Sri Lanka Institute of Information Technology


Assignment 1

Group 02

# Database Design, Implementation and Security

**Database Management Systems For Security – IE2042**

B.Sc. (Hons) in Information Technology – Cyber Security

## Group Details

Group Number: 02

Project Title: Database Design , Implementation, and Security

| | Student ID | Student Name | Email | Contact Number |
|---|---|---|---|---|
| 1 | IT22357762 | Dewmini P.L.T | IT22357762@my.sliit.lk | 0741688488 |
| 2 | IT22315496 | Anuradha D.P.G.C | IT22315496@my.sliit.lk | 0713952609 |
| 3 | IT22599186 | Yashodini Jayasinghe | IT22599186@my.sliit.lk | 0767096940 |
| 4 | IT22002174 | Dineth T.H.V | IT22002174@my.sliit.lk | 0766604096 |

# Term of references

*A report submitted in fulfilment of the requirement for the module IE2042, Sri Lanka Institute of Information Technology*

# Contents

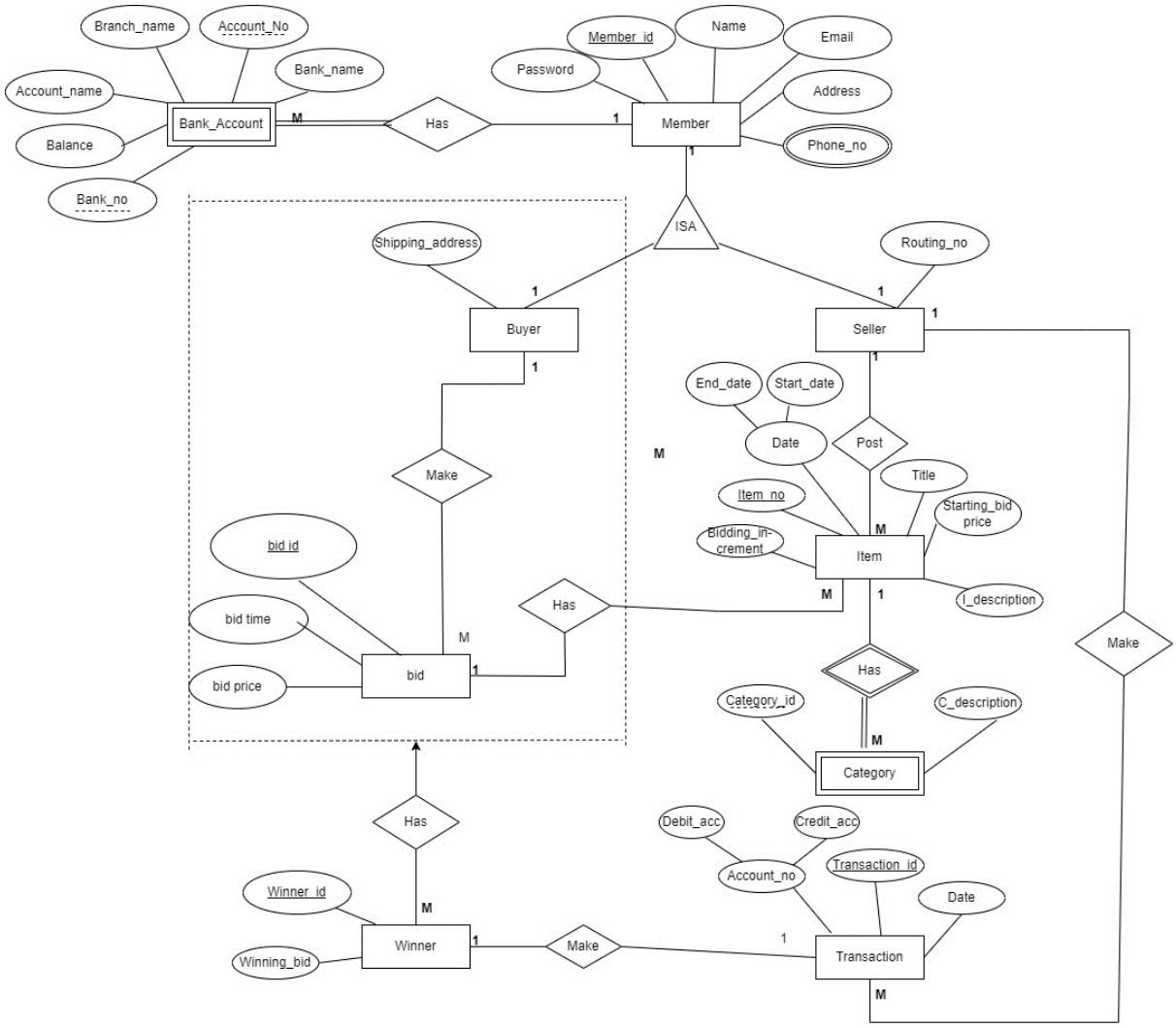
# 1 Database scenario

Consider an ONLINE AUCTION database system in which members (buyers and sellers) participate in the sale of items. The online site has members, each of whom is identified by a unique member number and is described by an e-mail address, name, password, home address, and phone number. A member may be a buyer or a seller. A buyer has a shipping address recorded in the database. A seller has routing number recorded in the database. Both buyers and sellers can have multiple bank accounts associated with their account. This needs to include Bank Name, Bank No, Branch Name, Account Name, Account ID, and Account balance. Items are placed by a seller for sale and are identified by a unique item number assigned by the system. Items are also described by an item title, a description, starting bid price, bidding increment, the start date of the auction, and the end date of the auction. Items are also categorized based on a classification system which includes a Category ID and Description. Buyers make bids for items they are interested in and are currently up for sale. The bid price and time of bid is recorded. The bidder at the end of the auction with the highest bid price is declared the winner. A transaction between buyer and seller may then proceed for the winning bid. Information on when the transaction took place, Credit account and debit account as well as who the seller and buyer for a transaction is as well as details of the winning bid must be captured.

# 2 Assumptions

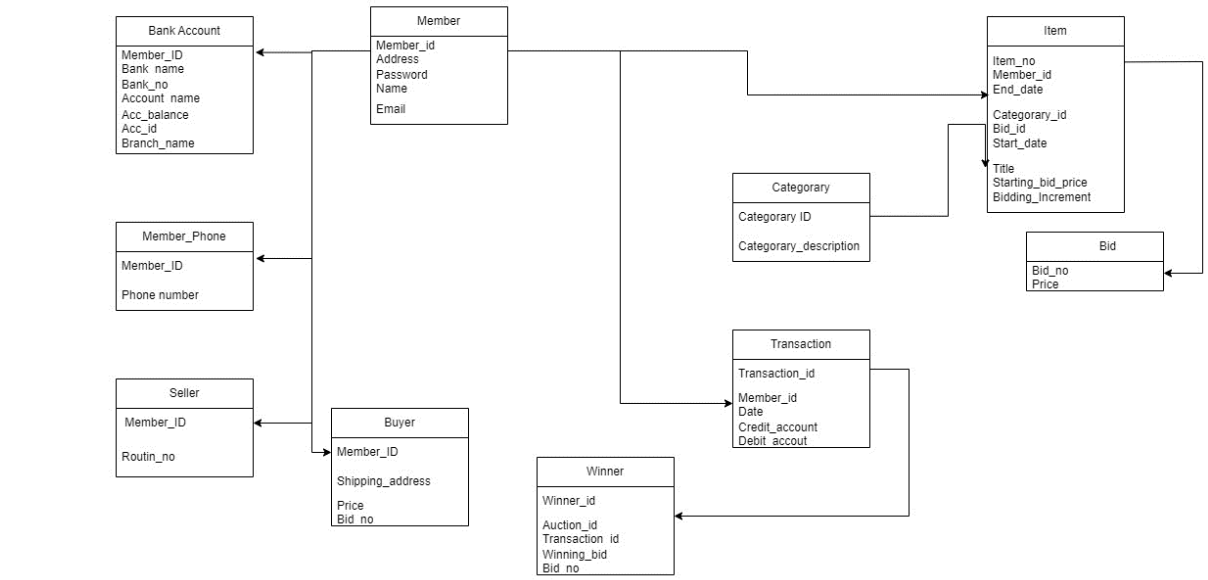
- Each member can have only one account.
- Members can have multiple telephone numbers and they are dependent from the memer_id.
- Buyer can bid for the item when the seller posts it.
- The bank account is dependent on the Member.
- Buyers can bid for many items as well as sellers can post multiple items.
- Category depends on the item.
- Multiple buyers can bid for multiple items and among them the highest is considered as the winner.
- Once the winner is declared, the winner can make the transaction according to the item he won.
- As the seller posts  many products they are proceeded with many transactions.
- Account name and branch name can be identified by the account id.
- Bank no reveals the name of the bank.

# 3 Enhanced Entity Relationship Diagram




[online auction.drawio]

# 4 Schema of the database



# 5 Relational Schema

Member ( Member_Id , password , email , address, name )

Member_Phone ( Member_Id , phone_no )

Bank_Account (Member_Id , Account_No, bank_no bank_name, account_name , acc_balance, branch_name)

Buyer ( Member_Id , shipping_address )

Seller ( Member_Id , routin_no )

Bid (Bid_id, Bid_time, Bid_price,Member_id)

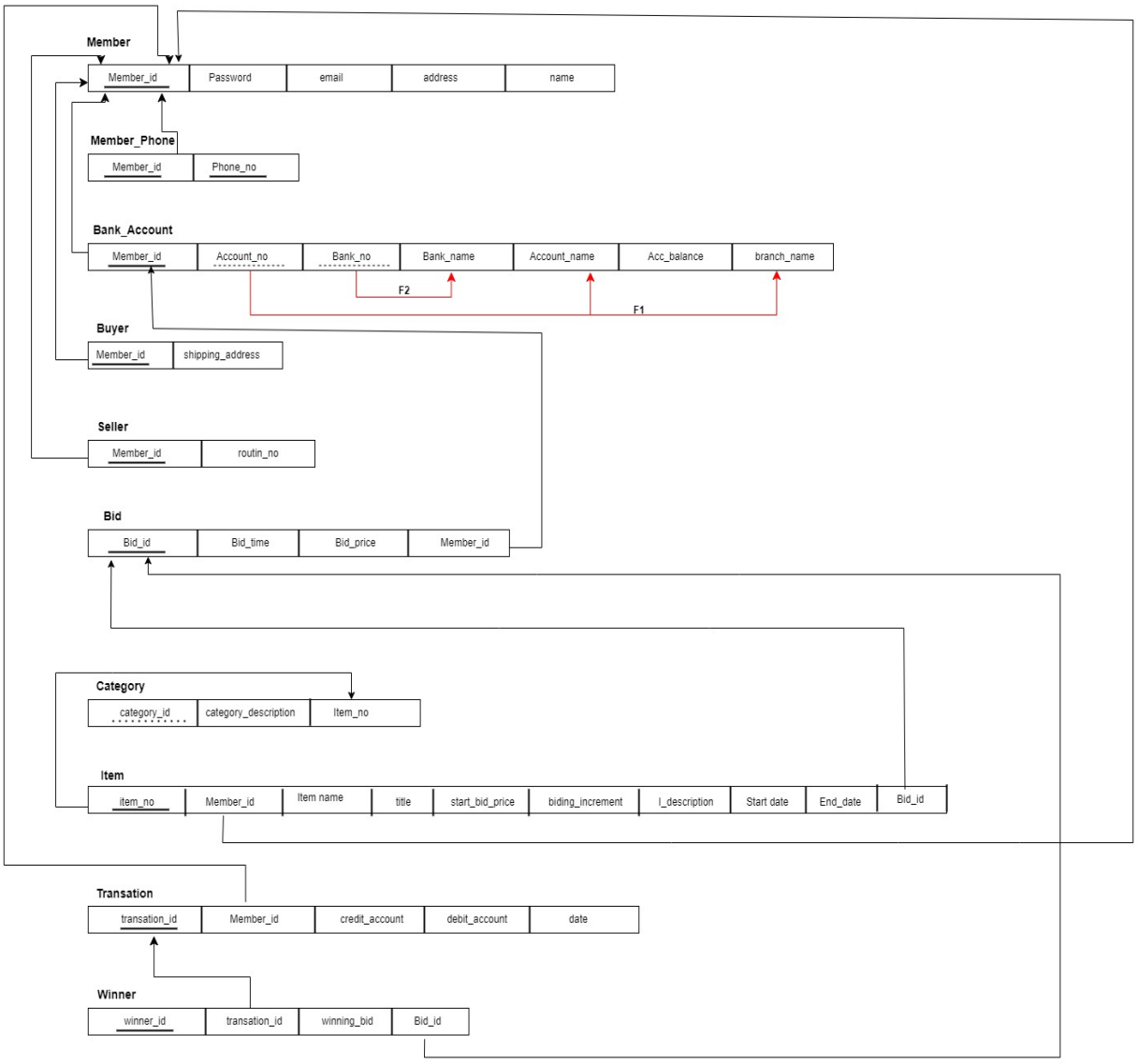
Category (Item_no, category_id , category_description )

Item ( item_no , member_id , title , start_bid_price , biding_increment ,I_description, Start_date, End_date,Bid_id,Item_name)

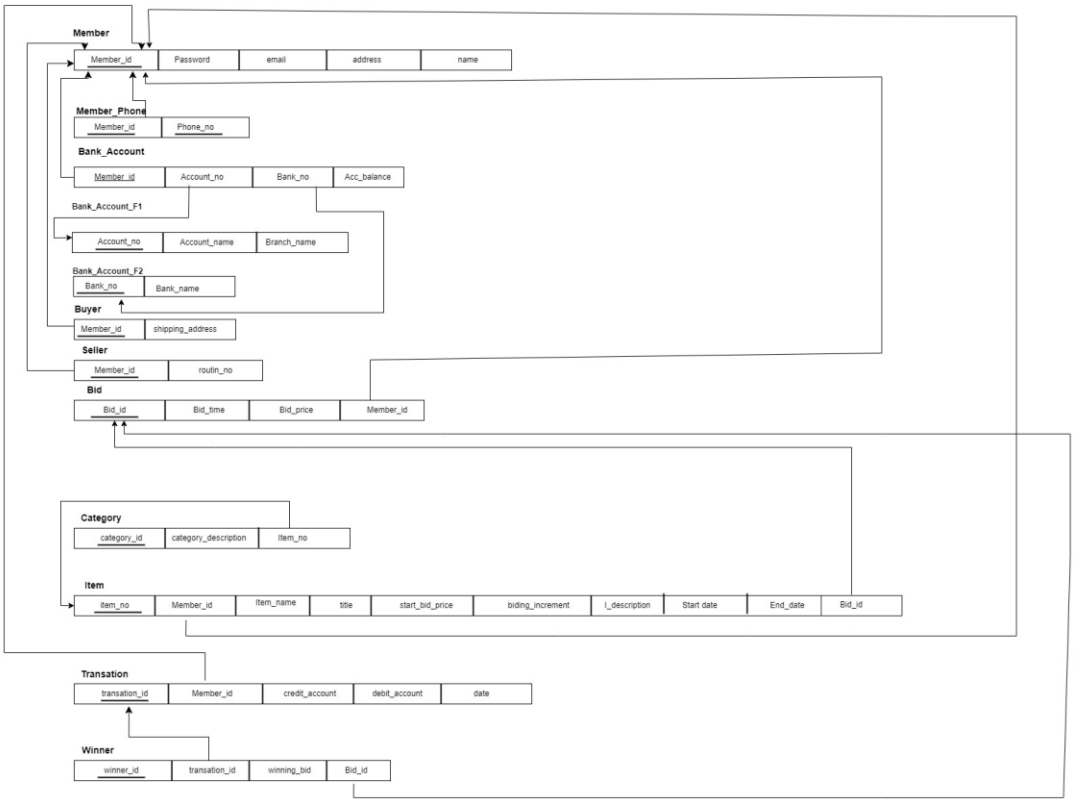
Transaction ( Transaction_id , credit_account , debit_account , date, member_id )

Winner ( winner_id , transaction_id , winning_bid , Bid_Id,)

# 6 Unnormalize relation schema

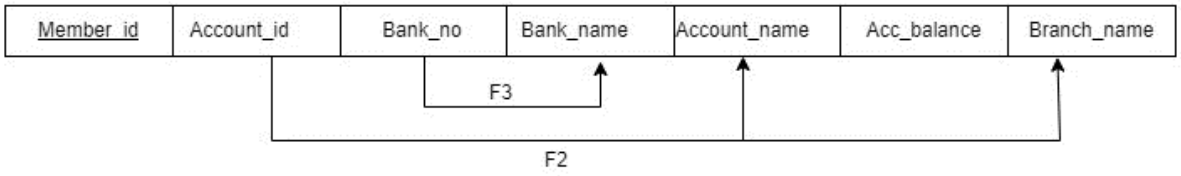


# 7 Normalized relational schema.




# 8 Functional Dependencies

**Bank_account Table**



# 9 SQL Codes – DDL

## 9.1 Table Creation

```
Create database DMSS_Project;
Use  DMSS_Project;

---creating member table--

Create table Member(
Member_id char(10),
Password varchar(10),
email varchar(255),
address varchar(255),
name varchar(255),

Constraint CHK_Memeber_Id Check (Member_id like'M%'),
Constraint Member_PK Primary Key (Member_id),
);

---creating member_phone--

Create table Member_phone(
Member_id char(10),
Phone_no char(10),

Constraint Member_phone_PK Primary Key (Member_id,Phone_no),
Constraint Member_phone_FK Foreign key (Member_id) References Member(Member_id)
);

---creating bank account f2--

Create table Bank_Account_F2(

Bank_no varchar(10),
Bank_Name varchar(255),
```

```
Constraint Bank_Account_F2_PK Primary Key (Bank_no)
);

--creating bank account f1--

Create table Bank_Account_F1(

Account_no char(10),
Account_name varchar(255),
Branch_name varchar(255),
Constraint Bank_Account_F1_PK Primary Key(Account_no),


);
---Creating Bank Account ---

Create table Bank_Account(

Member_id char(10),
Account_no char(10),
Bank_no varchar(10),

Constraint Bank_Account_CHK Check (Account_no >=0),
constraint Bank_Account_PK Primary key (Member_id),
constraint Bank_Account_FK_1 Foreign Key (Member_id) References Member(Member_id),
constraint Bank_Account_FK_2 Foreign Key (Account_no) References Bank_Account_F1(Account_no),
constraint Bank_Account_FK_3 Foreign Key (Bank_no) References Bank_Account_F2(Bank_no),
);
```

```
---creating Buyer---

Create table Buyer(

Member_id char(10),
shopping_address varchar(255),


constraint Buyer_PK Primary key (Member_id),
constraint Buyer_FK_ Foreign Key (Member_id) References Member(Member_id),
);

---creating Seller---

Create table Seller(
Member_id char(10),
routing_no varchar(255),

Constraint Seller_PK Primary Key (Member_id),
Constraint Seller_FK Foreign Key (Member_id) References Member(Member_id)
);
```

```
Create table Bid(

 Bid_ID char(10),
 Bid_time time,
 Bid_price int,
 Member_id char(10)

 Constraint BId_PK Primary Key (Bid_ID),
 Constraint BId_FK Foreign Key (Member_Id) References Member(Member_id)

);


---creating Category ---

Create table Category(
Category_id char(10),
Category_description varchar(255),
item_no char(10),

constraint Category_PK Primary Key (item_no),


);



 ---creating item ---

Create table Item(
 Item_no char(10),
 Member_id char(10),
 title varchar(255),
 start_bid_price int,
 biding_increment varchar(255),
 I_description varchar(255),
 start_date date,
 end_date date,
 Bid_ID char(10),
 Item_name varchar(255),

 constraint Item_PK Primary Key (Item_no),
 constraint Item_FK_1 Foreign Key (Member_id) References Member(Member_id),
 constraint Item_FK_3 Foreign Key (Bid_ID) References Bid(Bid_ID),
 );

 --- Add the foriegn key item no to the category table--
ALTER TABLE Category
 Add constraint Category_FK_1 Foreign Key (item_no) References Item(Item_no);

 --creating Transaction1--
Create table Transaction1(
```

```
Transaction1_id char(10),
Member_id char(10),
credit_account varchar(255),
debit_account varchar(255),
date date,

constraint Transaction1_PK Primary key(Transaction1_id),
constraint Transaction1_FK Foreign key (Member_id) References Member(Member_id)

);

Create table Winner(

Winner_id char(10),
Transaction1_id char(10),
winning_bid varchar(255),
Bid_id char(10),

constraint Winner_PK Primary key(Winner_id),
constraint Winner_FK_1 Foreign Key (Transaction1_id) References Transaction1(Transaction1_id),
constraint Winner_FK_2 Foreign Key (Bid_id) References Bid(Bid_id),

);
```

## 9.2 Data Insertion

```
/*Data insert into Member table*/

INSERT INTO Member(Member_id, Password, email, address, name)
VALUES ('M123456', 'abc123', 'abc@gmail.com', '123 Main St', 'Saman');
INSERT INTO Member(Member_id, Password, email, address, name)
VALUES ('M123457', 'mno123', 'cdf@gmail.com', '63 Chaple St', 'Michel');
INSERT INTO Member(Member_id, Password, email, address, name)
VALUES ('M1234548', 'klm123', 'klm@gmail.com', '28 Bakers St', 'Sherlock');
INSERT INTO Member (Member_id, Password, email, address, name)
VALUES ('M123459', 'xyz789', 'xyz@gmail.com', '55 Elm St', 'Alice');
INSERT INTO Member (Member_id, Password, email, address, name)
VALUES ('M123460', 'pqr456', 'pqr@gmail.com', '123 Oak St', 'Bob');



/*Data insert into Member_phone table*/

INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123456', '123456787');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123457', '456789432');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M1234548', '987654321');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123459', '555555555');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123459', '555123456');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123460', '777777777');
INSERT INTO Member_phone (Member_id, Phone_no)
VALUES ('M123460', '777888888');
```

```sql
/*Data insert into Bank_Account_F2 table*/

INSERT INTO Bank_Account_F2 (Bank_no, Bank_Name)
VALUES ('B001', 'Sampath_Bank');
INSERT INTO Bank_Account_F2 (Bank_no, Bank_Name)
VALUES ('B002', 'Peoples_Bank');
INSERT INTO Bank_Account_F2 (Bank_no, Bank_Name)
VALUES ('B003', 'HNB_Bank');

/*Data insert into Bank_Account_F1 table*/

INSERT INTO Bank_Account_F1 (Account_no, Account_name, Branch_name)
VALUES ('123456', 'John De Silva', 'Colombo 03' );

INSERT INTO Bank_Account_F1 (Account_no, Account_name, Branch_name)
VALUES ('20078123', 'Michel Perera ', 'Kolpitty' );

INSERT INTO Bank_Account_F1 (Account_no, Account_name, Branch_name)
VALUES ('98345621', 'Sherlock Homes', 'Galle');

-- Bank account details for member with Account_no '98345621' (M123459)
INSERT INTO Bank_Account_F1 (Account_no, Account_name, Branch_name)
VALUES ('98345622', 'Alice Smith', 'Colombo 04');

-- Bank account details for member with Account_no '98345621' (M123460)
INSERT INTO Bank_Account_F1 (Account_no, Account_name, Branch_name)
VALUES ('98345623', 'Bob Johnson', 'Colombo 05');
```

```sql
/*Data insert into Bank_Account table*/

INSERT INTO Bank_Account (Member_id, Account_no, Bank_no)
VALUES ('M123456', '123456', 'B001');
INSERT INTO Bank_Account (Member_id, Account_no, Bank_no)
VALUES ('M123457', '20078123', 'B002');
INSERT INTO Bank_Account (Member_id, Account_no, Bank_no)
VALUES ('M1234548', '98345621', 'B003');
INSERT INTO Bank_Account (Member_id, Account_no, Bank_no)
VALUES ('M123459', '98345622', 'B003');
INSERT INTO Bank_Account (Member_id, Account_no, Bank_no)
VALUES ('M123460', '98345623', 'B003');


/*Data insert into Buyer table*/

INSERT INTO Buyer (Member_id, shopping_address)
VALUES ('M123456', '456 Elm St');
INSERT INTO Buyer (Member_id, shopping_address)
VALUES ('M123457', '122 Park Street');
INSERT INTO Buyer (Member_id, shopping_address)
VALUES ('M123459', '456 Elm St');


/*Data insert into Seller table*/

INSERT INTO Seller (Member_id, routing_no)
VALUES ('M1234548', '123456789');
INSERT INTO Seller (Member_id, routing_no)
VALUES ('M123460', '987654321');
```

```sql
/*Data insert into Bid table*/
INSERT INTO Bid(Bid_ID, Bid_time, Bid_price, Member_id)
VALUES ('BID23', '2023-10-30 12:00:00', '32000', 'M123456');
INSERT INTO Bid(Bid_ID, Bid_time, Bid_price, Member_id)
VALUES ('BID24', '2023-10-20 12:15:30', '23000', 'M123457');
INSERT INTO Bid (Bid_ID, Bid_time, Bid_price, Member_id)
VALUES ('BID25', '2023-11-30 12:15:30', '23000', 'M123457');
INSERT INTO Bid(Bid_ID, Bid_time, Bid_price, Member_id)
VALUES ('BID26', '2023-10-20 12:15:30', '28000', 'M123459');


/*Data insert into Category table*/

INSERT INTO Category (Category_id, Category_description, item_no)
VALUES ('C001', 'Electronics', 'ITEM001');
INSERT INTO Category (Category_id,Category_description , item_no)
VALUES ('C002', 'Cosmetic', 'ITEM002');
INSERT INTO Category (Category_id, Category_description, item_no)
VALUES ('C003', 'Toys', 'ITEM003');
```

```sql
/*Data insert into Item table*/

INSERT INTO Item (Item_no, Member_id, Category_id, title, start_bid_price, biding_increment, I_description, start_date, end_date,Bid_ID,Item_name)
VALUES ('ITEM001', 'M123456', 'C001', 'Smartdevice', '500', '10', 'Brand new smartdevice', '2023-10-25', '2023-11-13', 'BID23', 'Laptop');

INSERT INTO Item (Item_no, Member_id, Category_id, title, start_bid_price, biding_increment, I_description, start_date, end_date,Bid_ID,Item_name)
VALUES ('ITEM002', 'M123457', 'C002', 'Foundation', '1500', '10', 'Uniform color to the complexion', '2023-09-02', '2023-11-05', 'BID24', 'Li_Foundation');

INSERT INTO Item (Item_no, Member_id, Category_id, title, start_bid_price, biding_increment, I_description, start_date, end_date,Bid_ID,Item_name)
VALUES ('ITEM003', 'M123457', 'C003', 'WoodenElephant', '5000', '10', 'WoodenToys', '2023-10-25', '2023-11-12', 'BID25', 'ToyElephant');

/*Data insert into Transaction1 table*/

INSERT INTO Transaction1 (Transaction1_id, Member_id, credit_account, debit_account, date)
VALUES ('T001', 'M123456', '98345621','123456',  '2023-10-20');
INSERT INTO Transaction1 (Transaction1_id, Member_id, credit_account, debit_account, date)
VALUES ('T002', 'M123457', '98345621' ,'12345679', '2023-10-20');
INSERT INTO Transaction1 (Transaction1_id, Member_id, credit_account, debit_account, date)
VALUES ('T003', 'M123456' , '98345621', '123456', '2023-10-20');

/*Data insert into Winner table*/
INSERT INTO Winner(Winner_id, Transaction1_id, winning_bid, Bid_id)
VALUES ('W001', 'T002', '500.00', 'BID23');
INSERT INTO Winner (Winner_id, Transaction1_id, winning_bid, Bid_id)
VALUES ('W002', 'T002', '1500.00', 'BID24');
INSERT INTO Winner (Winner_id, Transaction1_id, winning_bid, Bid_id)
VALUES ('W003', 'T003', '5000.00', 'BID25');
```

# 10 Triggers

```
--Tigger 01--

GO
Create Trigger Change_Item_No1
On item
After UPDATE
As
Begin
Declare @olditem_no char(10), @newitem_no char(10)
Select @olditem_no = Item_no from deleted
Select @newitem_no = Item_no from inserted
Update Item set Item_no=@newitem_no where Item_no=@olditem_no
End

--Tigger 02--

Go
Create Trigger change_members
On Member
After Update As
Begin
Declare @old_member_id char(10), @new_member_id char(10)
    Select @old_member_id =Member_id from deleted
    Select @new_member_id = Member_id from inserted
Update Member set Member_id =@new_member_id where Member_id=@old_member_id
End
```

Trigger 1 is used to change the item no in the Item table. Here by, trigger capture the old and new values of the data and delete and insert in the virtual tables respectively. And then update the new value where it matches to the old value.

Trigger 2 is used to update the member table. Once it's triggered member id can be changed into a new value.

# 11 Indexes

```
--Index--

--Index 01--

Create INDEX member_details
ON Member (Member_id,name, email,address);

--index 02--

Create INDEX Item_info
On Item (Item_no,Item_name);
```

This has enhanced the data retrieval performance for the queries which involved with the column Member_id , name, email and address.

2nd one has enhanced the data retrieval performance for queries including with the column Item_no, Item_name.

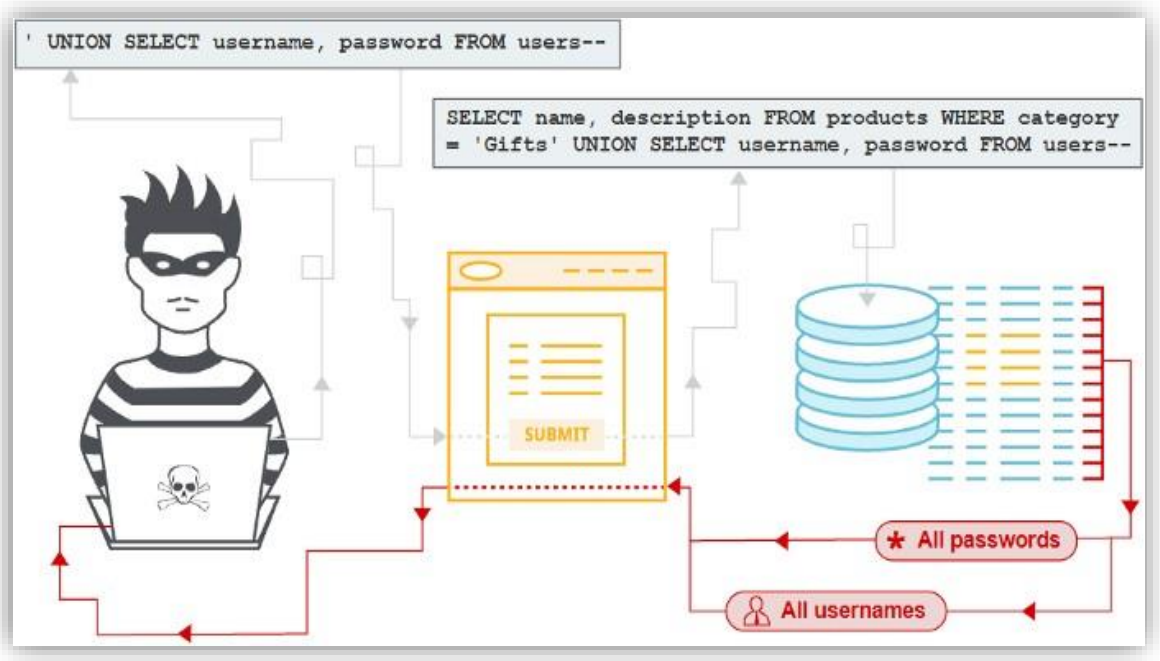
# 12 Stored Procedures

```
--procedure--

--Procedure 01 --
GO
Create Procedure Get_Member_Details_Sampathbank (@bank_name varchar(255), @name_person varchar(255) out, @address_person varchar(255) out) As
begin
    Select @name_person = M.name, @address_person = M.address
    from Member M, Bank_Account_F2 BA, Bank_Account B
    where B.Member_id = M.Member_id and B.Bank_no = BA.Bank_no and BA.Bank_Name = @bank_name
End

Exec Get_Member_Details_Sampathbank 'sampath bank', @name_person output, @address_person output
Print 'Details of the area: ' + @name_person + @address_person



--procedure 02--
GO
create Procedure Get_bid_laptop (@bid_item varchar(255),@name varchar(255) out, @email varchar(255) out,@bidprice varchar(255) out) As
begin
    Select @name = M.name, @email=M.email
    from Member M, Item I, Bid B
    where B.Member_id=M.Member_id and B.Bid_ID=I.Bid_ID and I.Item_Name=@bid_item
End

Exec Get_bid_laptop 'Laptop', @name output, @email output, @bidprice output
Print 'Details are :' +@name+@email+@bidprice


--procedure 03 --
GO
CREATE PROCEDURE Get_name_sell(@total_bid_price INT, @name VARCHAR(255) OUT) AS
BEGIN
    SELECT @name = M.name
    FROM Member M
    WHERE M.Member_id IN (
        SELECT S.Member_id
        FROM Seller S
        JOIN Bid B ON S.Member_id = B.Member_id
        GROUP BY S.Member_id
        HAVING SUM(B.Bid_price) = @total_bid_price
    );
END

  Procedure 04
GO
create Procedure Get_incre_bid (@name varchar(255),@Incre_bid_price INT out ) As
begin
    select @Incre_bid_price =B.Bid_price*115/100
    from Bid B, Member M
    where M.name = @name
End

Exec Get_incre_bid 'Saman',@Incre_bid_price output
Print 'OutPut' + @Incre_bid_price
```

# 13 Database Vulnerabilities

## 13.1 SQL Injection

SQL injection (SQLi) is an online security vulnerability that allows an attacker to impede database queries that an application submits to its server. Thanks to this strategy, an attacker can usually review data that they wouldn't otherwise be able to acquire. everything that the program itself is able to access, including data that is owned by other users. An attacker might often change or delete this data, changing the operation of the program permanently. In some situations, a denial-of-service attack or an escalated SQL injection attack may be initiated, compromising the underlying server or other back-end infrastructure.




### 13.1.1 Impacts of SQL Injection

Sensitive data that might be accessed without authorization by a successful SQL injection attack includes credit card numbers, passwords, and personal user information. Several high-profile data breaches in recent years have been caused by SQL injection attacks, which have damaged reputations and resulted in sanctions. An attacker may be able to establish a persistent backdoor into a business's system, which might lead to a long-term breach that is not discovered for a while

### 13.1.2 SQL Injection Examples

Numerous unique SQL injection attacks, vulnerabilities, and tactics exist that manifest in diverse scenarios. Common SQL injection examples are: ✓ Retrieving hidden data, which involves changing a SQL query to provide more results.

- ✓ Undermining application logic, which involves modifying a query to obstruct the programmer's reasoning.
- ✓ UNION attacks, which allow you to get information from many database tables.
- ✓ Analyzing the database, from which you may get details about its version and organization.
- ✓ Blind SQL injection, in which the application's response does not include the answers to a query you control.

### 13.1.3 Blind SQL vulnerabilities

SQL injection scenarios often include blind vulnerabilities. As such, the application's responses do not include the results of the SQL query or details on any problems with the database. Although blind vulnerabilities may still be utilized to get unauthorized access to data, doing so usually calls for more sophisticated technology.

Depending on the specifics of the vulnerability and the database in question, blind SQL injection vulnerabilities may be exploited using any of the following techniques:

- ✓ The logic of the query may be changed to generate a noticeable difference in the application's outcome in response to the satisfaction of a single condition. This might include producing an error, such a divide-by-zero, conditionally or adding a new condition to some Boolean logic.
- ✓ By conditionally initiating a time delay in the query processing, you may infer the truth of a condition based on how long the application takes to respond.
- ✓ It is possible to initiate an out-of-band network contact by using OAST procedures. This strategy is quite powerful and useful when other methods aren't working. Data exfiltration through the out-of-band route is often possible instantaneously.

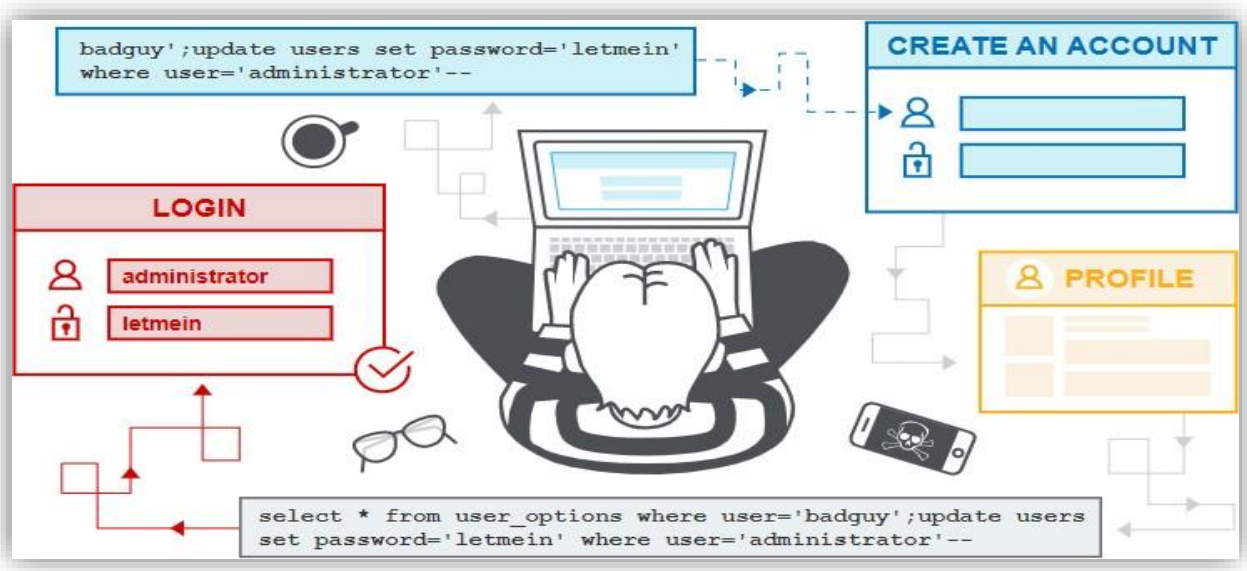
### 13.1.4 How to detect SQL injection vulnerabilities

- ✓ sending in the single quote character "and making sure there are no errors or strategies."
- ✓ entering some SQL-specific syntax, comparing it to the base (original) value of the entry point and to a new value, and then looking for systematic differences in the application responses.
- ✓ By adding Boolean criteria to the program, such as OR 1=1 and OR 1=2, and then looking for changes in the responses.
- ✓ Allows the submission of payloads designed to impede the execution of a SQL query, as well as the identification of response time fluctuations.
- ✓ Sending out OAST payloads designed to initiate an out-of-band network contact in response to a SQL query and monitoring the subsequent interactions.

### 13.1.5 Second – order SQL injection

An application may unsafely insert first-order SQL into a SQL query after processing user input obtained via an HTTP request.

The term "stored SQL injection" refers to the practice of an application saving user input from an HTTP request for potential use in the future. In order to do this, the data is often stored in a database; yet, at this point, there is no vulnerability. The program obtains the previously stored data and utilizes it dangerously in a SQL query when replying to a




subsequent HTTP request.

### 13.1.6 Countermeasures

Most SQL injection attacks may be prevented by substituting prepared statements, also known as parameterized queries, for string concatenation inside the query.

Parameterized queries may be used to treat untrusted input as data in just two places: the WHERE clause and the values in an INSERT or UPDATE statement. If they include untrusted information, they are unable to handle other query parts such as the ORDER BY clause and table or column names. A new approach, such whitelisting permissible input values, will need to be adopted by application functionality that enters untrusted data into such areas of the query, or utilize alternative logic to carry out the appropriate action.

To successfully prevent SQL injection, a parameterized query must always be a hard-coded constant and never include any variable data from any source. Refrain from analyzing each piece of data separately to decide whether it is reliable and continue using string concatenation in the query when it is considered secure. Too often, assumptions regarding whether data is tainted may be broken by altering other code, and the provenance of data might be wrongly detected.

## Security controls for PII:

- ✓ Change Management: This involves tracking and verifying changes to the configuration of IT systems, including adding or removing user accounts, that might affect security.
- ✓ Preventing data loss involves setting up mechanisms to track confidential information moving inside and outside of an enterprise and identify unusual patterns that may point to a security breach.
- ✓ Data masking: making sure that only the information absolutely necessary for the specific transaction is kept or sent, and that any more information is either concealed or deleted.
- ✓ Ethical walls: establishing screening protocols to prevent individuals or departments within an organization from accessing personally identifiable information (PII) that is unrelated to their job or that could present a conflict of interest.
- ✓ Monitoring of privileged users: this involves keeping an eye on all privileged access to databases and files, creating new users, and granting privileges; it also involves banning and alerting when any questionable behavior is detected.

- ✓ User tracking: putting in place mechanisms to monitor user behavior both online and while using company networks in order to spot careless disclosure of private information, account breach, or malevolent insiders.

## 13.2 Denial of service attack

An attack known as a denial of service (DoS) attempts to prevent a particular server, service, or network from operating normally This is accomplished by saturating the target with an excessive number of false requests or traffic. A denial-of-service (DoS) attack aims to prevent authorized users from accessing the target system or network, potentially leading to a temporary or permanent loss of services.

There are several types of DOS attacks including:

01. Volume base attacks

    These kinds of attacks overload the target with a large amount of traffic. The target's bandwidth and resources may be consumed by a rush of incoming packets, requests, or connections.
02. Protocol-based attacks

    These kinds of attacks take advantage of vulnerabilities in communication protocols. A SYN flood attack, for instance, uses the TCP/IP protocol's three-way handshake process to overload the target with SYN requests and exhaust its resources.
03. Application layer attacks

    These attacks, often called Layer 7 attacks, concentrate on particular services or applications inside the target's infrastructure. Attackers overwhelm a specific program by sending a lot of requests, which causes it to slow down or crash.
04. Distributed Denial of Service (DDOS) attacks

    Attacks using DDoS are coordinated denial-of-service attacks launched by several compromised computers, frequently as part of a botnet. Because DDoS attacks originate from a variety of sources, it can be difficult to discern between genuine and attack traffic, making mitigation more complex.

### 13.2.1 Impacts of Denial-of-Service Attack

**Enhanced Operational Costs:** It takes a lot of resources to mitigate a DoS attack. To defend against attacks, organizations could have to spend money on specialist DDoS mitigation services, hardware, or software. Operational expenses may also rise due to the labor and time needed for incident response, investigation, and recovery activities.

**Reputation Damage:** A lengthy denial-of-service (DoS) attack has the potential to harm an organization's or company's reputation. Consumers who frequently see delayed performance or service interruptions may come to doubt the company's ability to secure their information and deliver dependable services. Trust reconstruction following a major attack can be difficult.

**Disruption of Services:** The main effect of a denial-of-service assault is the interruption of services. When fraudulent traffic overwhelms the targeted system or network, it prevents genuine users from accessing it. For companies that depend on internet services for sales, this outage may result in losses.

### 13.2.2 How to detect DDOS attack vulnerabilities

- ✓ Regular security audits
- ✓ Network monitoring
- ✓ Anomaly detection
- ✓ Traffic analysis
- ✓ Application-level security

### 13.2.3 Denial of Service Attack Prevention Techniques

Check the network for problems.

Identifying your network's vulnerabilities is the first step toward properly protecting it. Examine every gadget connected to your network. This method entails collecting system data, describing the network's functionality, and locating any vulnerabilities that may already exist. With this level of expertise, you could recognize network vulnerabilities, evaluate their importance, and seal off any gaps to stop the incursion. Audits are valuable, even if they take time. It is always better for a team member to discover a security flaw than for an outsider, no matter how serious it may be.

It is important to ensure the safety of your existing infrastructure.

If you wish to successfully fend off a DoS attack, you need to ensure that your fortress's whole perimeter is guarded. This calls for the use of multi-level security techniques like intrusion prevention and threat management systems. These systems may use content filtering, firewalls, load balancing, VPNs, anti-spam, and security layers to detect and prevent attacks before they overwhelm your network. Software cannot do the task on its own; hardware is required. The best defense against DoS assaults for your network is edge micro-segmentation.

Collaborate with DDoS response teams.

To stay up to date on the newest risks and best practices, be aware of and work with groups such as CERTs (Computer Emergency Response Teams).

User education

User Awareness: Inform users and staff members on security best practices, like identifying dubious emails or links that could result in denial-of-service attacks (like phishing scams that infect bots).