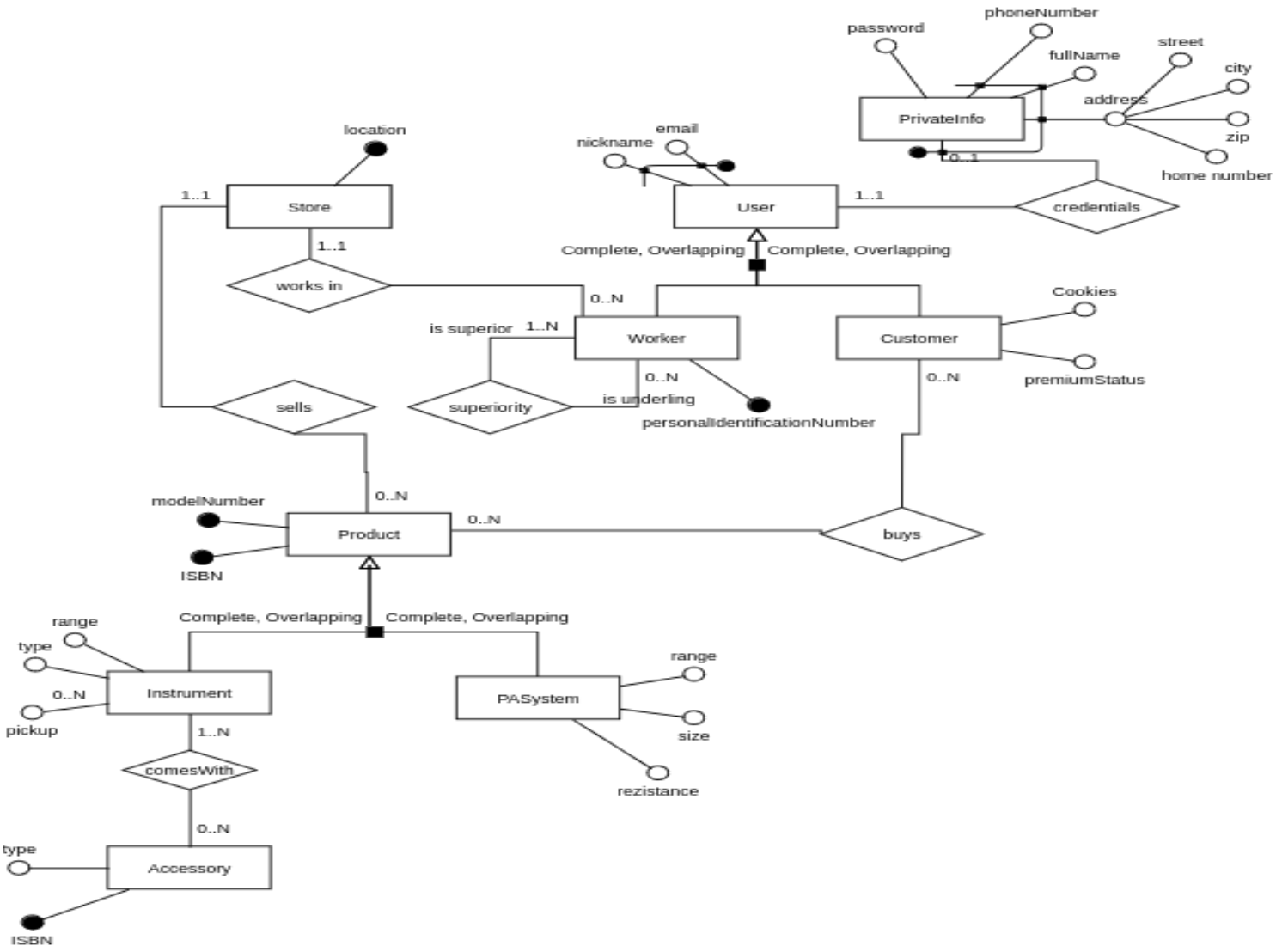
Music Store database cp03

ER diagram:



Relational model:

as a picture:

- User (userID, nickname, email)
- PrivateInfo (fullName, homeNumber, street, city, zip, userNickname, userMail, phoneNumber, password)
  - FK: (userNickname, userMail) ⊆ User(nickname, email)
- WorkerUser (workerId, personalIdentificationNumber, userNickname, userMail, location)
  - FK: WorkerUser(userNickname, userMail) ⊆ User(nickname, email)
  - FK: WorkerUser(location) ⊆ Store(location)
- superiority (superiorityId, superiorId, underlingId,isSuperior, isUnderling)
  - FK: superiority(isSuperior) ⊆ WorkerUser(personalIdentificationNumber)
  - FK: superiority(isUnderling) ⊆ WorkerUser(personalIdentificationNumber)
  - FK: superiority(superiorId) ⊆ WorkerUser(workerId)
  - FK: superiority(underlingId) ⊆ WorkerUser(workerId)
- CustomerUser (customerID, userNickname, userMail, cookies, premiumStatus)
  - FK: (nickname, email) ⊆ User(nickname, email)
- buys (purchaseId, ISBN, modelNumber, nickname, email)
  - FK: buys(ISBN, modelNumber) ⊆ Product(ISBN, modelNumber)
  - FK: buys(nickname, email) ⊆ CustomerUser(nickname, email)
- Store (storeId, location)
- Product (productID, ISBN, modelNumber, location)
  - FK: Product(location) ⊆ Store(location)
- InstrumentProduct (instrumentId, modelNumber, ISBN, range, type)
  - FK: InstrumentProduct(ISBN, modelNumber) ⊆ Product(ISBN, modelNumber)
- Pickup(pickupId, modelNumber, name)
- InstrumentPickup (installedOnId, instrumentId, pickupID, modelNumber, pickupModelNumber)
  - FK: InstrumentPickup(modelNumber) ⊆ InstrumentProduct(modelNumber)
  - FK: InstrumentPickup(pickupModelNumber) ⊆ Pickup(modelNumber)
  - FK: InstrumentPickup(instrumentId) ⊆ InstrumentProduct(instrumentId)
  - FK: InstrumentPickup(pickupID) ⊆ Pickup(pickupID)
- PASystemProduct (systemId, ISBN, range, type, resistance)
  - FK: PASystemProduct(ISBN) ⊆ Product(ISBN)
- Accessory (accessoryId, ISBN, type, comesWith)
  - FK: Accessory(comesWith) ⊆ InstrumentProduct(modelNumber)

as a html code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>RM for DBS</title>
</head>

<body>

<h1>Relational model for music store database</h1>

<ul>
  <li> User (<u>userID</u>, <u>nickname, email</u>)</li>
</ul>

<ul>
  <li> PrivateInfo (<u>fullName, homeNumber, street, city, zip, userNickname, userMail, phoneNumber</u>, password)
    <ul>
```

```
        <li>FK: (userNickname, userMail) ⊆ User(nickname, email)</li>
      </ul>
    </li>
</ul>

<ul>
  <li> WorkerUser (<u>workerId</u>, <u>personalIdentificationNumber</u>, <u>userNickname, userMail</u>, location)
    <ul>
      <li>FK: WorkerUser(userNickname, userMail) ⊆ User(nickname, email)</li>
      <li>FK: WorkerUser(location) ⊆ Store(location)</li>
    </ul>
  </li>

  <li> superiority (<u>superiorityId</u>, superiorId, underlingId,<u>isSuperior, isUnderling</u>)
    <ul>
      <li>FK: superiority(isSuperior) ⊆ WorkerUser(personalIdentificationNumber)</li>
      <li>FK: superiority(isUnderling) ⊆ WorkerUser(personalIdentificationNumber)</li>
      <li>FK: superiority(superiorId) ⊆ WorkerUser(workerId)</li>
      <li>FK: superiority(underlingId) ⊆ WorkerUser(workerId)</li>
    </ul>
  </li>
</ul>

<ul>
  <li> CustomerUser (<u>customerID</u>, <u>userNickname, userMail</u>, cookies, premiumStatus)
    <ul>
      <li>FK: (nickname, email) ⊆ User(nickname, email)</li>
    </ul>
  </li>

  <li> buys (<u>purchaseId</u>, <u>ISBN, modelNumber</u>, <u>nickname, email</u>)
    <ul>
      <li>FK: buys(ISBN, modelNumber) ⊆ Product(ISBN, modelNumber)</li>
      <li>FK: buys(nickname, email) ⊆ CustomerUser(nickname, email)</li>
    </ul>
  </li>
</ul>

<ul>
  <li> Store (<u>storeId</u>, <u>location</u>)</li>
</ul>

<ul>
  <li> Product (<u>productID</u>, <u>ISBN</u>, <u>modelNumber</u>, location)
    <ul>
      <li>FK: Product(location) ⊆ Store(location)</li>
    </ul>
  </li>
</ul>

<ul>
  <li> InstrumentProduct (<u>instrumentId</u>, <u>modelNumber, ISBN</u>, range, type)
    <ul>
```

```
        <li>FK: InstrumentProduct(ISBN, modelNumber) ⊆ Product(ISBN, modelNumber)</li>
      </ul>
    </li>

    <li>Pickup(<u>pickupId</u>, <u>modelNumber</u>, name)</li>

    <li> InstrumentPickup (<u>installedOnId</u>, instrumentId, pickupID, modelNumber, pickupModelNumber)
      <ul>
        <li>FK: InstrumentPickup(modelNumber) ⊆ InstrumentProduct(modelNumber)</li>
        <li>FK: InstrumentPickup(pickupModelNumber) ⊆ Pickup(modelNumber)</li>
        <li>FK: InstrumentPickup(instrumentId) ⊆ InstrumentProduct(instrumentId)</li>
        <li>FK: InstrumentPickup(pickupID) ⊆ Pickup(pickupID)</li>
      </ul>
    </li>
  </ul>

  <ul>
    <li> PASystemProduct (<u>systemId</u>, <u>ISBN</u>, range, type, resistance)
      <ul>
        <li>FK: PASystemProduct(ISBN) ⊆ Product(ISBN)</li>
      </ul>
    </li>
  </ul>

  <ul>
    <li> Accessory (<u>accessoryId</u>, <u>ISBN</u>, type, comesWith)
      <ul>
        <li>FK: Accessory(comesWith) ⊆ InstrumentProduct(modelNumber)</li>
      </ul>
    </li>
  </ul>

  </body>
  </html>
```

all SQL queries:

table creation:

```sql
-- Creating tables
=============================================================================================
CREATE TABLE Users (
  userID SERIAL UNIQUE,
  nickname VARCHAR(50) NOT NULL,
   email VARCHAR(50) NOT NULL
      CHECK ( email LIKE '%@%'),
   PRIMARY KEY (nickname, email)
);



CREATE TABLE PrivateInfo (
```

```
    userNickname VARCHAR(50) NOT NULL,
    userMail VARCHAR(50) NOT NULL,
    fullName VARCHAR(50),
    homeNumber VARCHAR(50),
    street VARCHAR(50),
    city VARCHAR(50),
    zip VARCHAR(50),
    phoneNumber VARCHAR(12),
    password VARCHAR(50),
    PRIMARY KEY (userNickname, userMail, fullName, street, city, zip, phoneNumber),
    FOREIGN KEY (userNickname, userMail) REFERENCES Users(nickname, email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);



CREATE TABLE WorkerUser (
    workerId SERIAL UNIQUE,
    personalIdentificationNumber VARCHAR(50) UNIQUE NOT NULL,
    userNickname VARCHAR(50) NOT NULL,
    userMail VARCHAR(50) NOT NULL,
    location VARCHAR(50),
    PRIMARY KEY (userNickname, userMail),
    FOREIGN KEY (userNickname, userMail) REFERENCES Users(nickname, email)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (location) REFERENCES Store(location)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);



CREATE TABLE CustomerUser (
    customerId SERIAL UNIQUE,
    userNickname VARCHAR(50) NOT NULL,
    userMail VARCHAR(50) NOT NULL,
    cookies INTEGER,
    premiumStatus BOOLEAN,
    PRIMARY KEY (userNickname, userMail),
    FOREIGN KEY (userNickname, userMail) REFERENCES Users(nickname, email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);



CREATE TABLE Superiority (
    superiorityId SERIAL PRIMARY KEY,
   superiorId INTEGER,
    underlingId INTEGER,
    isSuperior VARCHAR(50),
    isUnderling VARCHAR(50),
    FOREIGN KEY (isSuperior) REFERENCES WorkerUser(personalIdentificationNumber)
        ON DELETE CASCADE
```

```
    ON UPDATE CASCADE,
  FOREIGN KEY (isUnderling) REFERENCES WorkerUser(personalIdentificationNumber)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (superiorId) REFERENCES WorkerUser(workerId)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (underlingId) REFERENCES WorkerUser(workerId)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT check_notSuperiorToMyself CHECK (superiorId != underlingId),
  CONSTRAINT check_uniqueSuperiority UNIQUE (isSuperior, isUnderling)
);



CREATE TABLE buys (
  purchaseId SERIAL PRIMARY KEY,
  ISBN VARCHAR(50) NOT NULL,
  modelNumber VARCHAR(50) NOT NULL,
  userNickname VARCHAR(50) NOT NULL,
  userMail VARCHAR(50) NOT NULL,
  CONSTRAINT check_uniqueBuyingInfo UNIQUE (userNickname, userMail, ISBN),
  FOREIGN KEY (ISBN, modelNumber) REFERENCES Product(ISBN, modelNumber)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (userNickname, userMail) REFERENCES CustomerUser(userNickname, userMail)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);



CREATE TABLE Store (
  storeId SERIAL UNIQUE,
  location VARCHAR(50) PRIMARY KEY NOT NULL
);



CREATE TABLE Product (
  productID SERIAL UNIQUE,
  ISBN VARCHAR(50) NOT NULL,
  modelNumber VARCHAR(50) NOT NULL,
  location VARCHAR(50),
  PRIMARY KEY (ISBN, modelNumber),
  FOREIGN KEY (location) REFERENCES Store(location)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);



CREATE TABLE InstrumentProduct (
  instrumentId SERIAL UNIQUE,
  ISBN VARCHAR(50) UNIQUE NOT NULL,
  modelNumber VARCHAR(50) PRIMARY KEY NOT NULL,
```

```
    range VARCHAR(50) CHECK (range LIKE '%Hz-%Hz'),
    type VARCHAR(50) CHECK (type IN ('key', 'string', 'wind', 'percussion', 'special')),
    FOREIGN KEY (modelNumber, ISBN) REFERENCES Product(modelNumber, ISBN)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);



CREATE TABLE Pickup (
    pickupId SERIAL UNIQUE,
    modelNumber VARCHAR(50) PRIMARY KEY NOT NULL,
    name VARCHAR(50) NOT NULL,
    CONSTRAINT check_pickupType
        CHECK (name IN ('single coil', 'humbucker', 'piezo', 'lipstick', 'active', 'passive'))
);



CREATE TABLE InstrumentPickup (
    installedOnId SERIAL PRIMARY KEY,
    instrumentId INTEGER NOT NULL,
    pickupID INTEGER,
    modelNumber VARCHAR(50) NOT NULL,
    pickupModelNumber VARCHAR(50),
    FOREIGN KEY (modelNumber) REFERENCES InstrumentProduct(modelNumber)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (pickupModelNumber) REFERENCES Pickup(modelNumber)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (instrumentId) REFERENCES InstrumentProduct(instrumentId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (pickupID) REFERENCES Pickup(pickupID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT check_noPickupAtAll
        CHECK (
            (pickupID IS NULL AND pickupModelNumber IS NULL) OR
            (pickupID IS NOT NULL AND pickupModelNumber IS NOT NULL)
        )
);



CREATE TABLE includes(
    ISBN VARCHAR(50),
    modelNumber VARCHAR(50),
    PRIMARY KEY (ISBN, modelNumber),
    FOREIGN KEY (modelNumber) REFERENCES InstrumentProduct(modelNumber)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (ISBN) REFERENCES Accessory(ISBN)
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

```
);



CREATE TABLE PASystemProduct (
   systemId SERIAL UNIQUE,
   ISBN VARCHAR(50) PRIMARY KEY NOT NULL,
   modelNumber VARCHAR(50) UNIQUE NOT NULL,
   range VARCHAR(50) CHECK (range LIKE '%Hz-%Hz'),
   type VARCHAR(50) CHECK (type IN ('active', 'passive')),
   resistance VARCHAR(50) CHECK (resistance LIKE '%Ohm'),
   FOREIGN KEY (ISBN, modelNumber) REFERENCES Product(ISBN, modelNumber)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);



CREATE TABLE Accessory (
   accessoryId SERIAL UNIQUE,
   ISBN VARCHAR(50) PRIMARY KEY NOT NULL,
   type VARCHAR(50),
   comesWith VARCHAR(50) references InstrumentProduct(modelNumber)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);



-- New types for easy data insertion
================================================================================
CREATE TYPE WorkerUserType AS (
   personalIdentificationNumber VARCHAR(50),
   userNickname VARCHAR(50),
   userMail VARCHAR(50),
   location VARCHAR(50)
);



CREATE TYPE CustomerUserType AS (
   userNickname VARCHAR(50),
   userMail VARCHAR(50),
   cookies INTEGER,
   premiumStatus BOOLEAN
);



CREATE TYPE PASystemProductType AS (
   ISBN VARCHAR(50),
   modelNumber VARCHAR(50),
   range VARCHAR(50),
   type VARCHAR(50),
   resistance VARCHAR(50)
);
```

```
CREATE TYPE InstrumentProductType AS (
   ISBN VARCHAR(50),
   modelNumber VARCHAR(50),
   range VARCHAR(50),
   type VARCHAR(50)
);



CREATE TYPE privateInfoType AS (
   userNickname VARCHAR(50),
   userMail VARCHAR(50),
   fullName VARCHAR(50),
   homeNumber VARCHAR(50),
   street VARCHAR(50),
   city VARCHAR(50),
   zip VARCHAR(50),
   phoneNumber VARCHAR(10),
   password VARCHAR(50)
);



CREATE TYPE intPair AS (
   frst INTEGER,
   scnd INTEGER
);



-- Insert functions for easy and safe hereditary data insertion ============================================================
CREATE OR REPLACE FUNCTION insertWorkerUser(
   i_personalIdentificationNumber VARCHAR(50),
   i_userNickname VARCHAR(50),
   i_userMail VARCHAR(50),
   i_location VARCHAR(50)
) RETURNS VOID AS $$
DECLARE
   e_userId INTEGER;
BEGIN
   SELECT userID INTO e_userId FROM Users WHERE nickname = i_userNickname AND email = i_userMail;
   IF NOT FOUND THEN
      INSERT INTO Users(nickname, email) VALUES (i_userNickname, i_userMail);
   END IF;

  INSERT INTO WorkerUser(personalIdentificationNumber, userNickname, userMail, location)
     VALUES (i_personalIdentificationNumber, i_userNickname, i_userMail, i_location);

END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertWorkers(i_users WorkerUserType[])
RETURNS VOID AS $$
```

```
DECLARE
   userRecord WorkerUserType;
BEGIN
   FOREACH userRecord IN ARRAY i_users
   LOOP
      PERFORM insertWorkerUser(userRecord.personalIdentificationNumber,
                           userRecord.userNickname,
                           userRecord.userMail,
                           userRecord.location);
   END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertCustomerUser(
   i_userNickname VARCHAR(50),
   i_userMail VARCHAR(50),
   i_cookies INTEGER,
   i_premiumStatus BOOLEAN
) RETURNS VOID AS $$
DECLARE
   e_userId INTEGER;
BEGIN
   SELECT userID INTO e_userId FROM Users WHERE nickname = i_userNickname AND email = i_userMail;
   IF NOT FOUND THEN
      INSERT INTO Users(nickname, email) VALUES (i_userNickname, i_userMail);
   END IF;

  INSERT INTO CustomerUser(userNickname, userMail, cookies, premiumStatus)
     VALUES (i_userNickname, i_userMail, i_cookies, i_premiumStatus);

END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertCustomers(i_users CustomerUserType[])
RETURNS VOID AS $$
DECLARE
   userRecord CustomerUserType;
BEGIN
   FOREACH userRecord IN ARRAY i_users
      LOOP
         PERFORM insertCustomerUser(userRecord.userNickname,
                           userRecord.userMail,
                           userRecord.cookies,
                           userRecord.premiumStatus);
      END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE FUNCTION insertInstrumentProduct(
   i_ISBN VARCHAR(50),
```

```
    i_modelNumber VARCHAR(50),
    i_range VARCHAR(50),
    i_type VARCHAR(50)
) RETURNS VOID AS $$
DECLARE
    e_productID INTEGER;
BEGIN
    SELECT productID INTO e_productID FROM Product WHERE modelNumber = i_modelNumber;
    IF NOT FOUND THEN
        INSERT INTO Product(ISBN, modelNumber) VALUES (i_ISBN, i_modelNumber);
    END IF;

   INSERT INTO InstrumentProduct(ISBN, modelNumber, range, type)
       VALUES (i_ISBN, i_modelNumber, i_range, i_type);
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertInstruments(i_instruments InstrumentProductType[])
RETURNS VOID AS $$
DECLARE
    instrumentRecord InstrumentProductType;
BEGIN
    FOREACH instrumentRecord IN ARRAY i_instruments
    LOOP
        PERFORM insertInstrumentProduct(instrumentRecord.ISBN,
                                        instrumentRecord.modelNumber,
                                        instrumentRecord.range,
                                        instrumentRecord.type);
    END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertPAProduct(
    i_ISBN VARCHAR(50),
    i_modelNumber VARCHAR(50),
    i_range VARCHAR(50),
    i_type VARCHAR(50),
    i_resistance VARCHAR(50)
) RETURNS VOID AS $$
DECLARE
    e_productID INTEGER;
BEGIN
    SELECT productID INTO e_productID FROM Product WHERE modelNumber = i_modelNumber;
    IF NOT FOUND THEN
        INSERT INTO Product(ISBN, modelNumber) VALUES (i_ISBN, i_modelNumber);
    END IF;

   INSERT INTO PASystemProduct(ISBN, modelNumber, range, type, resistance)
       VALUES (i_ISBN, i_modelNumber, i_range, i_type, i_resistance);
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION insertPASystems(i_systems PASystemProductType[])
RETURNS VOID AS $$
DECLARE
   systemRecord PASystemProductType;
BEGIN
   FOREACH systemRecord IN ARRAY i_systems
   LOOP
      PERFORM insertPAProduct(systemRecord.ISBN,
                              systemRecord.modelNumber,
                              systemRecord.range,
                              systemRecord.type,
                              systemRecord.resistance);
   END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertPrivateInfo(
   i_userNickname VARCHAR(50),
   i_userMail VARCHAR(50),
   i_fullName VARCHAR(50),
   i_homeNumber VARCHAR(50),
   i_street VARCHAR(50),
   i_city VARCHAR(50),
   i_zip VARCHAR(50),
   i_phoneNumber VARCHAR(10),
   i_password VARCHAR(50)
) RETURNS VOID AS $$
DECLARE
   e_userId INTEGER;
BEGIN
   SELECT userID INTO e_userId FROM Users WHERE nickname = i_userNickname AND email = i_userMail;
   IF FOUND THEN
      INSERT INTO PrivateInfo(userNickname, userMail, fullName, homeNumber, street, city, zip, phoneNumber, password)
        VALUES (i_userNickname, i_userMail, i_fullName, i_homeNumber, i_street, i_city, i_zip, i_phoneNumber, i_password);
   ELSE
      RAISE EXCEPTION 'User % % not found, create new user using insertWorkerUser() or insertCustomerUser()', i_userNickname,
i_userMail;
   END IF;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertPrivateInfos(i_infos privateInfoType[])
RETURNS VOID AS $$
DECLARE
   infoRecord privateInfoType;
BEGIN
   FOREACH infoRecord IN ARRAY i_infos
   LOOP
      PERFORM insertPrivateInfo(infoRecord.userNickname,
```

```
                infoRecord.userMail,
                infoRecord.fullName,
                infoRecord.homeNumber,
                infoRecord.street,
                infoRecord.city,
                infoRecord.zip,
                infoRecord.phoneNumber,
                infoRecord.password);
    END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertSuperiorityRelation(superior_workerId INTEGER, underling_workerId INTEGER)
    RETURNS void AS $$
DECLARE
    superior_pn VARCHAR(50);
    underling_pn VARCHAR(50);
BEGIN
    SELECT personalIdentificationNumber INTO superior_pn FROM WorkerUser WHERE workerId = superior_workerId;
    SELECT personalIdentificationNumber INTO underling_pn FROM WorkerUser WHERE workerId = underling_workerId;

    IF superior_pn IS NOT NULL AND underling_pn IS NOT NULL THEN
        INSERT INTO Superiority (superiorId, underlingId, isSuperior, isUnderling)
        VALUES (superior_workerId, underling_workerId, superior_pn, underling_pn);
    ELSE
        RAISE EXCEPTION 'Invalid worker IDs: % %', superior_workerId, underling_workerId;
    END IF;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertSuperiorityRelations(i_pairs intPair[])
RETURNS VOID AS $$
DECLARE
    pair intPair;
BEGIN
    FOREACH pair IN ARRAY i_pairs
    LOOP
        PERFORM insertSuperiorityRelation(pair.frst, pair.scnd);
    END LOOP;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertInstrumentPickup(
    i_instrumentId INTEGER,
    i_pickupID INTEGER
) RETURNS VOID AS $$
DECLARE
    e_instrument_mn INTEGER;
    e_pickupID_mn INTEGER;
BEGIN
```

```
    SELECT modelNumber INTO e_instrument_mn FROM InstrumentProduct WHERE instrumentId = i_instrumentId;
    SELECT modelNumber INTO e_pickupID_mn FROM Pickup WHERE pickupId = i_pickupID;
    IF e_instrument_mn IS NOT NULL AND e_pickupID_mn IS NOT NULL THEN
        INSERT INTO InstrumentPickup(instrumentId, pickupID, modelNumber, pickupModelNumber)
        VALUES (i_instrumentId, i_pickupID, e_instrument_mn, e_pickupID_mn);
    ELSE
        RAISE EXCEPTION 'Invalid instrument ID: % or pickup ID: %', i_instrumentId, i_pickupID;
    END IF;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION insertInstrumentPickups(i_pairs intPair[])
RETURNS VOID AS $$
DECLARE
    pair intPair;
BEGIN
    FOREACH pair IN ARRAY i_pairs
    LOOP
        PERFORM insertInstrumentPickup(pair.frst, pair.scnd);
    END LOOP;
END;
$$ LANGUAGE plpgsql;



-- Inserting data in to tables
================================================================================================
    -- Insert at least 10 records into each table
    -- !!! NOTE: Some numeric values could be flawed, check server for data that was inputted

INSERT INTO Store(location)
  VALUES ('Prague/Muzeum'),
            ('Prague/Andel'),
            ('Brno/Namesti Svobody'),
            ('Praha/Namesti Republiky'),
            ('Ostrava/Namesti Svobody'),
            ('Online'),
            ('Berlin/Alexanderplatz'),
            ('Berlin/Kurfurstendamm'),
            ('Vienna/Stephansplatz'),
            ('Vienna/Praterstern');


SELECT insertWorkers(ARRAY[
    ('123456789', 'worker1', 'worker1@gmail.com', 'Prague/Muzeum'),
    ('987654321', 'worker2', 'worker2@gmail.com', 'Prague/Andel'),
    ('123123123', 'worker3', 'worker3@gmail.com', 'Brno/Namesti Svobody'),
    ('321321321', 'worker4', 'worker4@gmail.com', 'Praha/Namesti Republiky'),
    ('456456456', 'worker5', 'worker5@gmail.com', 'Ostrava/Namesti Svobody'),
    ('654654654', 'worker6', 'worker6@gmail.com', 'Online'),
    ('789789789', 'worker7', 'worker7@gmail.com', 'Berlin/Alexanderplatz'),
    ('987987987', 'worker8', 'worker8@gmail.com', 'Berlin/Kurfurstendamm'),
    ('654654655', 'worker9', 'worker9@gmail.com', 'Vienna/Stephansplatz'),
    ('321321323', 'worker10', 'worker10@gmail.com', 'Vienna/Praterstern')
```

```
    ]::WorkerUserType[]);



SELECT insertCustomers(ARRAY[
    ('customer1', 'cus1@seznam.cz', 0, FALSE),
    ('customer2', 'cus2@seznam.cz', 505, FALSE),
    ('customer3', 'cus3@seznam.cz', 404, FALSE),
    ('customer4', 'cus4@seznam.cz', 436, TRUE),
    ('customer5', 'cus5@seznam.cz', 0, FALSE),
    ('customer6', 'cus6@seznam.cz', 0, FALSE),
    ('customer7', 'cus7@seznam.cz', 102, TRUE),
    ('customer8', 'cus8@seznam.cz', 0, FALSE),
    ('customer9', 'cus9@seznam.cz', 555, TRUE),
    ('customer10', 'cus10@seznam.cz', 291, FALSE)
    ]::CustomerUserType[]);



-- superiority should be like a tree
SELECT insertSuperiorityRelations(ARRAY[
    (11, 12),
    (11, 13),
    (11, 19),
    (13, 14),
    (14, 17),
    (14, 18),
    (15, 16),
    (15, 110),
    (15, 18),
    (19, 20)
    ]::intPair[]);

SELECT insertPrivateInfos(ARRAY[
    ('worker1', 'worker1@gmail.com', 'John Doe', '123', 'Main Street', 'Prague', '12345', '420123456789', 'password1'),
    ('worker2', 'worker2@gmail.com', 'Jane Doe', '456', 'Second Street', 'Prague', '54321', '420987654321', 'password2'),
    ('worker3', 'worker3@gmail.com', 'John Smith', '789', 'Third Street', 'Brno', '67890', '420123123123', 'password3'),
    ('worker4', 'worker4@gmail.com', 'Jane Smith', '012', 'Fourth Street', 'Praha', '09876', '420321321321', 'password4'),
    ('worker5', 'worker5@gmail.com', 'John Johnson', '345', 'Fifth Street', 'Ostrava', '54321', '420456456456', 'password5'),
    ('worker6', 'worker6@gmail.com', 'Jane Johnson', '678', 'Sixth Street', 'Online', '67890', '420654654654', 'password6'),
    ('customer4', 'cus4@seznam.cz', 'Bob Doe', '123', 'Main Street', 'Prague', '12345', '420123456789', 'password1'),
    ('customer9', 'cus9@seznam.cz', 'Janek Doe', '456', 'Second Street', 'Prague', '54321', '420987654321', 'password2'),
    ('customer10', 'cus10@seznam.cz', 'Pepa Smith', '789', 'Third Street', 'Brno', '67890', '420123123123', 'password3'),
    ('customer7', 'cus7@seznam.cz', 'Janek Smith', '012', 'Fourth Street', 'Praha', '09876', '420321321321', 'password4')
    ]::privateInfoType[]);

COPY Accessory(ISBN, type) FROM '/home/safor/Documents/Cvut/DBS/musicStoreSemestralaccessories.csv' WITH (FORMAT csv);

SELECT insertInstruments(ARRAY[
    ('123456789', 'instrument1', '20Hz-20kHz', 'key'),
    ('187654321', 'instrument2', '20Hz-20kHz', 'string'),
    ('123123123', 'instrument3', '20Hz-20kHz', 'wind'),
    ('121321321', 'instrument4', '20Hz-20kHz', 'percussion'),
    ('156121212', 'instrument5', '20Hz-20kHz', 'special'),
    ('154654654', 'instrument6', '20Hz-20kHz', 'key'),
```

```
  ('189789789', 'instrument7', '20Hz-20kHz', 'string'),
  ('187987987', 'instrument8', '20Hz-20kHz', 'wind'),
  ('154654654', 'instrument9', '20Hz-20kHz', 'percussion'),
  ('121321321', 'instrument10', '20Hz-20kHz', 'special')
  ]::InstrumentProductType[]);

SELECT insertPASystems(ARRAY[
  ('223456789', 'system1', '20Hz-20kHz', 'active', '8Ohm'),
  ('287654321', 'system2', '20Hz-20kHz', 'passive', '4Ohm'),
  ('223123123', 'system3', '20Hz-20kHz', 'active', '8Ohm'),
  ('221321321', 'system4', '20Hz-20kHz', 'passive', '4Ohm'),
  ('256121212', 'system5', '20Hz-20kHz', 'active', '8Ohm'),
  ('254654654', 'system6', '20Hz-20kHz', 'passive', '4Ohm'),
  ('289789789', 'system7', '20Hz-20kHz', 'active', '8Ohm'),
  ('287987987', 'system8', '20Hz-20kHz', 'passive', '4Ohm'),
  ('254654654', 'system9', '20Hz-20kHz', 'active', '8Ohm'),
  ('221321321', 'system10', '20Hz-20kHz', 'passive', '4Ohm')
  ]::PASystemProductType[]);

INSERT INTO Pickup(modelNumber, name) VALUES
                        ('23', 'single coil'),
                        ('32', 'humbucker'),
                        ('41', 'piezo'),
                        ('75', 'lipstick'),
                        ('61', 'active'),
                        ('67', 'passive'),
                        ('38', 'single coil'),
                        ('90', 'humbucker'),
                        ('100', 'piezo'),
                        ('1001', 'lipstick');

SELECT insertInstrumentPickups(ARRAY[
  (10, 5),
  (12, 6),
  (13, 7),
  (14, 8),
  (15, 9),
  (16, 10),
  (17, 1),
  (18, 2),
  (11, 3),
  (19, 4)
  ]::intPair[]);

INSERT INTO buys(ISBN, modelNumber, userNickname, userMail)
  VALUES  ('223456789', 'system1', 'customer1', 'cus1@seznam.cz'),
      ('287654321', 'system2', 'customer2', 'cus2@seznam.cz'),
      ('223123123', 'system3', 'customer3', 'cus3@seznam.cz'),
      ('221321321', 'system4', 'customer4', 'cus4@seznam,cz'),
      ('256121212', 'system5', 'customer5', 'cus5@seznam,cz'),
      ('254654654', 'system6', 'customer6', 'cus6@seznam,cz'),
      ('123456789', 'instrument1', 'customer7', 'cus7@seznam,cz'),
      ('187654321', 'instrument2', 'customer8', 'cus8@seznam,cz'),
      ('123123123', 'instrument3', 'customer9', 'cus9@seznam,cz'),
```

```
        ('121321321', 'instrument4', 'customer10', 'cus10@seznam,cz');


    -- put some products into one of the stores (location) -> the product is still in store
UPDATE Product SET location = 'Prague/Muzeum' WHERE modelNumber = 'instrument1';
UPDATE Product SET location = 'Prague/Andel' WHERE modelNumber = 'instrument2';
UPDATE Product SET location = 'Brno/Namesti Svobody' WHERE modelNumber = 'instrument3';
UPDATE Product SET location = 'Praha/Namesti Republiky' WHERE modelNumber = 'instrument4';
UPDATE Product SET location = 'Ostrava/Namesti Svobody' WHERE modelNumber = 'instrument5';
UPDATE Product SET location = 'Online' WHERE modelNumber = 'instrument6';
UPDATE Product SET location = 'Berlin/Alexanderplatz' WHERE modelNumber = 'system1';
UPDATE Product SET location = 'Berlin/Kurfurstendamm' WHERE modelNumber = 'system2';
UPDATE Product SET location = 'Vienna/Stephansplatz' WHERE modelNumber = 'system3';
UPDATE Product SET location = 'Vienna/Praterstern' WHERE modelNumber = 'system4';

-- forgot to add link between accessory and instrument
ALTER TABLE accessory ADD COLUMN comesWith VARCHAR(50);
ALTER TABLE accessory
    ADD FOREIGN KEY (comesWith) REFERENCES instrumentproduct(modelNumber)
        ON DELETE SET NULL
        ON UPDATE CASCADE;

```

requests for data:

```
-- This query uses a LEFT OUTER JOIN to ensure that all instruments are listed,
-- even those without any associated pickups.
SELECT ip.instrumentId, ip.modelNumber, p.name AS pickup_name
FROM InstrumentProduct ip
LEFT OUTER JOIN InstrumentPickup ipu ON ip.instrumentId = ipu.instrumentId
LEFT OUTER JOIN Pickup p ON ipu.pickupModelNumber = p.modelNumber;
```

	instrumentid	modelnumber	pickup_name
1	10	instrument1	active
2	12	instrument3	passive
3	13	instrument4	single coil
4	14	instrument5	humbucker
5	15	instrument6	piezo
6	16	instrument7	lipstick
7	17	instrument8	single coil
8	18	instrument9	humbucker
9	11	instrument2	piezo
10	19	instrument10	lipstick

```
-- This query uses INNER JOIN to combine rows from
-- InstrumentProduct, Product, and Store tables
-- where there are matches in the ISBN and location columns, respectively.
```

```
SELECT ip.modelNumber, ip.type, s.location
FROM InstrumentProduct ip
    INNER JOIN Product p ON ip.ISBN = p.ISBN
    INNER JOIN Store s ON p.location = s.location;
```

	modelnumber	type	location
1	instrument1	key	Prague/Muzeum
2	instrument2	string	Prague/Andel
3	instrument3	wind	Brno/Namesti Svobody
4	instrument4	percussion	Praha/Namesti Republiky
5	instrument5	special	Ostrava/Namesti Svobody
6	instrument6	key	Online

```
-- This query searches for accessories
-- where the type starts with 'Bass'  and the ISBN starts with '97'.
SELECT *
FROM Accessory
WHERE type LIKE 'Bass%' AND ISBN LIKE '97%';
```

(500 results)

	accessoryid	isbn	type	comeswith
1	36004	9780416606423	Bass PedalJack Cable	<null>
2	36010	9780973524970	Bass PedalJack Cable	<null>
3	36032	9780997779943	Bass Case	<null>
4	36045	9780320085949	Bass PedalJack Cable	<null>
5	36061	9780024067432	Bass PedalJack Cable	<null>
6	36067	9781107468993	Bass PedalJack Cable	<null>
7	36070	9780098596180	Bass PedalJack Cable	<null>
8	36080	9780067117125	Bass Case	<null>
9	36089	9780055346537	Bass Case	<null>
10	36094	9780832948954	Bass PedalJack Cable	<null>
11	36098	9781241322083	Bass PedalJack Cable	<null>
12	36104	9780660574059	Bass PedalJack Cable	<null>
13	36108	9781782464372	Bass PedalJack Cable	<null>
14	36112	9780257790541	Bass Case	<null>
15	36123	9781557736598	Bass Case	<null>
16	36125	9781090164612	Bass Case	<null>
17	36135	9781804666364	Bass PedalJack Cable	<null>
18	36162	9780895706157	Bass PedalJack Cable	<null>
19	36165	9781169112599	Bass Case	<null>
20	36181	9781264177066	Bass Case	<null>
21	36185	9780913470275	Bass Case	<null>
22	36209	9781961722705	Bass Case	<null>
23	36212	9781154548464	Bass PedalJack Cable	<null>
24	36224	9781512094916	Bass Case	<null>
25	36234	9781001791432	Bass Case	<null>
26	36236	9781117999470	Bass Case	<null>
27	36237	9781776299362	Bass PedalJack Cable	<null>
28	36252	9780395109199	Bass Case	<null>
29	36261	9781212881151	Bass PedalJack Cable	<null>
30	36262	9781980750048	Bass PedalJack Cable	<null>
31	36266	9780076471249	Bass Case	<null>

```
-- This query groups instruments by type and counts them,
-- only showing those types where there are more than 5 instruments.
SELECT type, COUNT(*) AS num_instruments
FROM Accessory
GROUP BY Accessory.type
HAVING COUNT(*) > 5;
```

	type	num_instruments
1	Bass PedalJack Cable	2297
2	Guitar Case	2258
3	Cymbal Stand	2308
4	Pick	2307
5	Tom Head	2275
6	Snare Head	2293
7	Earplugs	2263
8	Effects PedalExpression Pedal	2292
9	Bass Case	2303
10	Power Cable	2239
11	Kick Head	2277
12	Mute	2305
13	Guitar Strings	2302
14	XLR Cable	2280

```
-- This UNION operation combines and removes duplicates,
-- listing all unique ISBNs across both tables.
SELECT ISBN FROM InstrumentProduct
UNION
SELECT ISBN FROM PASystemProduct;
```

	isbn
1	821321321
2	187987987
3	121321321
4	123123123
5	954654654
6	223456789
7	189789789
8	187654321
9	156121212
10	223123123
11	154114654
12	921321321
13	256121212
14	254654654
15	123456789
16	287654321
17	153654654
18	221321321
19	289789789
20	287987987

```
-- This query uses a nested SELECT to find instruments
-- that have an ISBN that is also listed in the Accessory table.
-- (could be good to eliminate duplicate ISBNs)
SELECT *
FROM InstrumentProduct
WHERE ISBN IN (SELECT ISBN FROM Accessory);
```

instrumentid	isbn	modelnumber	range	type