***APEX CODES***

**VehicleOrderTriggerHandler**

Code-

public class VehicleOrderTriggerHandler {

public static void handleTrigger(List<Vehicle\_Order\_\_c> newOrders, Map<Id, Vehicle\_Order\_\_c> oldOrders,

Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate) {

if (isBefore) {

if (isInsert || isUpdate) {

preventOrderIfOutOfStock(newOrders);

validateOrderData(newOrders);

}

}

if (isAfter) {

if (isInsert || isUpdate) {

updateStockOnOrderPlacement(newOrders, oldOrders, isUpdate);

}

}

}

private static void preventOrderIfOutOfStock(List<Vehicle\_Order\_\_c> orders) {

Set<Id> vehicleIds = new Set<Id>();

for (Vehicle\_Order\_\_c order : orders) {

if (order.Vehicle\_\_c != null) {

vehicleIds.add(order.Vehicle\_\_c);

}

}

if (!vehicleIds.isEmpty()) {

Map<Id, Vehicle\_\_c> vehicleMap = new Map<Id, Vehicle\_\_c>();

for (Vehicle\_\_c vehicle : [SELECT Id, Name FROM Vehicle\_\_c WHERE Id IN :vehicleIds]) {

vehicleMap.put(vehicle.Id, vehicle);

}

for (Vehicle\_Order\_\_c order : orders) {

if (order.Vehicle\_\_c != null && !vehicleMap.containsKey(order.Vehicle\_\_c)) {

order.addError('Selected vehicle does not exist.');

}

}

}

}

private static void validateOrderData(List<Vehicle\_Order\_\_c> orders) {

for (Vehicle\_Order\_\_c order : orders) {

if (order.Vehicle\_\_c == null) {

order.addError('Please select a vehicle for this order.');

}

if (String.isBlank(order.Status\_\_c)) {

order.Status\_\_c = 'Pending';

}

}

}

private static void updateStockOnOrderPlacement(List<Vehicle\_Order\_\_c> newOrders,

Map<Id, Vehicle\_Order\_\_c> oldOrders,

Boolean isUpdate) {

List<Id> confirmedOrderIds = new List<Id>();

for (Vehicle\_Order\_\_c order : newOrders) {

if (order.Status\_\_c == 'Confirmed') {

confirmedOrderIds.add(order.Id);

}

}

if (!confirmedOrderIds.isEmpty()) {

System.debug('Orders confirmed: ' + confirmedOrderIds.size());

}

}

}

**VehicleOrderTrigger**

Code-

trigger VehicleOrderTrigger on Vehicle\_Order\_\_c (before insert, before update, after insert, after update) {

VehicleOrderTriggerHandler.handleTrigger(

Trigger.new,

Trigger.oldMap,

Trigger.isBefore,

Trigger.isAfter,

Trigger.isInsert,

Trigger.isUpdate

);

}

**VehicleOrderBatch**

Code-

global class VehicleOrderBatch implements Database.Batchable<sObject> {

global Database.QueryLocator start(Database.BatchableContext bc) {

return Database.getQueryLocator([

SELECT Id, Status\_\_c, Vehicle\_\_c

FROM Vehicle\_Order\_\_c

WHERE Status\_\_c = 'Pending'

]);

}

global void execute(Database.BatchableContext bc, List<Vehicle\_Order\_\_c> orderList) {

Set<Id> vehicleIds = new Set<Id>();

// Collect vehicle IDs

for (Vehicle\_Order\_\_c order : orderList) {

if (order.Vehicle\_\_c != null) {

vehicleIds.add(order.Vehicle\_\_c);

}

}

if (!vehicleIds.isEmpty()) {

// Get vehicle information (simplified - no stock fields for now)

Map<Id, Vehicle\_\_c> vehicleMap = new Map<Id, Vehicle\_\_c>();

for (Vehicle\_\_c vehicle : [SELECT Id, Name FROM Vehicle\_\_c WHERE Id IN :vehicleIds]) {

vehicleMap.put(vehicle.Id, vehicle);

}

List<Vehicle\_Order\_\_c> ordersToUpdate = new List<Vehicle\_Order\_\_c>();

// Process each order

for (Vehicle\_Order\_\_c order : orderList) {

if (vehicleMap.containsKey(order.Vehicle\_\_c)) {

// Simply confirm the order (enhance this logic as needed)

order.Status\_\_c = 'Confirmed';

ordersToUpdate.add(order);

}

}

// Update orders

if (!ordersToUpdate.isEmpty()) {

try {

update ordersToUpdate;

System.debug('Updated ' + ordersToUpdate.size() + ' orders to Confirmed status.');

} catch (Exception e) {

System.debug('Error updating orders: ' + e.getMessage());

}

}

}

}

global void finish(Database.BatchableContext bc) {

System.debug('Vehicle order batch job completed successfully.');

// Get job details

AsyncApexJob job = [SELECT Id, Status, JobItemsProcessed, TotalJobItems, NumberOfErrors

FROM AsyncApexJob

WHERE Id = :bc.getJobId()];

System.debug('Job Status: ' + job.Status);

System.debug('Job Items Processed: ' + job.JobItemsProcessed + '/' + job.TotalJobItems);

System.debug('Errors: ' + job.NumberOfErrors);

}

}

**VehicleOrderBatchScheduler**

Code-

global class VehicleOrderBatchScheduler implements Schedulable {

global void execute(SchedulableContext sc) {

VehicleOrderBatch batchJob = new VehicleOrderBatch();

Database.executeBatch(batchJob, 50);

}

}

**Schedule code in the Execute Anonymous Window (Debug)**

Code-

String cronExp = '0 0 12 \* \* ?'; // every day at 12 PM

System.schedule('Daily Vehicle Order Processing', cronExp, new VehicleOrderBatchScheduler());