COP4710 Database Design
Final Project
Video Games Database and In-Store Order System

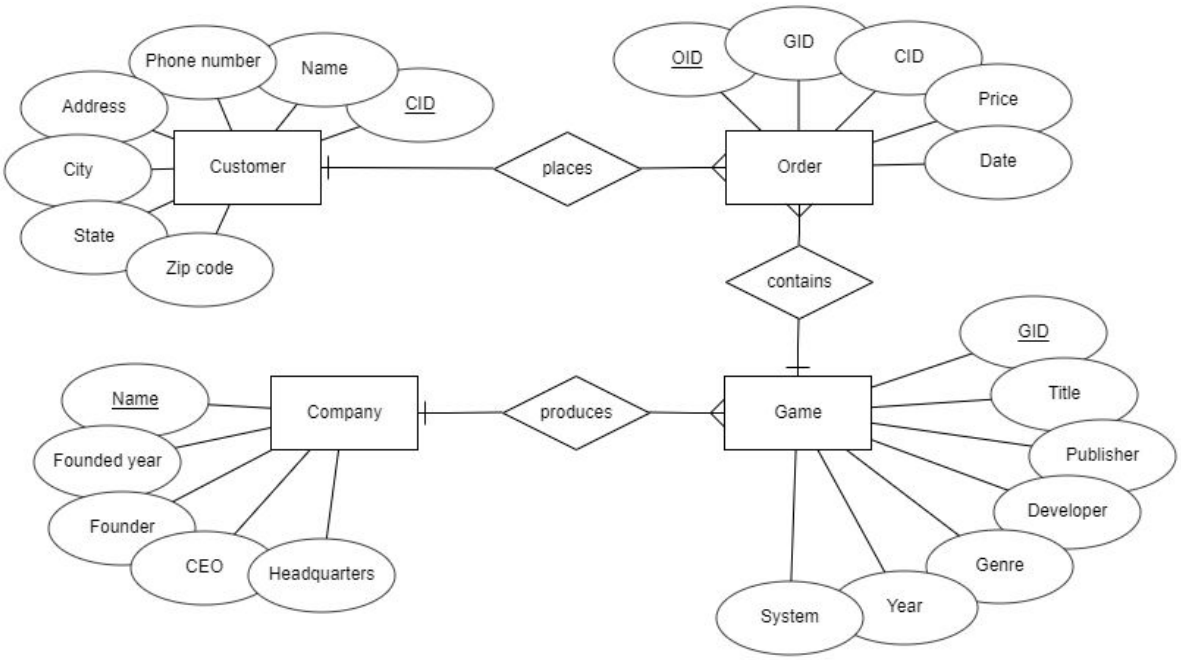
Edmund Lee
Matthew Kramer
Meera Patel

I. Overview

Our group came up with the idea of a video games database, where all of the games and game companies in this database would be available to the public for viewing. Any game can be searched by different parameters such as company, game genre, or title. However, this database is not only used by the public; game distributing stores can also use this as their database system to record transactions, as we also include tables to store orders and customer data, which are only accessible by employees of the store.

II. Database Design

The following entity-relationship (ER) diagram explains the relationship between each table in the database. The table Orders is the main relation that links to both Customers and Games, as it holds attributes GID and CID. Any name entered in Company is unique and can be matched to any game published.



III. Client Functionality

Before connecting to the database, the user must log in with with a username and password. There are two types of accounts that have different privileges on the database end: a “customer” account, which is for public users and has somewhat restricted access, and an “employee” account, which requires an employee key to be activated and has full access to the database.

The user interface is text-based, and upon running the program, there are three possible selections initially. Upon starting the database, the user will be prompted with three choices in the main menu:

- Enter Display Menu
- Enter Search Menu
- Enter Modify Menu (Employees only)

Before getting into the features of each menu, note that public users can only access the “Display Menu” and “Search Menu” functions, as these two menus cannot change any information in the database. Furthermore, these public users may only search and display values from the Games and Company tables. The Orders and Customer tables contain sensitive information that only employees may access. If a public user tries to access any of the “Modify Menu” functions, then they will get a message saying that they do not have permission to access those commands.

Inside the Display Menu, the user will be prompted with six choices:
- Display all games
- Display all companies
- Display all customers (Employees only)
- Display all orders (Employees only)
- Display all orders by ascending customer ID (Employees only)
- Display all orders by ascending game ID (Employees only)

If the user is a public user, then they will only have the ability to display all games or display all companies. If the user is an employee, they will have access to all options.

Inside the Search Menu, the user will be prompted with five possible choices.
- Search games by name
- Search games by genre
- Search games by company
- Search orders by customer ID (Employees only)
- Search orders by game ID (Employees only)

Once the user enters a choice, the system will then request an input from the user as a search parameter. Once the input is entered, it will be searched through the database and return only rows that correspond with the given parameter. Public users may only access the first three options, while employees may access all functionalities.

For the Modify Menu, only employees may use its functions, as this menu modifies the contents of the database and can create/delete user accounts. Inside the Modify Menu, we have:
- Add new game
- Add new company
- Add new customer
- Add new order
- Delete game

- Delete company
- Delete customer
- Update price of game
- Create a new user account
- Delete a user account

IV. Database Tables

The following tables represent the relationships depicted in the ER diagram and describe the characteristics of each table more thoroughly.

Company

Company (*Name:* CHARACTER VARYING(100), *Founder*: CHARACTER VARYING(100), *Year Founded:* INTEGER, *CEO*: CHARACTER VARYING(100), *Headquarters*: CHARACTER VARYING(50))

Foreign Keys: None
Primary Keys: *Name*
Not Null: *Name*

The Company table holds information about each company in the gaming industry. For example, Blizzard, Valve, and Nintendo publish games under these names. If a user wants to find out more information about a company and its history, they can go to this table.

Customer

Customer (*Name:* CHARACTER VARYING(30), *Address:* CHARACTER VARYING(50), *City:* CHARACTER VARYING (30), *State:* CHARACTER VARYING (2), *ZIP:* CHARACTER VARYING (5), *Phone Number:* CHARACTER VARYING(12), *CID:* CHARACTER VARYING(10))

Foreign Keys: None
Primary Keys: *CID*
Not Null: *CID*

This table holds customer's information identified by their customer ID (CID), which is unique to each customer. This attribute cannot be null and must be 10 characters long. This is linked to the Orders table, as Orders holds information about purchases and transactions linked to the CID. Employees would go to this table to find information about that customer if they needed to contact them for any reason.

Games

Games (*GID*: CHARACTER VARYING(10), *Title*: CHARACTER VARYING(30), *Publisher*: CHARACTER VARYING(30), *Developer*: CHARACTER VARYING (30), *Genre*: CHARACTER VARYING (20), *Year*: INTEGER, *System:* CHARACTER VARYING(30), *Price*: DOUBLE PRECISION)

Foreign Keys: None
Primary Keys: *GID*
Not Null: *GID*

This tables holds all information about games a store may currently own or have in stock. Users can go through this table if they would like to check if the store currently has a game available in their system. If the game exists in the system, then the store may have it for sale. Each game has a unique game ID, called GID. This is a 10-character code and cannot be null. This is linked to the Orders table to see what game a customer has bought. Price is an editable field as the value of a game may change as time passes.

Orders

Orders (*OID:* CHARACTER VARYING(10), *GID:* CHARACTER VARYING(10), *CID:* CHARACTER VARYING(10), *Sale:* DOUBLE PRECISION, *Date:* DATE)

Foreign Keys: *GID*, *CID*
Primary Keys: *OID*
Not Null: *OID*, *GID*, *CID*

This tables holds all transactions of games being sold. There are three non-null attributes in this table: OID, CID, and GID. These attributes are also unique as they hold linked information to each game or customer. The Sale attribute is the amount of money the game was sold for. As stated in the games table, games lose value over time, so the sale attribute in orders can be different for different games depending on when the game was sold. In addition, orders cannot be deleted as it is a list of transactions (i.e., a transaction history), so the store can keep record of what has been sold.

V. Database Initialization Queries

We implemented our database locally using Postgres, specifically pgAdmin 4. To connect our text-based UI (which was created using Python) to the database in pgAdmin, we used the Psycopg2 adapter. Below are the queries used to initialize the tables. Note that *postgres* is the default administrative superuser of the database, with all privileges granted.

```
-- Table: public."Company"
CREATE TABLE public."Company"
(
    "Name" character varying(100) COLLATE pg_catalog."default" NOT
      NULL,
    "Founder" character varying(100) COLLATE pg_catalog."default",
    "Year Founded" integer,
    "CEO" character varying(50) COLLATE pg_catalog."default",
    "Headquarters" character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT "Company_pkey" PRIMARY KEY ("Name"),
    CONSTRAINT ck_primarykeynotempty CHECK (length("Name"::text) > 0)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."Company"
    OWNER to postgres;

GRANT ALL ON TABLE public."Company" TO postgres;

-- Table: public."Customer"
CREATE TABLE public."Customer"
(
    "Name" character varying(30) COLLATE pg_catalog."default",
    "Address" character varying(50) COLLATE pg_catalog."default",
    "City" character varying(30) COLLATE pg_catalog."default",
    "State" character varying(2) COLLATE pg_catalog."default",
    "ZIP" character varying(5) COLLATE pg_catalog."default",
    "Phone Number" character varying(12) COLLATE pg_catalog."default",
    "CID" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Customer_pkey" PRIMARY KEY ("CID"),
    CONSTRAINT ck_primarykeynotempty CHECK (length("CID"::text) > 0)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."Customer"
    OWNER to postgres;
```

```
GRANT ALL ON TABLE public."Customer" TO postgres;

-- Table: public."Games"
CREATE TABLE public."Games"
(
    "GID" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    "Title" character varying(30) COLLATE pg_catalog."default",
    "Publisher" character varying(30) COLLATE pg_catalog."default",
    "Developer" character varying(30) COLLATE pg_catalog."default",
    "Genre" character varying(20) COLLATE pg_catalog."default",
    "Year" integer,
    "System" character varying(30) COLLATE pg_catalog."default",
    "Price" double precision,
    CONSTRAINT "Games_pkey" PRIMARY KEY ("GID"),
    CONSTRAINT ck_primarykeynotempty CHECK (length("GID"::text) > 0)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."Games"
    OWNER to postgres;

GRANT ALL ON TABLE public."Games" TO postgres;

-- Table: public."Orders"
CREATE TABLE public."Orders"
(
    "OID" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    "GID" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    "CID" character varying(10) COLLATE pg_catalog."default" NOT NULL,
    "Sale" double precision,
    "Date" date,
    CONSTRAINT "Orders_pkey" PRIMARY KEY ("OID", "GID", "CID"),
    CONSTRAINT ck_primarykeynotempty CHECK (length("OID"::text) > 0 AND
      length("GID"::text) > 0 AND length("CID"::text) > 0)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

```
ALTER TABLE public."Orders"
    OWNER to postgres;

GRANT ALL ON TABLE public."Orders" TO postgres;
```

VI. SQL Queries

Display Menu

All display functions used relatively straightforward and simple queries, as they only need to display all rows of a given table. For instance, to display all games, the following code is executed:

```
cur = conn.cursor()
cur.execute('SELECT * FROM public."Games";')
rs = cur.fetchall()
print_games(rs)
cur.close()
```

The cursor() class is used to execute a given PostgreSQL command in a connected database. The fetchall() command fetches the result set of the executed query. The print_games(rs) function was written manually in Python to display the entries of the table in a clearer, more readable manner rather than in tuples, as such:

```
Displaying all games
GID             Title                       Publisher       Developer       Genre           Year        System              Price
-------------------------------------------------------------------------------------------------------------------------------------
SUPSMA1999      Super Smash Bros            Nintendo        HAL Laboratory  Fighting        1999        Nintendo N64        150.00
SUPMAR1996      Super Mario 64              Nintendo        Nintendo        Action          1996        Nintendo N64        30.00
CALDUT2003      Call of Duty                Activision      Infinity        Action          2003        Valve Steam         59.99
MARKAR2014      Mario Kart 8                Nintendo        Nintendo        Racing          2014        Nintendo Wii U      60.00
PIKMIN2001      Pikmin                      Nintendo        Nintendo        Puzzle          2001        Nintendo GameCube   25.00
METGEA2009      Metal Gear Solid Rising     Konami          Konami          Action          2009        Microsoft Xbox 360  49.99
SONADV1999      Sonic Adventure             Sega            Sega            Action          1999        Sega Dreamcast      39.05
FIFWOR2010      2010 FIFA World Cup         Electronic Arts EA Canada       Sports          2010        Microsoft Xbox 360  29.99
MADNFL2003      Madden NFL 2003             Electronic Arts Electronic Arts Sports          2002        Nintendo GameCube   24.00
GTAFIV2013      Grand Theft Auto V          Rockstar Games  Rockstar Games  Action          2013        Microsoft Xbox 360  60.00
SPIMAN2004      Spider-Man 2                Activision      Treyarch        Fighting        2004        Sony Playstation 2  39.99
GUIHER2009      Guitar Hero 5               Activision      Neversoft       Music           2009        Microsoft Xbox 360  19.99
```

Likewise, for retrieving all rows of the Company, Customer, and Orders tables, the following PostgreSQL statements are executed, respectively:

```
SELECT * FROM public."Company";
SELECT * FROM public."Customer";
SELECT * FROM public."Orders";
```

For displaying all orders by ascending customer ID (CID) or ascending game ID (GID), the query on the Orders table is modified slightly:

```
SELECT * FROM public."Orders" ORDER BY "CID" ASC;
SELECT * FROM public."Orders" ORDER BY "GID" ASC;
```

These queries group together the orders by either CID or GID.

Search Menu

Any user can search the Games table by a game's name, genre, or publisher/company. The user's search input is read into a variable called *search*, trimmed of excess white space, and converted to all lowercase. The database is then queried in one of the following three ways, depending on which attribute the user decided to search by:

```
SELECT * FROM public."Games" WHERE LOWER("Title") LIKE \'%' +
      search + '%\';
SELECT * FROM public."Games" WHERE LOWER("Genre") LIKE \'%' +
      search + '%\';
SELECT * FROM public."Games" WHERE LOWER("Publisher") LIKE \'%' +
      search + '%\';
```

Employee accounts may also search the Orders table by CID or GID, to see how many orders a particular customer has made or how many orders have been placed for a particular game:

```
SELECT * FROM public."Orders" WHERE LOWER("CID") LIKE \'%' +
      search + '%\';
SELECT * FROM public."Orders" WHERE LOWER("GID") LIKE \'%' +
      search + '%\';
```

Modify Menu

Employees may add a new entry to the Games, Company, Customer, or Order table at any time. Depending on which table they are concerned with, we must read in the user input for every attribute of that table. For example, for inserting into the Games table, the following code is executed:

```
print("Enter the following information for the new game.")
gid = input("GID: ")
title = input("Title: ")
publisher = input("Publisher: ")
developer = input("Developer: ")
genre = input("Genre: ")
year = input("Year: ")
```

```
system = input("System: ")
price = input("Price: ")
```

After the inputs are stripped of extra white space and the *year* and *price* string inputs are converted to *int* and *float*, respectively, the data is inserted into the table using the following commands:

```
cur.execute('INSERT INTO public."Games" VALUES (\'%s\', \'%s\',
      \'%s\', \'%s\', \'%s\', %d, \'%s\', %f);' % (gid, title,
      publisher, developer, genre, year, system, price))
conn.commit()
```

Since the %s, %d, and %f placeholders are supported by Python, we use them to substitute in the inputs from the user into the query. Note that because the table is being updated, it is essential that we commit the change to the database.

The *INSERT* statements for the Company and Customer tables operate very similarly, but for the Orders table, we ask the user for an order ID (OID) as well as a GID and CID. The GID and CID must then be validated so that we know they exist in the other tables, and then the Sale amount is automatically pulled from the Price associated with the given GID. The Date attribute is generated by *CURRENT_DATE* in SQL. See the code below:

```
print("Enter the following information for the new order.")
oid = input("OID: ")
gid = input("GID: ")
customer = input("CID: ")

cur = conn.cursor()
cur.execute('SELECT "Price" FROM public."Games" WHERE "GID" = \''
      + gid + '\';')
record = cur.fetchone()
sale = record[0]

cur.execute('SELECT "CID" FROM public."Customer" WHERE "CID" =
      \'' + customer + '\';')
record = cur.fetchone()
cid = record[0]

cur.execute('INSERT INTO public."Orders" VALUES (\'%s\', \'%s\',
      \'%s\', \'%f\', CURRENT_DATE);' % (oid, gid, cid, sale))
conn.commit()
```

By attempting to *SELECT* the GID and CID inputs, we check whether or not those values exist in their respective tables. The above code is executed within a try-except block, so if the GID or CID is not found, an exception will be caught.

Employees may also delete from the Games, Company, or Customer tables. The user is asked to input the game GID, company Name, or customer CID, and one of the following is executed:

```
cur.execute('DELETE FROM public."Games" WHERE "GID" = \'%s\';' %
      (gid))
cur.execute('DELETE FROM public."Company" WHERE "Name" = \'%s\';'
      % (name))
cur.execute('DELETE FROM public."Customer" WHERE "CID" = \'%s\';'
      % (cid))
```

Users with employee accounts may also update the price of the game. They are asked to input the game's GID and the new price, which are read into variables called *gid* and *price*, and the following is executed:

```
cur.execute('UPDATE public."Games" SET "Price" = %f WHERE "GID" =
      \'%s\';' % (price, gid))
conn.commit()
```

Employees also have the privilege to create new user accounts or delete accounts of both types: customer or employee. Employee accounts require an employee key to be created; for demonstration purposes, we set this key to equal 1234. Below is the code used to create a new role for the database:

```
username = input("Enter new username: ")
password = pw.getpass("Enter new password: ")
confirmpass = pw.getpass("Confirm new password: ")
key = pw.getpass("Enter employee key (ignore if customer): ")

if password != confirmpass:
       print("Passwords do not match. Please try again.")

elif password == confirmpass and key != '1234':
      print("Creating customer account.")
      cur = conn.cursor()
      cur.execute("CREATE USER %s WITH PASSWORD '%s';"
            %(username, password))
```

```
            cur.execute("GRANT SELECT ON TABLE public.\"Games\" TO %s;
                  GRANT SELECT ON TABLE public.\"Company\" TO %s;"
                  %(username, username))
            conn.commit()

      elif password == confirmpass and key == '1234':
            print("Creating employee account.")
            cur = conn.cursor()
            cur.execute("CREATE USER %s WITH PASSWORD '%s';"
                  %(username, password))
            cur.execute("ALTER USER %s CREATEROLE; GRANT INSERT,
                  SELECT, UPDATE, DELETE ON TABLE public.\"Company\" TO
                  %s WITH GRANT OPTION; GRANT INSERT, SELECT, UPDATE,
                  DELETE ON TABLE public.\"Customer\" TO %s WITH GRANT
                  OPTION; GRANT INSERT, SELECT, UPDATE, DELETE ON TABLE
                  public.\"Games\" TO %s WITH GRANT OPTION; GRANT
                  INSERT, SELECT, UPDATE, DELETE ON TABLE
                  public.\"Orders\" TO %s WITH GRANT OPTION;"
                  %(username, username, username, username, username))
            conn.commit()
```

Different permissions are granted to the new role depending on whether or not the employee key is entered correctly. To delete an account, the user is asked for the username of the role they wish to delete, which is stored in a variable called *username*, and the following is executed:

```
cur.execute("REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public
FROM %s;
REVOKE ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public FROM %s;
REVOKE ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public FROM %s;
DROP USER %s;" %(username, username, username, username))
```

This removes the user from the Login/Group Roles on the database end in pgAdmin.

VII. Conclusions

Based on how this database is implemented, this system would be ideal for any retail game stores as it holds a database of games for the store, and provides security in holding sensitive information by restricting access from the public.