# IE2042 - Database Management System for Security [2023/Feb].

Sri Lanka Institute of Information Technology.

Year 2, Semester 1

| IT numbers | Name |
|---|---|
| IT21826368 | Nanayakkara Y.D.T. D |
| IT21822612 | Mendis H.R.M |
| IT21831904 | Weerasinghe K.M |
| IT21828348 | Dissanayake K.D.A.R. A |

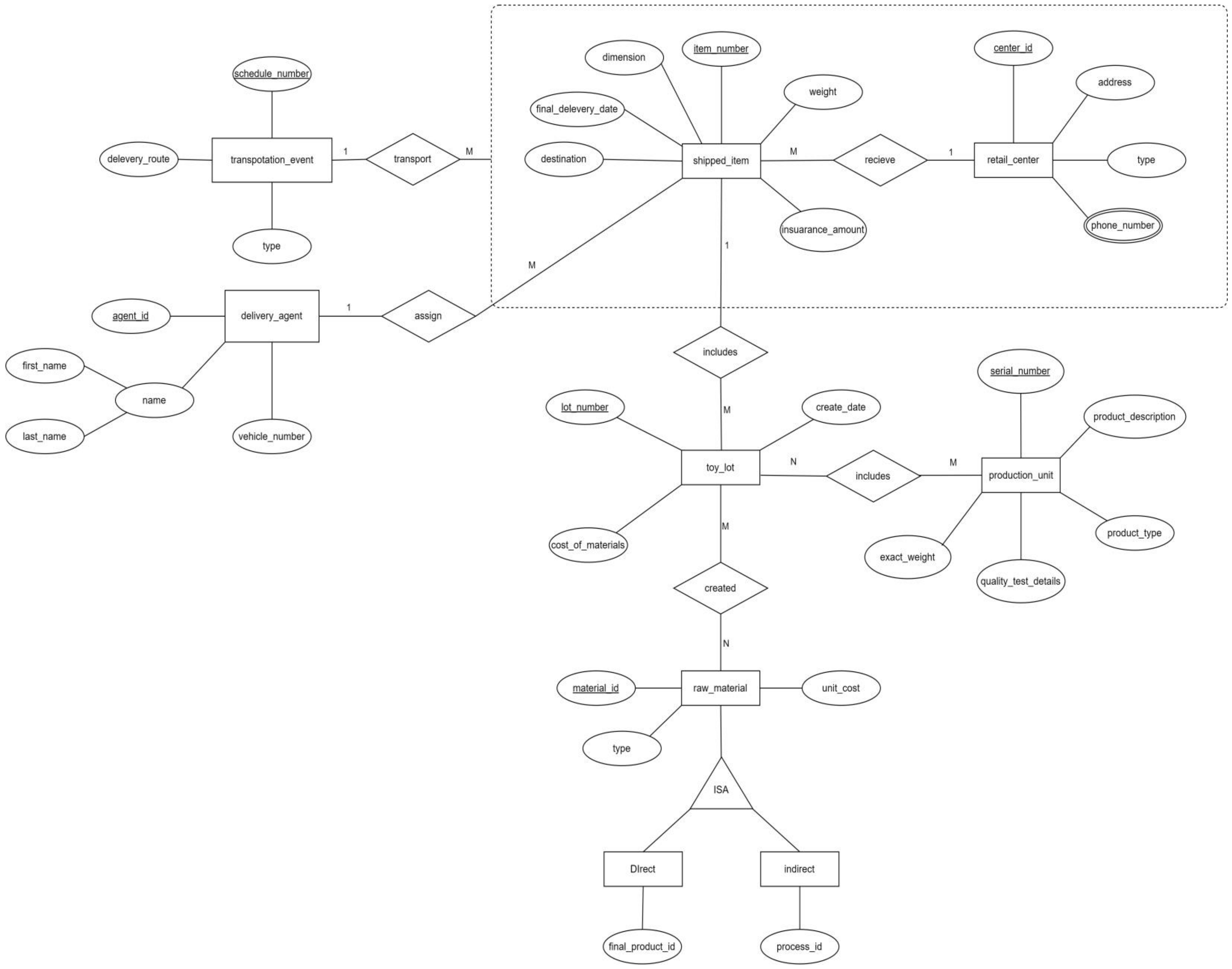
# Contents

# PART 1

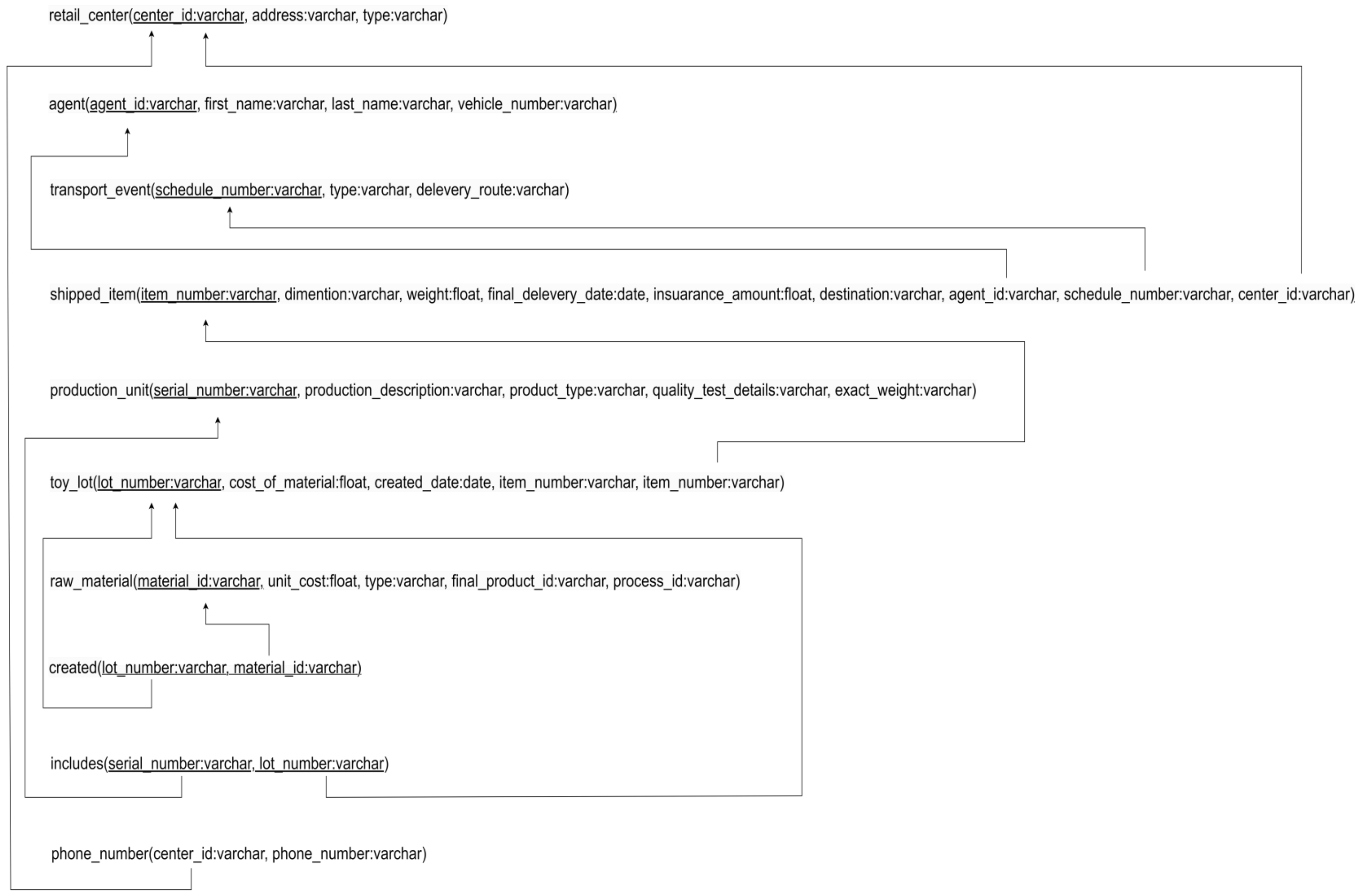
## 1. Assumptions

- TRANSPORT_EVENT is TRANSPORT SHIPPED_ITEMS that received to RETAIL_CENTER that we used aggregation modeling technique.
- One SHIPPED ITEM INCLUDES Many TOY_LOTS Therefore it is a One-to-Many relationship.
- Each SHIPPED ITEM consists of MULTIPLE TOY lots therefore its 1:M relationship.
- Many TOY LOT includes many PRODUCTION UNIT therefore it is a N:M relationship.
- Because there are two types of RAW MATERIALS ISA Relationship is used in ER.
- We used option 3 for ISA Relationship when logical model mapping because subclasses are disjoint and have less attributes.

## 2. Entity Relationship Diagram



## 3. Logic model



retail_center(center_id:varchar, address:varchar, type:varchar)

agent(agent_id:varchar, first_name:varchar, last_name:varchar, vehicle_number:varchar)

transport_event(schedule_number:varchar, type:varchar, delevery_route:varchar)

shipped_item(item_number:varchar, dimention:varchar, weight:float, final_delevery_date:date, insuarance_amount:float, destination:varchar, agent_id:varchar, schedule_number:varchar, center_id:varchar)

production_unit(serial_number:varchar, production_description:varchar, product_type:varchar, quality_test_details:varchar, exact_weight:varchar)

toy_lot(lot_number:varchar, cost_of_material:float, created_date:date, item_number:varchar, item_number:varchar)

raw_material(material_id:varchar, unit_cost:float, type:varchar, final_product_id:varchar, process_id:varchar)

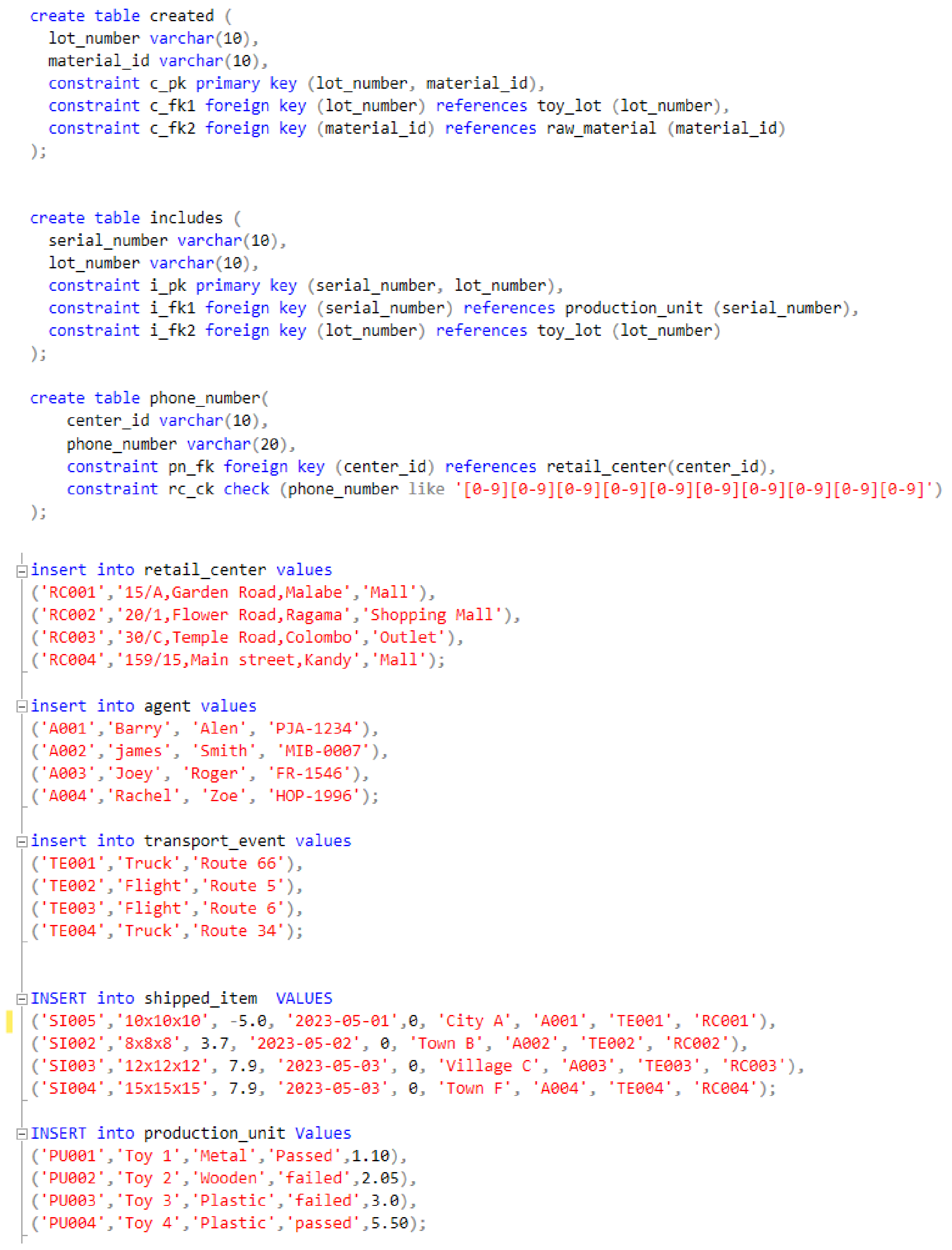
created(lot_number:varchar, material_id:varchar)

includes(serial_number:varchar, lot_number:varchar)

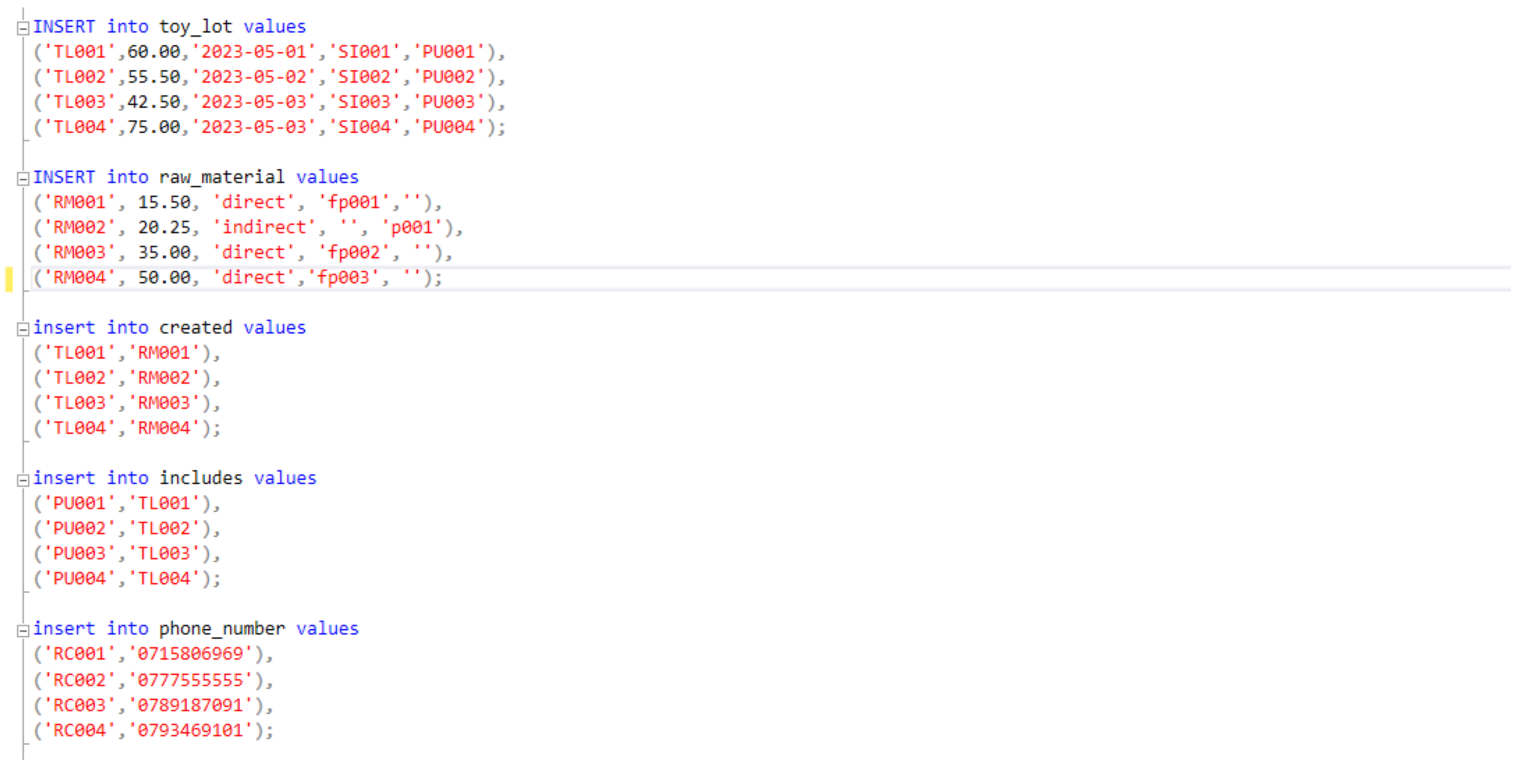
phone_number(center_id:varchar, phone_number:varchar)

## 4. SQL codes and sample data

```
create database ups;
use ups;

create table retail_center (
  center_id varchar(10),
  address varchar(100),
  type varchar(50),
  constraint rc_pk primary key(center_id)
);

create table agent (
  agent_id varchar(10),
  first_name varchar(50),
  last_name varchar(50),
  vehicle_number varchar(20),
  constraint a_pk primary key (agent_id),
  constraint a_ck check (vehicle_number like '[a-z][a-z]-[0-9][0-9][0-9][0-9]' or vehicle_number like '[a-z][a-z][a-z]-[0-9][0-9][0-9][0-9]')
);

create table transport_event (
  schedule_number varchar(10),
  type varchar(50),
  delivery_route varchar(100),
  constraint te_pk primary key (schedule_number)
);


create table shipped_item (
  item_number varchar(10),
  dimension varchar(20),
  weight float,
  final_delivery_date date,
  insurance_amount float,
  destination varchar(100),
  agent_id varchar(10),
  schedule_number varchar(10),
  center_id varchar(10),
  constraint si_pk primary key (item_number),
  constraint si_fk1 foreign key (agent_id) references agent (agent_id),
  constraint si_fk2 foreign key (center_id) references retail_center (center_id),
  constraint si_fk3 foreign key (schedule_number) references transport_event (schedule_number)
);

create table production_unit (
  serial_number varchar(10),
  production_description varchar(100),
  product_type varchar(50),
  quality_test_details varchar(100),
  exact_weight float,
  constraint pu_pk primary key (serial_number)
);

create table toy_lot (
  lot_number varchar(10),
  cost_of_material float,
  created_date date,
  item_number varchar(10),
  serial_number varchar(10),
  constraint tl_pk primary key (lot_number),
  constraint tl_fk foreign key (item_number) references shipped_item (item_number)
);

create table raw_material (
  material_id varchar(10),
  unit_cost float,
  type varchar(50),
  product_id varchar(10),
  process_id varchar(10),
  constraint rm_pk primary key (material_id),
  constraint rm_ck check (type in ('direct','indirect'))
);
```



```
create table created (
  lot_number varchar(10),
  material_id varchar(10),
  constraint c_pk primary key (lot_number, material_id),
  constraint c_fk1 foreign key (lot_number) references toy_lot (lot_number),
  constraint c_fk2 foreign key (material_id) references raw_material (material_id)
);


create table includes (
  serial_number varchar(10),
  lot_number varchar(10),
  constraint i_pk primary key (serial_number, lot_number),
  constraint i_fk1 foreign key (serial_number) references production_unit (serial_number),
  constraint i_fk2 foreign key (lot_number) references toy_lot (lot_number)
);

create table phone_number(
    center_id varchar(10),
    phone_number varchar(20),
    constraint pn_fk foreign key (center_id) references retail_center(center_id),
    constraint rc_ck check (phone_number like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
);

insert into retail_center values
('RC001','15/A,Garden Road,Malabe','Mall'),
('RC002','20/1,Flower Road,Ragama','Shopping Mall'),
('RC003','30/C,Temple Road,Colombo','Outlet'),
('RC004','159/15,Main street,Kandy','Mall');

insert into agent values
('A001','Barry', 'Alen', 'PJA-1234'),
('A002','james', 'Smith', 'MIB-0007'),
('A003','Joey', 'Roger', 'FR-1546'),
('A004','Rachel', 'Zoe', 'HOP-1996');

insert into transport_event values
('TE001','Truck','Route 66'),
('TE002','Flight','Route 5'),
('TE003','Flight','Route 6'),
('TE004','Truck','Route 34');


INSERT into shipped_item  VALUES
('SI005','10x10x10', -5.0, '2023-05-01',0, 'City A', 'A001', 'TE001', 'RC001'),
('SI002','8x8x8', 3.7, '2023-05-02', 0, 'Town B', 'A002', 'TE002', 'RC002'),
('SI003','12x12x12', 7.9, '2023-05-03', 0, 'Village C', 'A003', 'TE003', 'RC003'),
('SI004','15x15x15', 7.9, '2023-05-03', 0, 'Town F', 'A004', 'TE004', 'RC004');

INSERT into production_unit Values
('PU001','Toy 1','Metal','Passed',1.10),
('PU002','Toy 2','Wooden','failed',2.05),
('PU003','Toy 3','Plastic','failed',3.0),
('PU004','Toy 4','Plastic','passed',5.50);
```



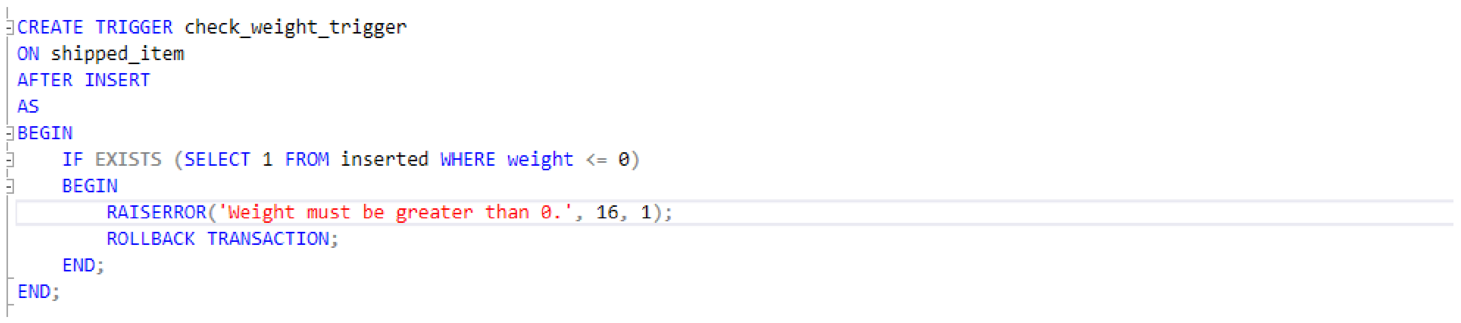
```sql
INSERT into toy_lot values
('TL001',60.00,'2023-05-01','SI001','PU001'),
('TL002',55.50,'2023-05-02','SI002','PU002'),
('TL003',42.50,'2023-05-03','SI003','PU003'),
('TL004',75.00,'2023-05-03','SI004','PU004');

INSERT into raw_material values
('RM001', 15.50, 'direct', 'fp001',''),
('RM002', 20.25, 'indirect', '', 'p001'),
('RM003', 35.00, 'direct', 'fp002', ''),
('RM004', 50.00, 'direct','fp003', '');

insert into created values
('TL001','RM001'),
('TL002','RM002'),
('TL003','RM003'),
('TL004','RM004');

insert into includes values
('PU001','TL001'),
('PU002','TL002'),
('PU003','TL003'),
('PU004','TL004');

insert into phone_number values
('RC001','0715806969'),
('RC002','0777555555'),
('RC003','0789187091'),
('RC004','0793469101');
```

## 5. Triggers

1. This trigger updates the insurance amount for newly inserted items in the SHIPPED_ITEM table based on their weight. 1000 for 1 kg.

```
create trigger update_insurance_amount
on shipped_item
for insert
as
begin
  update shipped_item
  set insurance_amount =shipped_item.weight * 1000
end;
```

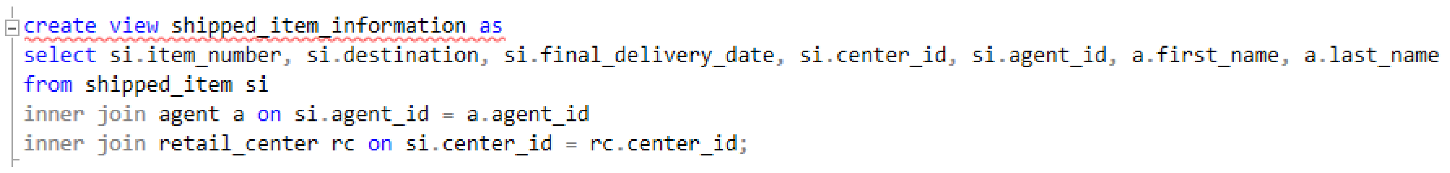
2. This trigger raises an error if user enters minus value for weight for SHIPPED_ITEM.



```
CREATE TRIGGER check_weight_trigger
ON shipped_item
AFTER INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE weight <= 0)
    BEGIN
        RAISERROR('Weight must be greater than 0.', 16, 1);
        ROLLBACK TRANSACTION;
    END;
END;
```

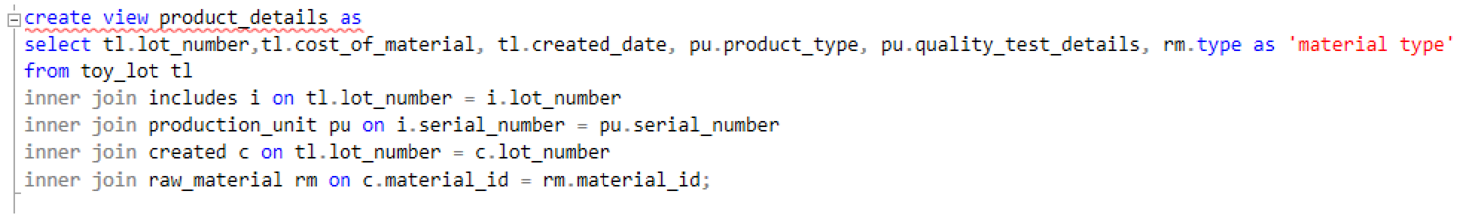
## 6. Views

1. Delivery Agent



```
create view shipped_item_information as
select si.item_number, si.destination, si.final_delivery_date, si.center_id, si.agent_id, a.first_name, a.last_name
from shipped_item si
inner join agent a on si.agent_id = a.agent_id
inner join retail_center rc on si.center_id = rc.center_id;
```

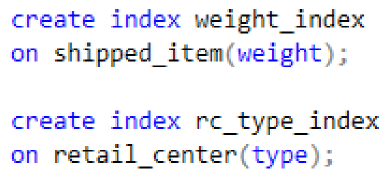
2. Production Manager



```
create view product_details as
select tl.lot_number,tl.cost_of_material, tl.created_date, pu.product_type, pu.quality_test_details, rm.type as 'material type'
from toy_lot tl
inner join includes i on tl.lot_number = i.lot_number
inner join production_unit pu on i.serial_number = pu.serial_number
inner join created c on tl.lot_number = c.lot_number
inner join raw_material rm on c.material_id = rm.material_id;
```

## 7. Index

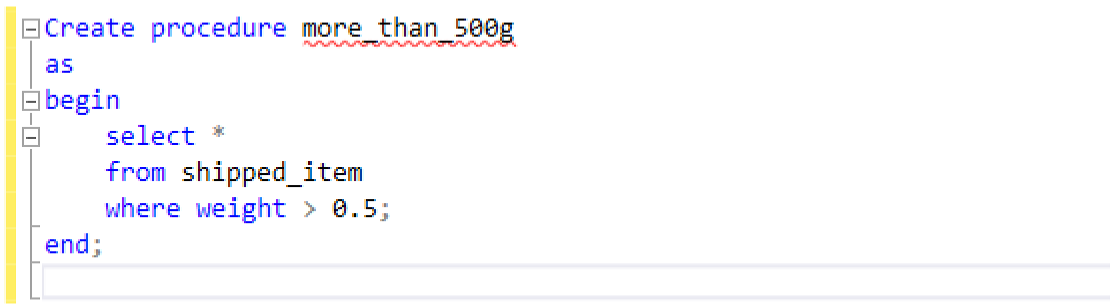
Following shows the Indexes for Weight and Retail Center



```
create index weight_index
on shipped_item(weight);

create index rc_type_index
on retail_center(type);
```

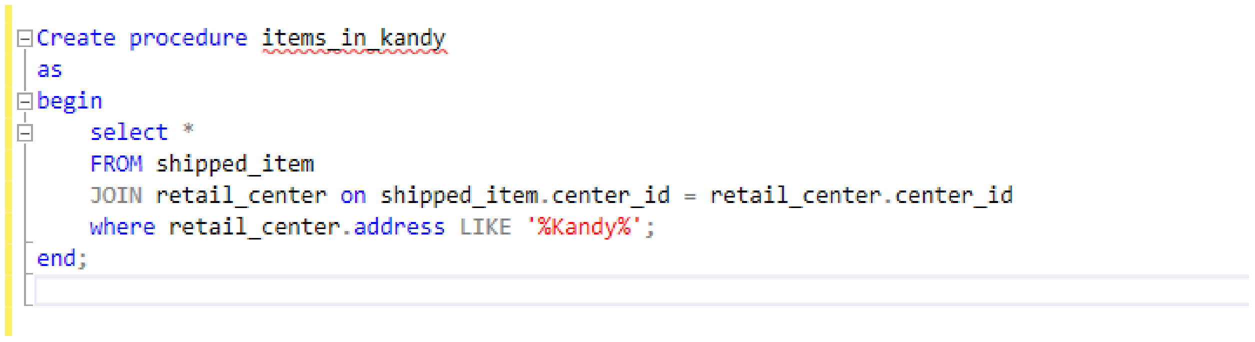
## 8. Procedures

1. List all the shipped items that weigh more than 500g.



```
Create procedure more_than_500g
as
begin
    select *
    from shipped_item
    where weight > 0.5;
end;
```

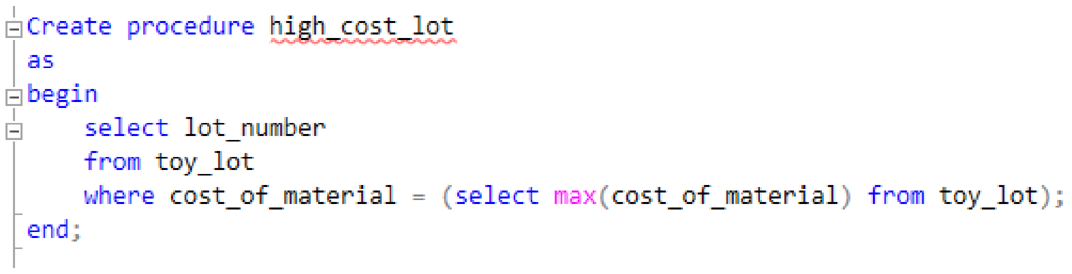
2. List all the shipped items where the city of the retail center is Kandy.



```
Create procedure items_in_kandy
as
begin
    select *
    FROM shipped_item
    JOIN retail_center on shipped_item.center_id = retail_center.center_id
    where retail_center.address LIKE '%Kandy%';
end;
```

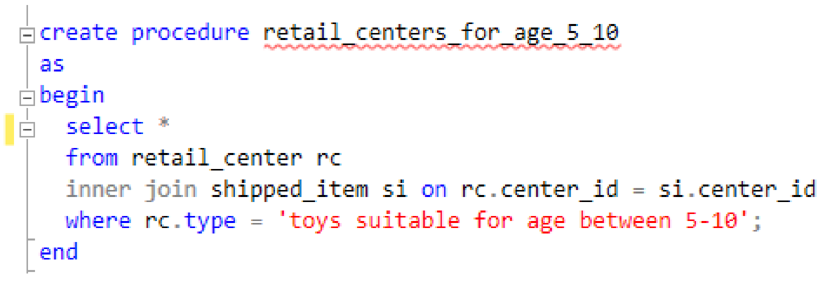
3. List down the lot numbers which has the highest unit cost values of the raw

Materials.



```
Create procedure high_cost_lot
as
begin
    select lot_number
    from toy_lot
    where cost_of_material = (select max(cost_of_material) from toy_lot);
end;
```

4. List all the retail centers taken by the type “toys suitable for age between 5-10”



```
create procedure retail_centers_for_age_5_10
as
begin
  select *
  from retail_center rc
  inner join shipped_item si on rc.center_id = si.center_id
  where rc.type = 'toys suitable for age between 5-10';
end
```

5.

# PART 2

## 1. Database vulnerabilities focusing on techniques and impact.

Databases have been targeted by attackers because they are the main part in an organization or a company, storing records, data, and other confidential information. The major problem here is not having enough protection. When hackers or attackers gain access to sensitive data, they can alter, extract, and damage these databases, resulting in integrity violations, and losses for the victims.

Two common database vulnerabilities are,

1. **SQL Injection:**

Injection Attacks are which allow the attacker to inject a code into a program, queries, or a malicious code on to a computer or a server to execute remote commands that can read or alter a database data or a website data. From these injections the commonly used method is SQL injection.

Structured Query Language (SQL) Injection is a code injection technique which allows the attacker to interface with a database that has a web security vulnerability. Most commonly the attacker can insert a malicious SQL query as input data to the vulnerable SQL application and manipulate the applications backend. The attack allows the attacker to compromise the Database and gain unauthorized access.

There are types of SQL code injection techniques which can be categorized depending on the situation, path, and attack techniques. Some of the techniques are:

a. Union-Based SQL Injection:

UNION is an Operator which combines the result of two or more individuals queries into a distinct result set within a single row in SQL. Using this technique, the attacker can retrieve data from another table or database.

b. Boolean-based SQL Injection:

In this technique, the attacker injects SQL code that forces the database to return a different result based on a true or false condition. Then the result gives the attacker valuable data even though no data from the database are recovered.

c. Error-Based SQL Injection:0.25

In this technique the attacker uses malicious code to generate an error message which contains sensitive data like the database name, table name or column name. This is also known as blind SQL injections.

d. Out-of-Band SQL Injection:

In this technique type when the attacker attacks the application using the SQL injection but does not receive a response from the attacked application on the same channel but instead cause the application to send data to a remote endpoint that they own.

e. Time-Based SQL Injection:

This SQL injection causes the database to delay using a heavy query or a specific DBMS function. Which will help to deduct some information depending on the time it takes to get the response from the server. Then it will be useful for the blind or deep blind SQL injection attacks.

- Impact of Successful SQL injection

A successful SQL code Injection can result in loss of damage to company, or organization as well as the user of the data. Data breach results in unauthorized access, which can hold up for ransomware attacks. If the attacker decides to corrupt the data, the user or the organization can have a data loss which cannot be recovered. Some attacks result in financial losses due to financial data or sensitive data like passwords. Mostly if the compromised database belongs to a company mainly their reputation will be damaged permanently. The Laws like HIPAA or GDPR compliances will also be violated due to the attacks. For the businesses it will lose its productivity and loss of revenue can happen.

- Identifying a SQL Injection attack

Identifying a SQL injection attack can be challenging but there are some common signs which may indicate an attack is in progress. SQL Injection attacks mostly involve malicious SQL code into user inputs. Looking for unexpected or malformed input in webforms, search boxes, or other user input field can be helpful when identifying a SQL Injection. Sometimes database errors can happen such as syntax errors or invalid queries. Monitoring database logs for these types of errors can indicate an attack. When an attack happens the increase of the network traffic can happen result of sending multiple requests to the server to execute the injected SQL code. Mostly performance issues, changes in data and suspicious database activities are the aftermath of the successful SQL Injection. From identifying them we can stop them in the progress or mitigate further damage.

2. **DDoS attacks:**

DDoS, or distributed denial of service, is a form of cyberattack that prevents a targeted server, service, or network from functioning normally by flooding the target or the area around it with internet traffic, making it unreachable to the legitimate users. Simply described, a DDoS attack involves several computers attacking a single computer, preventing the system from being accessed by the legitimate user. DDoS attacks cause a website to operate poorly or go offline.

Types of DDoS attacks

a. A volumetric attack

Volumetric attack is one of the most prevalent DDoS attack methods. In volumetric attacks, the network layer is flooded with what looks to be real traffic, which uses all the available bandwidth and creates traffic in the system. A number of approaches, such as the use of botnets or zombie computers, exploiting holes in network architecture, and using sources of amplified traffic, are used to conduct a volumetric attack.

b. Protocol attacks.

Protocol attacks make use of weaknesses in layers 3 and 4 of the protocol stack, which could disrupt services as a result. SYN attack is an example of such an attack.

c. A resource or application layer attacks

In this type of attack, it targets web application packets and disrupts the transmission of data between hosts. HTTP protocol violations, SQL injection, cross site scripting and other layer 7 attacks.

- Impact of DDOS attacks

These attacks can seriously impair web services, resulting in costs for businesses and harm to their reputations. They may cause systems to become unresponsive, which would hurt productivity and revenue. DDoS assaults may also be employed to divert attention from other sinister acts.

Collateral damage might result from unintentional involvement of computers belonging to innocent people. Investments in security measures are necessary to defend against DDoS attacks, raising operating expenses. To lessen the effects of such attacks, swift mitigation and cooperation are crucial.

- How Identify DDOS attacks

When identifying a DDOS attack we can watch for Network or service slowdowns, odd traffic patterns, increased resource usage, difficulty accessing services, odd DNS or IP resolution requests, abnormalities in network traffic. From identifying these attacks, we can easily mitigate further damage.

## How to mitigate database vulnerabilities and applying countermeasures to Overcome.

Database vulnerabilities can have major consequences like data breaches, unauthorized access, financial losses, reputation and brand losses and compliance violations for an organization or a company. Because of this mitigating these database vulnerabilities are very important. Some of the mitigating ways and countermeasures for apply are:

Scanning the vulnerabilities can always help to find the vulnerabilities. Then we can calculate the risks and find more ways to mitigate those vulnerabilities. Knowing the vulnerabilities is the simplest way to know the countermeasures for them and be ready. We need to understand and identify the sensitive data which we store in the databases to provide more security. We can then reduce the access privileges and some excessive privilege problems like insider attacks.

Monitoring the database in real time will help to identify the progressing attacks. We can detect unusual access to the database and malicious web requests. Attacks like SQL Injections can be stopped on constant monitoring. When comes to the user input all the inputs should be treated as untrusted. Any user can introduce the risk of SQL Injections therefore it is important to treat the inputs untrusted even if the inputs are authenticated and from the internal users. Then we can reduce the risks of code injections.

User Right Management is another countermeasure which can stop the Code Injection attacks. Managing the access of the user like identifying and removing excessive rights and dormant users, aggregate access rights, review, approve or reject user rights can use to solve some excessive privilege problems too.

For DDoS attacks we can use mitigation services to detect and block unwanted traffic. Ensuring a scalable network infrastructure with load balancing and traffic sharpening mechanisms also helps with the heavy traffic. traffic filtering, rate restriction, and intrusion detection systems can do as more countermeasures to DDoS attacks. Using the latest technology helps in every vulnerability cause the upgraded technology will safeguard more than the older. Using new encryption methods, encrypting devices are some examples for this. Educate the workforce on risk mitigation techniques including identifying the threats and vulnerabilities, best practices around internet, devices and password management will help to reduce the risks on both above vulnerabilities.

As we discussed above, understanding the database vulnerabilities and implanting the countermeasures will enable us to protect the valuable assets against hackers.