Triggers

1. **Ensure Detail Available Before Insert**

```
DELIMITER //

CREATE TRIGGER ensureDetailAvailableBeforeInsert

BEFORE INSERT ON Cars

FOR EACH ROW

BEGIN

  IF NOT EXISTS (

    SELECT *

    FROM Details

    WHERE make = NEW.make

      AND model = NEW.model

      AND year = NEW.year

  ) THEN

    INSERT INTO Details (make, model, year)

    VALUES (NEW.make, NEW.model, NEW.year);

  END IF;

END//
```

```
DELIMITER ;
```

2. **Ensure Detail Available Before Update**

```
DELIMITER //
CREATE TRIGGER ensureDetailAvailableBeforeUpdate
BEFORE UPDATE ON Cars
FOR EACH ROW
BEGIN
  IF NOT EXISTS (
    SELECT *
    FROM Details
    WHERE make = NEW.make
      AND model = NEW.model
      AND year = NEW.year
  ) THEN
    INSERT INTO Details (make, model, year)
    VALUES (NEW.make, NEW.model, NEW.year);
  END IF;
END//

DELIMITER ;
```

3. Enforce Positive Mileage and Price Before Inserts

Ensure that mileage and price in the Cars table are positive values before inserts.

```
DELIMITER //

CREATE TRIGGER EnforcePositiveValues
BEFORE INSERT ON Cars
FOR EACH ROW
BEGIN
   DECLARE MESSAGE_TEXT VARCHAR(60);

   IF NEW.mileage < 0 THEN
      SIGNAL SQLSTATE '45000';
      SET MESSAGE_TEXT = 'Mileage cannot be negative.';
   END IF;

   IF NEW.price < 0 THEN
      SIGNAL SQLSTATE '45000';
      SET MESSAGE_TEXT = 'Price cannot be negative.';
   END IF;
END;

//

DELIMITER ;
```

4. **Enforce Positive Mileage and Price Before Updates**

```
DELIMITER //

CREATE TRIGGER EnforcePositiveValues_Update
BEFORE UPDATE ON Cars
FOR EACH ROW
BEGIN
    DECLARE MESSAGE_TEXT VARCHAR(60);

    IF NEW.mileage < 0 THEN
        SIGNAL SQLSTATE '45000';
        SET MESSAGE_TEXT = 'Mileage cannot be negative.';
    END IF;

    IF NEW.price < 0 THEN
        SIGNAL SQLSTATE '45000';
        SET MESSAGE_TEXT = 'Price cannot be negative.';
    END IF;
END;

//

DELIMITER ;
```

TRANSACTION

Adjust car prices based on their demand and features. If a car is highly in demand based on sales trends and has good features, its price is increased by a user specified percentage e.g 10%. If a car has been sitting in inventory for too long based on status and mileage, its price is reduced by user specified percentage e.g 15%.
We used a serializable isolation level in this database transaction to ensure a strict consistency. It ensures that the results of concurrent transactions are the same as if the transactions were executed sequentially, rather than in parallel.

```
DELIMITER //

CREATE PROCEDURE AdjustCarPrices(
    IN increase DECIMAL(10, 2),
    IN decrease DECIMAL(10, 2)
)
BEGIN
    DECLARE totalNotSold INT;
    DECLARE salesThreshold INT DEFAULT 5;
    DECLARE featureThreshold DECIMAL(10, 2) DEFAULT 0.7;
    DECLARE ratingThreshold INT DEFAULT 4;

    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

    START TRANSACTION;

    SELECT COUNT(*) INTO totalNotSold
    FROM Cars
    WHERE status = 'available';

    UPDATE Cars
    SET price = price * (1 + (increase / 100))
    WHERE VIN IN (
        SELECT VIN
        FROM (
            SELECT c.VIN,
                   (COUNT(c.VIN) / temp.total) AS SalesTrendScore,
                   ((CASE WHEN c.mileage < 150000 THEN 1 ELSE 0 END +
                     CASE WHEN c.year > 2003 THEN 1 ELSE 0 END) / 2) AS FeatureScore
```

```
      FROM Cars c
      JOIN (SELECT make, model, year, COUNT(VIN) AS total
         FROM Cars
         GROUP BY make, model, year) AS temp
      ON (c.make = temp.make AND c.model = temp.model AND c.year =
temp.year)
      WHERE c.status != 'available'
      GROUP BY c.VIN
      HAVING SalesTrendScore > salesThreshold AND FeatureScore >
featureThreshold
    ) AS SubQuery
  );

  UPDATE Cars
  SET price = price * (1 - (decrease / 100))
  WHERE status = 'available' AND mileage > 200000;

UPDATE Cars c
JOIN (
  SELECT c.VIN
  FROM Cars c
  NATURAL JOIN Reviews r
  GROUP BY c.VIN
  HAVING AVG(r.rating) >= ratingThreshold AND COUNT(r.rating) >= 5
) AS qualified_cars
ON c.VIN = qualified_cars.VIN
SET c.price = c.price * (1 + (increase / 200));

  IF ROW_COUNT() > 0 THEN
    COMMIT;
    SELECT 'Price adjustments committed successfully.' AS Message;
  ELSE
    ROLLBACK;
    SELECT 'No price adjustments made.' AS Message;
  END IF;
END //

DELIMITER ;
```

PROCEDURE: Vehicle Display Optimize

The Vehicle Display Optimizer selects the best cars for display in the showroom/yard of the dealership. For each car in our inventory, we calculate a Display Score based on these factors

For each car in our inventory, we calculate a Display Score based on these factors

DisplayScore = (w1 * Customer_Preference_Match) +

(w2 * Vehicle_Feature_Score) +

(w3 * Sales_Trend_Score) +

(w4 * Inventory_Score)

w1, w2, w3, w4 are the weights assigned to each factor (which can be adjusted based on the priority of each factor).

Customer_Preference_Match: How well the car matches typical customer preferences for price, type, manufacturer, etc.

Vehicle_Feature_Score: A score based on important vehicle features (condition, mileage, price).

Sales_Trend_Score: A score based on recent sales data.

Inventory_Score: A score based on stock availability and whether the car is newly added.

```
 DELIMITER //

CREATE PROCEDURE VehicleDisplayOptimizer(
   IN weight1 DECIMAL(5, 4), -- Weight for Customer_Preference_Match
   IN weight2 DECIMAL(5, 4), -- Weight for Vehicle_Feature_Score
   IN weight3 DECIMAL(5, 4), -- Weight for Sales_Trend_Score
   IN weight4 DECIMAL(5, 4), -- Weight for Inventory_Score
   IN higher_input_price INT,
   IN higher_input_mileage INT,
   IN input_transmission VARCHAR(50),
   IN input_driveWheel VARCHAR(50)
```

```
)
BEGIN
    DECLARE totalWeight DECIMAL(5, 4);
    SET totalWeight = weight1 + weight2 + weight3 + weight4;

    -- Ensure that weights do not exceed 1
    IF totalWeight > 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The sum of weights exceeds 1. Please adjust the weights.';
    END IF;

     DROP TEMPORARY TABLE IF EXISTS TempDisplayScores;
    -- Create a temporary table to hold display scores
    CREATE TEMPORARY TABLE TempDisplayScores (
        VIN VARCHAR(17) PRIMARY KEY,
        totalScore DECIMAL(10, 2)
    );

    -- Insert calculated scores into the temporary table

    INSERT INTO TempDisplayScores (VIN, totalScore)
SELECT  c.VIN, (weight1 * customer.Customer_Preference_Score + weight2 *
feature.Vehicle_Feature_Score + weight3 * sale.Sales_Trend_Score + weight4
*inventory.Inventory_Score) AS totalScore
FROM Cars c NATURAL JOIN



  -- Customer Preference Match
        (SELECT c.VIN,  SUM((c.price < higher_input_price) * 0.25 + (c.mileage <
higher_input_mileage) * 0.25 + (d.transmission = input_transmission) * 0.25 + (d.driveWheel =
input_driveWheel) * 0.25) AS Customer_Preference_Score
FROM Cars c JOIN Details d ON c.make = d.make AND c.model = d.model AND c.year =
d.year
GROUP BY c.VIN) AS customer NATURAL JOIN

        -- Vehicle Feature Score
            (SELECT c.VIN, ( (c.mileage < 150000) + (c.year > 2003) + (AVG(r.rating) >= 4) ) / 3.0
            AS Vehicle_Feature_Score
            FROM Cars c
            NATURAL JOIN Reviews r
            GROUP BY c.VIN) AS feature NATURAL JOIN

        -- Sales Trend Score
        (SELECT c.VIN, temp2.Sales_Trend_Score
            FROM Cars c NATURAL JOIN
            (SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score
            FROM Cars c JOIN
            (SELECT c.make, c.model, c.year, COUNT(c.VIN) AS total
            FROM Cars c GROUP BY c.make, c.model, c.year) AS temp
            ON (c.make = temp.make AND c.model = temp.model AND c.year = temp.year)
            WHERE c.status != 'available'
            GROUP BY c.make, c.model, c.year) AS temp2) AS sale NATURAL JOIN

        -- Inventory Score
        (SELECT c.VIN, temp2.Inventory_Score
            FROM Cars c NATURAL JOIN

            (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS
            Inventory_Score
            FROM Cars c,
            (SELECT COUNT(*) AS total_not_sold
            FROM Cars c
            WHERE c.status = 'available') AS temp
            WHERE c.status = 'available'
            GROUP BY c.make, c.model, c.year) AS temp2) AS inventory;

END //

DELIMITER ;
```