

Student Name: Eoghan Hynes

Student ID Number: A00107408

Class Group: AL_KSWPT_R_5 MSc Software Eng.

Class: Database Systems 5

Project: MySQL

Table of Contents

<u>Chapter</u>	<u>Page Numbers</u>
1. Database Concept	3
Tables with sample data	3-4
Expected Functionality	5
Errors & Anomalies	5
2. Logical Design	6
ER Diagram	6
1 st Normal Form	6
2 nd Normal Form	6-7
3 rd Normal Form	7-8
Cardinality	8
Final Design	8
Errors & Anomalies	8
3. Physical Design	
Implementation of Database schema	9-10
Implementation of Test Data	11-23
Table Indexing	24-25
Query Optimisation	26-27
Errors & Anomalies	28
4. User Base	
Define Users and their Permissions	28-30
Test user roles	30-34
Errors & Anomalies	34
5. Auditing & Logging (Business Rules & housekeeping)	
Procedures & Triggers	35-42
Design Update	43-44
Errors & Anomalies	44-45
6. Front End Technology	45-46
7. Initial Rollout	46
Errors & Anomalies	46
8. Sign-Off	
A user manual of the database application.	47-49

1. Database Concept

My client 'Game World Ltd.', needs an inventory & credit control database to keep track of purchases and sales of their stock. They are an electronic device distributor dealing specifically with Game Consoles, PCs and VR headsets for the Computer Gaming market, a sector of the economy seeing strong growth, worth over €284m last year. The database will have to keep track of stock purchases, sales, suppliers and customers as well as debit and credit amounts owed and due.

Game World Ltd. have credit accounts with five suppliers. These are Steam, Sony, Microsoft, Facebook & Dell. They have 30 days credit terms agreed with Steam and Sony, 60 days with Microsoft and Dell, & 90 days with Facebook.

My client Game World Ltd. are slowly building up their credit customer base since the economic collapse of 2008 and currently deal regularly with 8 credit customers. These eight customers are 'Game Stop', 'GameXchange', 'Garry's Game Loft', 'Toy World', 'Games Ltd.', 'Toy Land', 'Larry's Game Lab' & 'Best Buy'. My client has a strict 30 day credit agreement with all of his customers.

Tables with sample data:

Table: Suppliers				
ID	Name	Address	email	Amount Owed
1001	Steam	Capel St, Dublin	steam@steam.com	10000.00
1002	Sony	High St, Dublin	sony@sony.ie	1342.99
1003	Microsoft	Store St, Dublin	microsoft@msn.com	0
1004	Facebook Inc.	Grand Canal Square, Dublin	support@book.net	7100.0
1005	Dell Ireland	CheryWood Park, Dublin	admin@dell.com	100,000.00

Table: Customers				
ID	Name	Address	Email	Amount Due
1001	Game Stop	16, Henry Street, Dublin	custcare@amestop.com	2500.23
1002	GameXchange	73, Talbot St, Dublin	gamexchange@msn.ie	1050.00
1003	Garry's Game Loft	High Street, Dublin	ggl@yahoo.net	3000,00
1004	Toy World	Dundrum Centre, Dublin	ToyWorld@gmail.com	300.00
1005	Games Ltd	Naas Road, Dublin	contact@gamesltd.ie	769.87
1006	Toy Land	Naas Road, Dublin	admin@toyland.ie	3000.00
1007	Larry's Game Lab	Blanchardstown, Dublin	Lgl@gmail.com	2400,99
1008	Best Buy	23, Store St, Dublin	support@bestbuy.com	246.56

Table: Products			
Make	Model	Type	Price
Steam	Alienware A	Console	599.00
Steam	Alienware B	Console	499.00
Steam	Maingear Drift	PC	1099.99
Steam	Syber X	Console	499.00
Steam	Syber P	Console	399.00

Steam	Syber I	Console	275.00
Steam	SteamVR	VRHeadset	550.00
Sony	PS4	Console	349.99
Sony	PlayStationVR	VRHeadset	349.99
Microsoft	Xbox One	Console	379.99
Microsoft	Xbox 360	Console	249.99
Microsoft	HoloLens	VRHeadset	3000.00
Facebook	Oculus Rift	VRHeadset	600.00
Dell	Alienware x51	PC	16489.00
Dell	Area 51	PC	3398.99
Dell	Inspiron	PC	1091.01
Sony	VAIO	PC	900.00
Sony	PS3	Console	250.00

Table: Consoles				
Model	Mem	GPU	CPU	Quantity
Alienware B	4GB DDR	Asus	Intel Corei3	4
Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	4
Cyber X	16GB DDR	Nvidia Gerforce 980	Intel Core i7	5
Cyber P	8GB DDR	Nvidia Geforce 960	Intel Core i5	2
Cyber I	4GB	Nvidia Geforce 750	Intel Corei3	2
Xbox One	8GB	AMD	D3D11	2
Xbox 360	526MB	ATI Xenos	ATI Xenon	6
PS3	525MB	AMD	Nvidia	7
PS4	8GB	Intel Corei7	Nvidia	2

Table: VR Headsets				
Model	Resolution	Field Of View	Refresh Rate	Quantity
SteamVR	2160x1200	100	90Hz	4
PlaystationVR	1080p	110	100Hz	3
HoloLens	1080p	100	90Hz	5
Oculus Rift	1080p	110	120Hz	5

Table: PCs				
Model	CPU	R.A.M	Memory	Quantity
Maingear Drift	Intel Corei7	16GB	1TB SSD	5
Alienware x51	Intel Corei7	16GB	1TB	5
Area 51	Intel Corei7	16GB	1TB	4
Ispiron	Intel Corei5	8GB	500GB	2
VAIO	Intel Corei3	4GB	300GB	3

Expected Functionality

Tables : 'Consoles' 'VRHeadsets' and 'PCs' are children of the Products table. The products table records the prices of all of the products that my client stocks, and the children tables record the quantities of each model currently in stock.

Amounts owed to suppliers and owed by customers are calculated by multiplying product prices by the amounts purchased and sold.

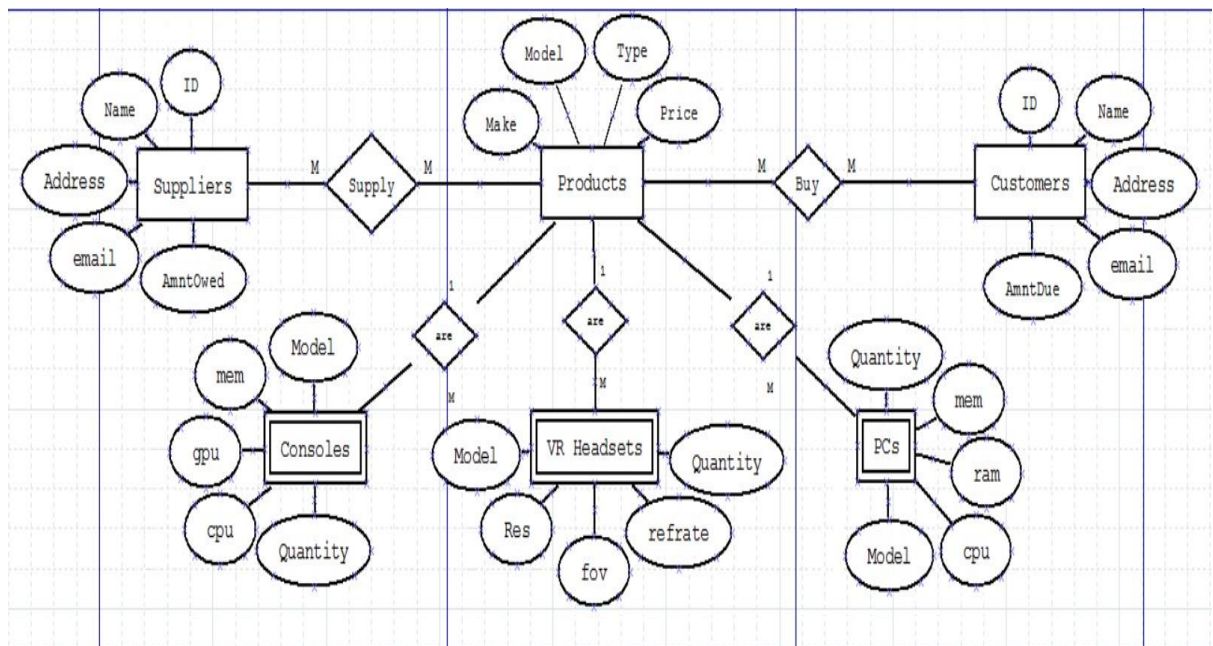
- Fundamentally, the database must keep count of the number of each model in stock. Purchases and Sales of these products will affect the amounts in stock. My client wants a Stock Checking procedure built into the database to produce a convenient list of stock holdings at any time.
- My client wants the Database to cater for low stock to allow for reordering, especially if there aren't enough of a given model to fulfil a sale. High stock counts may be used to calculate discounts in future to shift old stock such as older model consoles that aren't selling well.
- The Price of Products must also be recorded so total amounts from purchases and sales can be calculated. My client is also anxious to flag a warning if any customer account goes over 10,000 in credit having been badly stung during the economic crash in 2008.

I will use MySQL Workbench 6.3 for the design of this database. I will design the user profiles using MS DOS. I will design the front end using Mars Eclipse. This report will include screen shots from the applications.

Errors & Anomalies

The database will not allow for product returns from customers or to suppliers in this iteration.

2. Logical Design



Tables in 1st Normal Form

Table : Suppliers				
ID	Name	Address	Email	Amount Owed

Table: Customers				
ID	Name	Address	Email	Amount Due

Table: Products			
Make	Model	Type	Price

Table: Consoles				
Model	Memory	G.P.U	C.P.U	Quantity

Table: VR Headsets				
Model	Resolution	Field Of View	Refresh Rate	Quantity

Table: PCs				
Model	C.P.U	R.A.M	Memory	Quantity

Tables in 2nd Normal Form

Table : Suppliers				
ID	Name	Address	Email	Amount Owed

Table: Purchases					
Purchase ID	SupplierID	Model	Quantity	Date	Total Amount

Table: Customers				
ID	Name	Address	Email	Amount Due

Table: Sales					
SaleID	CustomerID	Model	Quantity	Date	Total Amount

Table: Products			
Make	Model	Type	Price

Table: Consoles					
Make	Model	Memory	G.P.U	C.P.U	Quantity
Foreign Key Make References Products(Make)					

Table: VR Headsets					
Make	Model	Resolution	Field Of View	Refresh Rate	Quantity

Table: PCs					
Make	Model	C.P.U	R.A.M	Memory	Quantity
Foreign Key Make References Products(Make)					

Tables in 3rd Normal Form

Table : Suppliers				
ID	Name	Address	Email	Amount Owed

Table: Purchases					
Purchase ID	SupplierID	Model	Quantity	Date	Total Amount

Table: Customers				
ID	Name	Address	Email	Amount Due

Table: Sales					
SaleID	CustomerID	Model	Quantity	Date	Total Amount

Table: Products			
Make	Model	Type	Price

Table: Product Console	
Make	Model

Table: Consoles				
Model	Memory	G.P.U	C.P.U	Quantity

Table: Product Headset	
Make	Model

Table: VR Headsets				
Model	Resolution	Field Of View	Refresh Rate	Quantity

Table: Product PC	
Make	Model

Table: PCs				
Model	C.P.U	R.A.M	Memory	Quantity

Cardinality

The M:M relationships between Suppliers and Products, and Customers and Products, become separate relation tables called Purchases and Sales. These new tables have their own Primary Keys and contain foreign keys referencing Suppliers and Customers.

The 1:M weak entity relationships become tables with Foreign Keys 'Model' referencing the Products parent table.

Final Design

The final design of the database will consist of the eleven tables shown above in 3NF.

Errors & Anomalies

Tables: 'Product Console', 'Product Headset' & 'Product PC' will only be needed if a supplier changes their name or releases a new product that my client wants to trade in.

3. Physical Design

GameWorld DDL:

```

create database GameWorld;
use GameWorld;

drop table if exists Products;
drop table if exists Consoles;
drop table if exists VRHeadsets;
drop table if exists PCs;

drop table if exists ProductConsole;
drop table if exists ProductHeadset;
drop table if exists ProductPC;

drop table if exists Customers;
drop table if exists Sales;

drop table if exists Suppliers;
drop table if exists Purchases;

create table if not exists Products(
  `make` varchar(10),
  `model` varchar(15) character set utf8,
  `type` varchar(105),
  `price` float,
  primary key (`model`)
);

create table if not exists Consoles(
  `model` varchar(15),
  `mem` varchar(10),
  `gpu` varchar(20),
  `cpu` varchar(15),
  `quantity` int,
  primary key (`model`),
  foreign key (`model`) references Products(`model`)
  on delete cascade
);

create table if not exists VRHeadsets(
  `model` varchar(15),
  `res` varchar(10),
  `fov` varchar(10),
  `refrate` varchar(10),
  `quantity` int,
  primary key (`model`),
  foreign key (`model`) references Products(`model`)
  on delete cascade
);

create table if not exists PCs(
  `model` varchar(15),
  `cpu` varchar(15),
  `ram` varchar(10),
  `mem` varchar(10),
  `quantity` int,
  primary key (`model`),
  foreign key (`model`) references Products(`model`)
  on delete cascade
);

create table if not exists ProductConsole(
  `make` varchar(10),
  `model` varchar(15)
);

```

```

create table if not exists ProductHeadset(
  `make` varchar(10),
  `model` varchar(15)
);

create table if not exists ProductPC(
  `make` varchar(10),
  `model` varchar(15)
);

create table if not exists Customers(
  `ID` int not null,
  `name` varchar(255),
  `address` varchar(255),
  `email` varchar(255),
  `AmntDue` float,
  primary key(`ID`)
);

create table if not exists Sales(
  `SaleID` int not null auto_increment,
  `CustomerID` int,
  `CustomerName` varchar(255),
  `model` varchar(15),
  `quantity` int,
  `day` date,
  `TotalAmnt` int,
  `paid` bool,
  primary key (`SaleID`),
  foreign key (`CustomerID`) references Customers(`ID`) on delete cascade,
  foreign key (`model`) references Products(`model`) on delete cascade
)auto_increment = 1000;

create table if not exists Suppliers(
  `ID` int,
  `name` varchar(255),
  `address` varchar(255),
  `email` varchar(255),
  `AmntOwed` int,
  primary key (`ID`)
);

create table if not exists Purchases(
  `PurchaseID` int not null auto_increment,
  `SupplierID` int,
  `SupplierName` varchar(255),
  `model` varchar(15),
  `quantity` int,
  `day` date,
  `TotalAmnt` int,
  `paid` bool,
  primary key (`PurchaseID`),
  foreign key (`SupplierID`) references Suppliers(`ID`) on delete cascade,
  foreign key (`model`) references Products(`model`) on delete cascade
)auto_increment = 1000;

```

Implementation of Test Data

GameWorld DML:

Initialise Supplier Information, amount owed set to zero:

```
insert into Suppliers ('ID', 'name', 'address', 'email', 'AmntOwed') values('1001', 'Steam', 'Capel St, Dublin', 'steam@steam.com', '0');
insert into Suppliers ('ID', 'name', 'address', 'email', 'AmntOwed') values('1002', 'Sony', 'City West, Dublin', 'sony@sony.ie', '0');
insert into Suppliers ('ID', 'name', 'address', 'email', 'AmntOwed') values('1003', 'Microsoft', 'Sandyford, Dublin', 'microsoft@msn.com', '0');
insert into Suppliers ('ID', 'name', 'address', 'email', 'AmntOwed') values('1004', 'Facebook', 'Grand Canal Square, Dublin', 'support@book.net', '0');
insert into Suppliers ('ID', 'name', 'address', 'email', 'AmntOwed') values('1005', 'Dell', 'CheryWood Park, Dublin', 'admin@dell.com', '0');
```

Table: Suppliers

	ID	name	address	email	AmntOwed
▶	1001	Steam	Capel St, Dublin	steam@steam.com	0
	1002	Sony	City West, Dublin	sony@sony.ie	0
	1003	Microsoft	Sandyford, Dublin	microsoft@msn.com	0
	1004	Facebook	Grand Canal Square, Dublin	support@book.net	0
	1005	Dell	CheryWood Park, Dublin	admin@dell.com	0

Initialise Customer Information, amount due set to zero:

```
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1001', 'Game Stop', '16, Henry Street, Dublin', 'custcare@gamestop.com', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1002', 'GameXchange', '73, Talbot St, Dublin', 'gamexchange@msn.ie', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1003', 'Garry's Game Loft', 'High Street, Dublin', 'ggl@yahoo.net', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1004', 'Toy World', 'Dundrum Centre, Dublin', 'ToyWorld@gmail.com', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1005', 'Games Ltd', 'Naas Road, Dublin', 'contact@gamesltd.ie', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1006', 'Toy Land', 'Naas Road, Dublin', 'admin@toyland.ie', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1007', 'Larry's Game Lab', 'Blanchardstown, Dublin', 'lgl@gmail.com', '0');
insert into Customers('ID', 'name', 'address', 'email', 'AmntDue') values('1008', 'Best Buy', '23, Store St, Dublin', 'support@bestbuy.com', '0');
```

Table: Customers

	ID	name	address	email	AmntDue
	1001	Game Stop	16, Henry Street, Dublin	custcare@gamestop.com	0
	1002	GameXchange	73, Talbot St, Dublin	gamexchange@msn.ie	0
	1003	Garry's Game Loft	High Street, Dublin	ggl@yahoo.net	0
	1004	Toy World	Dundrum Centre, Dublin	ToyWorld@gmail.com	0
	1005	Games Ltd	Naas Road, Dublin	contact@gamesltd.ie	0
	1006	Toy Land	Naas Road, Dublin	admin@toyland.ie	0
	1007	Larry's Game Lab	Blanchardstown, Dublin	Lgl@gmail.com	0
▶	1008	Best Buy	23, Store St, Dublin	support@bestbuy.com	0

Insert Product Information that my client stocks including unit prices:

```
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Alienware A', 'Console', '599.00');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Alienware B', 'Console', '499.00');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Maingear Drift', 'PC', '1099.99');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Syber X', 'Console', '499.00');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Syber P', 'Console', '399.00');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'Syber I', 'Console', '275.00');
insert into Products('make', 'model', 'type', 'price') values('Steam', 'SteamVR', 'VRHeadset', '550.00');
insert into Products('make', 'model', 'type', 'price') values('Sony', 'PS4', 'Console', '349.99');
insert into Products('make', 'model', 'type', 'price') values('Sony', 'PlayStationVR', 'VRHeadset', '349.99');
insert into Products('make', 'model', 'type', 'price') values('Microsoft', 'Xbox One', 'Console', '379.99');
insert into Products('make', 'model', 'type', 'price') values('Microsoft', 'Xbox 360', 'Console', '249.99');
insert into Products('make', 'model', 'type', 'price') values('Microsoft', 'HoloLens', 'VRHeadset', '3000.00');
insert into Products('make', 'model', 'type', 'price') values('Facebook', 'Oculus Rift', 'VRHeadset', '600.00');
insert into Products('make', 'model', 'type', 'price') values('Dell', 'Alienware x51', 'PC', '16489.00');
insert into Products('make', 'model', 'type', 'price') values('Dell', 'Area 51', 'PC', '3398.99');
insert into Products('make', 'model', 'type', 'price') values('Dell', 'Inspiron', 'PC', '1091.01');
insert into Products('make', 'model', 'type', 'price') values('Sony', 'VAIO', 'PC', '900.00');
insert into Products('make', 'model', 'type', 'price') values('Sony', 'PS3', 'Console', '250.00');
```


Table: Products

	make	model	type	price
▶	Dell	Alienware x51	PC	16489
	Dell	Area 51	PC	3398.99
	Dell	Inspiron	PC	1091.01
	Facebook	Oculus Rift	VRHeadset	600
	Microsoft	HoloLens	VRHeadset	3000
	Microsoft	Xbox 360	Console	249.99
	Microsoft	Xbox One	Console	379.99
	Sony	PlayStationVR	VRHeadset	349.99
	Sony	PS3	Console	250
	Sony	PS4	Console	349.99
	Sony	VAIO	PC	900
	Steam	Alienware A	Console	599
	Steam	Alienware B	Console	499
	Steam	Maingear Drift	PC	1099.99
	Steam	SteamVR	VRHeadset	550
	Steam	Syber I	Console	275
	Steam	Syber P	Console	399
	Steam	Syber X	Console	499

Populate Stock Tables, initialise stocks to zero:

```

insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Alienware B','4GB DDR','Asus','Intel Corei3','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Alienware A','8GB DDR','Nvidia Geforce','Intel Corei7','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Syber X','16GB DDR','Nvidia Gerforce 980','Intel Core i7','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Syber P','8GB DDR','Nvidia Geforce 960','Intel Core i5','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Syber I','4GB','Nvidia Geforce 750','Intel Corei3','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Xbox One','8GB','AMD','D3D11','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('Xbox 360','526MB','ATI Xenos','ATI Xenon','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('PS3','525MB','Nvidia','AMD','0');
insert into Consoles(`model`,`mem`,`gpu`,`cpu`,`quantity`) values('PS4','8GB','Nvidia','Intel Corei7','0');

insert into VRHeadsets(`model`,`res`,`fov`,`refrate`,`quantity`) values('SteamVR','2160x1200','100','90Hz','0');
insert into VRHeadsets(`model`,`res`,`fov`,`refrate`,`quantity`) values('PlaystationVR','1080p','110','100Hz','0');
insert into VRHeadsets(`model`,`res`,`fov`,`refrate`,`quantity`) values('HoloLens','1080p','100','90Hz','0');
insert into VRHeadsets(`model`,`res`,`fov`,`refrate`,`quantity`) values('Oculus Rift','1080p','110','120Hz','0');

insert into PCs(`model`,`cpu`,`ram`,`mem`,`quantity`) values('Maingear Drift','Intel Corei7','16GB','1TB SSD','0');
insert into PCs(`model`,`cpu`,`ram`,`mem`,`quantity`) values('Alienware x51','Intel Corei7','16GB','1TB','0');
insert into PCs(`model`,`cpu`,`ram`,`mem`,`quantity`) values('Area 51','Intel Corei7','16GB','1TB','0');
insert into PCs(`model`,`cpu`,`ram`,`mem`,`quantity`) values('Inspiron','Intel Corei5','8GB','500GB','0');
insert into PCs(`model`,`cpu`,`ram`,`mem`,`quantity`) values('VAIO','Intel Corei3','4GB','300GB','0');

```

Table: Consoles

	model ▲	mem	gpu	cpu	quantity
▶	Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	0
	Alienware B	4GB DDR	Asus	Intel Corei3	0
	PS3	525MB	Nvidia	AMD	0
	PS4	8GB	Nvidia	Intel Corei7	0
	Syber I	4GB	Nvidia Geforce 750	Intel Corei3	0
	Syber P	8GB DDR	Nvidia Geforce 960	Intel Core i5	0
	Syber X	16GB DDR	Nvidia Gerforce 980	Intel Core i7	0
	Xbox 360	526MB	ATI Xenos	ATI Xenon	0
	Xbox One	8GB	AMD	D3D11	0

Table: VRHeadsets

	model ▲	res	fov	refrate	quantity
▶	HoloLens	1080p	100	90Hz	0
	Oculus Rift	1080p	110	120Hz	0
	PlaystationVR	1080p	110	100Hz	0
	SteamVR	2160x1200	100	90Hz	0

Table: PCs

	model ▲	cpu	ram	mem	quantity
▶	Alienware x51	Intel Corei7	16GB	1TB	0
	Area 51	Intel Corei7	16GB	1TB	0
	Inspiron	Intel Corei5	8GB	500GB	0
	Maingear Drift	Intel Corei7	16GB	1TB SSD	0
	VAIO	Intel Corei3	4GB	300GB	0

Populate Make/Model tables as per 3NF:

```

insert into ProductConsole(`make`,`model`) values('Steam','Alienware A');
insert into ProductConsole(`make`,`model`) values('Steam','Alienware B');
insert into ProductConsole(`make`,`model`) values('Steam','Syber X');
insert into ProductConsole(`make`,`model`) values('Steam','Syber P');
insert into ProductConsole(`make`,`model`) values('Steam','Syber I');
insert into ProductConsole(`make`,`model`) values('Sony','PS4');
insert into ProductConsole(`make`,`model`) values('Sony','PS3');
insert into ProductConsole(`make`,`model`) values('Microsoft','Xbox One');
insert into ProductConsole(`make`,`model`) values('Microsoft','Xbox 360');

insert into ProductHeadset(`make`,`model`) values('Steam','SteamVR');
insert into ProductHeadset(`make`,`model`) values('Sony','PlayStationVR');
insert into ProductHeadset(`make`,`model`) values('Microsoft','HoloLens');
insert into ProductHeadset(`make`,`model`) values('Facebook','Oculus Rift');

insert into ProductPC(`make`,`model`) values('Steam','Maingear Drift');
insert into ProductPC(`make`,`model`) values('Dell','Alienware x51');
insert into ProductPC(`make`,`model`) values('Dell','Area 51');
insert into ProductPC(`make`,`model`) values('Dell','Inspiron');
insert into ProductPC(`make`,`model`) values('Sony','VAIO');
```

Table: ProductConsole

	make	model
►	Microsoft	Xbox One
	Microsoft	Xbox 360
	Sony	PS4
	Sony	PS3
	Steam	Alienware A
	Steam	Alienware B
	Steam	Syber X
	Steam	Syber P
	Steam	Syber I

Table: ProductHeadset

	make	model
►	Facebook	Oculus Rift
	Microsoft	HoloLens
	Sony	PlayStationVR
	Steam	SteamVR

Table: ProductPC

	make	model
►	Dell	Alienware x51
	Dell	Area 51
	Dell	Inspiron
	Sony	VAIO
	Steam	Maingear Drift

Procedure to Purchases Products:

The PurchaseProduct() procedure takes a 'model' and 'quantity' as parameters. It checks the model passed in argument one by calling the GetType() procedure which checks the model type, but can also be used to check if the model is not one that is being currently stocked. If the model is valid, It calculates the purchase amount from the Products table by multiplying the model's price by the quantity parameter and adds the total to the amount owed of the relevant supplier's record. Then it adds the quantity passed to the relevant quantity in stock of the product being purchased. It saves the purchase as a unique record in the Purchases table.

```
delimiter //
create procedure PurchaseProduct(model varchar(15), quantity int)
begin
    declare PurchaseAmnt int;
    declare SupplierIDNO int;
    declare SupplierName varchar(255);

    call GetType(model, @t);
    if (@t = 'FALSE') then
        select 'Model not stocked.';
    else
        -- Get the total cost of the Purchase for Purchase record.
        select `price` into PurchaseAmnt from Products
        where Products.`model` = model;

        set PurchaseAmnt = PurchaseAmnt * quantity;

        -- Get the Supplier's ID for the Purchase record.
        select s.`ID` into SupplierIDNO from Suppliers s
        join Products p on (s.`name` = p.`make`)
        where s.`name` = p.`make` and p.`model` = model;

        -- Get the Suppliers name for the Purchase record.
        select s.`name` into SupplierName from Suppliers s
        join Products p on (s.`name` = p.`make`)
        where s.`name` = p.`make` and p.`model` = model;

        -- Update the amount owed to the supplier
        update Suppliers s join Products p on (s.`name` = p.`make`)
        set s.`AmntOwed` = s.`AmntOwed` + (p.`price` * quantity)
        where s.`name` = p.`make` and p.`model` = model;
```



```

    if (@t = 'Console') then
        -- Update the stock holdings for the model
        update Consoles
        set Consoles.`quantity` = Consoles.`quantity` + quantity
        where Consoles.`model` = model;
    elseif (@t = 'VRHeadset' ) then
        update VRHeadsets
        set VRHeadsets.`quantity` = VRHeadsets.`quantity` + quantity
        where VRHeadsets.`model` = model;
    elseif (@t = 'PC') then
        update PCs
        set PCs.`quantity` = PCs.`quantity` + quantity
        where PCs.`model` = model;
    end if;
    -- Record the purchase in the purchases table.
    insert into Purchases(`SupplierID`,`SupplierName`,`model`,`quantity`,`day`,`TotalAmnt`,`paid`)
    values(SupplierIDNO,SupplierName,model,quantity,curdate(),PurchaseAmnt,FALSE);
end if;
end
//
delimiter ;

delimiter //
create procedure GetType(model varchar(15), out ProdType varchar(15))
begin
    declare MyType varchar(15);

    set MyType = 'FALSE';
    set ProdType = 'FALSE';

    select p.`type` into MyType from Products p where p.`model` = model;
    if (MyType = 'Console') then
        set ProdType = 'Console';
    elseif (MyType = 'VRHeadset') then
        set ProdType = 'VRHeadset';
    elseif ( MyType = 'PC') then
        set ProdType = 'PC';
    end if;
end
//
delimiter ;

call PurchaseProduct('Alienware A', 10);
call PurchaseProduct('Alienware B', 10);
call PurchaseProduct('PS3', 5);
call PurchaseProduct('PS4', 20);
call PurchaseProduct('Syber I', 5);
call PurchaseProduct('Syber P', 5);
call PurchaseProduct('Syber X', 5);
call PurchaseProduct('Xbox 360', 5);
call PurchaseProduct('Xbox One', 10);

```

Console Stocks:

	model	mem	gpu	cpu	quantity
▶	Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	10
	Alienware B	4GB DDR	Asus	Intel Corei3	10
	PS3	525MB	AMD	Nvidia	5
	PS4	8GB	Intel Corei7	Nvidia	20
	Syber I	4GB	Nvidia Geforce 750	Intel Corei3	5
	Syber P	8GB DDR	Nvidia Geforce 960	Intel Core i5	5
	Syber X	16GB DDR	Nvidia Gerforce 980	Intel Core i7	5
	Xbox 360	526MB	ATI Xenos	ATI Xenon	5
	Xbox One	8GB	AMD	D3D11	10

Suppliers Amounts Owed:

	ID	name	address	email	AmntOwed
	1005	Dell	CheryWood Park, Dublin	admin@dell.com	0
▶	1004	Facebook	Grand Canal Square, Dublin	support@book.net	0
	1003	Microsoft	Sandyford, Dublin	microsoft@msn.com	5050
	1002	Sony	City West, Dublin	sony@sony.ie	8250
	1001	Steam	Capel St, Dublin	steam@steam.com	16845

Purchases Records:

	PurchaseID	SupplierID	SupplierName	model	quantity	day	TotalAmnt
▶	1000	1001	Steam	Alienware A	10	2016-03-26	5990
	1001	1001	Steam	Alienware B	10	2016-03-26	4990
	1002	1002	Sony	PS3	5	2016-03-26	1250
	1003	1002	Sony	PS4	20	2016-03-26	7000
	1004	1001	Steam	Syber I	5	2016-03-26	1375
	1005	1001	Steam	Syber P	5	2016-03-26	1995
	1006	1001	Steam	Syber X	5	2016-03-26	2495
	1007	1003	Microsoft	Xbox 360	5	2016-03-26	1250
	1008	1003	Microsoft	Xbox One	10	2016-03-26	3800

```
call PurchaseProduct('HoloLens', 10);
call PurchaseProduct('Oculus Rift', 10);
call PurchaseProduct('PlaystationVR', 20);
call PurchaseProduct('SteamVR', 5);
```

Headset Stocks:

	model	res	fov	refrate	quantity
▶	HoloLens	1080p	100	90Hz	10
	Oculus Rift	1080p	110	120Hz	10
	PlaystationVR	1080p	110	100Hz	20
	SteamVR	2160x1200	100	90Hz	5

Suppliers Amounts Owed:

	ID	name	address	email	AmntOwed
▶	1005	Dell	CheryWood Park, Dublin	admin@dell.com	0
	1004	Facebook	Grand Canal Square, Dublin	support@book.net	6000
	1003	Microsoft	Sandyford, Dublin	microsoft@msn.com	35050
	1002	Sony	City West, Dublin	sony@sony.ie	29250
	1001	Steam	Capel St, Dublin	steam@steam.com	19595

Purchases Records:

	PurchaseID	SupplierID	SupplierName	model	quantity	day	TotalAmnt
▶	1000	1001	Steam	Alienware A	10	2016-03-26	5990
	1001	1001	Steam	Alienware B	10	2016-03-26	4990
	1002	1002	Sony	PS3	5	2016-03-26	1250
	1003	1002	Sony	PS4	20	2016-03-26	7000
	1004	1001	Steam	Syber I	5	2016-03-26	1375
	1005	1001	Steam	Syber P	5	2016-03-26	1995
	1006	1001	Steam	Syber X	5	2016-03-26	2495
	1007	1003	Microsoft	Xbox 360	5	2016-03-26	1250
	1008	1003	Microsoft	Xbox One	10	2016-03-26	3800
	1009	1003	Microsoft	HoloLens	10	2016-03-26	30000
	1010	1004	Facebook	Oculus Rift	10	2016-03-26	6000
	1011	1002	Sony	PlaystationVR	20	2016-03-26	7000
	1012	1001	Steam	SteamVR	5	2016-03-26	2750

```
call PurchaseProduct('Alienware x51', 15);
call PurchaseProduct('Area 51', 5);
call PurchaseProduct('Inspiron', 10);
call PurchaseProduct('Maingear Drift', 5);
call PurchaseProduct('VAIO', 10);
```

PC Stocks:

	model	cpu	ram	mem	quantity
▶	Alienware x51	Intel Corei7	16GB	1TB	15
	Area 51	Intel Corei7	16GB	1TB	5
	Inspiron	Intel Corei5	8GB	500GB	10
	Maingear Drift	Intel Corei7	16GB	1TB SSD	5
	VAIO	Intel Corei3	4GB	300GB	10

Suppliers Amounts Owed:

	ID	name	address	email	AmntOwed
▶	1005	Dell	CheryWood Park, Dublin	admin@dell.com	275240
	1004	Facebook	Grand Canal Square, Dublin	support@book.net	6000
	1003	Microsoft	Sandyford, Dublin	microsoft@msn.com	35050
	1002	Sony	City West, Dublin	sony@sony.ie	24250
	1001	Steam	Capel St, Dublin	steam@steam.com	25095

Purchases Records:

	PurchaseID	SupplierID	SupplierName	model	quantity	day	TotalAmnt
▶	1000	1001	Steam	Alienware A	10	2016-03-26	5990
	1001	1001	Steam	Alienware B	10	2016-03-26	4990
	1002	1002	Sony	PS3	5	2016-03-26	1250
	1003	1002	Sony	PS4	20	2016-03-26	7000
	1004	1001	Steam	Syber I	5	2016-03-26	1375
	1005	1001	Steam	Syber P	5	2016-03-26	1995
	1006	1001	Steam	Syber X	5	2016-03-26	2495
	1007	1003	Microsoft	Xbox 360	5	2016-03-26	1250
	1008	1003	Microsoft	Xbox One	10	2016-03-26	3800
	1009	1003	Microsoft	HoloLens	10	2016-03-26	30000
	1010	1004	Facebook	Oculus Rift	10	2016-03-26	6000
	1011	1002	Sony	PlaystationVR	20	2016-03-26	7000
	1012	1001	Steam	SteamVR	5	2016-03-26	2750
	1013	1005	Dell	Alienware ...	15	2016-03-26	247335
	1014	1005	Dell	Area 51	5	2016-03-26	16995
	1015	1005	Dell	Inspiron	10	2016-03-26	10910
	1016	1001	Steam	Maingear D...	5	2016-03-26	5500
	1017	1002	Sony	VAIO	10	2016-03-26	9000

```
call PurchaseProduct('PS5',5);
```

	Model not stocked.
▶	Model not stocked.

```
call PurchaseProduct('SteamVR', 1);
```

✓ 2627 21:23:05 call PurchaseProduct('SteamVR', 1) 1 row(s) affected 0.234 sec

Procedure to Sell Products:

Procedure SellProduct() takes a customer ID, a model and a quantity as arguments. It validates the customer account by calling CheckCustAcc(). If the Customer ID parameter corresponds to a valid credit account in the Customers table of the database, then the procedure checks if a valid model was passed in, by calling the GetType() procedure. If the model is valid, then it checks if there is enough of the model in stock using the quantity passed in the third argument calling the CheckStock() procedure. If these conditions are all met, the quantity of the relevant model in stock is deducted by the amount being sold. The relevant Customer's amount owed is increment by the price of the model multiplied by the quantity being sold. Finally, a record of the sale is recorded in the Sales table.

```

delimiter //
create procedure SellProduct(CustID int, model varchar(15), quantity int)
begin
    call CheckCustAcc(CustID, @c);
    if(@c = TRUE) then
        call GetType(model, @t);
        if (@t = 'FALSE') then
            select 'Model not in stock.';
        else
            call CheckStock(model, quantity, @s);
            if (@s = TRUE) then
                if (@t = 'Console') then
                    update Consoles
                    set Consoles.`quantity` = Consoles.`quantity` - quantity
                    where Consoles.`model` = model;
                    call UpdateCustSales(CustID, model, quantity);
                elseif (@t = 'VRHeadset') then
                    update VRHeadsets h
                    set h.`quantity` = h.`quantity` - quantity
                    where h.`model` = model;
                    call UpdateCustSales(CustID, model, quantity);
                elseif (@t = 'PC') then
                    update PCs p
                    set p.`quantity` = p.`quantity` - quantity
                    where p.`model` = model;
                    call UpdateCustSales(CustID, model, quantity);
                end if;
            else
                select 'Not enough', model, 'in stock to fulfil order.';
            end if;
        end if;
    else
        select 'Customer does not have a valid credit account.';
    end if;
end
//
delimiter ;

delimiter //
create procedure CheckCustAcc(CustID int, out ValidAccount bool)
begin
    declare AcntCheck int default 0;

    set ValidAccount = FALSE;

    select c.`ID` into AcntCheck from Customers c where c.`ID` = CustID;
    if(AcntCheck != 0) then
        set ValidAccount = TRUE;
    end if;
end
//
delimiter ;

```

```

delimiter //
create procedure CheckStock(model varchar(15), quantity int, out EnoughStock bool)
begin
    declare stock int default 0;
    set EnoughStock = FALSE;

    call GetType(model, @t);

    if (@t = 'Console') then
        select c.`quantity` into stock from Consoles c where c.`model` = model;
        set stock = stock - quantity;
        if(stock >= 0) then
            set EnoughStock = TRUE;
        end if;
    elseif (@t = 'VRHeadset') then
        select h.`quantity` into stock from VRHeadsets h where h.`model` = model;
        set stock = stock - quantity;
        if(stock >= 0) then
            set EnoughStock = TRUE;
        end if;
    elseif (@t = 'PC') then
        select p.`quantity` into stock from PCs p where p.`model` = model;
        set stock = stock - quantity;
        if(stock >= 0) then
            set EnoughStock = TRUE;
        end if;
    end if;
end
//
delimiter ;
delimiter //
create procedure UpdateCustSales(CustID int, model varchar(15), quantity int)
begin
    declare amount int default 0;
    declare CustName varchar(255);
    declare TotalAmount int default 0;

    -- Calculate amount for Sales record & Customer Amount owed.
    select p.`price` into amount from Products p
    where p.`model` = model;

    -- Get Customer's current amount for Sales record.
    select c.`AmntDue` into TotalAmount from Customers c
    where c.`ID` = CustID;

    set amount = amount * quantity;
    set TotalAmount = TotalAmount + amount;

    -- update Customer amount due.
    update Customers c
    set c.`AmntDue` = TotalAmount
    where c.`ID` = CustID;

    -- Get Customer's name for Sales record.
    select c.`name` into CustName from Customers c
    where c.`ID` = CustID;

    -- Record valid sale in the Sales table.
    insert into Sales(`CustomerID`, `CustomerName`, `model`, `quantity`, `day`, `TotalAmnt`, `paid`)
    values(CustID, CustName, model, quantity, curdate(), amount, FALSE);
end
//
delimiter ;

```

```
call SellProduct(1001,'PS3', 2);
call SellProduct(1001,'PS4', 3);
call SellProduct(1001,'HoloLens', 1);
call SellProduct(1001,'HoloLens', 1);
```

Console Stocks:

	model	mem	gpu	cpu	quantity
▶	Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	10
	Alienware B	4GB DDR	Asus	Intel Corei3	10
	PS3	525MB	Nvidia	AMD	3
	PS4	8GB	Nvidia	Intel Corei7	17
	Syber I	4GB	Nvidia Geforce 750	Intel Corei3	5
	Syber P	8GB DDR	Nvidia Geforce 960	Intel Core i5	5
	Syber X	16GB DDR	Nvidia Geforce 980	Intel Core i7	5
	Xbox 360	526MB	ATI Xenos	ATI Xenon	5
	Xbox One	8GB	AMD	D3D11	10

VRHeadset Stocks:

	model	res	fov	refrate	quantity
▶	HoloLens	1080p	100	90Hz	8
	Oculus Rift	1080p	110	120Hz	10
	PlaystationVR	1080p	110	100Hz	20
	SteamVR	2160x1200	100	90Hz	5

Customer Amounts Due:

	ID	name	address	email	AmntDue
▶	1001	Game Stop	16, Henry Street, Dublin	custcare@gamestop.com	7550
	1002	GameXchange	73, Talbot St, Dublin	gamexchange@msn.ie	0
	1003	Garry's Game Loft	High Street, Dublin	ggl@yahoo.net	0
	1004	Toy World	Dundrum Centre, Dublin	ToyWorld@gmail.com	0
	1005	Games Ltd	Naas Road, Dublin	contact@gamesltd.ie	0
	1006	Toy Land	Naas Road, Dublin	admin@toyland.ie	0
	1007	Larry's Game Lab	Blanchardstown, Dublin	Lgl@gmail.com	0
	1008	Best Buy	23, Store St, Dublin	support@bestbuy.com	0

Sales:

	SaleID	CustomerID	CustomerName	model	quantity	day	TotalAmnt
▶	1000	1001	Game Stop	PS3	2	2016-03-27	500
	1001	1001	Game Stop	PS4	3	2016-03-27	1050
	1002	1001	Game Stop	HoloLens	1	2016-03-27	3000
	1003	1001	Game Stop	HoloLens	1	2016-03-27	3000

```
call SellProduct(1004,'Inspiron', 4);
call SellProduct(1006,'HoloLens', 1);
call SellProduct(1001,'Xbox 360', 1);
call SellProduct(1008,'Syber P', 5);
```

PC Stocks:

	model	cpu	ram	mem	quantity
▶	Alienware x51	Intel Corei7	16GB	1TB	15
	Area 51	Intel Corei7	16GB	1TB	5
	Inspiron	Intel Corei5	8GB	500GB	6
	Maingear Drift	Intel Corei7	16GB	1TB SSD	5
	VAIO	Intel Corei3	4GB	300GB	10

VRHeadsets Stocks:

	model	res	fov	refrate	quantity
▶	HoloLens	1080p	100	90Hz	7
	Oculus Rift	1080p	110	120Hz	10
	PlaystationVR	1080p	110	100Hz	20
	SteamVR	2160x1200	100	90Hz	5

Console Stocks:

	model	mem	gpu	cpu	quantity
▶	Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	10
	Alienware B	4GB DDR	Asus	Intel Corei3	10
	PS3	525MB	Nvidia	AMD	3
	PS4	8GB	Nvidia	Intel Corei7	17
	Syber I	4GB	Nvidia Geforce 750	Intel Corei3	5
	Syber P	8GB DDR	Nvidia Geforce 960	Intel Core i5	0
	Syber X	16GB DDR	Nvidia Geforce 980	Intel Core i7	5
	Xbox 360	526MB	ATI Xenos	ATI Xenon	4
	Xbox One	8GB	AMD	D3D11	10

Customers Amounts Due:

	ID	name	address	email	AmntDue
▶	1001	Game Stop	16, Henry Street, Dublin	custcare@gamestop.com	7800
	1002	GameXchange	73, Talbot St, Dublin	gamexchange@msn.ie	0
	1003	Garry's Game Loft	High Street, Dublin	ggl@yahoo.net	0
	1004	Toy World	Dundrum Centre, Dublin	ToyWorld@gmail.com	4364
	1005	Games Ltd	Naas Road, Dublin	contact@gamesltd.ie	0
	1006	Toy Land	Naas Road, Dublin	admin@toyland.ie	3000
	1007	Larry's Game Lab	Blanchardstown, Dublin	Lgl@gmail.com	0
	1008	Best Buy	23, Store St, Dublin	support@bestbuy.com	1995

Sales records:

	SaleID	CustomerID	CustomerName	model	quantity	day	TotalAmnt
▶	1000	1001	Game Stop	PS3	2	2016-03-27	500
	1001	1001	Game Stop	PS4	3	2016-03-27	1050
	1002	1001	Game Stop	HoloLens	1	2016-03-27	3000
	1003	1001	Game Stop	HoloLens	1	2016-03-27	3000
	1004	1004	Toy World	Inspiron	4	2016-03-27	4364
	1005	1006	Toy Land	HoloLens	1	2016-03-27	3000
	1006	1001	Game Stop	Xbox 360	1	2016-03-27	250
	1007	1008	Best Buy	Syber P	5	2016-03-27	1995


```
call SellProduct(2004,'Inspiron', 4);
```

	Customer does not have a valid credit account.
▶	Customer does not have a valid credit account.

```
call SellProduct(1004,'PS7', 4);
```

	Model not in stock.
▶	Model not in stock.

```
call SellProduct(1004,'PS3', 14);
```

	Not enough	model	in stock to fulfil order.
▶	Not enough	PS3	in stock to fulfil order.

```
call SellProduct(1004,'VAIO', 2);
```

```
✓ 2626 21:21:25 call SellProduct(1004,'VAIO', 2) 1 row(s) affected 0.172 sec
```

Procedures to Auto-Populate Purchases & Sales tables to test Indexing & Query Optimisations:

```
delimiter //
create procedure PopulatePurchases(count int)
begin
    declare i int default 0;
    declare model varchar(15);
    declare amnt int;

    while (i < count) do
        set model = ELT(FLOOR(1 + (RAND() * (18))), "Alienware A", "Alienware B", "Syber X", "Syber P", "Syber I", "PS4", "PS3", "Xbox One",
            "Xbox 360", "SteamVR", "PlayStationVR", "Hololens", "Oculus Rift", "Maingear Drift", "Alienware x51",
            "Area 51", "Inspiron", "VAIO");

        set amnt = FLOOR(1 + (RAND() * (5)));
        call PurchaseProduct(model, amnt);
        set i = i+1;
    end while;
end
//
delimiter ;
```

```
call PopulatePurchases(5000);
```

```
SELECT * FROM purchases;
```

1 15:42:21 SELECT * FROM gameworld.purchases LIMIT 0, 5000 2145 row(s) returned 0.078 sec / 0.093 sec

Table: Purchases

	PurchaseID	SupplierID	SupplierName	model	quantity	day	TotalAmnt
▶	1000	1001	Steam	Alienware A	4	2016-03-31	2396
	1001	1005	Dell	Alienware x51	4	2016-03-31	65956
	1002	1005	Dell	Area 51	3	2016-03-31	10197
	1003	1001	Steam	Alienware A	4	2016-03-31	2396
	1004	1004	Facebook	Oculus Rift	4	2016-03-31	2400
	1005	1005	Dell	Alienware x51	4	2016-03-31	65956

	3140	1005	Dell	Inspiron	1	2016-03-31	1091
	3141	1001	Steam	SteamVR	2	2016-03-31	1100
	3142	1002	Sony	PlayStationVR	4	2016-03-31	1400
	3143	1005	Dell	Inspiron	3	2016-03-31	3273
	3144	1001	Steam	Alienware B	3	2016-03-31	1497

At this point in simulating test data, my client would owe all of his suppliers quite a lot of money, but the database has enough stock of all products to fill the sales table using a similar procedure generating random sales. The image above shows the beginning and ending of the Purchases table. Each simulated purchase will be for between one and five in quantity of a given model.

```

delimiter //
create procedure PopulateSales(count int)
begin
    declare i int default 0;
    declare CustID int;
    declare model varchar(15);
    declare amnt int;

    while (i < count) do
        set CustID = FLOOR(1 + (RAND() * (8)));
        set CustID = CustID + 1000;
        set model = ELT(FLOOR(1 + (RAND() * (18))), "Alienware A", "Alienware B", "Syber X", "Syber P", "Syber I", "PS4", "PS3", "Xbox One", "Xbox 360", "SteamVR",
        "PlayStationVR", "HoloLens", "Oculus Rift", "Maingear Drift", "Alienware x51", "Area 51", "Inspiron", "VAIO");

        set amnt = FLOOR(1 + (RAND() * (5)));
        call SellProduct(CustID, model, amnt);
        set i = i+1;
    end while;
end
//
delimiter ;

call PopulateSales(2000);

SELECT * FROM sales;

```

1 16:49:40 SELECT * FROM gameworld.sales LIMIT 0, 5000 1751 row(s) returned 0.015 sec / 0.015 sec

Table: Sales

	SaleID	CustomerID	CustomerName	model	quantity	day	TotalAmnt
▶	1000	1001	Game Stop	Syber I	1	2016-03-31	275
	1001	1007	Larry's Game Lab	Alienware A	3	2016-03-31	1797
	1002	1001	Game Stop	Alienware A	5	2016-03-31	2995
	1003	1006	Toy Land	SteamVR	4	2016-03-31	2200
	1004	1008	Best Buy	Maingear Drift	4	2016-03-31	4400
	1005	1006	Toy Land	Syber X	3	2016-03-31	1497
	2746	1004	Toy World	Xbox 360	2	2016-03-31	500
	2747	1006	Toy Land	Area 51	1	2016-03-31	3399
	2748	1005	Games Ltd	Alienware x51	3	2016-03-31	49467
	2749	1005	Games Ltd	SteamVR	1	2016-03-31	550
	2750	1001	Game Stop	HoloLens	1	2016-03-31	3000

The Sales table now contains 1751 records which I can use to test indexing and query optimisation. Each randomly generated sale will be for between one and five in quantity of a given model.

Indexing of Purchases and Sales tables:

```
EXPLAIN select * from Purchases
where SupplierName = 'Steam';
show status like 'Last_query_cost';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	Purchases	NULL	ALL	NULL	NULL	NULL	NULL	2075	10.00	Using where
		Variable_name	Value									
▶		Last_query_cost	423.999000									
✓	4	17:11:04	EXPLAIN select * from Purchases where SupplierName = 'Steam' 1 row(s) returned 0.047 sec / 0.000 sec									

```
ALTER TABLE Purchases ADD INDEX ix_name (SupplierName);
```

```
✓ 10 17:35:28 ALTER TABLE Purchases ADD INDEX ix_name (SupplierName) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.844 sec
```

```
EXPLAIN select * from Purchases
where SupplierName = 'Steam';
show status like 'Last_query_cost';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	Purchases	NULL	ref	ix_name	ix_name	768	const	905	100.00	NULL
		Variable_name	Value									
▶		Last_query_cost	207.999000									
✓	11	17:38:33	EXPLAIN select * from Purchases where SupplierName = 'Steam' 1 row(s) returned 0.000 sec / 0.000 sec									

The addition of an index, named ix_name in this case, to the 'SupplierName' column of the Purchases table, will allow my client to efficiently search his purchases records by supplier name.

Previous to adding the index, selection by supplier name 'Steam' searched almost the entire Purchases table, ie: 2075 rows filtering only 10 rows. This query took 0.047 secs with a cost of 423.999 units.

After the addition of the index to the Supplier Name fields, the query only searches 905 rows out of 2145 when looking for supplier name 'Steam' filtering 100, with a lower cost of 207.999 units and a recorded time of 0 secs.

With the current test data in use of only 2145 records, which would typically account for about three months of purchases by my client, the efficiency of a query on the purchases records by Supplier name will be improved by over 50% with the addition on an index on this column.


```
EXPLAIN select * from sales
where CustomerName = 'Toy World';
show status like 'Last_query_cost';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	sales	HULL	ALL	HULL	HULL	HULL	HULL	1683	10.00	Using where
	Variable_name		Value									
▶	Last_query_cost		345.599000									
✓	19 18:07:53 EXPLAIN select * from sales where CustomerName = 'Toy World' 1 row(s) returned 0.047 sec / 0.000 sec											

```
ALTER TABLE sales ADD INDEX ix_name (CustomerName);
```

```
✓ 21 18:18:41 ALTER TABLE sales ADD INDEX ix_name (CustomerName) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.687 sec
```

```
EXPLAIN select * from sales
where CustomerName = 'Toy World';
show status like 'Last_query_cost';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	sales	NULL	ref	ix_name	ix_name	768	const	231	100.00	NULL
	Variable_name		Value									
▶	Last_query_cost		73.199000									
✓	22 18:20:42 EXPLAIN select * from sales where CustomerName = 'Toy World' 1 row(s) returned 0.000 sec / 0.000 sec											

Adding an index on the 'CustomerName' column improves the efficiency of a Sales table search for a particular customer by name, in the example above, from searching the entire table, ie: 1683 records, to just 231 with an index filtering 100. With the index in place, the query time dropped from 0.047 sec to 0.000 sec, which is the same time saving as for the Purchases table index. The query cost dropped from 345.599 to 73.199, which is less than ¼ of the cost. This will improve the performance of the database and allow my client to quickly search his Sales records for particular customers by name.

Query Optimisations:

1. Design of a query to narrow the search for a particular product, which after having been sold on a particular date, is found to be faulty and needs to be traced to a purchase to aid the processing of it by its supplier in the absence of a serial number.

```
select distinct PurchaseID from Purchases p
inner join Sales s on s.model = p.model
where s.SaleID = 2750 and p.day not between s.day and curdate();
show status like 'Last_Query_Cost';
```

Variable_name	Value
Last_query_cost	52.199000

Complex

```
select * from Purchases
where model = 'HoloLens';
show status like 'Last_query_cost';
```

Variable_name	Value
Last_query_cost	52.199000

```
select * from Sales
where model = 'HoloLens';
show status like 'Last_Query_Cost';
```

Variable_name	Value
Last_query_cost	47.199000

Simple

We can see from the Sales table above that SaleID 2750 was for One HoloLens. If that HoloLens was defective and we had to trace it to a Purchase for our Supplier, the complex Query seen on the left with a cost of 51.199 units will be employed as it is far more efficient to use than the two simple queries seen on the right with a combined cost of 99.398 units and cannot compare dates, highlighting the benefit of table joins in a query. Hence, the complex query with the join will be used for this purpose in the database.

- Design of a query to find the largest amount total of a single purchase from a given supplier for budgeting purposes.

```
EXPLAIN select max(TotalAmt)
from Purchases
where SupplierID = 1005;
show status like 'Last query cost':
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	Purchases	NULL	ref	SupplierID	SupplierID	5	const	386	100.00	NULL
	Variable_name		Value									
▶	Last_query_cost		104.199000									

```
Explain select TotalAMnt from Purchases
where SupplierID = 1005
order by SupplierID
limit 1;
show status like 'Last_query_cost':
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	Purchases	NULL	ref	SupplierID	SupplierID	5	const	386	100.00	NULL
	Variable_name		Value									
▶	Last_query_cost		104.199000									

These two queries have the same cost and search the same number of rows. The first query is less complex requiring fewer lines of MySQL, hence it will be used in the database for this purpose.

- Design of a query to cross reference a customer's details with a model's details for a given sale. Testing the difference between an inner join and a left join.

```
Explain Extended
select * from sales s
inner join customers c on s.CustomerID = c.ID
inner join products p on p.model = s.model
where s.model = 'Xbox 360';
show status like 'Last_query_cost':
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	p	NULL	const	PRIMARY	PRIMARY	47	const	1	100.00	NULL
	1	SIMPLE	s	NULL	ref	CustomerID,model	model	48	const	105	100.00	Using index condition; Using where
	1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	4	gameworld.s.CustomerID	1	100.00	NULL
	Variable_name		Value									
▶	Last_query_cost		173.999000									

```
Explain Extended
select * from sales s
left join customers c on s.CustomerID = c.ID
left join products p on p.model = s.model
where s.model = 'Xbox 360';
show status like 'Last_query_cost':
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	s	NULL	ref	model	model	48	const	105	100.00	NULL
	1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	4	gameworld.s.CustomerID	1	100.00	NULL
	1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	47	gameworld.s.model	1	100.00	NULL
	Variable_name		Value									
▶	Last_query_cost		299.999000									

The two queries above reference the same amount of rows, but the inner join has a lower cost of 173.999 compared to the cost of the left join with a cost of 299.999. Hence, I will employ the inner join for this query in the database.

Errors & Anomalies

call PopulatePurchases(5000); resulted in a loss of connection to my MySQL server after about five minutes. It seemed to have crashed and could not be restarted. When I rebooted the computer, MySQL server started and upon a select * from Purchases, the query returned 2145 records, which will suffice for indexing and query optimisations. I called PopulateSales() with 2000 as an argument to save a similar crash, alas, after 50 calls to out of stock models, further procedure calls to SellProduct() fell out of sync and could not continue. Upon rebooting, a select * from Sales produced 1751 records which will suffice for testing of indexes.

A screenshot of a MySQL error message. It features a red 'x' icon on the left, followed by the text '68 16:25:47 call PopulateSales(1900) Error Code: 2014 Commands out of sync; you can't run this command now'. The background is light blue.

68 16:25:47 call PopulateSales(1900) Error Code: 2014 Commands out of sync; you can't run this command now

The Purchases and Sales table already have unique automatically incrementing primary keys which seem to behave as de facto indexes. Adding any further indexes on these tables does not seem to improve query performance. In fact, further indexes will only slow down table updates, as the indexes need to be updated as well.

4. User Base

Following is the design and testing of the database user accounts that will be needed by my client Game World Ltd.

Log into root account.

```
C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin>mysql -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Root user is the only user at this point:

```
mysql> select user from mysql.user;
+-----+
| user      |
+-----+
| mysql.sys |
| root      |
+-----+
2 rows in set (0.05 sec)

mysql>
```

Creation of user profiles. Create an account for Trevor in GameWorld IT department making him a power user. Trevor can access all tables in the GameWorld database with grants permissions. He will take on the roll of database administrator. He will grant other users permissions.

```
mysql> create user 'trevor'@'localhost' identified by 'trevorpass';
Query OK, 0 rows affected (0.07 sec)

mysql> grant all on gamewrold.* to 'trevor'@'localhost' with grant option;
Query OK, 0 rows affected (0.06 sec)

mysql>
```

Login as Trevor to test his grants:

```
C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin>mysql -utrevor -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

```
mysql> show grants;
+-----+
| Grants for trevor@localhost |
+-----+
| GRANT USAGE ON *.* TO 'trevor'@'localhost' |
| GRANT ALL PRIVILEGES ON `gamewrold`.* TO 'trevor'@'localhost' WITH GRANT OPTION |
+-----+
2 rows in set (0.00 sec)

mysql> _
```

Create an account for Alan in Purchases and Mary in Sales & Customer Care Department.

```
mysql> create user 'alan'@'localhost' identified by 'alanpass';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'mary'@'localhost' identified by 'marypass';
Query OK, 0 rows affected (0.00 sec)

mysql> _
```

Check all three users created successfully.

```
mysql> select user from mysql.user;
+-----+
| user |
+-----+
| alan |
| mary |
| mysql.sys |
| root |
| trevor |
+-----+
5 rows in set (0.00 sec)

mysql>
```

Now Trevor can grant privileges to the other users. Alan in the Purchases Department can select and update from the Purchases, Suppliers and Products tables. Mary in Sales & Customer Care can select and update from the Sales and Customers tables.

Command Prompt - mysql -utrevor -p

```
C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin>mysql -utrevor -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> grant select, update
      -> on gameworld.purchases
      -> to 'alan'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql> grant select, update
      -> on gameworld.suppliers
      -> to 'alan'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> grant select, update
      -> on gameworld.products
      -> to 'alan'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> grant select, update
      -> on gameworld.sales
      -> to 'mary'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> grant select, update
      -> on gameworld.customers
      -> to 'mary'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> _
```

Login as Alan to test privileges.

```

Command Prompt - mysql -u alan -p
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\eooha>cd C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin

C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin>mysql -u alan -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show grants;
+-----+
| Grants for alan@localhost |
+-----+
| GRANT USAGE ON *.* TO 'alan'@'localhost' |
| GRANT SELECT, UPDATE ON `gameworld`.`purchases` TO 'alan'@'localhost' |
| GRANT SELECT, UPDATE ON `gameworld`.`products` TO 'alan'@'localhost' |
| GRANT SELECT, UPDATE ON `gameworld`.`suppliers` TO 'alan'@'localhost' |
+-----+
4 rows in set (0.00 sec)

mysql>

```

Alan cannot select from Sales, Customers, Consoles, PCs.

```

mysql> select * from Sales;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'sales'
mysql> select * from Customers;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'customers'
mysql> select * from Consoles;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'consoles'
mysql> select * from PCs;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'pcs'
mysql>

```

Alan can select from Purchases, Products & Suppliers

```

mysql> select day from purchases
-> where PurchaseID = 1000;
+-----+
| day |
+-----+
| 2014-01-01 |
+-----+
1 row in set (0.00 sec)

mysql> select price from Products
-> where model = PS3;
ERROR 1054 (42S22): Unknown column 'PS3' in 'where clause'
mysql> select price from Products
-> where model = 'PS3';
+-----+
| price |
+-----+
| 250 |
+-----+
1 row in set (0.00 sec)

mysql> select name from Suppliers
-> where ID = 1005;
+-----+
| name |
+-----+
| Dell |
+-----+
1 row in set (0.00 sec)

mysql>

```


Alan can update Purchases, Products & Suppliers.

```
mysql> update Purchases
-> set paid = 1
-> where PurchaseID = 1002;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update Products
-> set price = 2500
-> where model = 'HoloLens';
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update suppliers
-> set AmntOwed = 5000
-> where ID = 1004;
Query OK, 1 row affected (0.11 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
-> where ID = 1001;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

Login as Mary to test privileges.

```
Command Prompt - mysql -umary -p
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\eogha>cd C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin

C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin>mysql -umary -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show grants;
+-----+
| Grants for mary@localhost |
+-----+
| GRANT USAGE ON *.* TO 'mary'@'localhost' |
| GRANT SELECT, UPDATE ON `gameworld`.`customers` TO 'mary'@'localhost' |
| GRANT SELECT, UPDATE ON `gameworld`.`sales` TO 'mary'@'localhost' |
+-----+
3 rows in set (0.00 sec)

mysql>
```

Mary cannot access Products or Purchases.

```
mysql> select * from Proucts;
ERROR 1142 (42000): SELECT command denied to user 'mary'@'localhost' for table 'proucts'
mysql> select * from Purchases;
ERROR 1142 (42000): SELECT command denied to user 'mary'@'localhost' for table 'purchases'
mysql> select * Consoles;
```


Mary can update Sales & Customers.

```
mysql> update sales
      -> set TotalAmnt = 4390
      -> where SaleID = 1006;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update Customers
      -> set address = '16/17, Henry Street, Dublin'
      -> where ID = 1001;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

Trevor Revokes Mary's Privileges.

```
mysql> revoke select on gameworld.* from mary@localhost;
ERROR 1141 (42000): There is no such grant defined for user 'mary' on host 'localhost'
mysql> revoke select on gameworld.sales from mary@localhost;
Query OK, 0 rows affected (0.07 sec)

mysql> revoke select on gameworld.customers from mary@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> revoke update on gameworld.customers from mary@localhost;
Query OK, 0 rows affected (0.03 sec)

mysql> revoke update on gameworld.sales from mary@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Mary can no longer access tables.

```
mysql> select * from sales;
ERROR 1142 (42000): SELECT command denied to user 'mary'@'localhost' for table 'sales'
mysql> update customers
      -> set AmntDue = 0
      -> where ID = 1008;
ERROR 1142 (42000): UPDATE command denied to user 'mary'@'localhost' for table 'customers'
mysql>
```

Trevor revokes Alan's privileges.

```
mysql> revoke select on gameworld.purchases from alan@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> revoke select on gameworld.suppliers from alan@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> revoke select on gameworld.products from alan@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Alan can no longer access tables.

```
mysql> select * from purchases;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'purchases'
mysql> select * from suppliers;
ERROR 1142 (42000): SELECT command denied to user 'alan'@'localhost' for table 'suppliers'
mysql> update products
  -> set price = 350
  -> where model = 'PS4';
ERROR 1143 (42000): SELECT command denied to user 'alan'@'localhost' for column 'model' in table 'products'
mysql>
```

Test Drop users.

```
mysql> drop user mary@localhost;
Query OK, 0 rows affected (0.03 sec)

mysql> drop user alan@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> drop user trevor@localhost;
Query OK, 0 rows affected (0.03 sec)

mysql>
```

Errors & Anomalies

User base functionality test successful. User base will include an account for the Sales, Purchases, Stores and Customer Care Departments. The Stores account will be able to select and update the stock tables Consoles, VRHeadsets & PCs etc.

5. Auditing & Logging (Business Rules & housekeeping)

In order to design and test credit terms between my client and his customers and suppliers, I now create procedures to receive and make payments.

1. First, a procedure to take payments from customers. In this iteration, a customer is invoiced by sale and payment is made in full for that sale. Hence, the TakePayment procedure takes a sale ID as its only parameter. The customer pays for the sale in full. The record is marked as Paid in the Sales record, and the amount paid is deducted from the amount owed by the customer. The Payment received record is then entered in the newly added PaymentsReceived Table.

```
delimiter //
create procedure TakePayment(SaleID int)
begin
    declare paid bool default False;
    declare CustID int;
    declare CustName varchar(255);
    declare amount int default 0;

    declare SaleCheck int default 0;

    select s.SaleID into SaleCheck from Sales s where s.SaleID = SaleID;
    if(SaleCheck != 0) then
        select s.paid into paid from sales s where s.SaleID = SaleID;
        if(paid = TRUE) then
            select 'Invoice already paid.';
        else
            select s.CustomerID into CustID from sales s where s.SaleID = SaleID;
            select s.CustomerName into CustName from sales s where s.SaleID = SaleID;
            select s.TotalAmnt into amount from sales s where s.SaleID = SaleID;

            update Sales s
            set s.paid = TRUE
            where s.SaleID = SaleID;

            update Customers c
            set c.AmntDue = c.AmntDue - amount
            where c.ID = CustID;

            insert into PaymentsReceived(`SaleID`, `CustomerID`, `CustomerName`, `day`, `Amnt`) values (SaleID, CustID, CustName, curdate(), amount);
        end if;
    else
        select 'No valid invoice.';
    end if;
end
//
delimiter ;
```

```
call TakePayment(1000);
call TakePayment(1001);
```

```
✓ 27 20:05:35 call TakePayment(1000) 1 row(s) affected 0.313 sec
✓ 39 20:29:33 call TakePayment(1001) 1 row(s) affected 0.250 sec
```

Table: Sales

	SaleID	CustomerID	CustomerName	model	quantity	day	TotalAmnt	paid
▶	1000	1005	Games Ltd	Maingear Drift	1	2016-04-03	1100	1
	1001	1005	Games Ltd	Syber X	1	2016-04-03	499	1
	1002	1007	Larry's Game Lab	Syber I	3	2016-04-03	825	0
	1003	1002	GameXchange	PS4	5	2016-04-03	1750	0
	1004	1008	Best Buy	Xbox 360	4	2016-04-03	1000	0

Table: PaymentsReceived

	PaymentID	SaleID	CustomerID	CustomerName	day	Amnt
▶	1	1000	1005	Games Ltd	2016-04-03	1100
	2	1001	1005	Games Ltd	2016-04-03	499
*	NULL	NULL	NULL	NULL	NULL	NULL

2. Similarly, I now create a procedure to make a payment to a supplier. In this iteration, payments are made in full for a given purchase. Hence, the MakePayment procedure takes a PurchaseID as its only parameter. The record is marked as Paid to the corresponding Purchases record, and the amount paid is deducted from the amount owed to the supplier. The Payment received record is then entered in the newly added PaymentsPaid Table.

```
drop procedure if exists MakePayment;

delimiter //
create procedure MakePayment(PurchaseID int)
begin
    declare paid bool default False;
    declare SupplierID int;
    declare SupplierName varchar(255);
    declare amount int default 0;

    declare PurchaseCheck int default 0;

    select p.PurchaseID into PurchaseCheck from Purchases p where p.PurchaseID = PurchaseID;
    if(PurchaseCheck != 0) then
        select p.paid into paid from Purchases p where p.PurchaseID = PurchaseID;
        if(paid = TRUE) then
            select 'Invoice already paid.';
        else
            select p.SupplierID into SupplierID from Purchases p where p.PurchaseID = PurchaseID;
            select p.SupplierName into SupplierName from Purchases p where p.PurchaseID = PurchaseID;
            select p.TotalAmnt into amount from Purchases p where p.PurchaseID = PurchaseID;

            update Purchases p
            set p.paid = TRUE
            where p.PurchaseID = PurchaseID;

            update Suppliers s
            set s.AmntOwed = s.AmntOwed - amount
            where s.ID = SupplierID;

            insert into PaymentsPaid('PurchaseID','SupplierID', 'SupplierName', 'day','Amnt') values (PurchaseID, SupplierID, SupplierName, curdate(),amount);
        end if;
    else
        select 'No valid invoice.';
    end if;
end
//
delimiter ;

call MakePayment(1000);
call MakePayment(1001);
```

```
✓ 12 11:01:27 call MakePayment(1000) 1 row(s) affected 0.265 sec
✓ 15 11:02:48 call MakePayment(1001) 1 row(s) affected 0.187 sec
```

Table: Purchases

	PurchaseID	SupplierID	SupplierName	model	quantity	day	TotalAmnt	paid
▶	1000	1005	Dell	Alienware x51	1	2016-04-03	16489	1
	1001	1001	Steam	Maingear Drift	4	2016-04-03	4400	1
	1002	1002	Sony	PlayStationVR	3	2016-04-03	1050	0
	1003	1003	Microsoft	Xbox 360	3	2016-04-03	750	0
	1004	1001	Steam	Alienware A	3	2016-04-03	1797	0

Table: PaymentsPaid

	PaymentID	PurchaseID	SupplierID	SupplierName	day	Amnt
▶	1000	1000	1005	Dell	2016-04-10	16489
	1001	1001	1001	Steam	2016-04-10	4400

- My Client wants a convenient stock checking procedure to produce a list of quantities of all products in stock. This procedure takes no arguments and uses cursors to iterate through the three stock tables in the database, Consoles, VRHeadsets and PCs. Column one and column five of each of these tables are the model and quantity in stock of each of the products. These are output to a comma separated file called stock.csv which could be viewed in Microsoft Excel as a spreadsheet.

DDL: StockCheck()

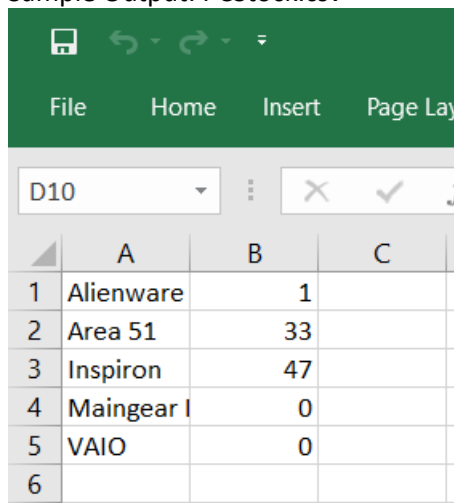
```
drop procedure if exists StockCheck;

delimiter //
create procedure StockCheck()
begin
    SELECT c.`model`, c.`quantity` from Consoles c
    INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/consolestock.csv'
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n';

    SELECT h.`model`, h.`quantity` from VRHeadsets h
    INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/headsetstock.csv'
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n';

    SELECT p.`model`, p.`quantity` from PCs p
    INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/pcstock.csv'
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n';
end ;
//
delimiter ;
```

Sample Output: PCStock.csv



	A	B	C
1	Alienware	1	
2	Area 51	33	
3	Inspiron	47	
4	Maingear I	0	
5	VAIO	0	
6			

4. My client wants functionality built into the database to cater for low stocks which will facilitate the reordering of the models in question. The following three triggers are invoked during a sale where stocks fall below a pre-defined threshold. Any models under this threshold are added to the Reorder table.

```

drop trigger if exists LowConsoleStock;
drop trigger if exists LowVRHeadsetStock;
drop trigger if exists LowPCStock;

delimiter //
create trigger LowConsoleStock
after update on Consoles
for each row
begin
    if(OLD.Quantity < 5) then
        insert into Reorder(`model`,`CurStock`,`day`) values(old.model,old.quantity,curdate());
    end if;
end;
//
delimiter ;

delimiter //
create trigger LowVRHeadsetStock
before update on VRHeadsets
for each row
begin
    if(OLD.Quantity < 5) then
        insert into Reorder(`model`,`CurStock`,`day`) values(old.model,old.quantity,curdate());
    end if;
end;
//
delimiter :
delimiter //
create trigger LowPCStock
before update on PCs
for each row
begin
    if(OLD.Quantity < 5) then
        insert into Reorder(`model`,`CurStock`,`day`) values(old.model,old.quantity,curdate());
    end if;
end;
//
delimiter ;

```

Table: Consoles

	model ▲	mem	gpu	cpu	quantity
▶	Alienware A	8GB DDR	Nvidia Geforce	Intel Corei7	0
	Alienware B	4GB DDR	Asus	Intel Corei3	0
	PS3	525MB	Nvidia	AMD	3
	PS4	8GB	Nvidia	Intel Corei7	0
	Syber I	4GB	Nvidia Geforce 750	Intel Corei3	0

```
call SellProduct(1008, 'PS4', 1);
```

Table: ReOrder

	ReorderID	model	CurStock	day
▶	1	PS4	0	2016-04-10

- My client has strict credit terms of 10,000 agreed with all of his customers. Therefore, I include a trigger to prevent any sale that would put a customer's account over this limit. Once the customer clears his account below a threshold that will allow the sale, a sale can continue. When the credit check on the customer fails, the trigger causes a roll back preventing updates to the sales or stock tables at the same time.

```
delimiter //
create trigger CreditLimit
before update on Customers
for each row
begin

declare message char(88) default 'This Transaction will put';

if(NEW.AmntDue > 10000) then
set message = concat(message, ' ', OLD.name, ' over credit limit. Current balance is', ' ', OLD.AmntDue);
SIGNAL SQLSTATE '12345' SET MESSAGE_TEXT = message;
call crazy;
end if;
end;
//
delimiter ;
```

```
call SellProduct(1002, 'Syber p', 1);
```

✖ 108 15:12:16 call SellProduct(1002, 'Sybe... Error Code: 1644. This Transaction will put GameXchange over credit limit. Current balance is 979683 0.125 sec

The customer's credit balances are unrealistically high from populating the Sales table with over two and a half thousand random sales for indexing tests, but it works well to test this trigger. A credit check fail on a customer's record will cause a role back on the query that triggered it, preventing the sale as a whole, including updates to stock and sales tables.

6. My client has strict 30 day credit agreements with his customers. The following trigger triggers during a sale if the customer has any unpaid invoices dating further than 30 days ago. If so, he will have to pay these invoices before he can buy any more products.

```

delimiter //
create trigger CreditDate
before update on Customers
for each row
begin

    declare CheckPaid bool default FALSE;
    declare CreditDays int default 0;
    declare message char(88) default 'Cannot continue transaction: ';

    DECLARE finished INTEGER DEFAULT 0;

    DECLARE col1,col2 int;
    DECLARE col8 bool default FALSE;
    DECLARE col6 double;

    -- sales_cursor iterates through the Sales table checking for unpaid invoices.
    DECLARE sales_cursor CURSOR FOR SELECT `SaleID`,`CustomerID`,`day`,`paid` FROM Sales;

    -- declare NOT FOUND handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

    OPEN sales_cursor;

    get_sales: LOOP
        FETCH sales_cursor INTO col1,col2,col6,col8;

        IF finished = 1 THEN
            LEAVE get_sales;
        END IF;

        -- if sale belongs to the customer and is not paid
        if(col2 = OLD.ID and col8 = FALSE)then
            set CreditDays = ( curdate() - col6 );
            if(CreditDays > 30) then
                set message = concat(message, ' ', OLD.name, ' has outstanding invoice No.: ', col1);
                SIGNAL SQLSTATE '12345' SET MESSAGE_TEXT = message;
                -- call crazy;
            end if;
        end if;
        ITERATE get_sales;
    END LOOP get_sales;
    CLOSE sales_cursor;
end;
//
delimiter ;

```

Table: Sales

	SaleID	CustomerID	CustomerName	model	quantity	day	TotalAmnt	paid
	1000	1005	Games Ltd	Maingear Drift	1	2016-03-04	1100	1
	1001	1005	Games Ltd	Syber X	1	2016-04-03	499	1
	1002	1007	Larry's Game Lab	Syber I	3	2016-02-05	825	0
	1003	1002	GameXchange	PS4	5	2016-01-03	1750	0

```
call SellProduct(1002, 'Syber P', 1);
```

194 17:28:12 call SellProduct(1002, 'Syber P', 1) Error Code: 1644. Cannot continue transaction: GameXchange has outstanding invoice No.: 1003 0.110 sec

7. My client has credit terms agreed with his suppliers. He wants functionality built into the database to help with book keeping which will tell if he has any overdue invoices to be paid to his suppliers as described in the logical concept at the beginning of this report. With this in mind, I have created a Procedure AgedAccounts() which adheres to his credit terms and writes any aged creditors accounts to a csv file for convenient viewing as a spreadsheet. This procedure will be invoked from the front end at the end of each month.

DDL: AgedAccounts

```
drop procedure if exists AgedAccounts;

delimiter //
create procedure AgedAccounts()
begin
    SELECT p.PurchaseID, p.SupplierName, p.`Day`, p.TotalAmnt
    INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/AgedAccounts.csv'
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
    FROM Purchases p
    WHERE ( (p.`paid` = FALSE) AND ( p.`SupplierName` = 'Steam' OR p.`SupplierName` = 'Sony') AND (curdate() - p.`day` > 30 )
           OR ( p.`SupplierName` = 'Microsoft' OR p.`SupplierName` = 'Dell') AND (curdate() - p.`day` > 60 )
           OR ( p.`SupplierName` = 'Facebook') AND (curdate() - p.`day` > 90 ) );
end ;
//
delimiter ;
```

DML: Change Purchases

```
update Purchases p
set p.`day` = '2014-01-01'
where p.`PurchaseID` = 1234 OR p.`PurchaseID` = 1471 OR p.`PurchaseID` = 1738;

call AgedAccounts();
```

AgedAccounts.csv:

File Home Insert Page Layout Formulas Data					
T29					
	A	B	C	D	E
1	1234	Dell	01/01/2014	49467	
2	1471	Sony	01/01/2014	1000	
3	1738	Dell	01/01/2014	13596	
4					
5					

8. My client runs a special offer advertisement campaign offering a 2% discount to help sell old stock. He wants this functionality automated in the database. The following trigger takes 2% of the total amount of a sale where the quantity in stock of the model is above a given threshold. It deducts the difference from the amount due which is applied to the customer.

```
drop trigger if exists SpecialOffer;

delimiter //
create trigger SpecialOffer
before insert on Sales
for each row
begin

    declare CheckQuantity int default 0;
    declare difference int default 0;

    set difference = NEW.TotalAmnt;

    select c.quantity into CheckQuantity from Consoles c where NEW.model = c.model;
    select h.quantity into CheckQuantity from VRHeadsets h where NEW.model = h.model;
    select p.quantity into CheckQuantity from PCs p where NEW.model = p.model;

    if(CheckQuantity > 100) then
        set NEW.TotalAmnt = (NEW.TotalAmnt / 1.02);
        set difference = difference - NEW.TotalAmnt;
    end if;

    update Customers
    set AmntDue = AmntDue - difference
    where ID = NEW.CustomerID;

end;
//
delimiter ;

update PCs
set quantity = 105
where model = 'VAIO';

call SellProduct(1001, 'VAIO', 1);
```

Table: Sales

2705	1001	Game Stop	VAIO	1	2016-04-14	882	0
------	------	-----------	------	---	------------	-----	---

Design Update

1. Addition of 'Paid' (Boolean) column to Sales and Purchases tables to allow for the creation of triggers to test for Customer and supplier credit terms.
2. Addition of PaymentsReceived & PaymentsPaid tables.

Updated DDL:

```
create table if not exists Sales(
  `SaleID`      int not null auto_increment,
  `CustomerID`  int,
  `CustomerName` varchar(255),
  `model`       varchar(15),
  `quantity`    int,
  `day`         date,
  `TotalAmnt`   int,
  `paid`        bool,
  primary key (`SaleID`),
  foreign key (`CustomerID`) references Customers(`ID`) on delete cascade,
  foreign key (`model`) references Products(`model`) on delete cascade
)auto_increment = 1000;

create table if not exists PaymentsReceived(
  `PaymentID`   int not null auto_increment,
  `SaleID`      int,
  `CustomerID`  int,
  `CustomerName` varchar(255),
  `day`         date,
  `Amnt`        int,
  primary key (`PaymentID`),
  foreign key (`SaleID`) references Sales(`SaleID`) on delete cascade,
  foreign key (`CustomerID`) references Customers(`ID`) on delete cascade
)auto_increment = 1000;

create table if not exists Purchases(
  `PurchaseID`  int not null auto_increment,
  `SupplierID`  int,
  `SupplierName` varchar(255),
  `model`       varchar(15),
  `quantity`    int,
  `day`         date,
  `TotalAmnt`   int,
  `paid`        bool,
  primary key (`PurchaseID`),
  foreign key (`SupplierID`) references Suppliers(`ID`) on delete cascade,
  foreign key (`model`) references Products(`model`) on delete cascade
)auto_increment = 1000;

create table if not exists PaymentsPaid(
  `PaymentID`   int not null auto_increment,
  `PurchaseID`  int,
  `SupplierID`  int,
  `SupplierName` varchar(255),
  `day`         date,
  `Amnt`        int,
  primary key (`PaymentID`),
  foreign key (`PurchaseID`) references Purchases(`PurchaseID`) on delete cascade,
  foreign key (`SupplierID`) references Suppliers(`ID`) on delete cascade
)auto_increment = 1000;
```

3. Addition of table 'Reorder' to add functionality to cater for low stock triggers . A trigger will execute during a sale and if the sale puts the quantity in stock of the product under a certain threshold it is added to this table to facilitate reordering. Also, the PurchaseProduct() has been updated to remove a purchased product from the new Reorder table.

DDL Table: ReOrder

```
create table if not exists ReOrder(
  `ReorderID`    int not null auto_increment,
  `model`        varchar(15),
  `CurStock`    int,
  `day`          date,
  primary key (`ReorderID`)
)auto_increment 0;
```

DDL: PurchaseProduct()

```
-- Record the purchase in the purchases table.
insert into Purchases(`SupplierID`,`SupplierName`,`model`,`quantity`
call RemoveFromReOrder(model);
end if;
```

DDL: RemoveFromReorder()

```
delimiter //
create procedure RemoveFromReOrder(model varchar(15))
begin
  delete from ReOrder
  where ReOrder.model = model;
end
//
delimiter ;
```

Errors & Anomalies

1. Calling PopulatePurchases(2000) did not result in a system crash but calling PopulateSales(2000) resulted in the same out of sync error as before after calling 50 stock items that were out of stock, but now the sales and purchases tables have a Paid column to allow for design and testing of credit terms. The Sales table currently hold 1690 records for design and testing purposes.

```
call PopulateSales(2000);
```

	Not enough stock to fulfill order.
▶	Not enough stock to fulfill order.

Result 50 ×

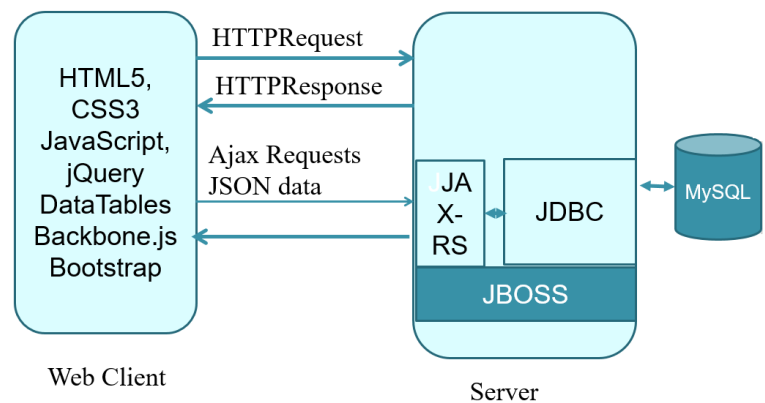
❌ 91 19:13:47 call PopulateSales(2000) Error Code: 2014 Commands out of sync; you can't run this command now

2. Table PaymentsReceived auto increment on PaymentID began at 0 instead of 1000.

3. File I/O in MySQL is protected by the 'secure_file_priv' global variable. Either you deactivate the variable, or you read and write files to the directory set by this variable, which on the windows operating system is C:\ProgramData\MySQL\MySQL Server 5.7\Uploads\. This was used in the StockCheck() and AgedAccounts() procedures to create .csv files for stock reports and aged accounts information. The Output file must not exist in advance, so they need to be used and deleted each time a stock check or aged account check is needed.
4. Signalling an SQLState from a trigger seems to terminate execution of the triggering query, causing a roll back in select, update, deletes performed earlier in the query. As in the CreditLimit trigger, before an update on Customers, I don't have to worry about the updates to stock tables that happen in the SellProduct() procedure because if the customer is over his credit limit, these updates are rolled back by the trigger.

6. Front End Technology

The front end is a web client that connects remotely to the MySQL server over a network via a JBoss server. This facilitates access of the database to the different users from different departments, ie; Purchases, Sales, Stock and Customer Services. It also provides for mobile access to the database. The front end connects to a JBoss java enabled server which takes restful JJAX requests from the client and retrieves / updates the relevant data from the database. It sends the data to the client as JSON data which the client renders accordingly. Java



Database Connectivity (**JDBC**) is an application programming interface (API) which defines how a client may access a database.

```

var ModalInvoice = Backbone.View.extend({
  model: PC,

  events:{
    "click #btnSubmit": "saveInvoice",
    "click #btnCancel": "cancelInvoice",
  },

  saveInvoice: function(e) {
    this.model.call({
      SellProduct('#custID', '#model', '#quantity');
    });

    return false;
  }
});
  
```


The front end consists of forms such as invoices, and purchases that the users fill out. These forms are viewed as backbone.js invoked modals in the client as seen here. Upon submitting the forms, the procedures and triggers in the database are invoked and the data in the database is updated accordingly. Seen here is an example of an invoice modal. Clicking Submit saves the values from the fields and sends them as JSON data to the server as seen in the code below. The server wraps the JSON data in MySQL and queries the database with it. The Java code running on the server for saving an invoice, saves the values entered in the fields of the form to the database.

7. Initial Rollout

The front end is more user friendly than having to use the command line to enter data directly into the database. It also allows the users to access the data remotely. The design of the database has not had to change to suit the front end.

The front end provides an element of security by using user logins to only serve the forms permitted to a given user. It also ensures data integrity by limiting the values of fields in the database that a user can change.

In future rollouts, this functionality will be scaled up to allow an invoice or stock purchase to consist of many models of different types of products.

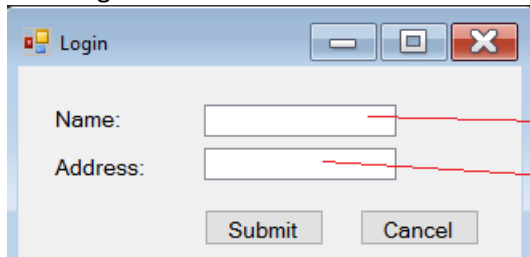
Errors & Anomalies

This iteration of the database only allows the purchase or sale of one model type per transaction.

8. Sign-Off

The user interface for the database is intuitive. The date field of an invoice or purchase autocompletes with the current date. You can only select customers or suppliers, and models included in the drop down menus. Enter quantities being purchased or sold and click submit when finished. User instructions are as follows:

User Login.

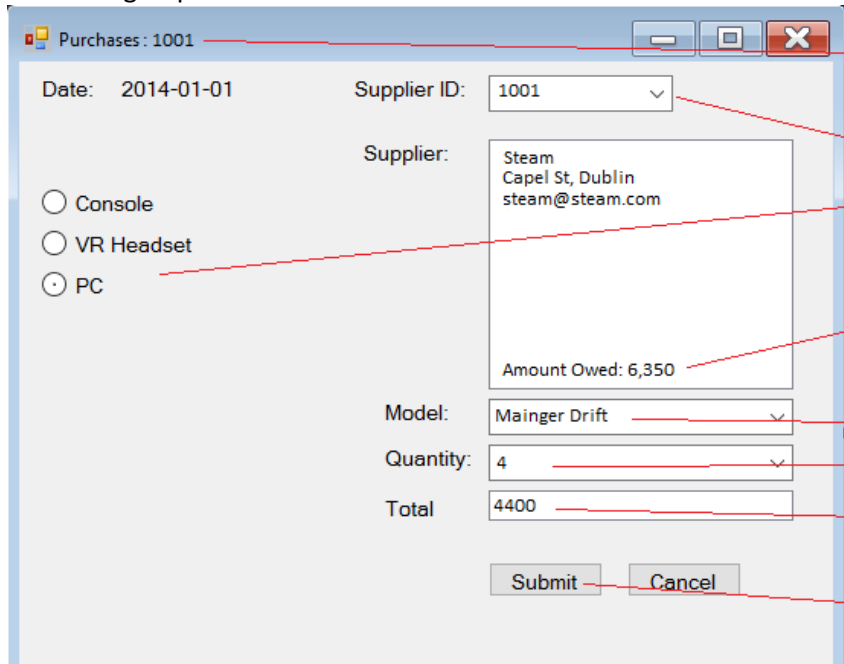


The User Login window contains two text input fields: 'Name:' and 'Address:'. Below these fields are two buttons: 'Submit' and 'Cancel'. Red lines point from the text labels to the respective input fields.

Enter User Name.

Enter Password.

Purchasing Department Interface:



The Purchasing Department Interface window displays the following fields and controls:

- Date:** 2014-01-01
- Supplier ID:** 1001 (dropdown menu)
- Supplier:** Steam, Capel St, Dublin, steam@steam.com
- Product Type:** Radio buttons for Console, VR Headset, and PC (PC is selected).
- Amount Owed:** 6,350
- Model:** Mainger Drift (dropdown menu)
- Quantity:** 4
- Total:** 4400
- Buttons:** Submit and Cancel

Red lines point from the text labels to the corresponding fields in the interface.

Purchase ID no automatically generated.

Select Supplier by ID number.

Select Product Type

Amount Owed.

Select Model from menu.

Input Quantity.

Total of Purchase.

Submit Purchase to database.

Suppliers

Supplier ID: 1005

Name: Dell

Address: CheryWood Park, Dublin

email: admin@dell.com

Purchase: 1008, 1005, Dell, Area 51, 4, 2016-04-03, 13596,

Total:

Purchase Total: 13596

Supplier Total: 4563,99

☐ Paid

Submit Cancel

Select Supplier By ID.

Edit Supplier Details.

View Purchase Record

Edit Purchase Details.

Submit Changes.

Sales Department Interface:

Invoice: 1001

☐ Paid

Customer ID: 1005

Model: Syber X

Quantity: 1

Total: 499

Cancel

Games Ltd
Naas Road,
Dublin
contact@gamesltd.ie
Amnt Due: 7,560

Invoice No. Generated Automatically.

Select Customer by ID

Customer Details.

Select Product.

Select model from drop down menu.

Enter Quantity.

Total of invoice.

Submit button hidden on Invoice Payment.

Customer Care Department Interface:

The screenshot shows a web application window titled "Customers". It contains several form fields and buttons. Red lines with labels point to specific elements:

- Select Customer By ID.** points to the "Customer ID" dropdown menu.
- Edit Customer Details** points to the "Name" and "Address" text input fields.
- View Customer Invoices** points to the "Invoices" dropdown menu.
- Edit Invoice Details.** points to the "Invoice Total" text input field and the "Paid" radio button.
- Submit Changes.** points to the "Submit" button.

The form fields contain the following data:

- Customer ID: 1007
- Name: Larry's Game Lab
- Address: Blanchardstown, Dublin
- email: Lgl@gmail.com
- Invoices: 1008, 1007, Larry's Game Lab, PlayStationVR, 2, 2016-04-03, 700, 0
- Invoice Total: 700
- Customer Total: 9670.54

Buttons: Submit, Cancel