# Online PC Game Store

By Brian Canoni

4/23/14

Professor Alan Laboseur

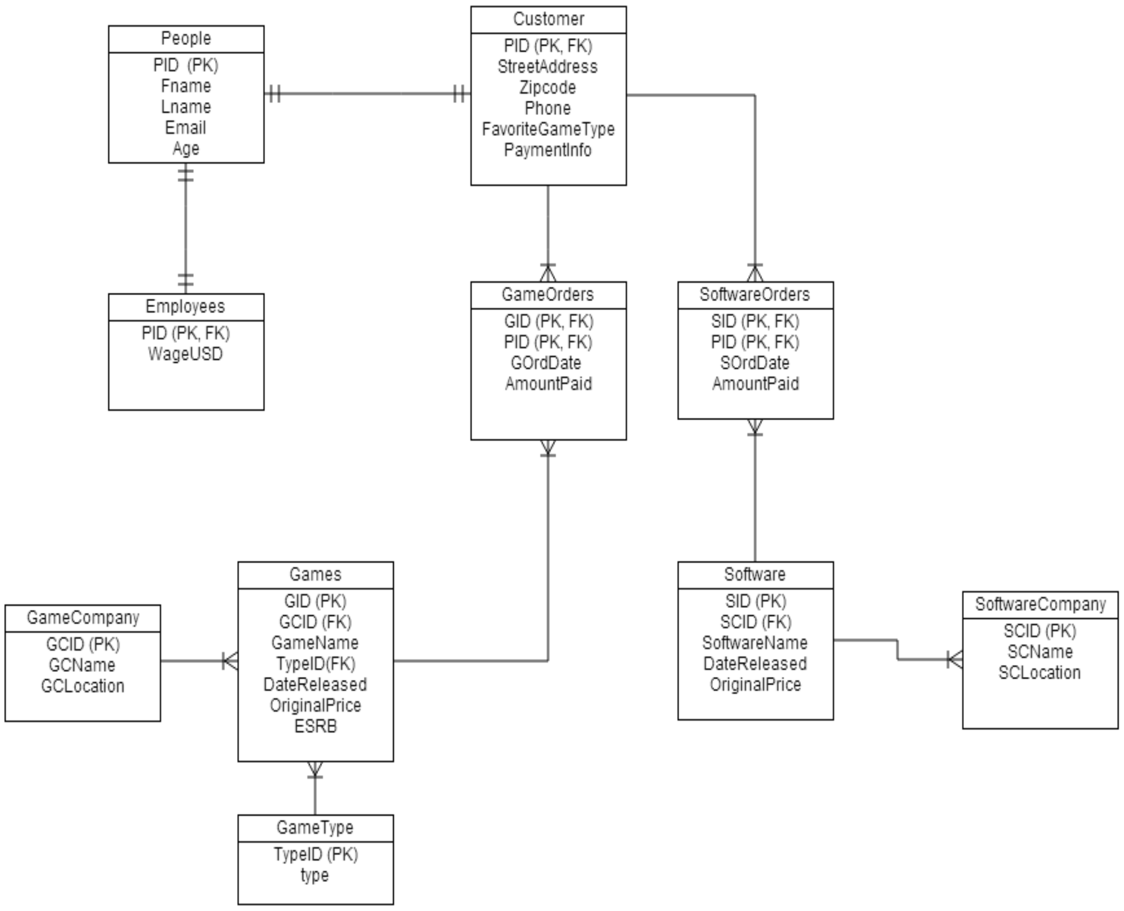
# Table of Contents

# Executive Summary

The main goal of this database is to organize and improve the business of selling computer games online, from the approval of games all the way down to the delivery of games to the customer. Additionally the feature of selling software online has been added and this database allowing for software to be sold right beside the games. By centralizing and very accurately modeling the company data, everything from data integrity to the scalability of the company is improved.  This database will smooth the process of adding new games to the system and software, yet also be able to remember customers and their information so that repeat customers can easily and quickly make new purchases.

In the following pages I will explain each of the tables within my design and their purpose in creating an easy to use relational database that will streamline business. People who will use this system will be Customers who buy the games, game companies/software companies who will create the content and Employees who will approve products to be sold at the store negotiating a price and fee. Employees based on position will also be allowed to change values in the tables such as game price.

# Entity Relationship Diagram



Following This diagram I will go into detail explaining each of the above tables starting with the people table.

# People Table

The People table is used to keep track of people who will be using this database (customers and employees). It stores information such as their name and email.

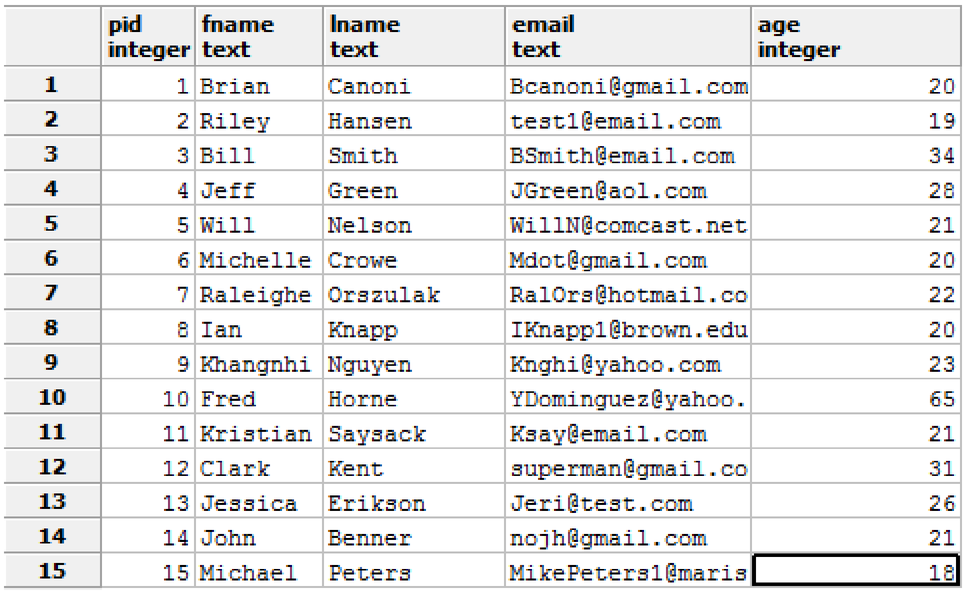
**Functional Dependencies:**

Pid -> Fname, Lname , email , age

**CreateStatement:**

```
CREATE TABLE if not exists people
(
pid serial NOT NULL,
fname text NOT NULL,
lname text NOT NULL,
email text NOT NULL,
age integer NOT NULL,
CONSTRAINT people_pkey PRIMARY KEY (pid),
CONSTRAINT people_email_key UNIQUE (email)
);
```

**Sample Data:**



| | pid integer | fname text | lname text | email text | age integer |
|---|---|---|---|---|---|
| 1 | 1 | Brian | Canoni | Bcanoni@gmail.com | 20 |
| 2 | 2 | Riley | Hansen | test1@email.com | 19 |
| 3 | 3 | Bill | Smith | BSmith@email.com | 34 |
| 4 | 4 | Jeff | Green | JGreen@aol.com | 28 |
| 5 | 5 | Will | Nelson | WillN@comcast.net | 21 |
| 6 | 6 | Michelle | Crowe | Mdot@gmail.com | 20 |
| 7 | 7 | Raleighe | Orszulak | RalOrs@hotmail.co | 22 |
| 8 | 8 | Ian | Knapp | IKnapp1@brown.edu | 20 |
| 9 | 9 | Khangnhi | Nguyen | Knghi@yahoo.com | 23 |
| 10 | 10 | Fred | Horne | YDominguez@yahoo. | 65 |
| 11 | 11 | Kristian | Saysack | Ksay@email.com | 21 |
| 12 | 12 | Clark | Kent | superman@gmail.co | 31 |
| 13 | 13 | Jessica | Erikson | Jeri@test.com | 26 |
| 14 | 14 | John | Benner | nojh@gmail.com | 21 |
| 15 | 15 | Michael | Peters | MikePeters1@maris | 18 |

# Customer Table

The customer table is used to keep track of customers of the game buying service. It will store necessary information for the transaction such as address and phone number. It is important to remember customers so that repeat transactions can be easier for them to complete.

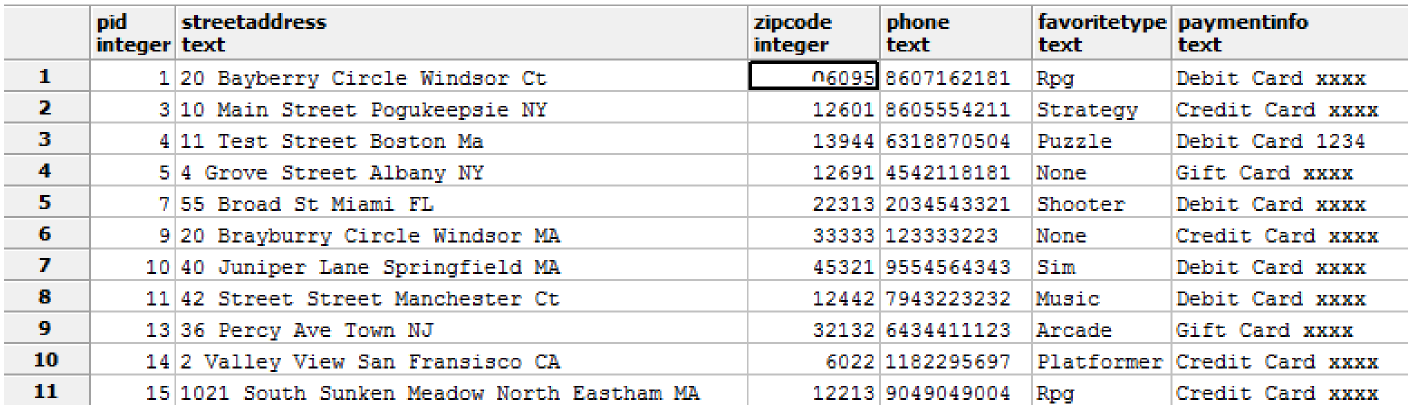
**Functional Dependencies:**

Pid -> StreetAddress, Zipcode, Phone, FavoriteGameType, PaymentInfo

**Create Statement:**

```
Create table if not exists Customer
(
pid serial not null references people(pid) primary key,
Streetaddress text not null,
zipcode int not null,
phone text not null ,
favoritetype text,
paymentinfo text not null
);
```

**Sample Data:**



| | pid integer | streetaddress text | zipcode integer | phone text | favoritetype text | paymentinfo text |
|---|---|---|---|---|---|---|
| 1 | 1 | 20 Bayberry Circle Windsor Ct | 06095 | 8607162181 | Rpg | Debit Card xxxx |
| 2 | 3 | 10 Main Street Pogukeepsie NY | 12601 | 8605554211 | Strategy | Credit Card xxxx |
| 3 | 4 | 11 Test Street Boston Ma | 13944 | 6318870504 | Puzzle | Debit Card 1234 |
| 4 | 5 | 4 Grove Street Albany NY | 12691 | 4542118181 | None | Gift Card xxxx |
| 5 | 7 | 55 Broad St Miami FL | 22313 | 2034543321 | Shooter | Debit Card xxxx |
| 6 | 9 | 20 Brayburry Circle Windsor MA | 33333 | 123333223 | None | Credit Card xxxx |
| 7 | 10 | 40 Juniper Lane Springfield MA | 45321 | 9554564343 | Sim | Debit Card xxxx |
| 8 | 11 | 42 Street Street Manchester Ct | 12442 | 7943223232 | Music | Debit Card xxxx |
| 9 | 13 | 36 Percy Ave Town NJ | 32132 | 6434411123 | Arcade | Gift Card xxxx |
| 10 | 14 | 2 Valley View San Fransisco CA | 6022 | 1182295697 | Platformer | Credit Card xxxx |
| 11 | 15 | 1021 South Sunken Meadow North Eastham MA | 12213 | 9049049004 | Rpg | Credit Card xxxx |

# Employee Table

In the Employee holds all the employees of the company. It contains their wages as well this will help in keeping track of current employees.

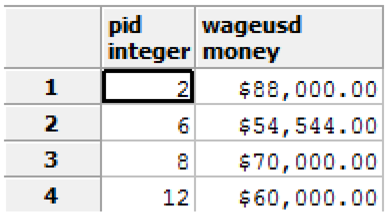
**Functional Dependencies:**

Pid -> WagesUSD

**Create Statement:**

```
Create table if not exists Employees
(
pid serial not null references people(pid) primary key,
WageUSD money not null
);
```

**Sample Data:**



| | pid<br>integer | wageusd<br>money |
|---|---|---|
| 1 | 2 | $88,000.00 |
| 2 | 6 | $54,544.00 |
| 3 | 8 | $70,000.00 |
| 4 | 12 | $60,000.00 |

# GameOrders

This table keeps track of orders a customer has placed from the online inventory in terms of games. This will be a record of all game purchases and will also keep track of when the purchase was made for better sorting.

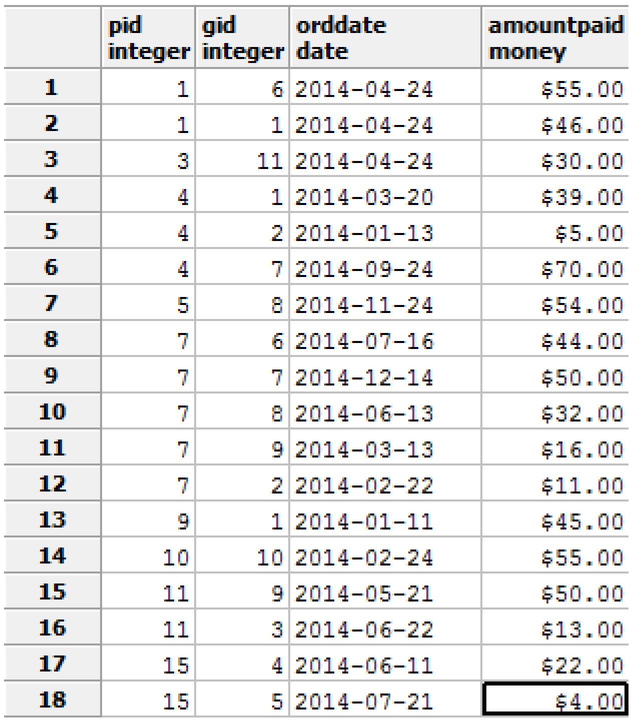
**Functional Dependencies:**

Pid, gid -> OrdDate,AmountPaid

**Table Creation:**

```
Create table if not exists GameOrders
(
pid serial not null references customer(pid),
gid serial not null references games(gid),
OrdDate DATE not null,
AmountPaid money not null,
primary key (pid,gid)
);
```

**Sample Data:**



| | pid integer | gid integer | orddate date | amountpaid money |
|---|---|---|---|---|
| 1 | 1 | 6 | 2014-04-24 | $55.00 |
| 2 | 1 | 1 | 2014-04-24 | $46.00 |
| 3 | 3 | 11 | 2014-04-24 | $30.00 |
| 4 | 4 | 1 | 2014-03-20 | $39.00 |
| 5 | 4 | 2 | 2014-01-13 | $5.00 |
| 6 | 4 | 7 | 2014-09-24 | $70.00 |
| 7 | 5 | 8 | 2014-11-24 | $54.00 |
| 8 | 7 | 6 | 2014-07-16 | $44.00 |
| 9 | 7 | 7 | 2014-12-14 | $50.00 |
| 10 | 7 | 8 | 2014-06-13 | $32.00 |
| 11 | 7 | 9 | 2014-03-13 | $16.00 |
| 12 | 7 | 2 | 2014-02-22 | $11.00 |
| 13 | 9 | 1 | 2014-01-11 | $45.00 |
| 14 | 10 | 10 | 2014-02-24 | $55.00 |
| 15 | 11 | 9 | 2014-05-21 | $50.00 |
| 16 | 11 | 3 | 2014-06-22 | $13.00 |
| 17 | 15 | 4 | 2014-06-11 | $22.00 |
| 18 | 15 | 5 | 2014-07-21 | $4.00 |

# SoftwareOrders

This table keeps track of the online software orders placed just like the game orders. Since there are some fundamental differences between software and games their orders will be in a separate table to help organize.

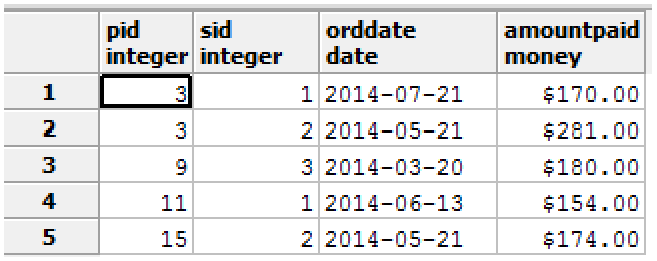
**Functional Dependencies:**

Pid, sid -> OrdDate,AmountPaid

**Table Creation:**

```
Create table if not exists SoftwareOrders
(
pid serial not null references customer(pid),
sid serial not null references software(sid),
OrdDate DATE not null,
AmountPaid money not null,
primary key (pid,sid)
);
```

**Sample Data:**



| | pid integer | sid integer | orddate date | amountpaid money |
|---|---|---|---|---|
| 1 | 3 | 1 | 2014-07-21 | $170.00 |
| 2 | 3 | 2 | 2014-05-21 | $281.00 |
| 3 | 9 | 3 | 2014-03-20 | $180.00 |
| 4 | 11 | 1 | 2014-06-13 | $154.00 |
| 5 | 15 | 2 | 2014-05-21 | $174.00 |

# Games

This table stores the information of games with details such as its name and release date. Games here have been approved for sale by employees of the company and are given a price which can later be modified for sales later on.

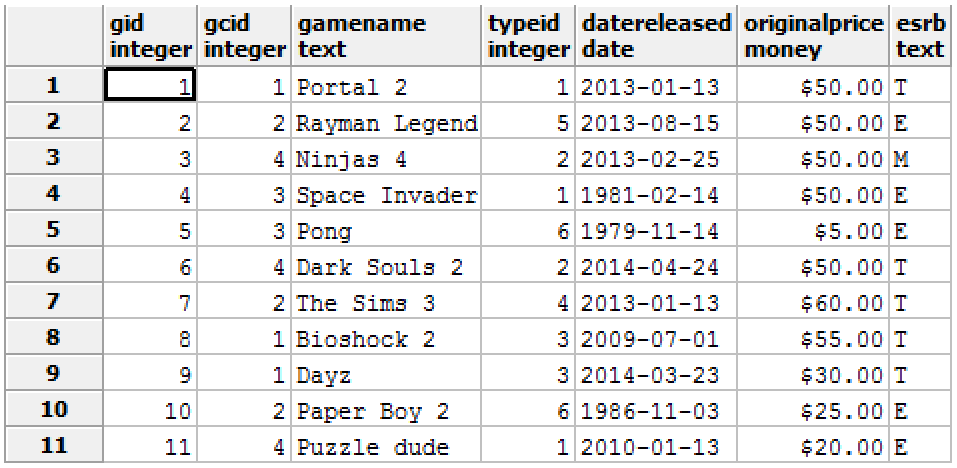
**Functional Dependencies:**

Gid -> GCID, GameName, TypeID , DateReleased, OriginalPrice, ESRB

**Table Creation:**

```
Create table if not exists SoftwareOrders
(
pid serial not null references customer(pid),
sid serial not null references software(sid),
OrdDate DATE not null,
AmountPaid money not null,
primary key (pid,sid)
);
```

**Sample Data:**



| | gid integer | gcid integer | gamename text | typeid integer | datereleased date | originalprice money | esrb text |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Portal 2 | 1 | 2013-01-13 | $50.00 | T |
| 2 | 2 | 2 | Rayman Legend | 5 | 2013-08-15 | $50.00 | E |
| 3 | 3 | 4 | Ninjas 4 | 2 | 2013-02-25 | $50.00 | M |
| 4 | 4 | 3 | Space Invader | 1 | 1981-02-14 | $50.00 | E |
| 5 | 5 | 3 | Pong | 6 | 1979-11-14 | $5.00 | E |
| 6 | 6 | 4 | Dark Souls 2 | 2 | 2014-04-24 | $50.00 | T |
| 7 | 7 | 2 | The Sims 3 | 4 | 2013-01-13 | $60.00 | T |
| 8 | 8 | 1 | Bioshock 2 | 3 | 2009-07-01 | $55.00 | T |
| 9 | 9 | 1 | Dayz | 3 | 2014-03-23 | $30.00 | T |
| 10 | 10 | 2 | Paper Boy 2 | 6 | 1986-11-03 | $25.00 | E |
| 11 | 11 | 4 | Puzzle dude | 1 | 2010-01-13 | $20.00 | E |

# GameType

This table stores the different types a game can be so that customers can find games they are looking for by searching by genre. It is in its own table so that game types can be added or changed more easily.

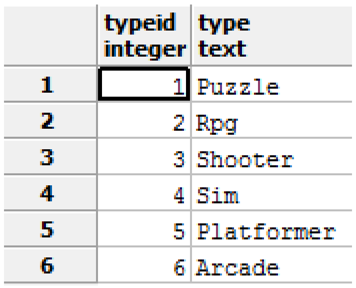
**Functional Dependencies:**

TypeID -> Type

**Table Creation:**

```
create table if not exists Gametype
(
TypeID serial primary key,
type text
);
```

**Sample Data:**



| | typeid integer | type text |
|---|---|---|
| 1 | 1 | Puzzle |
| 2 | 2 | Rpg |
| 3 | 3 | Shooter |
| 4 | 4 | Sim |
| 5 | 5 | Platformer |
| 6 | 6 | Arcade |

# Software

This table stores the software that is approved for sale for the software side of the online service.

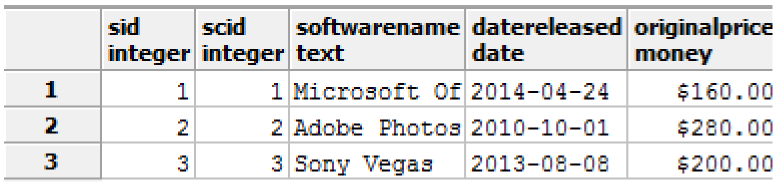
**Functional Dependencies:**

Sid -> SCID, GSoftwareName,  DateReleased, OriginalPrice

**Table Creation:**

```
Create table if not exists Software
(
Sid serial not null primary key,
scid serial not null references gamecompany(gcid),
softwarename text not null,
datereleased date not null,
originalprice money not null
);
```

**Sample Data:**



| | sid integer | scid integer | softwarename text | datereleased date | originalprice money |
|---|---|---|---|---|---|
| 1 | 1 | 1 | Microsoft Of | 2014-04-24 | $160.00 |
| 2 | 2 | 2 | Adobe Photos | 2010-10-01 | $280.00 |
| 3 | 3 | 3 | Sony Vegas | 2013-08-08 | $200.00 |

# GameCompany

This table will store all of the companies that produce games that we are currently working with. Storing them in this fashion will better organize which games are created by what company so that deals with specific game companies can be made more easily.

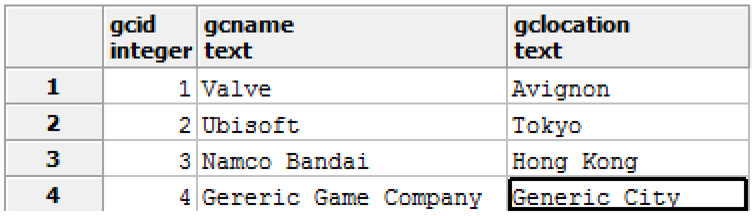
**Functional Dependencies:**

GCID -> GCName , GCLocation

**Table Creation:**

```
Create table if not exists GameCompany
(
gcid serial not null primary key,
gcname text,
gclocation text
);
```

**Sample Data:**



| | gcid integer | gcname text | gclocation text |
|---|---|---|---|
| 1 | 1 | Valve | Avignon |
| 2 | 2 | Ubisoft | Tokyo |
| 3 | 3 | Namco Bandai | Hong Kong |
| 4 | 4 | Gereric Game Company | Generic City |

# SoftwareCompany

This table will store all of the companies that produce software that we are currently working with. Storing them in this fashion will better organize what programs are created by what company so that deals with specific game companies can be made more easily.

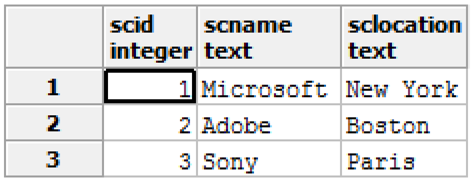
**Functional Dependencies:**

SCID -> SCName , SCLocation

**Table Creation:**

```
Create table if not exists SoftwareCompany
(
scid serial not null primary key,
scname text,
sclocation text
);
```

**Sample Data:**



| | scid integer | scname text | sclocation text |
|---|---|---|---|
| 1 | 1 | Microsoft | New York |
| 2 | 2 | Adobe | Boston |
| 3 | 3 | Sony | Paris |

# Views

An example of two common views that could be used with this system would be a GamesList and a SoftwareList which would display all the information of each of the products.

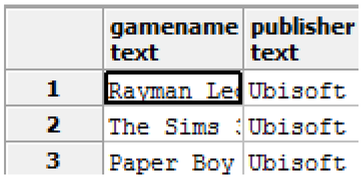
## GamesList

**Create Statement:**

```
create view GamesList
(GameName, Publisher, Type, DateReleased, OriginalPrice, ESRB) AS
select g.gameName, gc.gcname , gt.type , g.datereleased, g.originalprice, g.esrb
from games g, gamecompany gc, gametype gt
where g.gcid = gc.gcid and gt.typeid = g.typeid;
```

This provides a view containing all of the information about all of the current games in the database. An example of use of this view is as follows

```
##Finds all the games produced by Ubisoft
select GameName, Publisher
from gameslist
where publisher = 'Ubisoft'
```

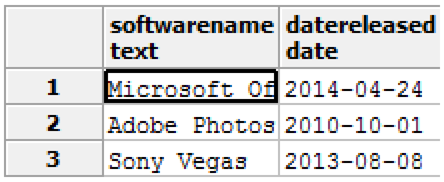


| | gamename text | publisher text |
|---|---|---|
| 1 | Rayman Le | Ubisoft |
| 2 | The Sims | Ubisoft |
| 3 | Paper Boy | Ubisoft |

## SoftwareList

This like the gameslist view provides easy access to all of the information of each available software product. Following this is another example of use for this particular view.

```
##Finds the name and release date of all products
select s.softwarename,s.datereleased
from softwareList s;
```



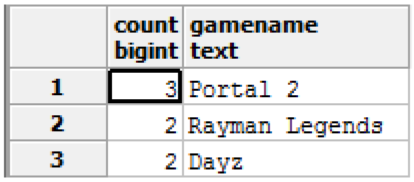
| | softwarename text | datereleased date |
|---|---|---|
| 1 | Microsoft Of | 2014-04-24 |
| 2 | Adobe Photos | 2010-10-01 |
| 3 | Sony Vegas | 2013-08-08 |

# Reports

Reports such as the two following examples can be used to check up on how well things are running and also find interesting facts about the business.

**Top 3 most bought games**

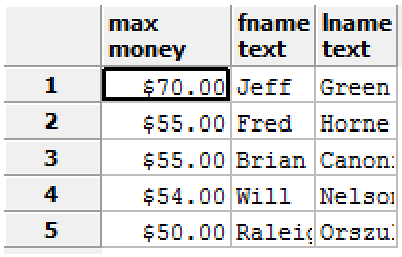
```
Select count(go.gid), g.gamename
From gameorders go, games g
where go.gid = g.gid
Group by go.gid,g.gamename
Order by count(go.gid) desc
Limit 3;
```



| | count bigint | gamename text |
|---|---|---|
| 1 | 3 | Portal 2 |
| 2 | 2 | Rayman Legends |
| 3 | 2 | Dayz |

**Top 5 customers and the single most expensive game they have purchased prices:**

```
Select max(go.amountPaid) , p.fname, p.lname
From gameorders go, customer c, people p
Where go.pid = c.pid and c.pid = p.pid
Group by go.amountpaid, p.fname, p.lname
Order by max(go.amountpaid) desc
Limit 5;
```



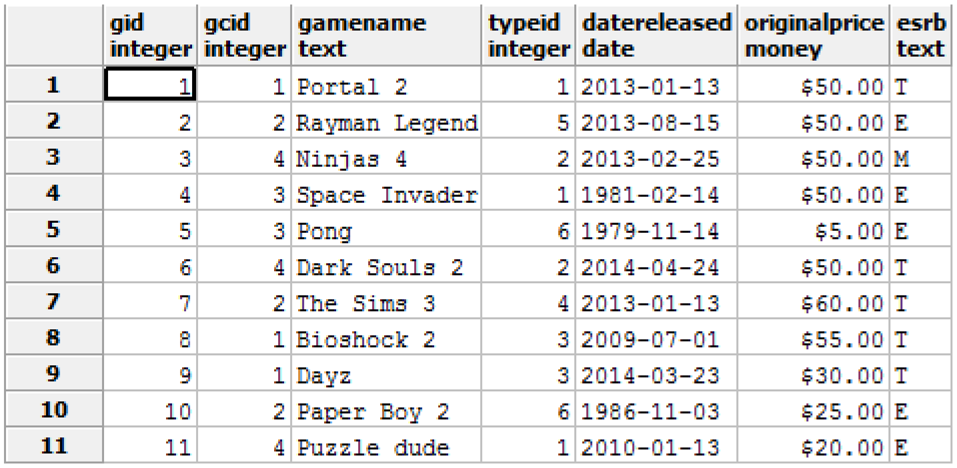
| | max money | fname text | lname text |
|---|---|---|---|
| 1 | $70.00 | Jeff | Green |
| 2 | $55.00 | Fred | Horne |
| 3 | $55.00 | Brian | Canon |
| 4 | $54.00 | Will | Nelso |
| 5 | $50.00 | Ralei | Orszu |

# Stored Procedures

A stored procedure that would most likely be used very often in my database would be the ability to add a game to the games table. This would occur very often because new games are being released rapidly. By utilizing a stored procedure such as this it will make the job of adding a new game less confusing for an employee who is potentially less savvy with databases.

```
Create function addNewGame (gamecompanyid int,title text, genreID int, theDate date, price money,
esrbCode text)
Returns void AS $$
Declare newGameID int;
Begin
        Insert into games(gid, gcid, gamename, typeid, datereleased, originalprice, esrb)
        Values (newGameID, gamecompanyid, title, genreID, thedate, price, esrbCode);
End;
$$ Language plpgsql
```

Using the addNewGame procedure will make adding new games easy.



| | gid integer | gcid integer | gamename text | typeid integer | datereleased date | originalprice money | esrb text |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Portal 2 | 1 | 2013-01-13 | $50.00 | T |
| 2 | 2 | 2 | Rayman Legend | 5 | 2013-08-15 | $50.00 | E |
| 3 | 3 | 4 | Ninjas 4 | 2 | 2013-02-25 | $50.00 | M |
| 4 | 4 | 3 | Space Invader | 1 | 1981-02-14 | $50.00 | E |
| 5 | 5 | 3 | Pong | 6 | 1979-11-14 | $5.00 | E |
| 6 | 6 | 4 | Dark Souls 2 | 2 | 2014-04-24 | $50.00 | T |
| 7 | 7 | 2 | The Sims 3 | 4 | 2013-01-13 | $60.00 | T |
| 8 | 8 | 1 | Bioshock 2 | 3 | 2009-07-01 | $55.00 | T |
| 9 | 9 | 1 | Dayz | 3 | 2014-03-23 | $30.00 | T |
| 10 | 10 | 2 | Paper Boy 2 | 6 | 1986-11-03 | $25.00 | E |
| 11 | 11 | 4 | Puzzle dude | 1 | 2010-01-13 | $20.00 | E |

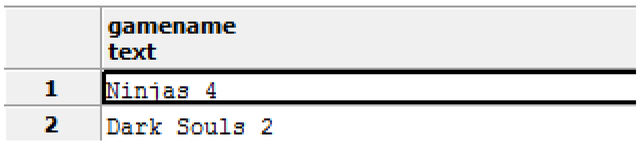
# Triggers

An example of a very useful trigger would be that when a new customer is added and their favorite game type is declared to then return all of the games that also belong to the same game type. This would hopefully lead to the customer finding games that they would want more quickly and leading to a purchase.

```
Create function addedCustomerTrigger()
Declare custID integer;
Returns trigger As $$
Begin
        Select g.gamename
        From customer c, games g, gametype gt
        Where gt.typeid = g.typeid and c.favoritetype = gt.type and c.pid =custID

Return null;
```

If for example a customer was added who liked RPG games than the following results would be returned This would be very useful as it will direct the customer to games that are their favorite kind.



|   | gamename<br>text |
|---|---|
| 1 | Ninjas 4 |
| 2 | Dark Souls 2 |

# Security

Security for this database is as following.

Employees

- Have access to the games, and software tables to be able to add products and edit them.
- They can add Game and software companies and edit their tables.
- They also have access to all of the stored procedures and the orders table
- Access to the game type table

Admins/Managers

- Have access to all of the tables, functions, and triggers

Customers

- They only have access to editing parts of their own information
- They can place orders for both games and software but aren't allowed to alter the tables beyond that.

**Executive Security Summary**

In summary customers are limited to only their own information and making purchases. While employees and more specifically the admins can alter most of the current data to suit the needs of the company and to add new products as well as take them away or change the price.

# Known Problems

Some of the current known problems are as follows:

- Adding a new game with a game type that hasn't been used yet can cause problems because the game type has to be added first this can be annoying.
- In retrospect games can be part of multiple game types and limiting the game to one could potentially not connect the customer to as many games
- Games are typically produced by a series of companies rather than one and this database only allows for one to be named
- Game companies and software companies are so similar that I probably could have created a companies entity and then branched the two out of that.

# Future Enhancements

Some of the potential future enhancements that could be included are as follows:

- A trigger to block users that are underage from purchasing a game if it is rated in an age range older than what they are.
- A way to purchase games in bundles.
- An easier way for sales to be implemented rather than going in and changing the price manually.
- Customers to have accounts with usernames and passwords