Deliverable 1:

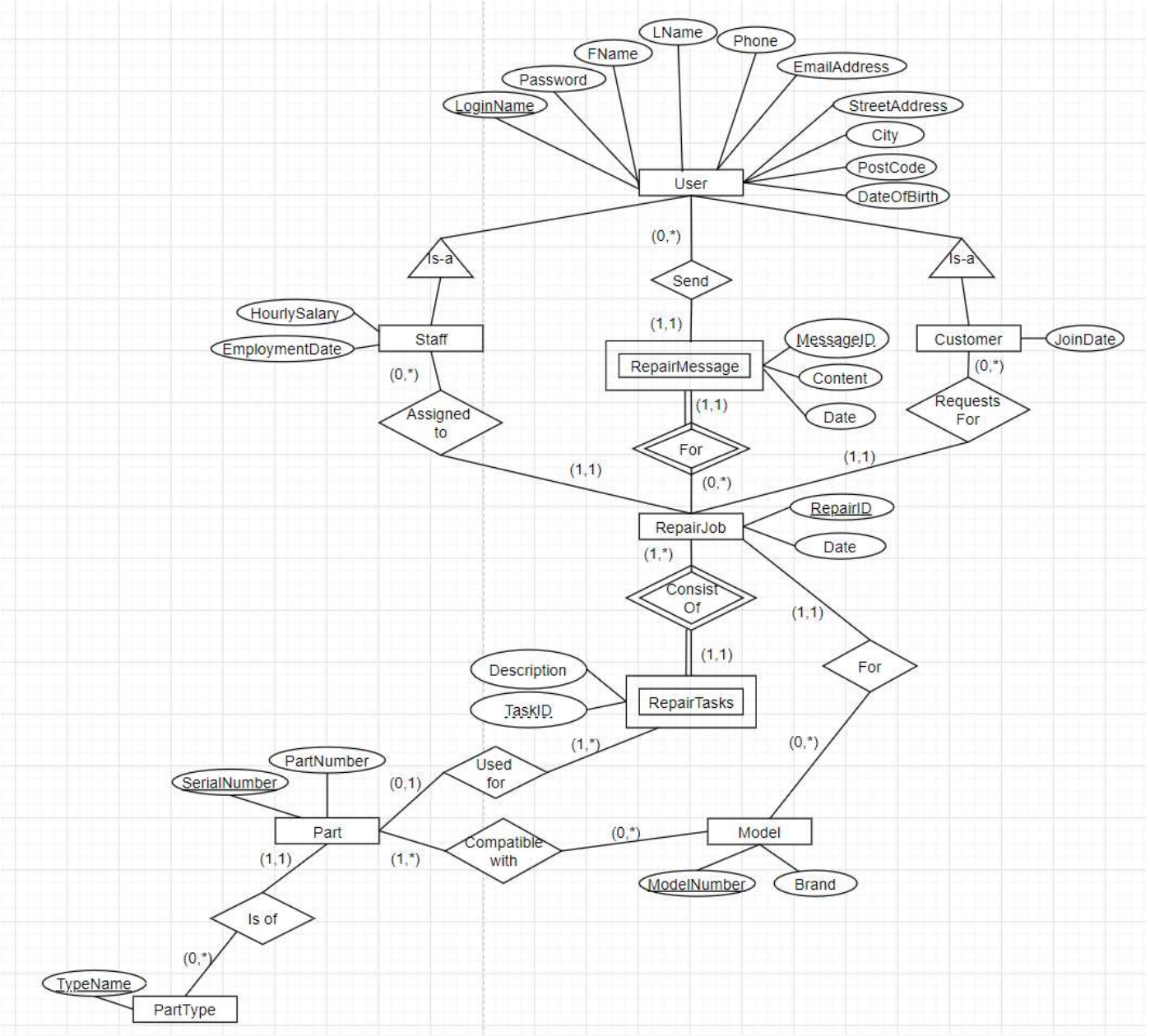
Task 1: Database Application Description

Key: Entities Attributes Relationships Cardinalities

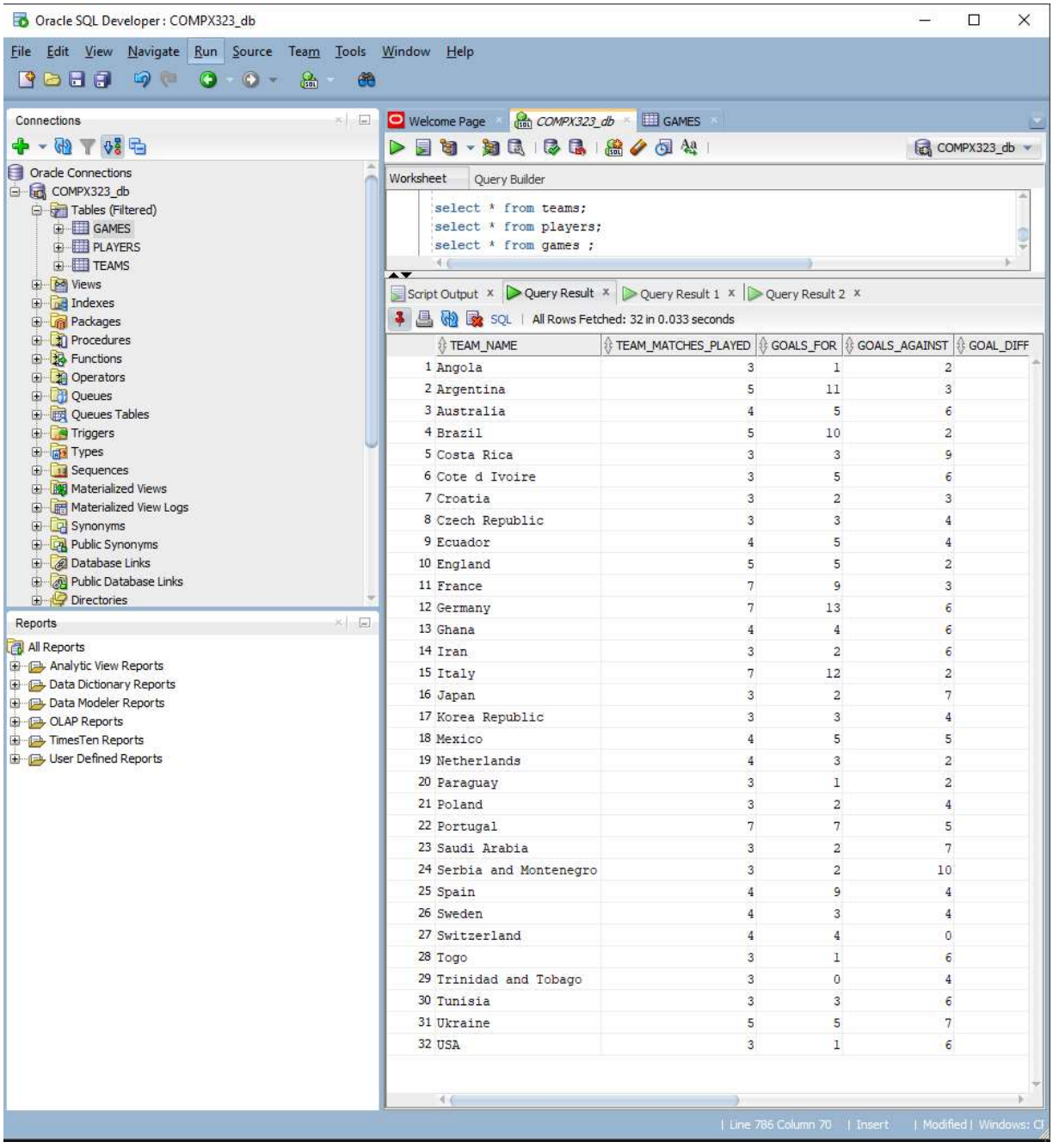
The creation of a repair shop job management app where Users will be able to manage dealing between one another through a database backend. There will be 2 types of users: Customers and Staff. Customers can login and check RepairJobs which they have arranged with the repair shop. Staff members can login to the app to manage sorting out the RepairJobs for the Customers. Customers and Staff are Generalised Users with LoginName, Password, Email, Phone, Address, DateOfBirth... Etc but Staff members will have a HourlySalary and EmploymentDate and Customers will have a JoinDate. Customers can send RepairMessages to Staff regarding the repairs currently in progress for which RepairMessages can be sent back by the Staff to update the Customers. Any number of Parts can be Used for a RepairTask to inform the Staff of Parts that need to be used for the repair, Each RepairTask will be Assigned to a single Staff member for them to handle, A RepairJob can consist of one or more RepairTasks. RepairTasks will have TaskDetails to describe what the Staff will need to perform to complete that part of the repair. The inventory management functionality of the app will allow for Staff to order in new Parts and check the parts currently present. Parts all have a unique SerialNumber will have a given PartType (which will have a TypeName of Screen, Battery Keyboard etc) Parts must be compatible with one or more Models (Apple Macbook Pro 15, MSI GS65, ... etc) of device rather than being for a specific device. Users will be able to view their account details and change their Password, Email, Phone, Address but not LoginName, DateOfBirth, HourlySalary, EmploymentDate, JoinDate etc as these will all be managed by the repair shop owner admin to prevent users from changing their Unique ID (LoginName) or other sensitive information which shouldnt be able to be freely edited. Staff users will be able to setup an account for first time Customers with a User account through the creating one for them on the “Customer Signup Page” on the Staff section of the app. Staff users will be added to the database directly by the repair shop owner admin so a Staff sign up page will not need to be implemented.

We expect that the majority of the difficulties will be faced in complexity of the queries to achieve the specific app functionality, We hope to make as much of the sorting of data be sorted via dropdowns (Such as PartType and Brand) to make searching for required Parts easier and more reliable than having users type in keywords. Other challenges will most likely be how we decide to handle the way which Customers can also be Staff members, we will most likely need to have the app detect if a person is both a Staff member and a Customer for it to present them with a options screen to select ether to access the Staff or Customer section of the app.

Task 2: ER Diagram (Revised) :



Task 3: Oracle login DB instance with fifa world cup test data:



Deliverable 2:

Task 1: Relational Schema:

User(LoginName, Password, FName, LName, Phone, EmailAddress, StreetAddress, City, PostCode, DateOfBirth)
Staff(LoginName, HourlySalary, EmploymentDate)
Customer(LoginName, JoinDate)
RepairJob(RepairID, Date, CustomerLogin, StaffLogin, ModelNumber)
Part(SerialNumber, PartNumber, TaskID, RepairID, TypeName)
Model(ModelNumber, Brand)
PartType(TypeName)
RepairMessage(MessageID, RepairID, Content, Date, LoginName)
RepairTasks(TaskID, RepairID, Description)
CompatibleWith(SerialNumber, ModelNumber)

Task 2: Relations in Oracle:

--(Chosen Tables were: UserAccount, DeviceModel, Staff, Customer, RepairJob)

```
create sequence Repair_seq
minvalue 1
start with 1
increment by 1;

CREATE TABLE UserAccount (
 LoginName VARCHAR(32) PRIMARY KEY,
 Password VARCHAR(32) NOT NULL,
 FName VARCHAR(32) NOT NULL,
 LName VARCHAR(32) NOT NULL,
 Phone VARCHAR(16) NOT NULL,
 EmailAddress VARCHAR(64) NOT NULL,
 StreetAddress VARCHAR(64) NOT NULL,
 City VARCHAR(32) NOT NULL,
 PostCode VARCHAR(4) NOT NULL,
 DateOfBirth DATE NOT NULL
);

CREATE TABLE DeviceModel (
 ModelNumber VARCHAR(32) PRIMARY KEY,
 Brand VARCHAR(16) NOT NULL
);

CREATE TABLE Staff (
 LoginName VARCHAR(32) PRIMARY KEY,
 HourlySalary DECIMAL NOT NULL,
 EmploymentDate DATE NOT NULL,
 FOREIGN KEY (LoginName) REFERENCES UserAccount(LoginName)
);

CREATE TABLE Customer (
 LoginName VARCHAR(32) PRIMARY KEY,
 JoinDate DATE NOT NULL,
 FOREIGN KEY (LoginName) REFERENCES UserAccount(LoginName)
);
```

```
CREATE TABLE RepairJob (
 RepairID INTEGER PRIMARY KEY,
 RepairDate DATE,
 CustomerLogin VARCHAR(32) NOT NULL,
 StaffLogin VARCHAR(32) NOT NULL,
 ModelNumber VARCHAR(32) NOT NULL,
 FOREIGN KEY (CustomerLogin) REFERENCES UserAccount(LoginName),
 FOREIGN KEY (StaffLogin) REFERENCES UserAccount(LoginName),
 FOREIGN KEY (ModelNumber) REFERENCES DeviceModel(ModelNumber)
);

--Adds checks as required for User Account Values
ALTER TABLE UserAccount ADD CONSTRAINT chk_Email
CHECK (regexp_like(EmailAddress, '^[A-Za-z0-9\.\_\-]{1,32}\@[A-Za-z0-9]{1,32}.[A-Za-z0-9.]{3,32}$'));
ALTER TABLE UserAccount ADD CONSTRAINT chk_Address
 CHECK (regexp_like(StreetAddress, '^[0-9]{1,5} [A-Za-z]{2,32}[ ]{0,1}[A-Za-z]{0,32}[ ]{0,1}[A-Za-z]{0,32}$'));
 ALTER TABLE UserAccount ADD CONSTRAINT chk_Postcode
 CHECK (regexp_like(PostCode, '^[0-9]{4}$'));

 --Adds checks as required for Staff Member Values
 ALTER TABLE Staff ADD CONSTRAINT chk_PayCorrect
 CHECK (HourlySalary > 0 AND HourlySalary < 999);

--Extra Tables (Not Used in Task 4: Application Project)

CREATE TABLE PartType (
 TypeName VARCHAR(32) PRIMARY KEY
);
CREATE TABLE RepairTask (
 TaskID INTEGER NOT NULL,
 RepairID INTEGER NOT NULL,
 Content VARCHAR(512) NOT NULL,
 constraint PK_RT primary key (TaskID, RepairID),
 FOREIGN KEY (RepairID) REFERENCES RepairJob(RepairID)
);
CREATE TABLE Part (
 SerialNumber VARCHAR(32) PRIMARY KEY,
 PartNumber VARCHAR(32) NOT NULL,
 TaskID INTEGER,
 RepairID INTEGER,
 TypeName VARCHAR(32) NOT NULL,
 FOREIGN KEY (TaskID, RepairID) REFERENCES RepairTask(TaskID, RepairID),
 FOREIGN KEY (TypeName) REFERENCES PartType(TypeName)
);
CREATE TABLE RepairMessage (
 MeassageID INTEGER NOT NULL,
 RepairID INTEGER NOT NULL,
 MContent VARCHAR(512) NOT NULL,
 MessageDate DATE NOT NULL,
 SenderLogin VARCHAR(32) NOT NULL,
 constraint PK_RM primary key (MeassageID, RepairID),
 FOREIGN KEY (RepairID) REFERENCES RepairJob(RepairID),
 FOREIGN KEY (SenderLogin) REFERENCES UserAccount(LoginName)
);

CREATE TABLE CompatibleWith (
 SerialNumber VARCHAR(32) NOT NULL,
 ModelNumber VARCHAR(32) NOT NULL,
 constraint PK_CW primary key (SerialNumber, ModelNumber),
 FOREIGN KEY (SerialNumber) REFERENCES Part(SerialNumber),
 FOREIGN KEY (ModelNumber) REFERENCES DeviceModel(ModelNumber)
);
```

Task 3: Test Data Sets:

Small Data:

```
INSERT INTO DeviceModel VALUES ('Predator 15', 'ACER');
INSERT INTO DeviceModel VALUES ('GS75 Stealth', 'MSI');
INSERT INTO DeviceModel VALUES ('GT76 Titan', 'MSI');
INSERT INTO DeviceModel VALUES ('GE66 Raider', 'MSI');
INSERT INTO DeviceModel VALUES ('Predator 17', 'ACER');
INSERT INTO DeviceModel VALUES ('Blade 17', 'RAZOR');
INSERT INTO DeviceModel VALUES ('Galaxy Book 13', 'SAMSUNG');
INSERT INTO DeviceModel VALUES ('Macbook 13', 'APPLE');
INSERT INTO DeviceModel VALUES ('Macbook 16', 'APPLE');
INSERT INTO DeviceModel VALUES ('Notebook 9 Pro 15', 'SAMSUNG');
INSERT INTO UserAccount VALUES ('ImStaffMember', 'QWERTYUIOP', 'John', 'Doe', '(07)9543632', 'ImStaffMember@slingshot.co.nz', '82 Hillcrest',
'Hamilton', '0687', to_date('1989/7/27', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('ImAlsoStaffMember', 'POIUYTREWQ', 'Marth', 'Helps', '(09)2566730', 'ImAlsoStaffMember@yahoo.com', '19
School Street', 'Wellington', '0978', to_date('1990/5/7', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('Sheeple4Apple', 'GeniusBar', 'Tim', 'Cook', '(09)0639482', 'Sheeple4Apple@icloud.com', '2 Jackson Street',
'Auckland', '0687', to_date('1978/9/29', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('XxX_ProGamer_XxX', 'Skyrim', 'Tod', 'Howard', '(01)2464205', 'XxX_Gaming_XxX@hotmail.com', '90 Ferry
Lane', 'Hamilton', '0687', to_date('1959/8/4', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('MSI_Butter', 'EVGAJam11', 'Podel', 'Meme', '(06)5386957', 'MSI_Butter@slingshot.co.nz', '17 Heron Close',
'Wellington', '0978', to_date('1990/10/26', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('Sammy1', 'GalaxyBrain1', 'Sam', 'Man', '(47)7164223', 'Sammy1@yahoo.com', '90 Jackson Street', 'Auckland',
'0687', to_date('1965/9/11', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('Nvidia_FanBoy', 'RTX_SUX', 'Lisa', 'Sue', '(24)7426905', 'Nvidia_FanBoy@gmail.com', '12 Tudor Road',
'ChristChurch', '3584', to_date('1966/4/18', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('MrMicrosoft', 'XBOX1', 'Bill', 'Gates', '(06)5424203', 'NoInsurance@slingshot.co.nz', '35 Lyndhurst Road',
'ChristChurch', '3584', to_date('1978/8/21', 'yyyy/mm/dd'));
INSERT INTO UserAccount VALUES ('MySystemCrashed', 'DataRecovery', 'Protogent', 'Virus', '(53)8324246', 'MySystemCrashed@slingshot.co.nz',
'45 Byron Street', 'ChristChurch', '3584', to_date('1988/7/2', 'yyyy/mm/dd'));
INSERT INTO Staff VALUES ('ImStaffMember', '29', to_date('1998/2/15', 'yyyy/mm/dd'));
INSERT INTO Staff VALUES ('ImAlsoStaffMember', '26', to_date('2000/7/7', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('Sheeple4Apple', to_date('2014/12/23', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('XxX_ProGamer_XxX', to_date('2013/3/10', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('MSI_Butter', to_date('2012/11/28', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('Sammy1', to_date('2012/6/23', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('Nvidia_FanBoy', to_date('2011/3/17', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('MrMicrosoft', to_date('2014/7/20', 'yyyy/mm/dd'));
INSERT INTO Customer VALUES ('MySystemCrashed', to_date('2012/12/20', 'yyyy/mm/dd'));
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2016/1/10', 'yyyy/mm/dd'), 'Sheeple4Apple', 'ImStaffMember', 'Macbook 13');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2015/4/15', 'yyyy/mm/dd'), 'Sheeple4Apple', 'ImAlsoStaffMember', 'Macbook 16');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2017/8/22', 'yyyy/mm/dd'), 'MSI_Butter', 'ImAlsoStaffMember', 'GT76 Titan');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2016/11/29', 'yyyy/mm/dd'), 'MySystemCrashed', 'ImStaffMember', 'Predator 15');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2016/7/29', 'yyyy/mm/dd'), 'Sammy1', 'ImStaffMember', 'Galaxy Book 13');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2014/6/19', 'yyyy/mm/dd'), 'MrMicrosoft', 'ImStaffMember', 'Blade 17');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2017/7/9', 'yyyy/mm/dd'), 'MySystemCrashed', 'ImAlsoStaffMember', 'GE66
Raider');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2016/3/22', 'yyyy/mm/dd'), 'Nvidia_FanBoy', 'ImStaffMember', 'Notebook 9 Pro 15');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2015/1/28', 'yyyy/mm/dd'), 'MySystemCrashed', 'ImStaffMember', 'Predator 17');
INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('2017/4/29', 'yyyy/mm/dd'), 'Nvidia_FanBoy', 'ImAlsoStaffMember', 'GS75 Stealth');
```

Table Contents:

UserAccount:

	LOGINNAME	PASSWORD	FNAME	LNAME	PHONE	EMAILADDRESS	STREETADDRESS	CITY	POSTCODE	DATEOFBIRTH
1	ImStaffMember	QWERTYUIOP	John	Doe	(07) 9543632	ImStaffMember@slingshot.co.nz	82 Hillcrest	Hamilton	0687	27/07/89
2	ImAlsoStaffMember	POIUYTREWQ	Marth	Helps	(09) 2566730	ImAlsoStaffMember@yahoo.com	19 School Street	Wellington	0978	07/05/90
3	Sheeple4Apple	GeniusBar	Tim	Cook	(09) 0639482	Sheeple4Apple@icloud.com	2 Jackson Street	Auckland	0687	29/09/78
4	XxXProGamerXxX	Skyrim	Tod	Howard	(01) 2464205	XxXGamingXxX@hotmail.com	90 Ferry Lane	Hamilton	0687	04/08/59
5	MSIButter	EVGAJam11	Podel	Meme	(06) 5386957	MSIButter@slingshot.co.nz	17 Heron Close	Wellington	0978	26/10/90
6	Sammy1	GalaxyBrain1	Sam	Man	(47) 7164223	Sammy1@yahoo.com	90 Jackson Street	Auckland	0687	11/09/65
7	NvidiaFanBoy	RTXSUX	Lisa	Sue	(24) 7426905	NvidiaFanBoy@gmail.com	12 Tudor Road	ChristChurch	3584	18/04/66
8	MrMicrosoft	XBOX1	Bill	Gates	(06) 5424203	NoInsurance@slingshot.co.nz	35 Lyndhurst Road	ChristChurch	3584	21/08/78
9	MyComputerCrashed	DataRecovery	Protogent	Virus	(53) 8324246	MySystemCrashed@slingshot.co.nz	45 Byron Street	ChristChurch	3584	02/07/88

Customer:

	LOGINNAME	JOINDATE
1	Sheeple4Apple	23/12/14
2	XxXProGamerXxX	10/03/13
3	MSIButter	28/11/12
4	Sammy1	23/06/12
5	NvidiaFanBoy	17/03/11
6	MrMicrosoft	20/07/14
7	MyComputerCrashed	20/12/12

Staff:

	LOGINNAME	HOURLYSALARY	EMPLOYMENTDATE
1	ImStaffMember	29	15/02/98
2	ImAlsoStaffMember	26	07/07/00

DeviceModel:

	MODELNUMBER	BRAND
1	Predator 15	ACER
2	GS75 Stealth	MSI
3	GT76 Titan	MSI
4	GE66 Raider	MSI
5	Predator 17	ACER
6	Blade 17	RAZOR
7	Galaxy Book 13	SAMSUNG
8	Macbook 13	APPLE
9	Macbook 16	APPLE
10	Notebook 9 Pro 15	SAMSUNG

RepairJob:

	REPAIRID	REPAIRDATE	CUSTOMERLOGIN	STAFFLOGIN	MODELNUMBER
1	1	10/01/16	Sheeple4Apple	ImStaffMember	Macbook 13
2	2	15/04/15	Sheeple4Apple	ImAlsoStaffMember	Macbook 16
3	3	22/08/17	MSIButter	ImAlsoStaffMember	GT76 Titan
4	5	29/07/16	Sammy1	ImStaffMember	Galaxy Book 13
5	6	19/06/14	MrMicrosoft	ImStaffMember	Blade 17
6	7	09/07/17	MyComputerCrashed	ImAlsoStaffMember	GE66 Raider
7	8	22/03/16	NvidiaFanBoy	ImStaffMember	Notebook 9 Pro 15
8	9	28/01/15	MyComputerCrashed	ImStaffMember	Predator 17
9	10	29/04/17	NvidiaFanBoy	ImAlsoStaffMember	GS75 Stealth
10	11	29/11/16	MyComputerCrashed	ImStaffMember	Predator 15

Large Dataset:

The large dataset was generated through the following Java code, Comments are included to describe the functionality as well as the reasoning for choices made:

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Random;

//Name:        Daniel Wilson
//Student ID:   1345359

//Custom coded program to allow for around 250,000 unique results to be created in SQL format for adding to the COMPX323
//Repairshop Database Program. Uses indexes into arrays with sections of strings to combine and create usernames, addresses,
//passwords etc as well as randomly generated values for dates.

class CreateDatabaseInformation {

  public static void main(String[] args) {
    CreateDatabaseInformation c = new CreateDatabaseInformation();
    c.Create();
  }
  public void Create() {
    //Used to store the
    ArrayList<String> output = new ArrayList<>();

    String[] FnamesP0 = {"Delma","Glory","Chi","Pearlene","Jeanelle","Beula","Delmer",
        "Donna","Giovanna","Caroline","Kindra","Piedad","Hyon","Hildred","Leida",
        "Domenic","Deann","Willena","Jude","Carina","Rich","Maida","Carmon","Rosaria",
        "Marcell","Moriah","Mac","Bernardina","Lorriane","Diane","Percy","Lila","Verda",
        "Trena","Britany","Kellee","Jolanda","Melaine","Lynnette","Rossana","Will",
        "Tabetha","Hattie","Gertie","Antione","Cammie","Aleshia","Esther","Andre"};

    String[] LnamesP0 = {"Franco","Pena","Wilcox","Cooper","Hawkins","Mcdaniel","Barker",
        "Owens","Hurst","Porter","Melendez","Sherman","Ray","Sharp","Houston",
        "Jefferson","Leblanc","Barajas","Rasmussen","Green","Dickson","Maida","Morrison","Barajas",
        "Newton","Chang","Herrera","Salazar","Chambers","Larson","Mcdowell","Curtis","Shelton",
        "Mahoney","Collier","Oneill","Wagner","Heath","Watkins","Brewer","Beasley",
        "Villa","Jenkins","Gertie","Vance","Ortega","Salazar","Cantrell","Cameron"};

    String[] StreetP0 = {"Chapel Hill","Chantry Close","Tudor Road","Birch Close","Heather Close","Ferry Lane","Park Place",
        "Arthur Street","Church View","School Hill","Ash Grove","Boundary Road","Sandy Lane","Byron Street","Newport Road",
        "Heron Close","Barley Close","Richmond Close","Rowan Drive","Silver Street","School Street","Bankside","Greenacres","The Paddocks",
        "Jasmine Close","Baker Street","Frog Lane","Cavendish Road","Second Avenue","Lower Road","Queensway","Kingsway","Rosemary Lane",
        "South Street","King Edward Road","Vicarage Close","Tower Road","Pear Tree Close","Lyndhurst Road","The Grange","Brookfield Road",
        "Derby Road","Vicarage Gardens","Willow Grove","Conway Road","Jackson Street","Railway Terrace","Devonshire Road","The Pastures"};

    String[] CityP0 = {"San Francisco", "Milwaukee", "Huntington", "Boise", "Scottsdale", "St. Louis", "Henderson", "Jersey City", "Greensboro", "Laredo",
        "Indianapolis", "Miami", "Omaha", "Arlington", "Memphis", "Chesapeake", "Cincinnati", "Fresno", "Columbus", "Riverside"};
    String[] PostCodeP0 = {"0091", "0693", "1456", "2987", "3412", "8208", "1237", "5487", "9034", "9492",
        "6574", "7398", "3584", "2869", "4583", "3256", "0687", "2347", "3486", "4568"};


    //An array of terms to be used for creating the model number

    String[] BrandP1 = {"APPLE", "MSI", "ACER", "ASUS", "SAMSUNG", "RAZOR", "LG"};

    String[] ModelP0 = {"Q-", "Z-", "Y-", "V-", "R-", "K-", "T-", "C-", "P-", "N-"};

    String[] ModelP1 = {"GX", "GW", "GM", "GP", "GD", "GS", "GH",
        "TX", "TW", "TM", "TP", "TD", "TS", "TH",
        "AX", "AW", "AM", "AP", "AD", "AS", "AH",
        "MX", "MW", "MM", "MP", "MD", "MS", "MH",
        "EX", "EW", "EM", "EP", "ED", "ES", "EH"};

    String[] ModelP2 = {"17", "10", "1G", "19", "18", "1Q", "14",
        "37", "30", "3G", "39", "38", "3Q", "34",
        "47", "40", "4G", "49", "48", "4Q", "44",
        "57", "50", "5G", "59", "58", "5Q", "54",
        "77", "70", "7G", "79", "78", "7Q", "74"};

    String[] ModelP3 = {"AF", "AX", "AG", "AT", "AN", "AQ", "AD",
        "BF", "BX", "BG", "BT", "BN", "BQ", "BD",
        "CF", "CX", "CG", "CT", "CN", "CQ", "CD",
        "EF", "EX", "EG", "ET", "EN", "EQ", "ED",
        "FF", "FX", "FG", "FT", "FN", "FQ", "FD"};
```

```
//Used for creating passwords

String[] PasswordP0 = {"Q", "Z", "Y", "V", "R", "K", "T", "C", "P", "N"};

String[] PasswordP1 = {"FH7", "EF0", "GWG", "CD9", "AF8", "JMQ", "FV4",
      "DC7", "SD0", "FAG", "SW9", "AD8", "CDQ", "GR4",
      "UO7", "DW0", "YLG", "DW9", "AI8", "ZHQ", "44",
      "FS7", "ET0", "NUG", "DA9", "XS8", "YJQ", "AR4",
      "VG7", "GE0", "DEG", "JE9", "CS8", "EFQ", "SW4"};

String[] PasswordP2 = {"351", "392", "3G3", "39H", "F8D", "3QS", "34V",
      "25J", "2SS", "2GQ", "29W", "28E", "2VF", "F4F",
      "35G", "39X", "3GD", "3Q", "68F", "3QW", "34F",
      "45E", "49V", "FGC", "4RD", "48S", "4QF", "4PG",
      "55J", "59S,", "IGH", "59T", "58S", "VQH", "54J"};

String[] PasswordP3 = {"3SG1", "3GR2", "3GD3", "39FH", "FK8D", "3IQS", "34HV",
      "EG5J", "2CSS", "2DSQ", "2ASW", "EDFE", "SDFF", "F4FF",
      "4D5E", "4R9V", "FGHC", "4RDD", "48WS", "4QGF", "4WPG",
      "55WD", "HK9F", "5GSG", "7FBE", "U8DF", "5QEB", "54FN",
      "55SJ", "5Q9S,", "IGGH", "59TY", "5W8S", "VGQH", "54J"};

//Used for creating Usernames

String[] UserP0 = {"Blue", "Red", "Green", "Lazy", "Angry", "Sad", "Happy", "Scared", "Big", "Tiny"};

String[] UserP1 = {"Mustang", "Chameleon", "Gazelle", "Deer", "Weasel", "Gorilla", "Bear",
      "Lamb", "Mandrill", "Hare", "Marten", "Puma", "Aoudad", "Fawn",
      "Snake", "Possum", "Wildcat", "Jerboa", "Bighorn", "Walrus", "Porcupine",
      "Woodchuck", "Budgerigar", "Wombat", "Cheetah", "Crab", "Platypus", "Toad",
      "Beaver", "Panther", "Baboon", "Dingo", "Stallion", "Vicuna", "Lynx"};

String[] UserP2 = {"BLL", "BVG", "BGB", "BVH", "FWD", "BQS", "BLV",
      "GLJ", "GSS", "GGQ", "GVW", "GWE", "GVF", "FLF",
      "LYE", "LVV", "FGC", "LRD", "LWS", "LQF", "LPG",
      "ILD", "HVF", "LGG", "7BE", "UWF", "QEB", "LFN",
      "LLJ", "LVS,", "IGH", "LVT", "LWS", "VQH", "LLJ"};


//Used for creating phone numbers
String[] PhoneP0 = {"(07)", "(06)", "(76)", "(47)", "(99)", "(01)", "(03)", "(53)", "(24)", "(31)", "(73)", "(48)", "(52)"};

String[] PhoneP1 = {"832", "653", "063", "354", "742", "246", "194", "538", "552", "269", "542", "716", "493"};

String[] PhoneP2 = {"32", "53", "63", "54", "42", "46", "94", "38", "52", "69", "42", "16", "93"};

String[] PhoneP3 = {"23", "57", "03", "53", "40", "46", "04", "38", "82", "05", "43", "86", "33"};


//Used to assign random email addresses
String[] EmailP0 = {"@gmail.com", "@hotmail.com", "@slingshot.co.nz", "@icloud.com", "@yahoo.com", "@live.com"};

int NUMBER = 250000;

Random rand = new Random();
//Sets the start year for generating random birthdates
int StartYearDOB = 1950;
//Start year for defining signup data / start work date
int StartYearStore = 1985;
//Start year for defining signup data / start work date
int StartYearRepair = 2014;
//For the default salary
int defSal = 25;
//Stores the word for incrementing the repair ID
String autoIncrement = "Repair_seq.NEXTVAL";
//Stores each of the strings for the created names
ArrayList<String> Models = new ArrayList<>();
ArrayList<String> Brands = new ArrayList<>();
ArrayList<String> Passwords = new ArrayList<>();
ArrayList<String> Users = new ArrayList<>();
ArrayList<String> FNames = new ArrayList<>();
ArrayList<String> LNames = new ArrayList<>();
ArrayList<String> Customer = new ArrayList<>();
ArrayList<String> Staff = new ArrayList<>();
ArrayList<String> Streets = new ArrayList<>();
ArrayList<String> Citys = new ArrayList<>();
ArrayList<String> PostCodes = new ArrayList<>();
ArrayList<String> Emails = new ArrayList<>();
```

```
ArrayList<String> PhoneNums = new ArrayList<>();
ArrayList<String> DOBs = new ArrayList<>();
ArrayList<String> SignupDates = new ArrayList<>();
ArrayList<String> HiringDates = new ArrayList<>();
ArrayList<String> HourlySalarys = new ArrayList<>();
ArrayList<String> RepairDates = new ArrayList<>();
//First creates User account data
int count = 0;
for(int y = 0; y < PasswordP0.length; y++){
  for(int q = 0; q < PasswordP1.length; q++){
    for(int t = 0; t < PasswordP2.length; t++){
      for(int j = 0; j < PasswordP3.length; j++){
        count++;
        //Adds a unique data to the array using a combination of all fields
        Users.add(UserP0[y] + UserP1[q] + UserP2[t] + j);
        Passwords.add(PasswordP0[y] + PasswordP1[q] + PasswordP2[t] + PasswordP3[j]);
        Models.add(ModelP0[y] + ModelP1[q] + ModelP2[t] + ModelP3[j]);
        if (count == NUMBER){
          //Sets the loop to make 400,000 results, (200,000 for Customer and Staff)
          for(int i = 0; i < NUMBER; i++) {
            //Generates random date values
            int randomStreet = rand.nextInt(48);
            int randomNumber = rand.nextInt(125) + 1;
            int randomCity = rand.nextInt(19);
            int randomBrand = rand.nextInt(6);
            int randomDOBYear = rand.nextInt(50);
            int randomDOBMonth = rand.nextInt(12) + 1;
            int randomDOBDay = rand.nextInt(29) + 1;
            int randomRepairYear = rand.nextInt(4);
            int randomRepairMonth = rand.nextInt(12) + 1;
            int randomRepairDay = rand.nextInt(29) + 1;
            int randomEmailProvider = rand.nextInt(5);
            int randomPH0 = rand.nextInt(12);
            int randomPH1 = rand.nextInt(12);
            int randomPH2 = rand.nextInt(12);
            int randomPH3 = rand.nextInt(12);
            int randomFName = rand.nextInt(33);
            int randomLName = rand.nextInt(33);
            FNames.add(FnamesP0[randomFName]);
            LNames.add(LnamesP0[randomLName]);
            //Creates an address to add, also adds a random city and post code
            Streets.add(randomNumber + " " + StreetP0[randomStreet]);
            Citys.add(CityP0[randomCity]);
            PostCodes.add(PostCodeP0[randomCity]);
            //Adds the brand
            Brands.add(BrandP1[randomBrand]);
            StringBuilder DOB = new StringBuilder();
            DOB.append(randomDOBYear + StartYearDOB);
            DOB.append("/");
            DOB.append(randomDOBMonth);
            DOB.append("/");
            DOB.append(randomDOBDay);
            DOBs.add(DOB.toString());
            StringBuilder RepairDate = new StringBuilder();
            int repairYear = randomRepairYear + StartYearRepair;
            RepairDate.append(repairYear);
            RepairDate.append("/");
            RepairDate.append(randomRepairMonth);
            RepairDate.append("/");
            RepairDate.append(randomRepairDay);
            RepairDates.add(RepairDate.toString());
            //Creates a email using the users username and randomEmailProvider
            Emails.add(Users.get(i) + EmailP0[randomEmailProvider]);
            PhoneNums.add(PhoneP0[randomPH0] + PhoneP1[randomPH1] + PhoneP2[randomPH2] + PhoneP3[randomPH3]);
            //Creates the Customer and Staff member data
            //Works out if a employee or customer needs to be created
            if((i % 10) == 0){
              int randomEmployYear = rand.nextInt(18);
              int randomEmployMonth = rand.nextInt(12) + 1;
              int randomEmployDay = rand.nextInt(29) + 1;
              //Creates a random salary between $25 and $40
              int randomSalary = rand.nextInt(15);
              Staff.add(Users.get(i));
              StringBuilder HiringDate = new StringBuilder();
              HiringDate.append(randomEmployYear + StartYearStore);
              HiringDate.append("/");
              HiringDate.append(randomEmployMonth);
              HiringDate.append("/");
              HiringDate.append(randomEmployDay);
```

```
            HiringDates.add(HiringDate.toString());
            HourlySalarys.add(randomSalary + defSal + "");
          }
          else {
            int randomSignupYear = rand.nextInt(13);
            int randomSignupMonth = rand.nextInt(12) + 1;
            int randomSignupDay = rand.nextInt(29) + 1;
            Customer.add(Users.get(i));
            StringBuilder Signup = new StringBuilder();
            Signup.append(randomSignupYear +1995);
            Signup.append("/");
            Signup.append(randomSignupMonth);
            Signup.append("/");
            Signup.append(randomSignupDay);
            SignupDates.add(Signup.toString());
          }
        }
        //Gets the length of the Customers and Staff
        int StaffNumber = Staff.size();
        int CustomerNumber = Customer.size();
        int RepairDatesNumber = RepairDates.size();
        int ModelsNum = Models.size();
        int UsersNum = Users.size();
        System.out.println("");
        for(int d = 0; d < ModelsNum; d++) {
          output.add("INSERT INTO DeviceModel VALUES ('" + Models.get(d) + "', '" + Brands.get(d) + "');");
        }
        System.out.println("");
        for(int u = 0; u < UsersNum; u++) {
          output.add("INSERT INTO UserAccount VALUES ('" + Users.get(u) + "', '" + Passwords.get(u) + "', '" + FNames.get(u) + "', '" + LNames.get(u) + "', '" +
PhoneNums.get(u)
              + "', '" + Emails.get(u) + "', '" + Streets.get(u) + "', '" + Citys.get(u) + "', '" + PostCodes.get(u) + "', to_date('" + DOBs.get(u) + "', 'yyyy/mm/dd'));");
        }
        System.out.println("");
        for(int s = 0; s < StaffNumber; s++) {
          output.add("INSERT INTO Staff VALUES ('" + Staff.get(s) + "', '" + HourlySalarys.get(s) + "', to_date('" + HiringDates.get(s) + "', 'yyyy/mm/dd'));");
        }
        System.out.println("");
        for(int c = 0; c < CustomerNumber; c++) {
          output.add("INSERT INTO Customer VALUES ('" + Customer.get(c) + "', to_date('" + SignupDates.get(c) + "', 'yyyy/mm/dd'));");
        }
        System.out.println("");
        for(int r = 0; r < RepairDatesNumber; r++){
          //Gets a random staff member and customer index for the
          int RandomStaffIndex = rand.nextInt(StaffNumber);
          int RandomCustomerIndex = rand.nextInt(CustomerNumber);
          output.add("INSERT INTO RepairJob VALUES (Repair_seq.NEXTVAL, to_date('" + RepairDates.get(r) + "', 'yyyy/mm/dd'), '" + Customer.get(RandomCustomerIndex)
+ "', '" + Staff.get(RandomStaffIndex)
              + "', '" + Models.get(r) + "');");
        }
        try{
          FileWriter writer = new FileWriter("outDataBaseData.sql", true);
          for (int l = 0; l < output.size(); l++) {
            writer.write(output.get(l) + "\n");
          }
          writer.close();
        }
        catch (Exception e){
          System.out.print("Error encountered: ");
          e.printStackTrace();
        }
      }
    }
  }
 }
}
```

Screenshots of Data Loaded:

```
SELECT Count(U.LoginName) FROM UserAccount U;
SELECT Count(D.ModelNumber) FROM DeviceModel D;
SELECT Count(S.LoginName) FROM Staff S;
SELECT Count(C.LoginName) FROM Customer C;
SELECT Count(R.RepairID) FROM RepairJob R;
```

	COUNT(U.LOGINNAME)
1	235283

	COUNT(D.MODELNUMBER)
1	250010

	COUNT(S.LOGINNAME)
1	23485

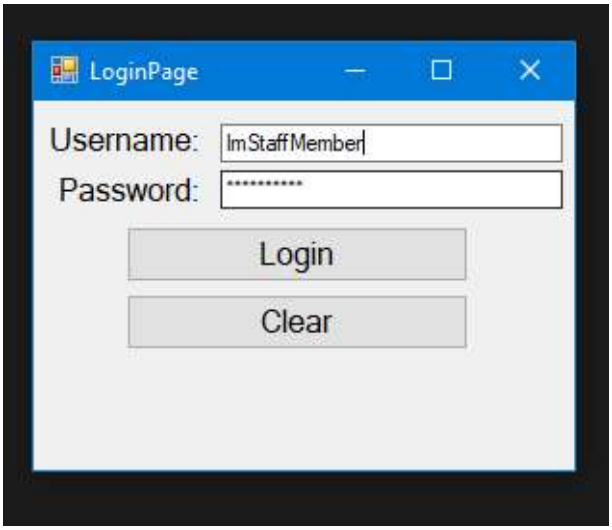
	COUNT(C.LOGINNAME)
1	211314

	COUNT(R.REPA...
1	248470

Task 4: Application:

For the set of tables we chose (UserAccount, DeviceModel, Staff, Customer, RepairJob) we were able to setup an application which allows for a Staff User to login and perform queries to find information or add data to the database. The following screenshots show the functionality and the queries behind the given functionality:

Login Page:



Simple login page for a staff member to access the app, Queries used after pressing the Login Button are:

To determine if a user exists with same login + password:
SQL.selectQuery("SELECT LoginName FROM UserAccount WHERE LoginName = '" + loginname + "' AND Password = '" + password + "'");
To determine if the user is a staff member:
SQL.selectQuery("SELECT LoginName FROM Staff WHERE LoginName = '" + loginname + "'");

Login Page Error checking:

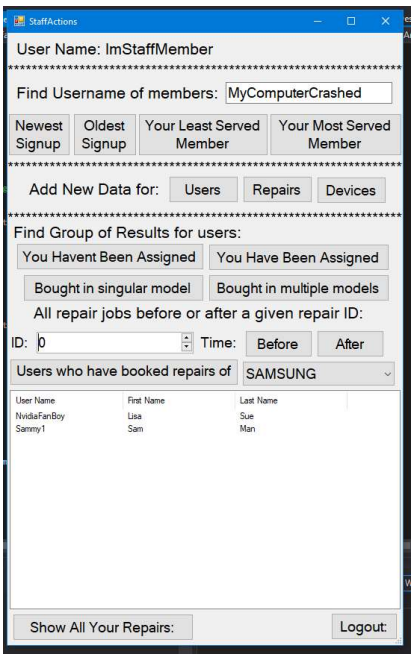
No matching User + Password exists from query:
MessageBox.Show("Error! Either Username or Password are incorrect!");

Matching User + Password exists from query, but user is not a staff member:
MessageBox.Show("Error: You are not a Staff Member, Please goto the Customer Login.");

User does not input values into textbox:
MessageBox.Show("Please make sure you enter a Loginname and Password");

Staff Actions:

Has numerous different buttons and dropdowns for a staff member to make quick and easy queries to to the database, the following queries were linked to the labeled buttons:



Locate Single Member Functions: puts a username in the "Find Usernames of Members" Textbox:
Newest / Oldest Signup:
"SELECT C.LoginName FROM Customer C WHERE C.JoinDate = (SELECT **MAX<For Newest>MIN<For Oldest>**(C2.JoinDate) FROM Customer C2)"
Least / Most Served Member:
"SELECT R.CustomerLogin FROM RepairJob R WHERE R.StaffLogin = '" + loginUser + "' GROUP BY R.CustomerLogin ORDER BY COUNT(R.CustomerLogin) ASC**<For Most>** DESC**<For Least>** FETCH FIRST 1 ROWS ONLY";
Have / Haven't Been Assigned:
"SELECT DISTINCT U.LoginName, U.FName, U.LName FROM UserAccount U, Customer C WHERE U.LoginName = C.LoginName AND C.LoginName NOT IN**<For Haven't>** IN**<For Have>** (SELECT DISTINCT R.customerlogin FROM RepairJob R WHERE R.StaffLogin = '" + loginUser + "')";

Locate Groups of Results: Puts data into the listView at the bottom of the app
Bought in a singular model:
"SELECT DISTINCT U.LoginName, U.FName, U.LName FROM UserAccount U, Customer C, RepairJob R WHERE C.LoginName = R.customerlogin AND U.LoginName = C.LoginName" +" AND EXISTS(SELECT R2.ModelNumber FROM RepairJob R2 WHERE R2.ModelNumber = R.ModelNumber AND R2.RepairID != R.RepairID)";
Bought in a multiple models:
"SELECT DISTINCT U.LoginName, U.FName, U.LName FROM UserAccount U, Customer C, RepairJob R WHERE C.LoginName = R.customerlogin AND U.LoginName = C.LoginName" +
" AND EXISTS(SELECT R2.ModelNumber FROM RepairJob R2 WHERE R2.ModelNumber != R.ModelNumber)";
Before / After a given Repair ID:
"SELECT R.RepairDate, R.CustomerLogin, R.ModelNumber FROM RepairJob R WHERE R.StaffLogin = '" + loginUser + "' AND R.repairdate < **<For Before>** > **<For After>** (SELECT R1.repairdate FROM RepairJob R1 WHERE R1.RepairID = " + ID + ")";
Booked repairs of Brand:

"SELECT DISTINCT U.LoginName, U.FName, U.LName FROM UserAccount U, RepairJob R WHERE R.StaffLogin = '" + loginUser + "' AND U.LoginName = R.CustomerLogin AND R.ModelNumber IN (SELECT D.ModelNumber FROM DeviceModel D WHERE D.Brand = '" + BrandN + "')";

Show all your repairs:

"SELECT R.RepairID, U.LoginName, R.RepairDate, R.ModelNumber FROM UserAccount U, RepairJob R WHERE R.StaffLogin = '" + loginUser + "' AND R.CustomerLogin = U.LoginName";

Add elements to combobox(For performing Booked repairs of Brand):

```
//Performs a query to populate the brand types
string s = "SELECT DISTINCT Brand FROM DeviceModel";
SQL.selectQuery(s);
//WHile there are results, add them to the drop down
while (SQL.dr.Read())
{
    comboBoxBrands.Items.Add(SQL.dr[0]);
}
listViewResults.View = View.Details;
```

Staff Actions Error checking:

Repair ID does not exist:

```
string check = "SELECT R1.repairdate FROM RepairJob R1 WHERE R1.RepairID = " + ID;
SQL.selectQuery(check);
if (SQL.dr.HasRows != true)
{
   MessageBox.Show("Repair ID '" + ID + "' does not exsist!");
   return;
}
```

All other query checks:

```
if (SQL.dr.HasRows) {<Perform Operation>}
else
{
     MessageBox.Show("No Matches Found");
     MessageBox.Show("No Repairs before repair ID: '" + ID + "' exist");
     MessageBox.Show("You do not have any repairs linked to your Username");
   Return;
  }
```

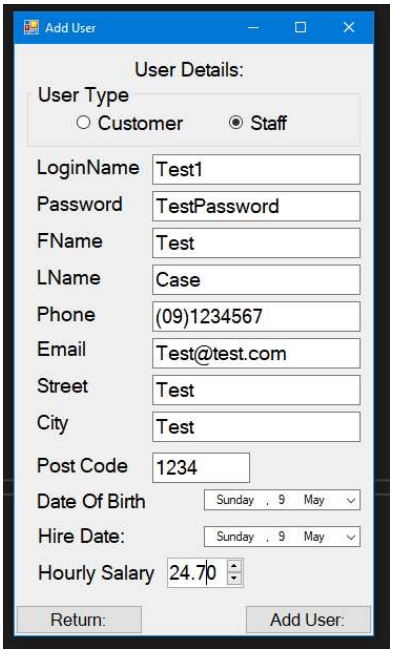
No user input from combo box:

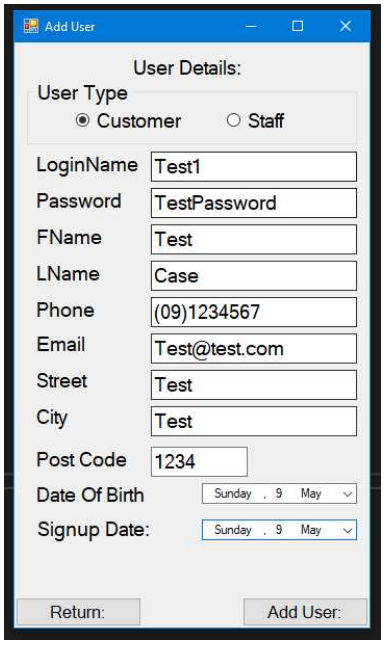
```
string BrandN = comboBoxBrands.Text.Trim();
if (BrandN == "")
{
   MessageBox.Show("Please select a Brand!");
   return;
}
```

No user id entered:

```
string ID = numericUpDownID.Text.Trim();
if(ID == "")
{
   MessageBox.Show("Please enter an ID!");
   return;
}
```

New Data for Staff / Customers:





Allows for a users to be added to the database, reads the data from the LoginName, Password, FName, LName, Phone, Email, Street, City, Postcode, Textboxes as well as the Date of Birth, from the TimeDatePicker. If the user is a Customer the Signup Date will be read from the TimeDatePicker, If the user is a Staff member the Hire Date will be read from the TimeDatePicker and the Hourly Salary will be read from the numericTextBox. Depending on the radio button chosen, the interface will change to show the correct controls required for adding a Customer or Staff member.

Add User:

Performs Query to check that no existing user has same username:

```
SQL.selectQuery("SELECT LoginName FROM UserAccount WHERE LoginName = '" + NUserName + "'");
 if (SQL.dr.HasRows)
 {
    MessageBox.Show("Error: Username already exists in database.");
    return;
 }
```

Performs query to add user to user table:

```
    string AddUserS = "INSERT INTO UserAccount VALUES ('" + NUserName + "', '" + NPassword + "', '" +
NFname + "', '" + NLName + "', '" + NPhonenum + "', '" + NEmail + "', '" + NStreetAddress + "', '" + NCity
    + "', " + NPostcode + ", to_date('" + NDOB.Year + "/" + NDOB.Month + "/" + NDOB.Day + "',
'yyyy/mm/dd'))";
    SQL.executeQuery(AddUserS);
```

Depending on radio button selected, either adds a Customer or Staff member:

```
      string customer = "INSERT INTO Customer VALUES('" + NUserName + "', to_date('" + nSignup.Year +
"/" + nSignup.Month + "/" + nSignup.Day + "', 'yyyy/mm/dd'))";
      SQL.executeQuery(customer);
      string staff = "INSERT INTO Staff VALUES('" + NUserName + "','" + NHourlySalary + "', to_date('" +
nHireDate.Year + "/" + nHireDate.Month + "/" + nHireDate.Day + "', 'yyyy/mm/dd'))";
      SQL.executeQuery(staff);
```

New Data for Staff / Customers Error Checking:

Regex on text changed to prevent invalid characters:

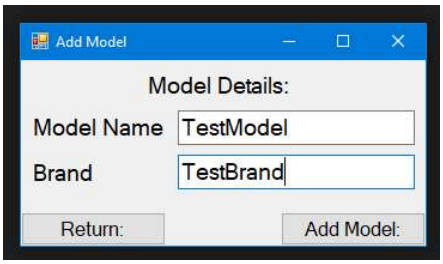
```
private void textBoxName_TextChanged(object sender, EventArgs e)
{
    if (System.Text.RegularExpressions.Regex.IsMatch(textBoxName.Text, "<Regex Pattern>"))
    {
        MessageBox.Show("Please only enter <Regex Types> for <Name>");
        textBoxName.Text = textBoxName.Text.Remove(textBoxName.Text.Length - 1);
        return;
    }
```

Regex for checking Pay, Phone, Email, Street and Postcode before submitting data:

```
string rPay = @"^[0-9]{1,3}\.[0-9]{2}$";
string rPhone = @"^\(\d{2}\)\d{7}\s$";
string rEmail = @"^[A-Za-z0-9]{1,32}\@[A-Za-z0-9]{1,16}.[A-Za-z0-9.]{3,16}$";
string rAddress = @"^[0-9]{1,5} [A-Za-z]{2,32}[ ]{0,1}[A-Za-z]{0,32}$";
string rPostCode = @"^[0-9][0-9][0-9][0-9]$";

    if (System.Text.RegularExpressions.Regex.IsMatch(textBoxModelName.Text, <rRegex>) != true)
    {
        //Lets the user know that only number can be input
        MessageBox.Show("Error: Please enter <Field> correctly in <Format of Data>");
        return;
    }
```

Add Model:



Allows for a model to be added to the database, reads the data from the “Model Name” text box and “Brand” text box.

Add Model:

```
//Checks first that there does not exist a model number with the same value
SQL.selectQuery("SELECT ModelNumber FROM DeviceModel WHERE ModelNumber = '" + NModel + "'");
if (SQL.dr.HasRows)
{
    SQL.dr.Read();
    string exisingModel = SQL.dr[0].ToString();
    MessageBox.Show("'" + exisingModel + "' already exists on the database!");
    return;
}
string AddModel = "INSERT INTO DeviceModel VALUES ('" + NModel + "', '" + NBrand + "')";
SQL.executeQuery(AddModel);
SQL.selectQuery("SELECT ModelNumber FROM DeviceModel WHERE ModelNumber = '" + NModel + "'");
if (SQL.dr.HasRows)
{
    SQL.dr.Read();
    string exisingModel = SQL.dr[0].ToString();
    MessageBox.Show("'" + exisingModel + "' successfully added!");
}
```

Add Model Error Checking:

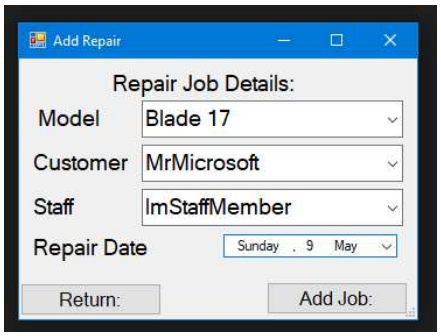
Regex on text changed to prevent invalid characters:

```
private void textBoxModelName_TextChanged(object sender, EventArgs e)
{
    if (System.Text.RegularExpressions.Regex.IsMatch(textBoxModelName.Text, "[^0-9A-Za-z/-]"))
    {
        MessageBox.Show("Please only enter Leters and Numbers with no spaces for  Model Number");
        textBoxModelName.Text = textBoxModelName.Text.Remove(textBoxModelName.Text.Length - 1);
        return;
    }
}
```

Regex for checking Model or Brand before submitting data:

```
string rModel = @"^[A-Za-z0-9/-]{1,32}$";
string rBrand = @"^[A-Za-z]{1,16}$";
```

Add Repair:



Allows for a repairJob to be added to the database, reads the data from the Model, Customer and Staff Drop Down Lists as well as the Repair Date from the TimeDatePicker. To help users to perform queries, drop downs are populated will all Model / Customer / Staff values currently on the DB

Populate drop downs:

```
string Customer = "SELECT LoginName FROM Customer";
string DeviceModel = "SELECT ModelNumber FROM DeviceModel";
string Staff = "SELECT LoginName FROM Staff";
        SQL.selectQuery(<QueryString>);
        //While there are results, add them to the drop down
        if(SQL.dr.HasRows){
            while (SQL.dr.Read())
            {
                comboBox<Type>.Items.Add(SQL.dr[0]);
            }
        }
        else
        {
            MessageBox.Show("Error: No <Type>s on DB");
            return;
        }
```

Add Repair Job:

```
string repair = "INSERT INTO RepairJob VALUES(Repair_seq.NEXTVAL, to_date('" + RepairDate.Year + "/" +
RepairDate.Month + "/" + RepairDate.Day + "', 'yyyy/mm/dd'), '" + CUserName + "', '" + SUserName + "', '" + Model +
"')";
SQL.executeQuery(repair);
```

Add Repair Error Checking

Check that a user has picked a value from the dropdown

```
        string <Type> = comboBox<Type>.Text;
        if (<Type> == "")
        {
            MessageBox.Show("Please select a <Type>");
            return;
        }
```