**VehicleOrderTriggerHandler:-**

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 && (isInsert || isUpdate)) {

            preventOrderIfOutOfStock(newOrders);

        }

        if (isAfter && (isInsert || isUpdate)) {

            updateStockOnOrderPlacement(newOrders);

        }

    }

    // ❌ Prevent placing an order if stock is zero

    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> vehicleStockMap = new Map<Id, Vehicle\_\_c>(

                [SELECT Id, Stock\_Quantity\_\_c FROM Vehicle\_\_c WHERE Id IN :vehicleIds]

            );

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

                Vehicle\_\_c vehicle = vehicleStockMap.get(order.Vehicle\_\_c);

                if (vehicle != null && vehicle.Stock\_Quantity\_\_c <= 0) {

                    order.addError('This vehicle is out of stock. Order cannot be placed.');

                }

            }

        }

    }

    // ✅ Decrease stock when an order is confirmed

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

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

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

            if (order.Vehicle\_\_c != null && order.Status\_\_c == 'Confirmed') {

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

            }

        }

        if (!vehicleIds.isEmpty()) {

            Map<Id, Vehicle\_\_c> vehicleStockMap = new Map<Id, Vehicle\_\_c>(

                [SELECT Id, Stock\_Quantity\_\_c FROM Vehicle\_\_c WHERE Id IN :vehicleIds]

            );

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

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

                Vehicle\_\_c vehicle = vehicleStockMap.get(order.Vehicle\_\_c);

                if (vehicle != null && vehicle.Stock\_Quantity\_\_c > 0) {

                    vehicle.Stock\_Quantity\_\_c -= 1;

                    vehiclesToUpdate.add(vehicle);

                }

            }

            if (!vehiclesToUpdate.isEmpty()) {

                update vehiclesToUpdate;

            }

        }

    }

}

**VehicleOrderTrigger:-**  
  
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:-**

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>();

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

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

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

            }

        }

        if (!vehicleIds.isEmpty()) {

            Map<Id, Vehicle\_\_c> vehicleStockMap = new Map<Id, Vehicle\_\_c>(

                [SELECT Id, Stock\_Quantity\_\_c FROM Vehicle\_\_c WHERE Id IN :vehicleIds]

            );

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

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

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

                Vehicle\_\_c vehicle = vehicleStockMap.get(order.Vehicle\_\_c);

                if (vehicle != null && vehicle.Stock\_Quantity\_\_c > 0) {

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

                    vehicle.Stock\_Quantity\_\_c -= 1;

                    ordersToUpdate.add(order);

                    vehiclesToUpdate.add(vehicle);

                }

            }

            if (!ordersToUpdate.isEmpty()) update ordersToUpdate;

            if (!vehiclesToUpdate.isEmpty()) update vehiclesToUpdate;

        }

    }

    global void finish(Database.BatchableContext bc) {

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

    }

}

**VehicleOrderBatchScheduler:-**

global class VehicleOrderBatchScheduler implements Schedulable {  
    global void execute(SchedulableContext sc) {  
        VehicleOrderBatch batchJob = new VehicleOrderBatch();  
        Database.executeBatch(batchJob, 50); // 50 = batch size } }