)
    REFERENCES `Branch` (`BranchID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_DutyRosterHistory_WorkingShift`
```

```
  FOREIGN KEY (`WorkingShiftID`)
  REFERENCES `WorkingShift` (`WorkingShiftID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);
```

Task Two

```
CREATE  PROCEDURE `updateRoster`(IN employeeIdparam int, IN branchIdparam int)
proc_label:BEGIN
        DECLARE countRost INT DEFAULT 0;
  DECLARE minShiftId INT DEFAULT 0;
  DECLARE notthereId INT DEFAULT 0;
  DECLARE empdutyType VARCHAR(20);
  DECLARE hourCount INT DEFAULT 0;

   SELECT COUNT(*) INTO countRost FROM DutyRoster where EmployeeID = employeeIdparam;

    IF countRost = 0 THEN
      select 'No Roster present for the passed employee' ;
      LEAVE proc_label;
```

```
 END IF;


      select min(workingshiftid)
 into minShiftId
 FROM DutyRoster
where EmployeeID = employeeIdparam;


  select Dutytype
into empdutyType
 from  WorkingShift
where Workingshiftid = minShiftId;


select min(workingshiftid)
      into notthereId
      from WorkingShift
      where WorkingShiftID not in (select workingshiftid
                                   FROM DutyRoster
                                   where EmployeeID =
employeeIdparam)
      and Dutytype = empdutyType;
```

```
   update DutyRoster set workingshiftid = notthereId, branchid = branchIdparam
where EmployeeID = employeeIdparam and workingshiftid = minShiftId;


        select sum(hour(TIMEDIFF(WorkingShiftEndTime, WorkingShiftStartTime)))
   into hourCount
        from WorkingShift, DutyRoster
        where DutyRoster.WorkingShiftID = WorkingShift.WorkingShiftID
        and EmployeeID = 1001;


        if (hourCount > 35)
   then
                select 'Working hour allocated exceeds standard hours of work (35
hours)';
   end if;



   commit;



END
```

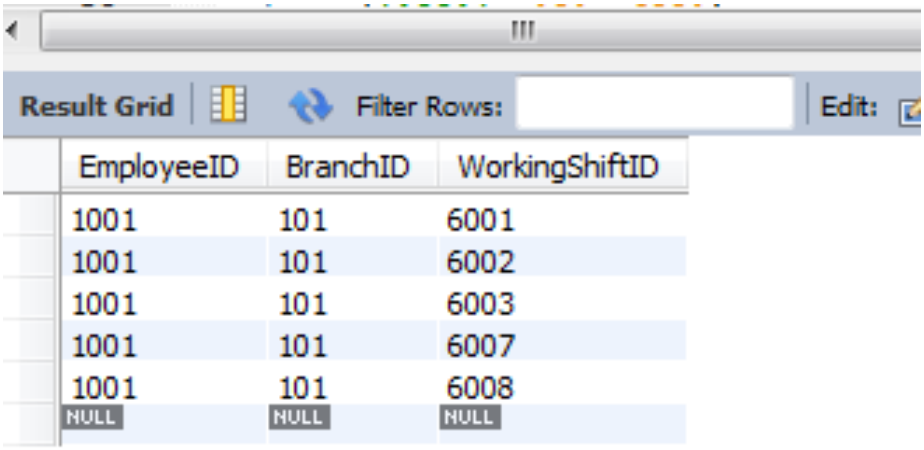
Task Three

```
DELIMITER $$
CREATE TRIGGER before_dutyroster_update
   BEFORE UPDATE ON DutyRoster
   FOR EACH ROW
BEGIN

   DECLARE countWeekend INT DEFAULT 0;

   INSERT INTO DutyRosterHistory
   SET EmployeeID = OLD.EmployeeID,
      BranchID = OLD.BranchID,
                  WorkingShiftID = OLD.WorkingShiftID,
      WeekStarting = date(now() + INTERVAL 1 + weekday(now()) DAY);



END$$
DELIMITER ;
```

Task Four

a)

select * from dutyroster;



Result Grid | Filter Rows: | Edit:

EmployeeID	BranchID	WorkingShiftID
1001	101	6001
1001	101	6002
1001	101	6003
1001	101	6007
1001	101	6008
NULL	NULL	NULL

insert into DutyRoster (EMPLOYEEID, BRANCHID, WORKINGSHIFTID)

values ('1001', 103, 6002);

insert into DutyRoster (EMPLOYEEID, BRANCHID, WORKINGSHIFTID)

values ('1001', 103, 6003);

insert into DutyRoster (EMPLOYEEID, BRANCHID, WORKINGSHIFTID)

values ('1001', 103, 6005);

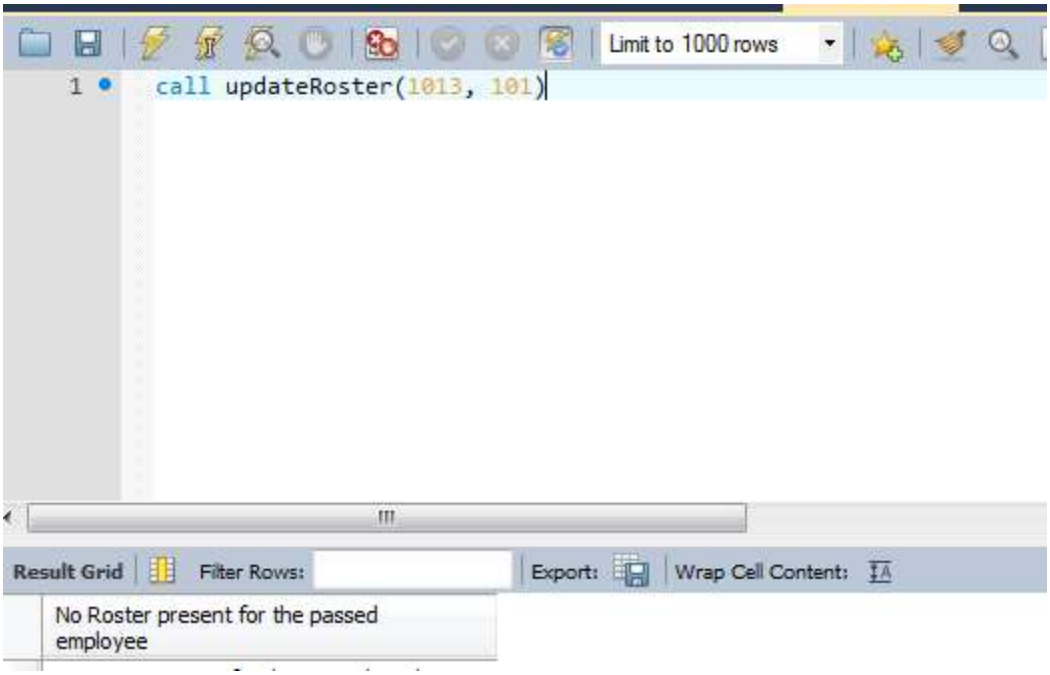
insert into DutyRoster (EMPLOYEEID, BRANCHID, WORKINGSHIFTID)

values ('1001', 103, 6007);

insert into DutyRoster (EMPLOYEEID, BRANCHID, WORKINGSHIFTID)

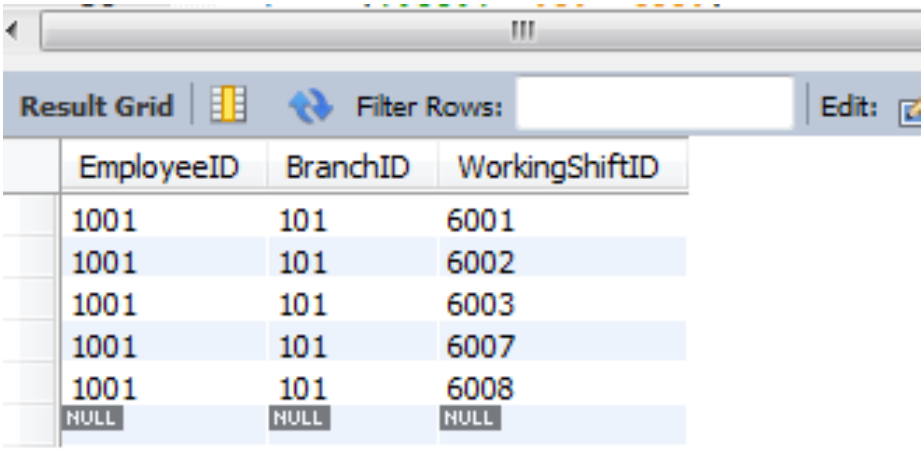
values ('1001', 103, 6001);

b) Update the roster for an employee with Employee ID 1013 to allocate work shifts in Branch 101

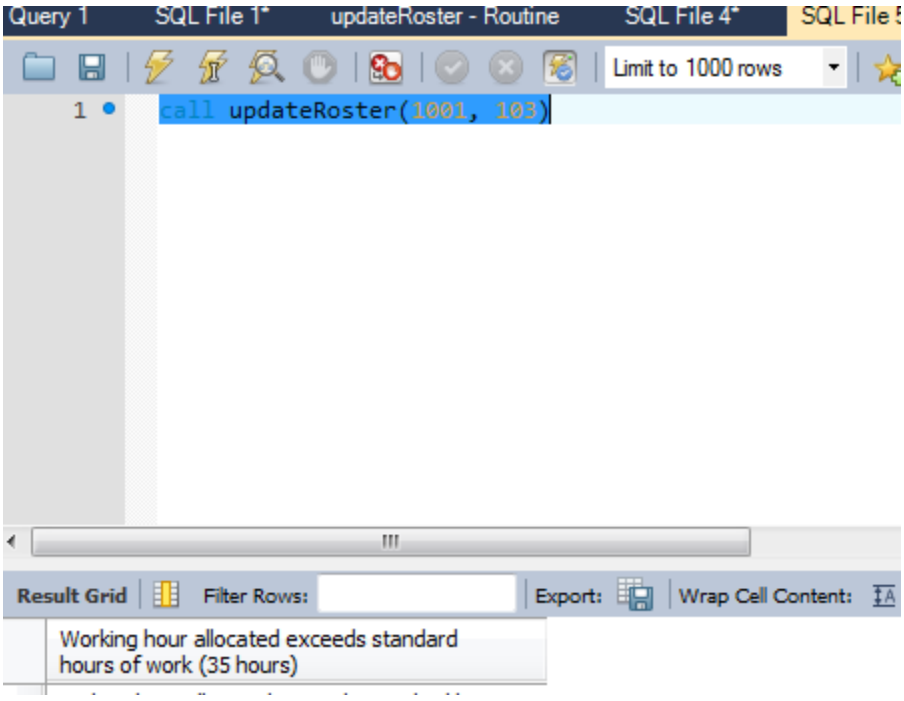


Update the roster for an employee with Employee ID 1001 to allocate work shifts in Branch 103.

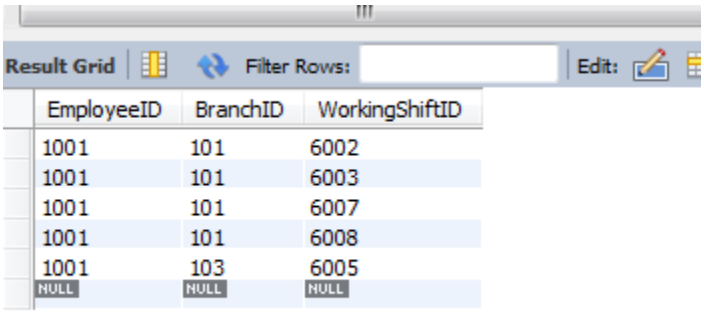
Data in dutyRoster before update



Calling procedure



Data in dutyRoster after update



Task five

```
UPDATE WorkingShift SET dutyType = 'Security'
WHERE WorkingShiftID = 5001;
```

a)

If the following statement would be executed concurrently with the above statement then it could cause a conflict as both the statements would try to acquire exclusive lock on table and would result in deadlock.

```
Delete from  WorkingShift WHERE WorkingShiftID =
5001;
```

b)

The following query would lock the table exclusively and would conflict with the above update statement.

```
UPDATE WorkingShift SET dutyType = 'Security'
```

C)

Deadlock occurs when two statements keeps on waiting for each other to acquire lock for a particular table or object. For example if the statement 1 is waiting for statement 2 to get completed and release the database object it is using whereas statement 2 is waiting for statement 1 to get complete and release the objects statement 1 has locked. This will result in mutual dependency and will keep on waiting for each other and will result in deadlock.

Task six

Since SBH updated the duty roster of its employees every week the database is a write intensive database and hence we need to make sure that the RAID system we use has good write performance. For this reason we should use RAID level 1 which is also known as disk mirroring, this configuration consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage.