

## Final Project - Database - Group 6 (Part One)

### Team Roles (Sorted by First Name Initial)

Boyan Qin - database designer

Gracia(Junzhu) Zhang - Project Manager(Organizing Part One Doc)

Jonathan Hernandez - Documenter(Presentation)

Lili Zheng - Security Specialist

Suraj Bashyal -SQL Developer

*Students will work in groups to create and submit a Word or PDF document and SQL code file that contains the following:*

#### 1. Database Concept

- *Define the purpose of your database (e.g., "Library Management System" or "Student Records Management").*
- *Identify the users of the database and the key functionalities it will provide.*

### E-commerce Management System

The rapid growth of e-commerce has pointed the need for a strong system that will efficiently manage products, orders, and customers effectively. Many sellers struggle to keep their inventory records accurate, process orders without hitches, and maintain customer data in a secure way. They often end up with delayed orders, dissatisfied customers, and operational inefficiency.

Our e-commerce database system would offer solutions to these issues with a safe, reliable, and easy-to-scale solution on catalog management, tracking of orders, and customer records. This would help improve operation efficiencies, ensure consistency in data, and allow sellers to provide an improved shopping experience for their customers.

### Users of the Database

- Sellers: Manage product catalogs, track orders, and analyze customer data.
- Admin Staff: Update products, manage orders, and support customers.

### Key Functionalities

- Catalog Management: Organize, update, and manage product details.
- Order Tracking: Monitor and update order statuses in real-time.
- Customer Records: Store customer profiles and purchase history.
- Data Security: Ensure data consistency and protection.

- Scalability: Handle growing data needs reliably.

## 2. Entity-Relationship (ER) Diagram

*Create an ER diagram for the database using tools such as Lucidchart, MS Visio, or hand-drawn diagrams.*

*Include the following:*

- *Entities, attributes, and relationships.*
- *Primary and foreign keys.*
- *Weak entities if applicable.*

### **Entities & attributes:**

Listing: listing ID, listing date, item id, item name, listing status, seller id, item listing price, item quantity, item description

Transactions: transaction id, transaction created date, quantity, listing id, item\_id, buyer\_id, payment id, checkout status, item final sales price, tax, shipping id

Users : User id, user full name, user registration date, seller document signed date, Date of birth, user address, user phone number, ID verification status, user status, is company seller, company name,

Item: Item id, category id, category name, item name

Payment instance: payment id, payment type, payment created date, payment status

Shipping: shipping id, shipping addr1, shipping addr2, shipping city, shipping state, shipping country, shipping zip code, shipping status, shipping provider, tracking number

### **Relationship:**

One listing can trigger many transactions and one transaction includes one listing. (1: M)

One listing can include one seller and one seller can list many listings. (M:1)

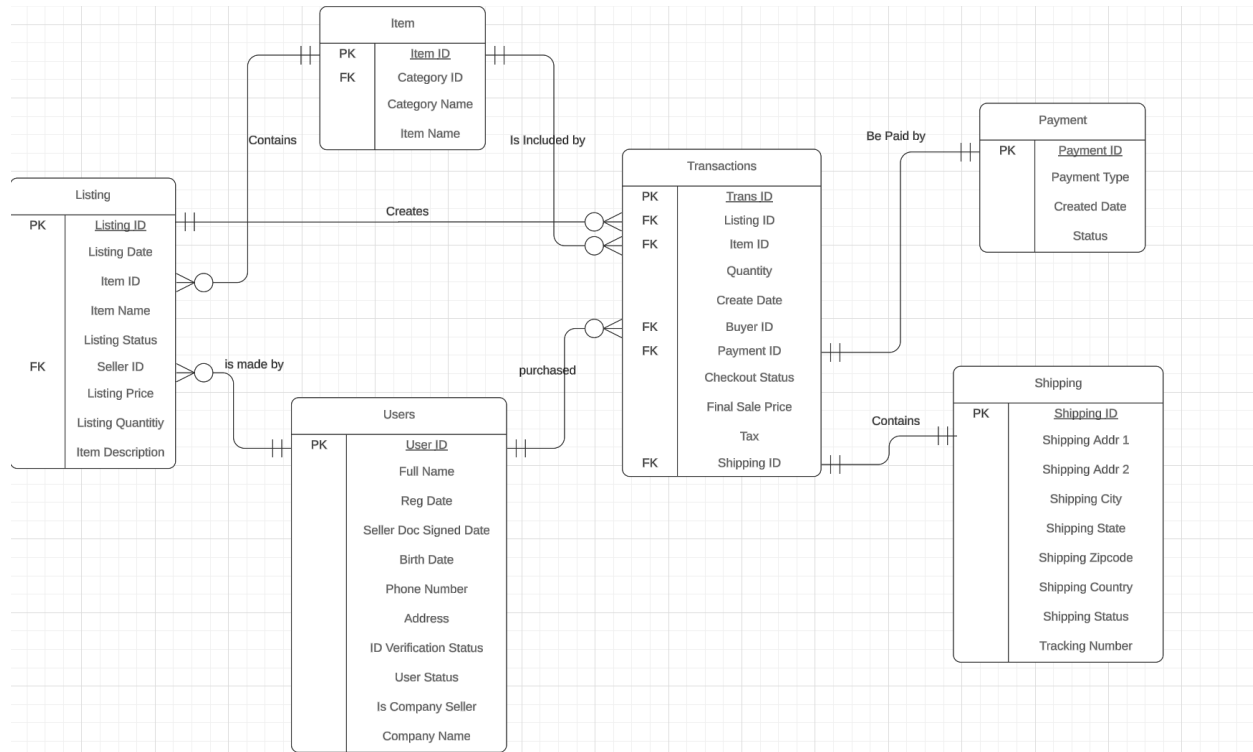
One listing can list one type of item and one item can be part of many listings. (M:1)

One transaction can have one buyer and one buyer can create many transactions (M:1)

One transaction can have one payment instance and one payment instance can be used in one transaction (1:1)

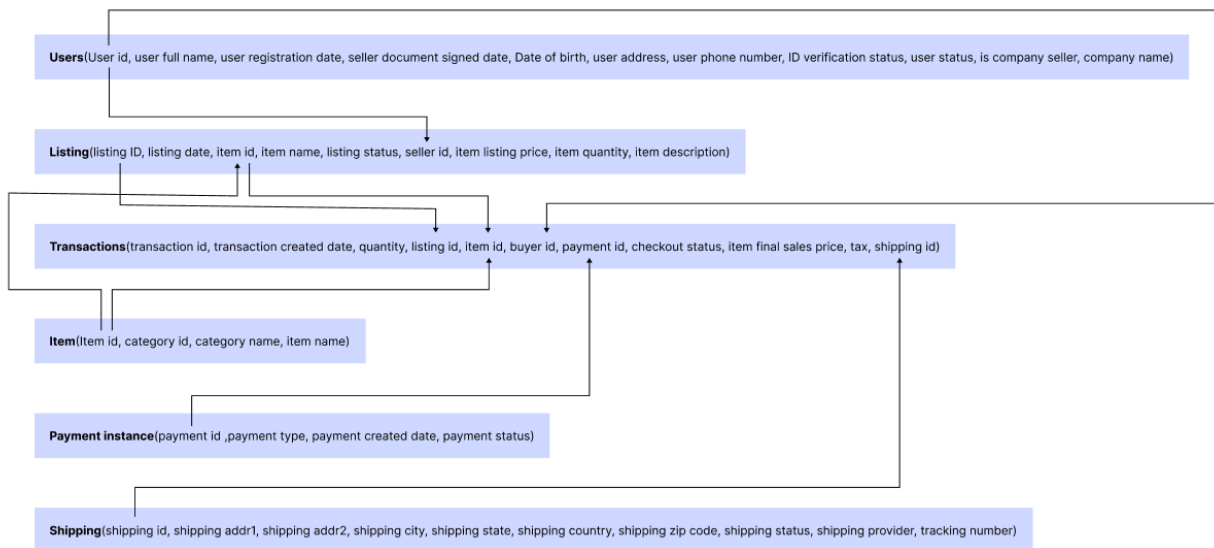
One transaction can have one shipping and one shipping can be part of one transaction (1:1)

### **Diagram:**



### 3. Relational Schema

- Convert the ER diagram into a relational schema.
- Clearly indicate the tables, their attributes, primary keys, and foreign keys.



### 4. Normalization

- Show the normalization process for one or two tables (e.g., First Normal Form to Third Normal Form).

The original item table works as follows:

Item ID	Item Name	Category ID	Category Name
I001	Smartphone	C001	Electronics
I002	TV	C001	Electronics
I003	Coffee Maker	C003	Appliances

### **1NF**

Item ID	Item Name	Category ID	Category Name
I001	Smartphone	C001	Electronics
I002	TV	C001	Electronics
I003	Coffee Maker	C003	Appliances

### **2NF**

#### **Item Table**

Item ID	Item Name
I001	Smartphone
I002	TV
I003	Coffee Maker

#### **Category Table**

Item ID	Category ID	Category Name
I001	C001	Electronics
I002	C001	Electronics
I003	C003	Appliances

### **3NF**

#### **Item Table**

Item ID	Item Name
---------	-----------

I001	Smartphone
I002	TV
I003	Coffee Maker

### Category Table

Category ID	Category Name
C001	Electronics
C003	Appliances

### Item\_Category Table

Item ID	Category ID
I001	C001
I002	C001
I003	C003

## 5. SQL Implementation

*Provide the SQL code for the following tasks:*

- *Creating tables with primary and foreign keys.*
- *Inserting at least 5 rows of data into each table.*
- *Performing basic SELECT queries (e.g., retrieving specific rows based on conditions).*
- *Joining two or more tables to retrieve data.*
- *Using INSERT, UPDATE, and DELETE commands to manipulate data.*

### Code:

```
CREATE TABLE Users (
    UserID VARCHAR(50) PRIMARY KEY, -- Unique ID for each user
    FullName VARCHAR(50), -- User full name
    RegDate TIMESTAMP, -- User registration date
    SellerDocSignedDate TIMESTAMP, -- Timestamp the seller
    agreement was signed
    BirthDate DATE, -- The user's birth date
    PhoneNumber VARCHAR(50), -- User's contact number
    Address VARCHAR(300), -- User's physical address
    IDVerificationStatus BOOLEAN, -- Whether the user has verified
    their ID
    UserStatus VARCHAR(50), -- Status of the user
```

```

        IsCompanySeller BOOLEAN, -- Whether the user is a company
seller
        CompanyName VARCHAR(100) -- The name of the company
    );
CREATE TABLE Items (
    ItemID VARCHAR(50) PRIMARY KEY, -- Unique ID for each type of
item
    CategoryID VARCHAR(50), -- Unique ID for each category
    CategoryName VARCHAR(100), -- Name of the category
    ItemName VARCHAR(100) -- Name of the item
);
CREATE TABLE Listings (
    ListingID VARCHAR(50) PRIMARY KEY, -- Unique ID for each
listing
    ListingDate TIMESTAMP, -- Timestamp the listing was created
    ListingStatus VARCHAR(50), -- Current status of the listing
    SellerID VARCHAR(50), -- Used to connect to the Users table
    ListingPrice DECIMAL(10, 2), -- Price of the listed item
    ListingQuantity INTEGER, -- Quantity of the item listed
    ItemDescription VARCHAR(100), -- Detailed description of the
item
    ItemID VARCHAR(50), -- Connects to the Items table
    ItemName VARCHAR(100), -- Name of the item
    FOREIGN KEY (SellerID) REFERENCES Users(UserID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID)
);
CREATE TABLE Payments (
    PaymentID VARCHAR(50) PRIMARY KEY, -- Unique ID for each
payment
    PaymentType VARCHAR(50), -- Type of payment method
    CreatedDate TIMESTAMP, -- Timestamp the payment was created
    Status VARCHAR(50) -- Current status of the payment
);
CREATE TABLE Shipping (
    ShippingID VARCHAR(50) PRIMARY KEY, -- Unique ID for each
shipping
    ShippingAddr1 VARCHAR(200), -- Address line 1 of shipping
    ShippingAddr2 VARCHAR(200), -- Address line 2 of shipping
    ShippingCity VARCHAR(50), -- City shipped to
    ShippingState VARCHAR(50), -- State shipped to
    ShippingZipcode VARCHAR(50), -- Postal code for delivery
    ShippingCountry VARCHAR(50), -- Country for delivery
    ShippingStatus VARCHAR(50), -- Current status of the shipping
    TrackingNumber VARCHAR(100) -- Tracking number of the shipping
);
CREATE TABLE Transactions (
    TransID VARCHAR(50) PRIMARY KEY, -- Unique ID for each
transaction
    ListingID VARCHAR(50), -- Connects to the Listing table
    ItemID VARCHAR(50), -- Connects to the Item table
    CreateDate TIMESTAMP, -- Timestamp the transaction was created

```

```

    BuyerID VARCHAR(50), -- Connects to the Users table
    PaymentID VARCHAR(50), -- Connects to the Payment table
    CheckoutStatus VARCHAR(50), -- Status of the transaction
    FinalSalesPrice DECIMAL(10, 2), -- Final sales price
    Tax DECIMAL(10, 2), -- Tax paid by the buyer
    ShippingID VARCHAR(50), -- Connects to the Shipping table
    Quantity INTEGER, -- Quantity of the item checked out
    FOREIGN KEY (ListingID) REFERENCES Listing(ListingID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID),
    FOREIGN KEY (BuyerID) REFERENCES Users(UserID),
    FOREIGN KEY (PaymentID) REFERENCES Payment(PaymentID),
    FOREIGN KEY (ShippingID) REFERENCES Shipping(ShippingID)
);
INSERT INTO Users (UserID, FullName, RegDate, SellerDocSignedDate,
BirthDate, PhoneNumber, Address, IDVerificationStatus, UserStatus,
IsCompanySeller, CompanyName)
VALUES
('U001', 'Alice Seller', '2023-01-01', '2023-01-02', '1990-05-01',
'1234567890', '123 Main St, Springfield, IL', 1, 'Active', 1,
'Alice Inc.'),
('U002', 'Bob Buyer', '2023-02-01', NULL, '1985-10-15',
'9876543210', '456 Market St, San Francisco, CA', 0, 'Active', 0,
NULL),
('U003', 'Charlie Seller', '2023-03-10', '2023-03-12',
'1980-07-20', '5551234567', '789 Oak St, Boston, MA', 1, 'Active',
1, 'Charlie Electronics'),
('U004', 'David Buyer', '2023-04-01', NULL, '1992-11-25',
'6789101112', '101 Maple St, Los Angeles, CA', 1, 'Active', 0,
NULL),
('U005', 'Eve Seller', '2023-05-15', '2023-05-16', '1995-02-10',
'1122334455', '202 Pine St, Miami, FL', 0, 'Inactive', 1, 'Eve
Fashion');
INSERT INTO Items (ItemID, ItemName, CategoryID, CategoryName)
VALUES
('I001', 'Smartphone', 'C001', 'Electronics'),
('I002', 'T-shirt', 'C002', 'Clothing'),
('I003', 'Coffee Maker', 'C003', 'Appliances'),
('I004', 'Laptop', 'C001', 'Electronics'),
('I005', 'Blender', 'C003', 'Appliances');
INSERT INTO Payments (PaymentID, PaymentType, CreatedDate, Status)
VALUES
('P001', 'Credit Card', '2023-06-01', 'Completed'),
('P002', 'PayPal', '2023-06-05', 'Completed'),
('P003', 'Bank Transfer', '2023-06-10', 'Pending'),
('P004', 'Credit Card', '2023-06-12', 'Completed'),
('P005', 'PayPal', '2023-06-15', 'Cancelled');
INSERT INTO Listings (ListingID, ListingDate, ListingStatus,
SellerID, ListingPrice, ListingQuantity, ItemDescription, ItemID,
ItemName)
VALUES
('L001', '2023-05-01', 'Active', 'U001', 699.99, 10, 'Brand new

```

```

smartphone with high-end features', 'I001', 'Smartphone'),
('L002', '2023-05-05', 'Active', 'U003', 19.99, 50, 'Cotton t-shirt
in various sizes', 'I002', 'T-shirt'),
('L003', '2023-05-10', 'Inactive', 'U004', 129.99, 5, 'High-quality
coffee maker with stainless steel body', 'I003', 'Coffee Maker'),
('L004', '2023-05-12', 'Active', 'U002', 899.99, 3, 'Laptop with
fast processor and high storage', 'I004', 'Laptop'),
('L005', '2023-05-15', 'Active', 'U005', 49.99, 20, 'Blender with
multiple speed settings', 'I005', 'Blender');
INSERT INTO Shipping (ShippingID, ShippingAddr1, ShippingAddr2,
ShippingCity, ShippingState, ShippingZipcode, ShippingCountry,
ShippingStatus, TrackingNumber)

```

#### VALUES

```

('S001', '123 Main St', 'Apt 1', 'Springfield', 'IL', '62701',
'USA', 'Shipped', '1Z999AA10123456784'),
('S002', '456 Market St', 'Suite 200', 'San Francisco', 'CA',
'94105', 'USA', 'Delivered', '1Z999AA10123456785'),
('S003', '789 Oak St', 'Unit 4B', 'Boston', 'MA', '02118', 'USA',
'In Transit', '1Z999AA10123456786'),
('S004', '101 Maple St', 'Floor 2', 'Los Angeles', 'CA', '90001',
'USA', 'Shipped', '1Z999AA10123456787'),
('S005', '202 Pine St', 'Apt 9', 'Miami', 'FL', '33101', 'USA',
'Returned', '1Z999AA10123456788');

```

```

INSERT INTO Transactions (TransID, ListingID, ItemID, CreateDate,
BuyerID, PaymentID, CheckoutStatus, FinalSalesPrice, Tax,
ShippingID, Quantity)

```

#### VALUES

```

('T001', 'L001', 'I001', '2023-06-01', 'U002', 'P001', 'Completed',
699.99, 20.99, 'S001', 1),
('T002', 'L002', 'I002', '2023-06-05', 'U004', 'P002', 'Completed',
19.99, 2.00, 'S002', 2),
('T003', 'L003', 'I003', '2023-06-10', 'U005', 'P003', 'Pending',
129.99, 7.50, 'S003', 1),
('T004', 'L004', 'I004', '2023-06-12', 'U001', 'P004', 'Completed',
899.99, 30.00, 'S004', 1),
('T005', 'L005', 'I005', '2023-06-15', 'U003', 'P005', 'Cancelled',
49.99, 3.00, 'S005', 1);

```

```

ALTER TABLE Transactions ADD CONSTRAINT fk_listing FOREIGN KEY
(ListingID) REFERENCES Listings(ListingID);

```

```

ALTER TABLE Transactions ADD CONSTRAINT fk_item FOREIGN KEY
(ItemID) REFERENCES Items(ItemID);

```

```

ALTER TABLE Transactions ADD CONSTRAINT fk_user FOREIGN KEY
(BuyerID) REFERENCES Users(UserID);

```

```

ALTER TABLE Transactions ADD CONSTRAINT fk_payment FOREIGN KEY
(PaymentID) REFERENCES Payments(PaymentID);

```

```

ALTER TABLE Transactions ADD CONSTRAINT fk_shipping FOREIGN KEY
(ShippingID) REFERENCES Shipping(ShippingID);

```

```

ALTER TABLE Listings ADD CONSTRAINT fk_item_listing FOREIGN KEY

```



```

(ItemID) REFERENCES Items(ItemID);
ALTER TABLE Listings ADD CONSTRAINT fk_user_listing FOREIGN KEY
(SellerID) REFERENCES Users(UserID);

-- Select all data from the Users table
-- This will give you every piece of information available for all
users
SELECT * FROM Users;

-- Select just the FullName and PhoneNumber of all users
-- If you only need contact details for users, this query will pull
that specific data
SELECT FullName, PhoneNumber FROM Users;

-- Retrieve all transactions where the BuyerID is 'U002'
-- This can help us see what 'U002' bought, when, and for how much
SELECT * FROM Transactions WHERE BuyerID = 'U002';

-- Get all active listings where the seller's ID is 'U001'
-- This will show you which items Seller 'U001' has listed and
their prices
SELECT * FROM Listings WHERE ListingStatus = 'Active' AND SellerID
= 'U001';

-- Fetch all items, but this time order them by ItemName
-- Useful if you want to see the list sorted alphabetically by item
name
SELECT * FROM Items ORDER BY ItemName;

-- Only show the first 3 rows from the Payments table
-- This is a quick way to get a sneak peek of the payment records
SELECT * FROM Payments LIMIT 3;

-- Joining Tables

-- Get listing details along with the item name by joining Listings
and Items tables on ItemID
-- This will let you see which items belong to each listing
SELECT Listings.ListingID, Listings.ListingPrice, Items.ItemName
FROM Listings
INNER JOIN Items ON Listings.ItemID = Items.ItemID;

-- Get a list of users and their listings, even if they have no
listings
-- This is a left join, meaning we show all users even if they
don't have items listed
SELECT Users.UserID, Listings.ListingID, Listings.ListingPrice
FROM Users
LEFT JOIN Listings ON Users.UserID = Listings.SellerID;

-- Fetch transaction details, buyer names, and item names

```

```

-- This gives a complete view of who bought what, and at what
price, from the transactions
SELECT Transactions.TransID, Users.FullName, Items.ItemName,
Transactions.FinalSalesPrice
FROM Transactions
JOIN Users ON Transactions.BuyerID = Users.UserID
JOIN Items ON Transactions.ItemID = Items.ItemID;

-- Using INSERT, UPDATE, and DELETE Commands

-- Add a new user to the Users table
-- This is how you'd create a new user in the system, for example a
new buyer
INSERT INTO Users (UserID, FullName, RegDate, SellerDocSignedDate,
BirthDate, PhoneNumber, Address, IDVerificationStatus, UserStatus,
IsCompanySeller, CompanyName)
VALUES ('U006', 'Frank Buyer', '2023-06-25', NULL, '1993-03-12',
'1239874560', '500 Broadway St, New York, NY', 0, 'Active', 0,
NULL);

-- Update a user's phone number
-- This query will change the contact info of 'U002' in the system
UPDATE Users
SET PhoneNumber = '5551234567'
WHERE UserID = 'U002';

-- Delete a user from the Users table
-- If you need to remove 'U006' from the system, this is the
command you'd use
DELETE FROM Users WHERE UserID = 'U006';

-- More Advanced Queries

-- Count how many listings are currently active
-- This is useful if you want to see the number of products
available for sale right now
SELECT COUNT(*) AS ActiveListings FROM Listings WHERE ListingStatus
= 'Active';

-- Calculate total sales for each item
-- This query totals up the sales for every item, helping to track
how much each item has sold for
SELECT ItemID, SUM(FinalSalesPrice) AS TotalSales
FROM Transactions
GROUP BY ItemID;

-- Show the total sales for each seller, but only those who have
made over $1000 in sales
-- A great way to identify top-performing sellers
SELECT SellerID, SUM(FinalSalesPrice) AS TotalSales
FROM Transactions

```

```

GROUP BY SellerID
HAVING SUM(FinalSalesPrice) > 1000;

-- Subqueries

-- Find users who bought items worth more than $100
-- This query first finds all the transactions over $100, then
-- fetches the buyer information for those
SELECT FullName
FROM Users
WHERE UserID IN (SELECT BuyerID FROM Transactions WHERE
FinalSalesPrice > 100);

-- Find listings priced higher than the average price in the
-- Listings table
-- This query compares the price of each listing to the average
-- price of all listings
SELECT *
FROM Listings
WHERE ListingPrice > (SELECT AVG(ListingPrice) FROM Listings);

```

## 6. Database Integrity and Security

- Briefly describe how integrity constraints (e.g., primary keys, foreign keys) and security measures (e.g., user authentication) are implemented in the database.

We combined primary keys, foreign keys, unique constraints and check constraints to ensure that the database maintains data integrity, preventing invalid and duplicate data and maintaining the relationships between different entities.

- **Primary keys:** Each table has a Primary Key constraint added to a column that serves as its unique identifier.
- **Foreign keys:** Foreign keys build relationships between tables by linking a column in one table to the primary key of another, ensuring referential integrity and consistent data across related tables.
- **Unique constraints:** Unique constraints guarantee that specific columns contain only unique values, preventing duplicate entries.
- **Check constraints:** Check constraints validate data by restricting the range of allowable values for a column.

To ensure security, roles and users are created with distinct permissions. Each user is assigned a unique login username and password. The system will clearly display the

roles assigned to each user, specifying the queries or actions they are authorized to perform, as well as the additional privileges available to admins within the group. For enhanced data security, we create a table to store encrypted phone numbers, manage the insertion of encrypted data, and facilitate the secure retrieval and decryption of the data.

## Database Integrity

```
**1. Primary Keys**
-- Primary keys uniquely identify each record in the table.
-- This ensures no duplicate rows for key entities.

ALTER TABLE Users
  ADD CONSTRAINT PK_Users PRIMARY KEY (UserID); -- Uniquely
identifies each user.

ALTER TABLE Shipping
  ADD CONSTRAINT PK_Shipping PRIMARY KEY (ShippingID); -- Uniquely
identifies each shipping record.

ALTER TABLE Items
  ADD CONSTRAINT PK_Items PRIMARY KEY (ItemID); -- Uniquely
identifies each item.

ALTER TABLE Categories
  ADD CONSTRAINT PK_Categories PRIMARY KEY (CategoryID); --
Uniquely identifies each category.

-- **2. Foreign Keys**
-- Foreign keys establish relationships between tables, ensuring
referential integrity.
-- Example: Ensuring that the UserID in the Shipping table exists
in the Users table.

ALTER TABLE Shipping
  ADD CONSTRAINT FK_Shipping_UserID FOREIGN KEY (UserID) REFERENCES
Users(UserID); -- Links shipping data to users.

ALTER TABLE Items
  ADD CONSTRAINT FK_Items_CategoryID FOREIGN KEY (CategoryID)
REFERENCES Categories(CategoryID); -- Links items to their
categories.

-- **3. Unique Constraints**
-- Unique constraints prevent duplicate entries in specific
columns.
-- Example: Ensuring that TrackingNumber in the Shipping table is
```

```

always unique.

ALTER TABLE Shipping
  ADD CONSTRAINT UQ_TrackingNumber UNIQUE (TrackingNumber);  --
Ensures no duplicate tracking numbers.

-- **4. Check Constraints**
-- Check constraints enforce valid data entry for specific
columns.
-- Example: Ensuring that IDVerificationStatus is either 0 or 1
(binary status).

ALTER TABLE Users
  ADD CONSTRAINT CHK_IDVerificationStatus CHECK
(IDVerificationStatus IN (0, 1));  -- Only allows 0 or 1 for
verification.

ALTER TABLE Shipping
  ADD CONSTRAINT CHK_ShippingStatus CHECK (ShippingStatus IN
('Shipped', 'Delivered', 'In Transit', 'Returned'));  -- Restricts
shipping status to specific values.

```

## Database Authentication and Role based access control

```

create role Group6_user;
grant select, update, insert on mydatabase.* To Group6_user;
create user 'Lili' identified by 'Lili1234' default role
'Group6_user';
create user 'Boyan' identified by 'Boyan1234' default role
'Group6_user';
create user 'Jonathan' identified by 'Jonathan1234' default role
'Group6_user';
create user 'Junzhu' identified by 'Junzhu1234' default role
'Group6_user';

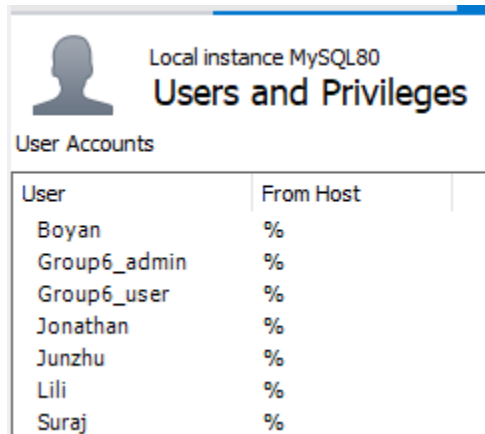
create role Group6_admin;
grant create, select, update, insert, delete on mydatabase.* To
Group6_admin;
create user 'Suraj' identified by 'Suraj1234' default role
'Group6_admin'

Test role-based access control
-- Log in as Group6_user and attempt to delete data (this should
fail)
-- Log in as Group6_admin and delete a record (this should succeed)

DELETE FROM Items WHERE ItemID = 'I001';

```

## Group 6 Users



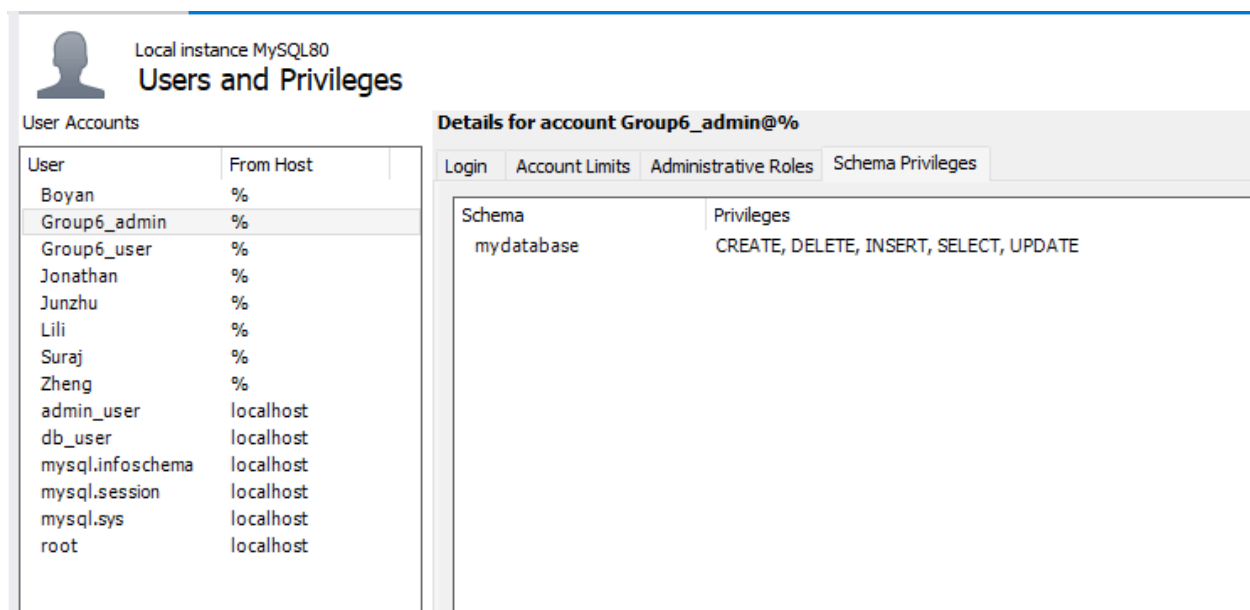
Local instance MySQL80

### Users and Privileges

User Accounts

User	From Host
Boyan	%
Group6_admin	%
Group6_user	%
Jonathan	%
Junzhu	%
Lili	%
Suraj	%

## Group6\_admin Privilege



Local instance MySQL80

### Users and Privileges

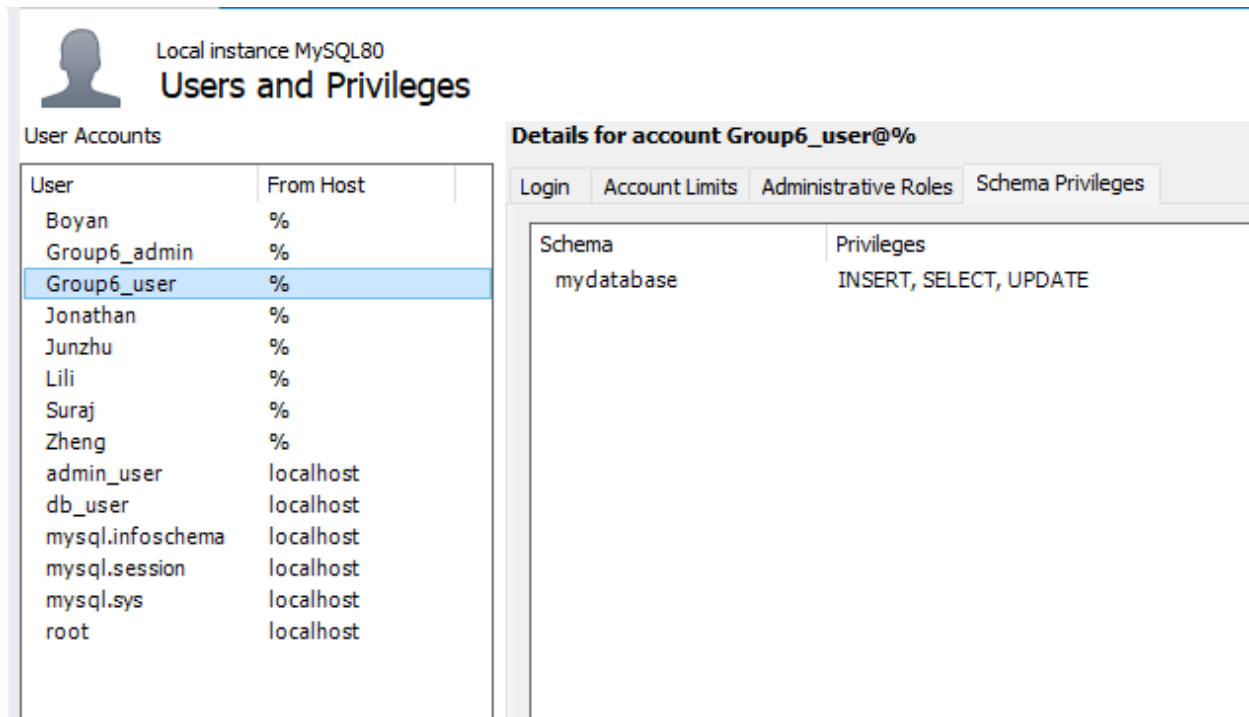
User Accounts

User	From Host
Boyan	%
Group6_admin	%
Group6_user	%
Jonathan	%
Junzhu	%
Lili	%
Suraj	%
Zheng	%
admin_user	localhost
db_user	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

#### Details for account Group6\_admin@%

Login	Account Limits	Administrative Roles	Schema Privileges
Schema			Privileges
mydatabase			CREATE, DELETE, INSERT, SELECT, UPDATE

## Group6\_user Privilege:



The screenshot shows the MySQL 'Users and Privileges' window for a local instance of MySQL 8.0. On the left, the 'User Accounts' table lists various users, with 'Group6\_user' selected. On the right, the 'Details for account Group6\_user@%' are shown, with the 'Schema Privileges' tab active. This tab displays that the 'mydatabase' schema has privileges for INSERT, SELECT, and UPDATE.

User	From Host
Boyan	%
Group6_admin	%
Group6_user	%
Jonathan	%
Junzhu	%
Lili	%
Suraj	%
Zheng	%
admin_user	localhost
db_user	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Schema	Privileges
mydatabase	INSERT, SELECT, UPDATE

## Data Security

```
CREATE TABLE EncryptedPhoneNumbers (
    UserID VARCHAR(10),
    EncryptedPhoneNumber VARBINARY(255), -- Encrypted phone number
    storage
    CONSTRAINT FK_EncryptedPhoneNumbers_UserID FOREIGN KEY (UserID)
    REFERENCES Users (UserID)
);

-- Add sample encrypted data (encryption would typically be done in
the application layer)

INSERT INTO EncryptedPhoneNumbers (UserID, EncryptedPhoneNumber)
VALUES ('U001', AES_ENCRYPT('1234567890', 'encryption_key')); --
Replace 'encryption_key' with your actual key

-- Verify encryption
-- Retrieve the encrypted phone number

SELECT * FROM EncryptedPhoneNumbers;

-- Verify encryption
-- Retrieve the encrypted phone number
```

```

SELECT EncryptedPhoneNumber FROM EncryptedPhoneNumbers WHERE UserID
= 'U001';

-- Decrypt the phone number

SELECT AES_DECRYPT(EncryptedPhoneNumber, 'encryption_key') AS
DecryptedPhoneNumber FROM EncryptedPhoneNumbers WHERE UserID =
'U001';

```

## Phone number encrypted using encryption key

The screenshot shows a SQL Server Enterprise Manager interface. In the background, a query window contains the following SQL code:

```

25 -- Verify encryption
26 -- Retrieve the encrypted phone number
27 • SELECT * FROM EncryptedPhoneNumbers;
28 -- Verify encryption
29 -- Retrieve the encrypted phone number
30 • SELECT EncryptedPhoneNumber FROM EncryptedPhoneNumbers WHERE UserID = 'U001';
31
32 -- Decrypt the phone number
33 • SELECT AES_DECRYPT(EncryptedPhoneNumber, 'encryption_key') AS DecryptedPhoneNumber

```

The results grid shows the following data:

EncryptedPhoneNumber
01.00

In the foreground, a dialog box titled "Edit Data for EncryptedPhoneNumber (VARBINARY)" is open. It shows a binary value: 00-2"W>000D\*5(00)+00. The data length is 16 bytes.

## Phone number Decrypted using encryption Key

The screenshot shows a SQL Server Enterprise Manager interface. In the background, a query window contains the following SQL code:

```

31
32 -- Decrypt the phone number
33 • SELECT AES_DECRYPT(EncryptedPhoneNumber, 'encryption_key') AS DecryptedPhoneNumber FROM EncryptedPhoneNumbers WHERE UserID = 'U001';

```

The results grid shows the following data:

DecryptedPhoneNumber
01.00

In the foreground, a dialog box titled "Edit Data for DecryptedPhoneNumber (VARBINARY)" is open. It shows a text value: 1234567890. The data length is 10 bytes.

The Action Output pane at the bottom shows the following actions:

#	Time	Action
29	22:09:58	create user 'Li'
30	22:09:58	create user 'Bo'
31	22:09:58	create user 'Jo'
32	22:09:58	create user 'Ju'
33	22:14:14	SELECT Encry
34	22:15:02	SELECT AES