BUG MASTER SYSTEM:

standardize the recording of bugs in systems.

Tables:

- Bug Master System is designed to standardize the recording of bugs in systems.
- The system has the following Tables.
  a) Incident Master: This table will be **updated by several roles** as roles will update relative table **trigger** will update the incident master, but **only specific views will be available** to different roles. The incident master has a backup table **using 'SAVEPOINT'**.
  b) Bug Master: The bug master table is designed for the engineer when a bug occurs in the system. This Role will update the Bug ID, System, and ETA in the Bug master table.
  c) Front_Desk: This table is designed for the front_desk who will be handling the customers from the front desk. They will refer the incident ID and enter the customer and status weather the customer is egarly waiting for a solution or agreed to grant some time for resolution this particular input will update the priority of the bug.
  d) Quality Team: Quality Team will update the resolved time total downtime and engineer who worked on the incident.
  e) Engineering Master table will be updated by engineer for related BUG ID and system and short resolution steps or steps of troubleshooting and resolving the bug.
  f) Engineering SME will refer the incident ID and BugID to provide the steps required to remove the BUG from system.

Views: Scematic views for specific user/role:

- Front_Desk_VIEW : Incident_ID, Active Bug, related customer, ETA and standard ETA columns are taken from the incident master table as a view. ETA is expected time of resolution and Standard ETA is average downtime of the bug from history calculated by aggregate functions and group by.
- Engineer_View: Engineer can see BugID and engineer assigned for resolving the particular bug. This role has access for the Knowledge_Base table of the Bug_ID and Resolution.
- Quality Team can see Engineer assigned for active bug and ETA , steps required to resolve particular BUG.

Logic and Queries:

1. Create Database

   Create database Bug_Master;

2. We need to use the same database.

   Use Bug_master;

3. Create _ Master Table to centralize all records

   Create table Incident_Master (Incident_ID varchar(50) Primary key,Bug_id varchar(50),Resolved_Timing timestamp, Invoked_Time timestamp, System_ID varchar(50), Live_Status varchar(30),Engineer_ID varchar(30),ETA int, Customer_ID Varchar(40),Customer_Status varchar(40));

4. Create Table System_Status for engineers

   Create table System_Status (Bug_id varchar(50) ,

   System_ID varchar(50), Live_Status varchar(30),Invoked_Time timestamp);

   create table Quality_Team (Incident_ID varchar(30), Resolved_time timestamp ,Engineer varchar(30));

5. Create table front_Office

   create table front_office (Incident_ID varchar(30), Customer varchar(40),Customer_ID varchar(40),Customer_status varchar(40));

6. Create table Engineering_Dept

   Create table engineeringDept(Engineer_Name varchar(30) , ID varchar(30), System_Speciality varchar(30));

7. Create table systems

   Create table systems (SystemID varchar(30), Bug_ID varchar(30)) ;

8. Create trigger for updating master table from Bug_Master : Bug_Insert_Master_Table

   DELIMITER //

   CREATE TRIGGER Bug_Insert_Master_Table AFTER INSERT ON System_Status1

   FOR EACH ROW

   BEGIN

```
    INSERT INTO Incident_Master1 ( Bug_id, Invoked_Time, System_ID, Live_Status,date)

    VALUES ( NEW.Bug_id, NEW.Invoked_Time, NEW.System_ID, NEW.Live_Status,
NEW.date);

    END //

    DELIMITER ;
```

9. Trigger to generate a Incident_ID in incident master once the bug_ID is entered in the bug master table.

```
    DELIMITER //

    CREATE TRIGGER before_insert_incident_master1

    BEFORE INSERT ON Incident_Master1

    FOR EACH ROW

    BEGIN

    SET NEW.Incident_ID = CONCAT(DATE_FORMAT(NEW.Invoked_Time, '%Y_%m_%d'),'_',
NEW.Bug_ID,'_', NEW.System_ID);

    END //

    DELIMITER ;
```

10. Trigger to update the incident master table once the front desk table column ‘Customer_ID’,’Customer_Status’ is updated in front desk table.

```
    DELIMITER //

    CREATE TRIGGER Front_Office_Update

    AFTER INSERT ON front_office

    FOR EACH ROW

    BEGIN

      UPDATE Incident_Master

      SET Customer_ID = NEW.Customer_ID, Customer_Status =new.Customer_Status

      WHERE Incident_ID = NEW.Incident_ID;

    END//


    Delimiter ;
```

11. Trigger to update the incident master table once the Quality table column ‘Customer_ID’,’ Engineer_ID’ , Resolved_Timing is updated in Quality table.

```
DELIMITER //
CREATE TRIGGER Quality_Team_Update
AFTER INSERT ON Quality_Team
FOR EACH ROW
BEGIN
  UPDATE Incident_Master
  SET Engineer_ID = NEW.Engineer_ID , Resolved_Timing = new.Resolved_Timing
  WHERE Incident_ID = NEW.Incident_ID;
END//

Delimiter ;
```

12. BY Given BUG ID we can check Avg downtime we can check standard Downtime.

```
SELECT BugID, AVG(Downtime) AS AverageDowntime
FROM BugDowntime
GROUP BY BugID;
```

13. This procedure will return average downtime for a given bugID

```
DELIMITER //
CREATE PROCEDURE GetAverageDowntimeForBug(IN P_BugID INT, OUT
P_AverageDowntime DECIMAL(10,2))
BEGIN
  SELECT AVG(Downtime) INTO P_AverageDowntime
  FROM BugDowntime
  WHERE BugID = P_BugID;
END //

DELIMITER ;
```

14. This CTE will return top 5 bugs which has highest downtime.

```
WITH Bug_Downtime_Ranked AS (
  SELECT
    BugID,
    Downtime,
    ROW_NUMBER() OVER (ORDER BY Downtime DESC) AS RowNum
  FROM
    BugDowntime
)
SELECT BugID, Downtime FROM BugDowntimeRanked WHERE RowNum <= 5;
```

15. TOP bugs which has highest downtime from each system.

```
WITH System_Total_Downtime_Ranked AS (
  SELECT
    System_ID,
    SUM(Downtime) AS TotalDowntime,
    ROW_NUMBER() OVER (ORDER BY SUM(Downtime) DESC) AS RowNum
  FROM
    SystemBugs
  GROUP BY
    SystemID
)
SELECT
  SystemID,
  TotalDowntime
FROM
  SystemTotalDowntimeRanked
WHERE
  RowNum <= 5;
```

16. Views for each role

```
CREATE VIEW frontdesk AS
SELECT Incident_ID, Bug_ID, ETA, Live_Status
FROM Incident_Master
WHERE Live_Status = 'Active';

CREATE VIEW Quality  AS
SELECT Incident_ID, Bug_ID, ETA, Live_Status, Engineer_ID,Engineer
FROM Incident_Master
WHERE Live_Status = 'Active';

CREATE VIEW Engineer  AS
SELECT Incident_ID, Bug_ID, ETA, Live_Status
FROM Incident_Master
WHERE Live_Status = 'Active';

CREATE VIEW Master_View  AS
select * from Incident_Master;
```

17. Lets create a backup for everyday @ specified time.

```
CREATE TABLE Daily_Backup AS SELECT * FROM Incident_Master WHERE 1=0;


DELIMITER //


CREATE PROCEDURE BackupMyTable()
BEGIN
   INSERT INTO Daily_Backup SELECT * FROM Incident_Master;
END //


DELIMITER ;


CREATE EVENT daily_backup
ON SCHEDULE EVERY 1 DAY
STARTS TIMESTAMP(CURRENT_DATE, '00:00:00')
DO
CALL BackupMyTable();




SET GLOBAL event_scheduler = ON;
```