A dog walker decides to get organized with the help of a database application. She has 3 dog-walking timeslots in each of 5 days of the work week, so 15 timeslots in all, which she has numbered 1 to 15. Each of her customers (dog owners) has a certain timeslot (only one, for simplicity), and one or more dogs. Each dog has a name, a unique id, and an owner name (i.e., a customer name). For example, customer ‘John’ has 2 dogs, ‘Spot’ and ‘Junior’, with ids 23 and 24, and John has slot 3 for the dog-walking appointment each week. The slot number is held in the customer table. The customer and dog data is already loaded in the database, with null slot number recorded for any asset unassigned customers. The customer names are unique but the dogs’ names are unique only for a certain owner. Multiple customers can be given the same slot number. The app needs to be able to assign a slot number to an existent customer, and report on all the dogs and the owner for each, for a given slot.

a. Using our client-server database applications technology, how should we hold the data on all the owners and dogs to support this application? Write the SQL that the program needs to use to create an empty table named customers and another table named dogs. Use the uniqueness of the names appropriately. Make the relationship between the two tables show up in this setup by an appropriate foreign key constraint.

```sql
CREATE TABLE customers (name VARCHAR(30), slot INT, primary key (name));
CREATE TABLE dogs (id INT, primary key, name VARCHAR(25), not null, owner VARCHAR(30) not null,
CHECK (name) REFERENCES customers (name));
```

b. Write SQL to assign customer ‘John’ (already in the customers table) to slot 3.

```sql
UPDATE customers SET slot = 3 WHERE name = 'John';
```

c. Write SQL to report on all the dogs and the owner of each assigned to slot 3.

```sql
SELECT d.name, d.owner FROM dogs d, customers c
WHERE d.owner = c.name AND c.slot = 3;
```
2. Continuing with the dog-walking app of problem 1.
a. Design two simple domain classes (as in pizza1 or music1) named Dog and Customer. Make sure there is an object reference in one of the classes that points to an object in the other class, like Track has a Product reference in music1. You can write "//getters and setters" rather than write the getters, setters and constructors in detail.

```
Dog class:
public class Dog {
    private String name;
    private Customer owner;
    private int i;
    //getters & setters
}
```

```
Customer class:
public class Customer {
    private String name;
    private int s;
    private List<Dog> dogs;
    //getters & setters
}
```

b. Design a DAO API, with method headers only (no code), and a throws clause as in pizza1 or music1, to do the required actions of the app. Return Dog objects for the retrieval method. Don’t worry about initializing the database here.

```
- or not
void assignDog (Customer cust, int slot) throws SQLException

let <Dog> findDog (int slot) throws SQLException
```
3. SQL. Note the core created.sql script at the end of the exam. Tear this off and use it here. You don't have to turn it in. For this problem, write SQL queries on the database (no Java here!) to find the following:

a. Find the tracks and their product codes that the user with email 'joe@cs.umb.edu' has downloaded. Report their track titles, product codes and download dates, in time order.

```sql
select t.title, p.product_code, d.download_date
from tracks t, products p, downloads d, site_users u
where t.product_id = p.product_id
and t.track_id = d.track_id
and d.user_id = u.user_id
and u.email_address = 'joe@cs.umb.edu'
order by d.download_date
```

b. For each product (i.e. each CD), find the time of the most recent download of any of its tracks. List product code and most recent download time (of any track) for that product.

```sql
select p.product_code, max(d.download_date)
from products p, tracks t, downloads d
where p.product_id = t.product_id
and t.track_id = d.track_id
group by p.product_code
```
4. Web background

A website is available at server http://pizzanow.com. Suppose this HTML file is located in the pizza3 directory of the web server’s root directory. Pizzapie.jpg is the circular pizza image used in pizza3, the web app you made a page flow for in homework 3.

```html
<html>
<body>
  <p>Here is the image of a pizza: </p>
  <img src="images/pizzapie.jpg" width="100" height="100">

  <p>Here it is again: </p>
  <img src="images/pizzapie.jpg" width="100" height="100">

  <a href = "content/menu.html"> See our menu </a>
</body>
</html>
```

a. Show the display on the screen for this page. Use a circle to show the pizzapie.

- [ ] Here is the image of a pizza:
  - [ ] Here it is again:
  - [x] See our menu

b. Give the full URL of this page, assuming the web server is using port 80.

```
```

c. Give the sequence of GET commands (one line each, each ending with HTTP/1.1) that occur when you browse to this page. Assume the browser knows how to avoid downloading a certain image twice.

```
GET /pizza3/index.html HTTP/1.1
GET /pizza3/images/pizzapie.jpg HTTP/1.1
```

d. Give the GET command that occurs if the user clicks on the link.

```
GET /pizza3/content/menu.html HTTP/1.1
```
5. Layers and Implementation. Consider the following method from music1:

```java
public InvoiceData checkout(Cart cart, long userId) throws ServiceException {
    Invoice invoice = null;
    try {
        User user = userDb.findUserById(userId);
        invoice = new Invoice(-1, user, new Date(), false, null, null);
        Set<LineItem> lineItems = new HashSet<LineItem>();
        for (CartItem item : cart.getItems()) {
            Product prod = prodDb.findProductById(item.getProductId());
            LineItem li = new LineItem(prod, invoice, item.getQuantity());
            lineItems.add(li);
        }
        invoice.setLineItems(lineItems);
        invoice.setTotalAmount(invoice.calculateTotal());
        invoiceDb.insertInvoice(invoice);
    } catch (SQLException e) {
        throw new ServiceException("Can't check out: ", e);
    }
    cart.clear();
    return new InvoiceData(invoice);
}
```

a. What DAO methods are called in this method? List them in execution order (name only)

- `findUserById`
- `findProductById`
- `insertInvoice`

b. Underline the domain object method calls, including any constructor calls. Note that transfer objects are not domain objects.

c. Clearly this execution depends on a good value for the variable invoiceDb. What kind of variable is this, local variable or field? `field`

How does this variable get set with a good value? Cite the class and method of the code that sets the value of this variable: `UserService constructor`

What code (class and method) called the method you just cited? `MusicSystemConfig`

---

d. What domain object(s) were passed down from the presentation layer in the call to this method?

- `Cart`
You can tear this off for easy reference. You don’t need to turn it in.

-- Note money as decimal here
CREATE TABLE product(
    product_id INT NOT NULL,
    product_code VARCHAR(10) NOT NULL,
    product_description VARCHAR(100) NOT NULL,
    product_price DECIMAL(10,2) NOT NULL,
    UNIQUE (product_code),
    PRIMARY KEY (product_id)
);

INSERT INTO product VALUES
    (1, '8601', '86 (the band) - True Life Songs and Pictures', '14.95');
INSERT INTO product VALUES
    (2, 'pf01', 'Paddlefoot - The first CD', '12.95');
INSERT INTO product VALUES
    (3, 'pf02', 'Paddlefoot - The second CD', '14.95');
INSERT INTO product VALUES
    (4, 'jr01', 'Joe Rut - Genuine Wood Grained Finish', '14.95');

-- create track before download: download has FK to track
CREATE TABLE track (  
    track_id INT NOT NULL PRIMARY KEY,
    product_id INT NOT NULL,
    track_number INT NOT NULL,
    title varchar(100) NOT NULL,
    sample_filename varchar(100) NOT NULL,
    FOREIGN KEY (product_id) REFERENCES product (product_id),
    UNIQUE (product_id, track_number)
);

INSERT INTO track VALUES (1, 1, 2, 'You Are a Star', 'star.mp3');
INSERT INTO track VALUES (2, 1, 3, 'Don't Make No Difference', 'no_difference.mp3');
INSERT INTO track VALUES (3, 2, 2, 'Whiskey Before Breakfast', 'whiskey.mp3');
INSERT INTO track VALUES (4, 2, 6, '64 Corvair, Part 2', 'corvair.mp3');
INSERT INTO track VALUES (5, 3, 1, 'Neon Lights', 'neon.mp3');
INSERT INTO track VALUES (6, 3, 3, 'Tank Hill', 'tank.mp3');
INSERT INTO track VALUES (7, 4, 1, 'Filter', 'filter.mp3');
INSERT INTO track VALUES (8, 4, 5, 'So Long Lazy Ray', 'so_long.mp3');

-- site_user: many unimplemented columns for our project
-- avoid identifier 'user', a reserved word in SQL
-- make them "not null" if ever implemented
-- note email_address is a second table key
CREATE TABLE site_user (  
    user_id INT NOT NULL,
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    email_address VARCHAR(50) NOT NULL,
    company_name VARCHAR(50),
    address1 VARCHAR(50),
    address2 VARCHAR(50),
    address3 VARCHAR(50),
    address4 VARCHAR(50),
    address5 VARCHAR(50),
    city VARCHAR(50),
    state VARCHAR(50),
    zip VARCHAR(50),
    country VARCHAR(50),
    phone VARCHAR(50),
    fax VARCHAR(50),
    notes VARCHAR(50),
    PRIMARY KEY (user_id)
);