Whitford Country Club

A Database Project by Mark Miller

Table of Contents:

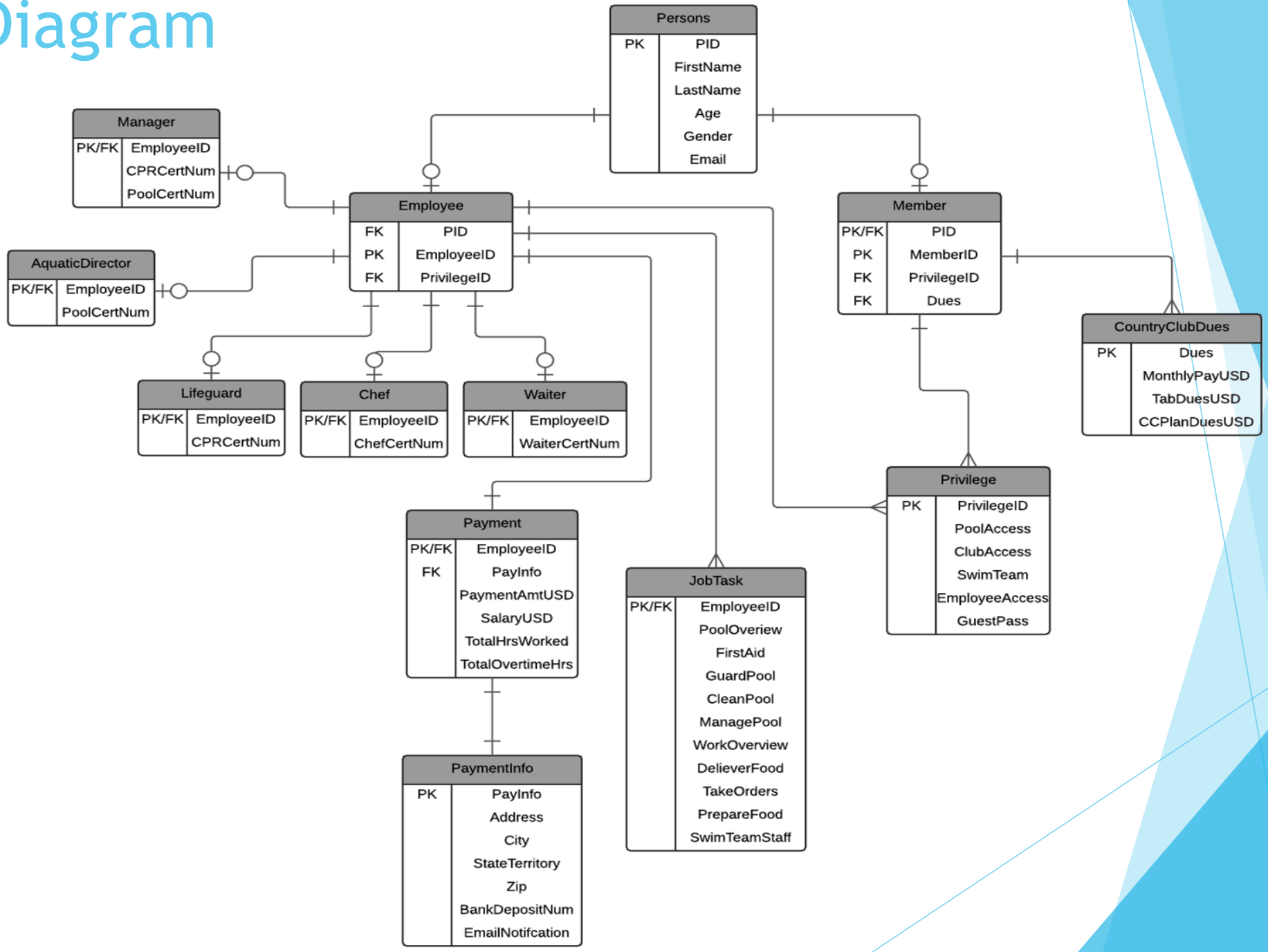
Executive Summary:

Whitford Country Club, located in the small town of Exton, is a pristine Country Club with a prestigious golf course for the area along with a fantastic community. WCC is also home to a terrific staff that runs the outside pool in the summer, yet has had trouble in the past with managing all of its Employees. So I decided to go ahead and make a database proposal for the CC.

The objective of this proposal is first to make sure all information is persevered and never lost. They had an incident in which some important data was lost and unable to be recovered. Secondly, to organize the data so it is easier to view and understand. Finally, the last was to all around create a better more organized database for the Whitford Country Club.

The Database is currently a work in progress but is a general idea and layout that is meant to server as a proposal for a new Database system. As we move forward we will continue to add, update, and revise the Database in order to make it the best possible solution for the problems that Whitford Country Club faces.

ER-Diagram


Persons
PK PID
FirstName
LastName
Age
Gender
Email
Manager
PK/FK EmployeeID
CPRCertNum
PoolCertNum
Employee
FK PID
PK EmployeeID
FK PrivilegeID
Member
PK/FK PID
PK MemberID
FK PrivilegeID
FK Dues
AquaticDirector
PK/FK EmployeeID
PoolCertNum
CountryClubDues
PK Dues
MonthlyPayUSD
TabDuesUSD
CCPlanDuesUSD
Lifeguard
PK/FK EmployeeID
CPRCertNum
Chef
PK/FK EmployeeID
ChefCertNum
Waiter
PK/FK EmployeeID
WaiterCertNum
Privilege
PK PrivilegeID
PoolAccess
ClubAccess
SwimTeam
EmployeeAccess
GuestPass
Payment
PK/FK EmployeeID
FK PayInfo
PaymentAmtUSD
SalaryUSD
TotalHrsWorked
TotalOvertimeHrs
JobTask
PK/FK EmployeeID
PoolOveriew
FirstAid
GuardPool
CleanPool
ManagePool
WorkOverview
DelieverFood
TakeOrders
PrepareFood
SwimTeamStaff
PaymentInfo
PK PayInfo
Address
City
StateTerritory
Zip
BankDepositNum
EmailNotifcation

Create Table Statements:

Persons Table

- This is the most important table as it is the beginning in which we distinguish every person with a PID, name and important information about themselves. The PID is the primary key as it acts as the beginning to distinguishing a member from an employee.

```
CREATE TABLE Persons(
    PID int not null unique,
    FirstName varchar not null,
    LastName varchar not null,
    Age int not null,
    Gender char(6) not null,
    Email varchar not null,
    unique (PID),
    primary key (PID)
);
```

Functional Dependencies:
PID → FirstName, LastName, Age, Gender, Email

Sample data:

pid integer	firstname characte...	lastname characte...	age integer	gender character	email characte...
1	Mark	Miller	19	Male	markmille...
2	Chris	Kolimingo	18	Male	chriskol1...
3	Mike	Love	18	Male	mikelove...
4	Joey	Baldja	19	Male	joeybaldj...
5	Stephen	Miller	26	Male	stephen...
6	Kerry	Schubert	24	Male	kerrysch...
7	Tori	Flaherty	19	Female	tori.flahe...
8	Sara	Schubert	26	Female	SaraSchu...
9	Kelsey	Smith	18	Female	kelseysm...
10	Alan	Labouseur	40	Male	www.3N...

Create Table Statements:
Employee Table

- The Employee Table places a crucial role as it links many of the other tables that are associated with an employee. We also define and store the EmployeeID into this table. This table references the Persons table through the PID and uses the EmployeeID as its primary key. There is a decision node placed on the employee side because of the choice between a employee or member.

```
CREATE TABLE Employee(
    PID int not null references Persons(PID),
    EmployeeID varchar not null unique,
    PrivilegeID int not null references Privilege(PrivilegeID),
    unique (EmployeeID),
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → PID, PrivilegeID

Sample data:

pid integer	employe... characte...	privilegeid integer
1	1000	100
2	1001	101
3	1002	102
4	1003	103
5	1004	104
6	1005	105
7	1006	106

Create Table Statements:
Member Table

- Another crucial table as it links all member related table to it by a MemberID. The MemberID servers as the primary key and references the Persons table by the PID. Again we place a decision node on the member side because you can either be a member or an employee.

```
CREATE TABLE Member(
    PID int not null references Persons(PID),
    MemberID Serial not null unique,
    Dues varchar not null references CountryClubDues(Dues),
    PrivilegeID int not null references Privilege(PrivilegeID),
    primary key (MemberID)
);
```

Functional Dependencies:
MemberID → PID, Dues, PrivilegeID

Sample data:

pid integer	memberid integer	dues characte...	privilegeid integer
12	1	d0001	111
23	2	d0002	122
25	3	d0003	124
26	4	d0004	125

Create Table Statements:

AquaticDirector Table

- This is one of many jobs that are associated with being an Employee. EmployeeID is again the primary key and references the Employee Table. The AqauticDirector also has a Pool Certificate Number associated with him. A decision node is placed on this side as it is a job decision.

```
CREATE TABLE AquaticDirector(
    EmployeeID varchar not null references Employee(EmployeeID),
    PoolCertNum int not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → PoolCertNum

Sample data:

employe... character...	poolcert... integer
1015	123456789
1020	190123456

Create Table Statements:
Lifeguard Table

- Another job associated with being an Employee and follows the same principles as the AquaticDirector table. It references EmployeeID and has an CPR Certificate Number associated with it too. We again place a decision node on this side of the table to show the choice of being a lifeguard.

```
CREATE TABLE Lifeguard(
    EmployeeID varchar not null references Employee(EmployeeID),
    CPRCertNum int not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → CPRCertNum

Sample data:

employe... character...	cprcertn... integer
1000	234567890
1001	345678901
1002	456789012
1003	567890123
1008	678901234
1012	789012345

Create Table Statements:
Chef Table

- Following the other job tables, this table sets the primary key as the EmployeeID and references the Employee table with it. The Chef table also has a Chef Certificate Number within it. The decision node is used on this side of the table to show the choice of being a chef.

```
CREATE TABLE Chef(
    EmployeeID varchar not null references Employee(EmployeeID),
    ChefCertNum int not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → ChefCertNum

Sample data:

employe... character...	chefcert... integer
1004	123000000
1005	124000000
1007	125000000
1016	126000000
1017	127000000
1019	128000000

Create Table Statements:
Waiter Table

- The waiter table follows all other job tables in that we set the EmployeeID as the primary key and reference the Employee table to get it. The Waiter table also as a Waiter Certificate number corresponding with it. A decision node is used to show a decision is made on choosing this job.

```
CREATE TABLE Waiter(
    EmployeeID varchar not null references Employee(EmployeeID),
    WaiterCertnum int not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → WaiterCertNum

Sample data:

employe... character...	waiterce... integer
1006	234000000
1013	235000000
1014	236000000
1018	237000000
1021	238000000

Create Table Statements:
Manager Table

- The final job table is the Manager table. This again sets the EmployeeID to be the primary key and references the Employee table to get it. It then also has both a CPR and Pool Certificate Number in its table. Like the rest a decision node is used to represent the choice between a manager or any other job.

```
CREATE TABLE Manager(
    EmployeeID varchar not null references Employee(EmployeeID),
    CPRCertNum int not null,
    PoolCertNum int not null,
    primary key (MemberID)
);
```

Functional Dependencies:
EmployeeID → PoolCertNum, CPRCertNum

Sample data:

employe... character...	cprcertn... integer	poolcert... integer
1009	444444444	222222222
1010	555555555	111111111
1023	666666666	333333333

Create Table Statements:

Privilege Table

- The privilege table represents the privileges that each employee or member has access to. The PrivilegeID is stored within this table and is referenced in both the employee and member table. The PrivilegeID is used as the primary key in this table.

```
CREATE TABLE Privilege(
    PrivilegeID int not null unique,
    PoolAccess boolean not null,
    ClubAccess boolean not null,
    SwimTeam boolean not null,
    EmployeeAccess boolean not null,
    GuestPass boolean not null,
    unique (PrivilegeID),
    primary key (PrivilegeID)
);
```

Functional Dependencies:
PrivilegeID → PoolAccess, ClubAccess, SwimTeam, EmployeeAccess, GuestPass

Sample data:

privilegeid integer	poolacce... boolean	clubaccess boolean	swimteam boolean	employe... boolean	guestpass boolean
100	true	false	true	true	false
101	true	false	true	true	false
102	true	false	true	true	false
103	true	false	true	true	false
104	false	true	false	true	false
105	false	true	false	true	false
106	true	true	false	true	false
107	false	true	false	true	false
108	true	false	true	true	false
109	true	true	true	true	true
110	true	true	true	true	true
111	true	true	true	false	true
112	true	false	true	true	false
113	true	true	false	true	false

Create Table Statements:

JobTask Table

- This table is very similar to the Privilege table in that it defines all of the job task assigned to a specific person. We use the EmployeeID as the primary key and references the Employee table once again.

```
CREATE TABLE JobTask(
    EmployeeID varchar not null references Employee(EmployeeID),
    PoolOverview boolean not null,
    FirstAid boolean not null,
    GuardPool boolean not null,
    CleanPool boolean not null,
    ManagePool boolean not null,
    WorkOverview boolean not null,
    DelieverFood boolean not null,
    TakeOrders boolean not null,
    PrepareFood boolean not null,
    SwimTeamStaff boolean not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → PoolOverview, FirstAid, GuardPool, CleanPool, ManagePool, WorkOverview, DelieverFood, TakeOrders, SwimTeamStaff

Sample data:

employe... character...	poolover... boolean	firstaid boolean	guardpool boolean	cleanpool boolean	manage... boolean	workove... boolean	delieverf... boolean	takeord... boolean	preparef... boolean	swimtea... boolean
1000	false	true	true	true	false	false	false	false	false	false
1001	false	true	true	true	false	false	false	false	false	false
1002	false	true	true	true	false	false	false	false	false	false
1003	false	true	true	true	false	false	false	false	false	false
1004	false	false	false	false	false	false	false	false	true	false

Create Table Statements:
CountryClubDues Table

- This table signifies all of the payments that a member must make to the Country Club. We use Dues as the primary key and store it within this table. Dues is referenced by the Member table in order to gain access.

```
CREATE TABLE CountryClubDues(
    Dues varchar not null unique,
    MonthlyPayUSD int not null,
    TabDuesUSD int not null,
    CCPlanDuesUSD int not null,
    unique (Dues),
    primary key (Dues)
);
```

Functional Dependencies:
Dues → MonthlyPay, TabDues, CCPlanDues

Sample data:

dues character...	monthly... integer	tabdues... integer	ccplandu... integer
d0001	300	100	200
d0002	500	50	200
d0003	250	100	200
d0004	300	150	200

Create Table Statements:
Payment Table

- This table uses a EmployeeID once again as its primary id and describes all of the factors that go into your payments and the total payment that you get.

```
CREATE TABLE Payment(
    EmployeeID varchar not null references Employee(EmployeeID),
    PayInfo char(50) not null references PaymentInfo(PayInfo),
    PaymentAmtUSD int not null,
    SalaryUSD int not null,
    TotalHrsWorked int not null,
    TotalOvertimeHrs int not null,
    primary key (EmployeeID)
);
```

Functional Dependencies:
EmployeeID → PAyInfo, PaymentAmt, Salary, TotalHrsWorked, TotalOvertimeHrs

Sample data:

employe... character...	payinfo character	payment... integer	salaryusd integer	totalhrs... integer	totalove... integer
1000	Mark-Mill...	5454	9	600	6
1001	Chris-Koli...	2700	9	300	0
1002	Mike-Lov...	2520	9	280	0
1003	Joey-Bal...	3240	9	360	0
1004	Stephen-...	2320	8	290	0
1005	Kerry-Sc...	1680	8	210	0
1006	Tori-Flah...	1200	8	150	0
1007	Sara-Sch...	3392	8	410	14
1008	Kelsey-S...	440	8	55	0
1009	Alan-Lab...	1000000	1000000	1	0

Create Table Statements:
PaymentInfo Table

- The final table in our database is the PaymentInfo table which specifies all of the needed information when a employee is being paid. The primary key is the PayInfo which is unqiue for every person.

```
CREATE TABLE PaymentInfo(
    PayInfo char(50) not null unique,
    Address char(25) not null,
    City char(25) not null,
    StateTerritory char(50) not null,
    zip int not null,
    BankDepositNum int not null,
    EmailNotification char(50) not null,
    primary key (PayInfo)
);
```

Functional Dependencies:
PayInfo → Address, City, StateTerritory, zip, BankDepositNum, EmailNotification

Sample data:

payinfo character	address character	city character	stateterr... character	zip integer	bankdep... integer	emailnot... character
Mark-Mill...	37 Hawk ...	Poughke...	NY	12601	11011	markmille...
Chris-Koli...	2 Rando...	Downing...	PA	19335	11012	chriskol1...
Mike-Lov...	3 Rando...	Downing...	PA	19335	11013	mikelove...
Joey-Bal...	4 Rando...	Downing...	PA	19335	11014	joeybaldj...
Stephen-...	5 Nowhe...	Poughke...	NY	12601	11015	stephen...
Kerry-Sc...	6 Nowhe...	Poughke...	NY	12601	11016	kerrysch...

LifeGuardInfo View

This View displays all employees who have the same privileges and job tasks that fits the Lifeguard description. It identifies them by their name and lists there EmployeeID in ascending order.

```
CREATE VIEW LifeGuardInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           JobTask.GuardPool,
           JobTask.CleanPool,
           JobTask.FirstAid,
           Privilege.PoolAccess,
           Privilege.SwimTeam,
           Privilege.EmployeeAccess
    from  Persons, JobTask, Privilege, Employee
    where Employee.EmployeeID = JobTask.EmployeeID
      and Employee.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Employee.PID
      and JobTask.GuardPool = true
      and JobTask.CleanPool = true
      and JobTask.FirstAid = true
      and Privilege.PoolAccess = true
      and Privilege.SwimTeam = true
      and Privilege.EmployeeAccess = true
      order by EmployeeID;
```

Sample data:

firstname characte...	lastname characte...	employe... characte...	guardpool boolean	cleanpool boolean	firstaid boolean	poolacce... boolean	swimteam boolean	employe... boolean
Mark	Miller	1000	true	true	true	true	true	true
Chris	Kolimingo	1001	true	true	true	true	true	true
Mike	Love	1002	true	true	true	true	true	true
Joey	Baldja	1003	true	true	true	true	true	true
Kelsey	Smith	1008	true	true	true	true	true	true
Alan	Labouseur	1009	true	true	true	true	true	true
Lauren	Kennen	1010	true	true	true	true	true	true
Erica	Marie	1012	true	true	true	true	true	true
Matt	Lake	1023	true	true	true	true	true	true

AquaticDirectorInfo View

This View displays all employees who have the same privileges and job tasks that fits the Aquatic Director description. It identifies them by their name and lists there EmployeeID in ascending order.

```
CREATE VIEW AquaticDirectorInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           JobTask.PoolOverview,
           JobTask.SwimTeamStaff,
           JobTask.WorkOverview,
           Privilege.PoolAccess,
           Privilege.ClubAccess,
           Privilege.EmployeeAccess
    from  JobTask, Privilege, Employee, Persons
    where Employee.EmployeeID = JobTask.EmployeeID
      and Employee.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Employee.PID
      and JobTask.PoolOverview = true
      and JobTask.SwimTeamStaff = true
      and JobTask.WorkOverview = true
      and Privilege.PoolAccess = true
      and Privilege.ClubAccess = true
      and Privilege.EmployeeAccess = true
      order by EmployeeID;
```

Sample data:

firstname charact...	lastname charact...	employe... charact...	poolover... boolean	swimtea... boolean	workove... boolean	poolacce... boolean	clubaccess boolean	employe... boolean
Alan	Labouseur	1009	true	true	true	true	true	true
Lauren	Kennen	1010	true	true	true	true	true	true
Matt	Lake	1023	true	true	true	true	true	true

ChefInfo View

This View displays all employees who have the same privileges and job tasks that fits the Chef description. It identifies them by their name and lists there EmployeeID in ascending order.

```
CREATE VIEW ChefInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           JobTask.PrepareFood,
           Privilege.EmployeeAccess,
           Privilege.ClubAccess
    from  JobTask, Privilege, Employee, Persons
    where Employee.EmployeeID = JobTask.EmployeeID
      and Employee.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Employee.PID
      and JobTask.PrepareFood = true
      and Privilege.EmployeeAccess = true
      and Privilege.ClubAccess = true
      order by EmployeeID asc;
```

Sample data:

firstname character...	lastname character...	employe... character...	preparef... boolean	employe... boolean	clubaccess boolean
Stephen	Miller	1004	true	true	true
Kerry	Schubert	1005	true	true	true
Sara	Schubert	1007	true	true	true
Alan	Labouseur	1009	true	true	true
Lauren	Kennen	1010	true	true	true
Matthew	Read	1016	true	true	true
Kate	Kolosecke	1017	true	true	true
Andrew	Austria	1019	true	true	true
Matt	Lake	1023	true	true	true

WaiterInfo View

This View displays all employees who have the same privileges and job tasks that fits the Waiter description. It identifies them by their name and lists there EmployeeID in ascending order.

```
CREATE VIEW WaiterInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           JobTask.TakeOrders,
           JobTask.DelieverFood,
           Privilege.EmployeeAccess,
           Privilege.ClubAccess
    from JobTask, Privilege, Employee, Persons
    where Employee.EmployeeID = JobTask.EmployeeID
      and Employee.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Employee.PID
      and JobTask.TakeOrders = true
      and JobTask.DelieverFood = true
      and Privilege.EmployeeAccess = true
      and Privilege.ClubAccess = true
      order by EmployeeID asc;
```

Sample data:

firstname character...	lastname character...	employe... character...	takeord... boolean	delieverf... boolean	employe... boolean	clubaccess boolean
Tori	Flaherty	1006	true	true	true	true
Alan	Labouseur	1009	true	true	true	true
Lauren	Kennen	1010	true	true	true	true
Minion	Liengtira...	1013	true	true	true	true
Mason	Murphy	1014	true	true	true	true
Andrew	Gorr	1018	true	true	true	true
Marc	Christen...	1021	true	true	true	true
Matt	Lake	1023	true	true	true	true

ManagerInfo View

This View displays all employees who have the same privileges and job tasks that fits the Manager description. It identifies them by their name and lists there EmployeeID in ascending order.

```
CREATE VIEW ManagerInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           JobTask.PoolOverview,
           JobTask.FirstAid,
           JobTask.GuardPool,
           JobTask.CleanPool,
           JobTask.ManagePool,
           JobTask.WorkOverview,
           JobTask.SwimTeamStaff,
           Privilege.PoolAccess,
           Privilege.ClubAccess,
           Privilege.SwimTeam,
           Privilege.EmployeeAccess
    from JobTask, Privilege, Employee, Persons
    where Employee.EmployeeID = JobTask.EmployeeID
      and Employee.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Employee.PID
      and  JobTask.PoolOverview = true
      and  JobTask.FirstAid = true
      and  JobTask.GuardPool = true
      and JobTask.CleanPool = true
      and JobTask.ManagePool = true
      and JobTask.WorkOverview = true
      and JobTask.SwimTeamStaff = true
      and Privilege.PoolAccess = true
      and Privilege.ClubAccess = true
      and  Privilege.SwimTeam = true
    order by EmployeeID asc;
```

Sample data:

firstname characte...	lastname characte...	employe... characte...	poolover... boolean	firstaid boolean	guardpool boolean	cleanpool boolean	manage... boolean	workove... boolean	swimtea... boolean	poolacce... boolean	clubaccess boolean	swimteam boolean	employe... boolean
Alan	Labouseur	1009	true	true	true	true	true	true	true	true	true	true	true
Lauren	Kennen	1010	true	true	true	true	true	true	true	true	true	true	true
Matt	Lake	1023	true	true	true	true	true	true	true	true	true	true	true

CCMemberInfo View

This view displays all of the current Country Club Members and identifies that they all have access to the specified Member privileges.

```
CREATE VIEW CCMemberInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Member.MemberID,
           Privilege.PoolAccess,
           Privilege.ClubAccess,
           Privilege.SwimTeam,
           Privilege.GuestPass
    from Member, Privilege, Persons
    where Member.PrivilegeID = Privilege.PrivilegeID
      and Persons.PID = Member.PID
    order by MemberID desc;
```

Sample data:

firstname characte...	lastname characte...	memberid integer	poolacce... boolean	clubaccess boolean	swimteam boolean	guestpass boolean
Jack	Ryan	4	true	true	true	true
Andrew	Arrigo	3	true	true	true	true
Rob	Lynch	2	true	true	true	true
Conor	Tunno	1	true	true	true	true

PaymentDeterminate View

In this View we are listing all of there names and employee numbers in a descending order. Then were are display the main components that going into factoring the persons total pay (PaymentAmt). These are the determinates used to find the total pay for that employee.

```
CREATE VIEW PaymentDeterminate AS
    select Persons.FirstName,
           Persons.LastName,
           Employee.EmployeeID,
           Payment.SalaryUSD,
           Payment.TotalOvertimeHrs,
           Payment.TotalHrsWorked
    from  Employee, Payment, Persons
    where Employee.EmployeeID = Payment.EmployeeID
      and Persons.PID = Member.PID
    order by EmployeeID desc;
```

Sample data:

firstname characte...	lastname characte...	employe... characte...	salaryusd integer	totalove... integer	totalhrs... integer
Matt	Lake	1023	8	0	200
Marc	Christen...	1021	6	0	380
Brian	Brucker	1020	6	0	400
Andrew	Austria	1019	7	0	200
Andrew	Gorr	1018	7	0	120

CCMemberDues View

This view does the same as the PayDeterminate Table, but instead it list all of the members. It then displays their MemberID and then displays each of the payments that add up to there total Dues.

```
CREATE VIEW CCMemberDuesInfo AS
    select Persons.FirstName,
           Persons.LastName,
           Member.MemberID,
           CountryClubDues.MonthlyPayUSD,
           CountryClubDues.TabDuesUSD,
           CountryClubDues.CCPlanDuesUSD
    from Member, CountryClubDues, Persons
    where Member.Dues = CountryClubDues.Dues
      and Persons.PID = Member.PID
    order by MemberID asc;
```

Sample data:

firstname characte...	lastname characte...	memberid integer	monthly... integer	tabdues... integer	ccplandu... integer
Conor	Tunno	1	300	100	200
Rob	Lynch	2	500	50	200
Andrew	Arrigo	3	250	100	200
Jack	Ryan	4	300	150	200

getPersonInfoByName Stored Procedure

This stored procedure takes a first name and last name and searches through the Persons table and finds all matches. When it finds this match it list the persons PID, first name, last name age, email, and gender.

```
CREATE OR REPLACE FUNCTION getPersonInfoByName(TEXT, TEXT, REFCURSOR) RETURNS refcursor AS
$$
declare
    PersonFName             Text        := $1;
    PersonLName             Text        := $2;
    ResultSet          refcursor        := $3;
begin
    open ResultSet for
        select *
        from Persons
        where FirstName like    PersonFName
        and   LastName  like    PersonLName;
    return ResultSet;
end;
$$
language plpgsql;
```

Sample data:

```
select getPersonInfoByName('%', 'Mil%','ref');
fetch all from ref;
```

pid integer	firstname characte...	lastname characte...	age integer	gender character	email characte...
1	Mark	Miller	19	Male	markmille...
5	Stephen	Miller	26	Male	stephen...

getEmployeeInfo Stored Procedure

This stored procedure takes a first name and last name and searches through the Employee table and finds all matches. When it finds this match it list the Employees information.

```
CREATE OR REPLACE FUNCTION getEmployeeInfo(TEXT, TEXT, REFCURSOR) RETURNS refcursor AS
$$
declare
    PersonFName             Text        := $1;
    PersonLName             Text        := $2;
    ResultSet           refcursor       := $3;
begin
    open ResultSet for
        select
                Persons.FirstName,
                Persons.LastName,
                Persons.Email,
                Persons.Gender,
                Persons.Age,
                Employee.PrivilegeID,
                Employee.EmployeeID,
                Payment.PayInfo,
                Payment.PaymentAmtUSD,
                Payment.SalaryUSD,
                Payment.TotalHrsWorked,
                Payment.TotalOvertimeHrs
        from  Persons, Employee, Payment
        where Persons.PID = Employee.PID
        and   Employee.EmployeeID = Payment.EmployeeID
        and   FirstName like    PersonFName
        and   LastName  like    PersonLName;
    return ResultSet;
end;
$$
language plpgsql;
```

Sample data:

```
select getEmployeeInfo('Al%', 'L%','ref');
fetch all from ref;
```

firstname characte...	lastname characte...	email characte...	gender character	age integer	privilegeid integer	employe... characte...	payinfo character	payment... integer	salaryusd integer	totalhrs... integer	totalove... integer
Alan	Labouseur	www.3N...	Male	40	109	1009	Alan-Lab...	1000000	1000000	1	0

getMemberInfo Stored Procedure

This stored procedure takes a first name and last name and searches through the Member table and finds all matches. When it finds this match it list the Members information.

```
CREATE OR REPLACE FUNCTION getMemberInfo(TEXT, TEXT, REFCURSOR) RETURNS refcursor AS
$$
declare
    PersonFName             Text        := $1;
    PersonLName             Text        := $2;
    ResultSet           refcursor       := $3;
begin
    open ResultSet for
        select
                Persons.FirstName,
                Persons.LastName,
                Persons.Email,
                Persons.Gender,
                Persons.Age,
                Member.PrivilegeID,
                Member.MemberID,
                Member.Dues,
                CountryClubDues.MonthlyPayUSD,
                CountryClubDues.TabDuesUSD,
                CountryClubDues.CCPlanDuesUSD
        from  Persons, Member, CountryClubDues
        where Persons.PID = Member.PID
        and   Member.Dues = CountryClubDues.Dues
        and   FirstName like    PersonFName
        and   LastName  like    PersonLName;
    return ResultSet;
end;
$$
language plpgsql;
```

Sample data:

```
select getMemberInfo('R%', 'L%', 'ref');
fetch all from ref;
```

firstname characte...	lastname characte...	email characte...	gender character	age integer	privilegeid integer	memberid integer	dues characte...	monthly... integer	tabdues... integer	ccplandu... integer
Rob	Lynch	rob.lynch...	Male	20	122	2	d0002	500	50	200

Report:
Employee Payment Info

This Report returns all the employees that we currently have and returns there Payment Information (PaymentInfo) to us so we have easy access to it.

```
select Employee.EmployeeID,
       Payment.PaymentAmtUSD,
       PaymentInfo.PayInfo,
       PaymentInfo.Address,
       PaymentInfo.City,
       PaymentInfo.StateTerritory,
       PaymentInfo.Zip,
       PaymentInfo.BankDepositNum,
       PaymentInfo.EmailNotification
from Employee, Payment, PaymentInfo
where Employee.EmployeeID = Payment.EmployeeID
and   Payment.PayInfo = PaymentInfo.PayInfo
order by Employee.EmployeeID;
```

Sample data:

employe... character...	payment... integer	payinfo character	address character	city character	stateterr... character	zip integer	bankdep... integer	emailnot... character
1000	5454	Mark-Mill...	37 Hawk ...	Poughke...	NY	12601	11011	markmille...
1001	2700	Chris-Koli...	2 Rando...	Downing...	PA	19335	11012	chriskol1...
1002	2520	Mike-Lov...	3 Rando...	Downing...	PA	19335	11013	mikelove...
1003	3240	Joey-Bal...	4 Rando...	Downing...	PA	19335	11014	joeybaldj...
1004	2320	Stephen-...	5 Nowhe...	Poughke...	NY	12601	11015	stephen...

Report:
Member Dues Info

This Report returns all the Members that we currently have and returns there current dues they owe (CountryClubDues) to us so we have easy access to it.

```
select Persons.PID
       Persons.FirstName,
       Perons.LastName,
       Member.MemberID,
       Member.Dues,
       CountryClubDues. MonthlyPayUSD,
       CountryClubDues.TabDuesUSD,
       CountryClubDues.CCPlanDuesUSD
from Member,  CountryClubDues
where Member.Dues =  CountryClubDues.Dues
  and Persons.PID = Member.PID
order by MemberID;
```

Sample data:

pid integer	firstname characte...	lastname characte...	memberid integer	dues characte...	monthly... integer	tabdues... integer	ccplandu... integer
12	Conor	Tunno	1	d0001	300	100	200
23	Rob	Lynch	2	d0002	500	50	200
25	Andrew	Arrigo	3	d0003	250	100	200
26	Jack	Ryan	4	d0004	300	150	200

Trigger Functions

These are the functions that allow the trigger to work. They check all tables to see if the EmployeeID is already in it. If it is it then returns true otherwise returns false.

```
create function insertIntoAquaticDirector(varchar) returns boolean AS
$$
declare
    EmployeeInput text :=$1;
begin
    if exists (select EmployeeID from Lifeguard where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Chef where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Waiter where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Manager where EmployeeID = EmployeeInput) then
        return true;
    else
        return false;
    end if;
end;
$$
language plpgsql;

create function insertIntoManager(varchar) returns boolean AS
$$
declare
    EmployeeInput text :=$1;
begin
    if exists (select EmployeeID from AquaticDirector where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Lifeguard where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Chef where EmployeeID = EmployeeInput) then
        return true;
    elsif exists (select EmployeeID from Waiter where EmployeeID = EmployeeInput) then
        return true;
    else
        return false;
    end if;
end;
$$
language plpgsql;
```

Trigger Function

This function takes the previous trigger functions we created for this and use it in this trigger so that whenever we try to add into a job table it will automatically check to see if the Employee is already in another table or not.

```
CREATE OR REPLACE FUNCTION newEmployee() RETURNS TRIGGER AS
$$
declare
    test1 boolean = insertIntoAquaticDirector(new.EmployeeID);
    test2 boolean = InsertIntoLifeguard(new.EmployeeID);
    test3 boolean = InsertIntoChef(new.EmployeeID);
    test4 boolean = insertIntoWaiter(new.EmployeeID);
    test5 boolean = insertIntoManager(new.EmployeeID);
begin
    if (test1 = true) then
        raise exception 'This Employee is already in another table';
    elsif (test2 = true) then
        raise exception 'This Employee is already in another table';
    elsif (test3 = true) then
        raise exception 'This Employee is already in another table';
    elsif (test4 = true) then
        raise exception 'This Employee is already in another table';
    elsif (test5 = true) then
        raise exception 'This Employee is already in another table';
    else
        return new;
    end if;
end;
$$
language plpgsql;
```

Inserts for the Trigger and Sample Data

These are the inserts for the trigger so that the trigger will work properly.

```
create trigger newEmployee
before insert on AquaticDirector
for each row
execute procedure newEmployee();

create trigger newEmployee
before insert on Lifeguard
for each row
execute procedure newEmployee();

create trigger newEmployee
before insert on Chef
for each row
execute procedure newEmployee();

create trigger newEmployee
before insert on Waiter
for each row
execute procedure newEmployee();

create trigger newEmployee
before insert on Manager
for each row
execute procedure newEmployee();
```

Sample Data For Trigger:

```
insert into AquaticDirector(EmployeeID, PoolCertNum) values
    ('1015','123456789');
```

```
ERROR:  This Employee is already in another table
CONTEXT:  PL/pgSQL function newemployee() line 12 at RAISE
********** Error **********

ERROR: This Employee is already in another table
SQL state: P0001
Context: PL/pgSQL function newemployee() line 12 at RAISE
```

Security:

The Whitford Vice president is someone appointed by the President, usually a member and has some access. He is only allowed to access, insert, and update on people and the jobs that they have.

```
CREATE ROLE Whitford_Administration;
GRANT SELECT, INSERT, UPDATE
ON ALL TABLES IN SCHEMA PUBLIC
TO Whitford_Administration;

CREATE ROLE Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Persons
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Member
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Employee
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
On Lifeguard
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Waiter
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Chef
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Waiter
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON AquaticDirector
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON Manager
TO Whitford_Vice_President;
GRANT SELECT, INSERT, UPDATE
ON JobTask
TO Whitford_Vice_President;
```

The Whitford Administration is a big help to the boss himself. They have access to all table but again they are only allowed to select, insert and update the tables. They do not have the the delete function.

```
CREATE ROLE Whitford_Administration;
GRANT SELECT, INSERT, UPDATE
ON ALL TABLES IN SCHEMA PUBLIC
TO Whitford_Administration;
```

The final security given is to the President or the Big Man on duty. He has access to all tables and all functions. He is the only one allowed to have access to the delete function in order to keep everything in tact.

```
CREATE ROLE Whitford_President;
GRANT SELECT, UPDATE, INSERT, DELETE
ON ALL TABLES IN SCHEMA PUBLIC
TO Whitford_President;
```

Implementation Notes

- Purpose of the Database:
  - The purpose of this database is to make the organization of data easier for the Whitford Country Club. Their prior means of keeping and organizing data was about as productive as a communication major is at Mairst. After working their it really girnd my gears and so I devised a better way to keep the data organized and in tact.
- Origination of Test Data:
  - All test data used in this data base was for the most part made up in order to provide a quick and simple way to present this to the Whitford board. It was also simple in order to satisfy the people that would find this otherwise harder to understand. Everyone named within the database is a real person, but any sort of pay, address and ages was made up to complete that data.

Known Issues:

- The database assumes that every member has the same plan and therefore will be paying the same, but in reality there are many types of plans that the Country Club offers.
- There is a lot of IDs involved and it can make it hard to read and understand at times
- This is data that is only given for the pool staff as there is a golf course on location that has its own staff which is not included in the database.
- Was unable to find the best way to represent dues and payments within the database.
- I am only a 19 year old college kid who is currently getting plowed by so much work that I guarantee that I have missed a crap load of important data. Hopefully no one notices and if so I quit.

Future Enhancements

- Find a way to easily connect everything without having a ton of IDs used
- Add all employee staff into the database
- Supply data that is correct and would make sense
- Find a better way to represent pay in the tables
- Make sure that I have all the important data that I need that will best suit the requirements for the Country Club