DBDS DataBase Design Specialists, Inc.

"We are always 5NF."

AAA Rentals – AAA Project

Database and Final Report

Memo: #6

By

Aakarsh Nadella

NN: 70

Data Architect

CONTENTS

1. Functional Dependency Analysis 1

2. Relational Database Design 9

3. Prototype of Relational Database Design 11

4. Results from the Prototype.....19

Appendix.....23

List of Figures

Figure 1: Sample Data 1 1

Figure 2: Sample Data 2 2

Figure 3: Repair Data.....4

Figure 4: Final Logical Data Model.....9

Figure 5: Final Physical Data Model 10

Figure 6: First Logical Data Model 23

Figure 7: Second Logical Data Model.....24

Figure 8: Old Physical Data Model 25

1. Functional Dependency Analysis

According to the data given in Memo1, we get the following results for first table

	ApartmentID	Number	NumberRents	ComplexID	RenterID	Name	StartDate	Rent	PaymentNumber	DateDue	DatePaid	Amount
ApartmentID	-				2&3	2&3	2&3	2&3	1&2	1&2	1&2	2&3
Number	4&5	-		4&5	2&3	2&3	2&3	2&3	1&2	1&2	1&2	2&3
NumberRents	4&5	1&9	-	1&4	1&3	1&3	1&3	1&3	1&2	1&2	1&2	1&3
ComplexID	1&4	1&4	1&4	-	1&3	1&3	1&3	1&3	1&2	1&2	1&2	1&3
RenterID	5&6	5&6		5&6	-		5&6	5&6	1&2	1&2	1&2	8&9
Name	5&6	5&6		5&6		-	5&6	5&6	1&2	1&2	1&2	8&9
StartDate	1&4	1&4	1&4		1&4	1&4	-	3&6	1&2	1&2	1&2	7&8
Rent	1&4	1&4	1&4	3&5	1&4	1&4	1&4	-	1&2	1&2	1&2	8&9
PaymentNumber									-			
DateDue	1&4	1&4	1&4	2&5	1&4	1&4		3&6	1&4	-	1&4	7&8
DatePaid	3&7	3&7	3&7	3&7	3&7	3&7	3&7	3&7	3&7	3&7	-	3&7
Amount	1&4	1&4	1&4	3&5	1&4	1&4	3&5	3&6		1&2	1&2	-

Figure 1: Sample Data 1

The dependencies from the above analysis are:

{ApartmentID} → {Number}

{ApartmentID} → {NumberRents}

{ApartmentID} → {ComplexID}

{Number} → {NumberRents}

{RenterID} → {NumberRents}

{RenterID} → {Name}

{Name} → {NumberRents}

{Name} → {RenterID}

{StartDate} → {ComplexID}

{PaymentNumber} → {ApartmentID}

{PaymentNumber} → {Number}

{PaymentNumber} → {NumberRents}

{PaymentNumber} → {ApartmentID}

{PaymentNumber} → {ComplexID}

{PaymentNumber} → {RenterID}

{PaymentNumber} → {Name}

{PaymentNumber} → {StartDate}

{PaymentNumber} → {Rent}

{PaymentNumber} → {DateDue}

{PaymentNumber} → {DatePaid}

{PaymentNumber} → {Amount}

{DateDue} → {StartDate}

{DateDue} → {ComplexID}

Now for Table 2, the analysis is:

	ApartmentID	Number	ComplexID	ProspectNumb	Name	Address	Phone	StatusCode
ApartmentID	-			10 & 11	10 & 11	10 & 11	10 & 11	10 & 11
Number	12 & 13	-	12 & 13	10 & 11	10 & 11	10 & 11	10 & 11	10 & 11
ComplexID	10 & 12	10 & 12	-	10 & 11	10 & 11	10 & 11	10 & 11	10 & 11
ProspectNumbe	11 & 12	11 & 12		-				
Name	11 & 12	11 & 12			-			
Address	10 & 16	10 & 16	10 & 16	10 & 16	10 & 16	-		10 & 16
Phone	10 & 16	10 & 16	10 & 16	10 & 16	10 & 16		-	10 & 16
StatusCode	11 & 12	11 & 12	11 & 14	11 & 14	11 & 14	11 & 14	11 & 14	-

Figure 2: Sample Data 2

The dependencies from the above analysis are:

{ApartmentID} → {Number}

{ApartmentID} → {ComplexID}

{ProspectNumber} → {ComplexID}

{ProspectNumber} → {Name}

{ProspectNumber} → {Address}

{ProspectNumber} → {Phone}

{ProspectNumber} → {StatusCode}

{Name} → {ComplexID}

{Name} → {ProspectNumber}

{Name} → {Address}

{Name} → {Phone}

{Name} → {StatusCode}

{Address} → {Phone}

{Phone} → {Address}

The functional dependencies from the Enterprise Statement:

{ApartmentID} → {Number, ComplexID}

{ApartmentID, RenterID} → {Rent}

{ApartmentID, RenterID} → {StartDate}

{ApartmentID} → {Number, ComplexID}

{ApartmentID, RenterID} → {ProspectNumber}

So, based on all FD's above, after removing redundancies using primary key and removing transitive dependencies, the irreducible cover generated is as follows

{ApartmentID} → {Number}

{ApartmentID} →{ComplexID}

{ApartmentID} →{NumberRents}

{RenterID} → {Name}

{ApartmentID, RenterID} → {Rent}

{ApartmentID, RenterID} → {StartDate}

{PaymentNumber} → {ApartmentID}

{PaymentNumber} → {RenterID}

{PaymentNumber} → {DateDue}

{PaymentNumber} → {DatePaid}

{PaymentNumber} → {Amount}

{ProspectNumber} → {Name}

{ProspectNumber} → {Phone}

{ProspectNumber} → {Address}

{ProspectNumber, ApartmentID} → {StatusCode}

Relational Headers derived from irreducible cover is:

{

{ApartmentID} → {Number, ComplexID, NumberRents}

{RenterID} → {Name}

{ApartmentID, RenterID} → {Rent, StartDate}

{PaymentNumber} → {ApartmentID, RenterID, DateDue, DatePaid, Amount}

{ProspectNumber} → {Name, Phone, Address}

{ProspectNumber, ApartmentID} → {StatusCode}

}

According to the data given in Memo 4, we get the following results for table

	ComplexID	AptID	Number	PersonID	Name	StartDate	EndDAte	JobCode	RepairNum	Description	DateOrdered	DateCompleted	Type
ComplexID	-	1&5	1&5	2&3	2&3	2&3	2&3		1&5	1&5	1&5	1&5	
AptID		-		2&3	2&3	2&3	2&3		2&3	2&3			
Number	2&5	2&5	-	2&3	2&3	2&3	2&3		2&3	2&3	2&5	2&5	2&5
PersonID		1&5	1&5	-					1&5	1&5	1&5	1&5	
Name		1&5	1&5		-				1&5	1&5	1&5	1&5	
StartDate	1&3	1&3	1&3	1&3	1&3	-	1&3		1&3	1&3	1&3	1&3	1&3
EndDAte	1&2	1&2	1&2	1&2	1&2	1&2	-		1&2	1&2	1&2	1&2	1&2
JobCode	1&2	1&2	1&2	1&2	1&2	1&2	1&3	-	1&2	1&2	1&2	1&2	1&2
RepairNum									-				
Description										-			
DateOrdered				2&3	2&3	2&3	2&3		2&3	2&3	-		
DateCompleted				2&3	2&3	2&3	2&3		2&3	2&3		-	
Type	2&4	2&4	2&4	2&3	2&3	2&3	2&3	3&4	1&5	1&5	1&5	1&5	-

Figure 3: Repair Data

The dependencies from the above analysis are:

{ComplexID} → {JobCode}

{ComplexID} → {Type}

{ApartmentID} → {ComplexID}

{ApartmentID} → {Number}

{ApartmentID} → {JobCode}

{ApartmentID} → {DateOrdered}

{ApartmentID} → {DateCompleted}

{ApartmentID} → {Type}

{Number} → {JobCode}

{PersonID} → {ComplexID}

{PersonID} → {Name}

{PersonID} → {StartDate}

{PersonID} → {EndDate}

{PersonID} → {JobCode}

{PersonID} → {Type}

{Name} → {ComplexID}

{Name} → {PersonID}

{Name} → {StartDate}

{Name} → {EndDate}

{Name} → {JobCode}

{Name} → {Type}

{StartDate} → {JobCode}

{EndDate} → {JobCode}

{RepairNum} → {ComplexID}

{RepairNum} → {ApartmentID}

{RepairNum} → {Number}

{RepairNum} → {PersonID}

{RepairNum} → {Name}

{RepairNum} → {StartDate}

{RepairNum} → {EndDate}

{RepairNum} → {JobCode}

{RepairNum} → {Description}

{RepairNum} → {DateOrdered}

{RepairNum} → {DateCompleted}

{RepairNum} → {Type}

{Description} → {ComplexID}

{Description} → {ApartmentID}

{Description} → {Number}

{Description} → {PersonID}

{Description} → {Name}

{Description} → {StartDate}

{Description} → {EndDate}

{Description} → {JobCode}

{Description} → {RepairNum}

{Description} → {DateOrdered}

{Description} → {DateCompleted}

{Description} → {Type}

{DateOrdered} → {ComplexID}

{DateOrdered} → {ApartmentID}

{DateOrdered} → {Number}

{DateOrdered} → {JobCode}

{DateOrdered} → {DateCompleted}

{DateOrdered} → {Type}

{DateCompleted} → {ComplexID}

{DateCompleted} → {ApartmentID}

{DateCompleted} → {Number}

{DateCompleted} → {JobCode}

{DateCompleted} → {DateOrdered}

{DateCompleted} → {Type}

The functional dependencies from the Enterprise Statement and the other requirements requested by AAA Rentals are:

{ComplexID} → {Description}

{StatusCode} → {Description}

{Type} → {Description}

{JobCode} → {Description}

To increase control over the information and limit user's access only to the data they should rightfully access, sub-types should be created for CEO, Manager and Maintenance Staff and the functional dependencies corresponding to these are as follows:

{PersonID} → {NextMeeting} (corresponding to CEO sub-type)

{PersonID} → {LastAccessedOn} (corresponding to Manager sub-type)

So, based on all FD's above, after removing redundancies using primary key and removing transitive dependencies, the irreducible cover generated is as follows

{ApartmentID} → {ComplexID}

{ApartmentID} → {Number}

{PersonID} → {Name}

{PersonID} → {StartDate}

{PersonID} → {EndDate}

{PersonID} → {JobCode}

{RepairNum} → {ApartmentID}

{RepairNum} → {PersonID}

{RepairNum} → {Description}

{RepairNum} → {DateOrdered}

{RepairNum} → {DateCompleted}

{RepairNum} → {Type}

{ComplexID} → {Description}

{StatusCode} → {Description}

{Type} → {Description}

{JobCode} → {Description}

{PersonID} → {NextMeeting} (corresponding to CEO sub-type)

{PersonID} → {LastAccessedOn} (corresponding to Manager sub-type)

Relational Headers derived from irreducible cover is:

{

{ApartmentID} → {Number, ComplexID}

{PersonID} → {Name, StartDate, EndDate, JobCode}

{RepairNum} → {Description, DateOrdered, DateCompleted, Type, ApartmentID, PersonID}

{ComplexID} → {Description, PersonID} (corresponding to Maintenance Staff sub-type)

{StatusCode} → {Description}

{JobCode} → {Description}

{Type} → {Description}

{PersonID} → {NextMeeting} (corresponding to CEO sub-type)

{PersonID} → {LastAccessedOn} (corresponding to Manager sub-type)

{PersonID} (corresponding to Maintenance Staff sub-type)

}

Looking on whole the final irreducible cover is:

{

{ApartmentID} → {Number, ComplexID, NumberRents}

{RenterID} → {Name}

{ApartmentID, RenterID} → {Rent, StartDate}

{PaymentNumber} → {ApartmentID, RenterID, DateDue, DatePaid, Amount}

{ProspectNumber} → {Name, Phone, Address}

{ProspectNumber, ApartmentID} → {StatusCode}

{PersonID} → {Name, StartDate, EndDate, JobCode}

{RepairNum} → {Description, DateOrdered, DateCompleted, Type, ApartmentID, PersonID}

{ComplexID} → {Description, PersonID} (corresponding to Maintenance Staff sub-type)

{StatusCode} → {Description}

{JobCode} → {Description}

{Type} → {Description}

{PersonID} → {NextMeeting} (corresponding to CEO sub-type)

{PersonID} → {LastAccessedOn} (corresponding to Manager sub-type)

{PersonID}

2. Relational Database Design

The final Logical Data Model, that is designed according to the functional dependency analysis carried out above is as follows:

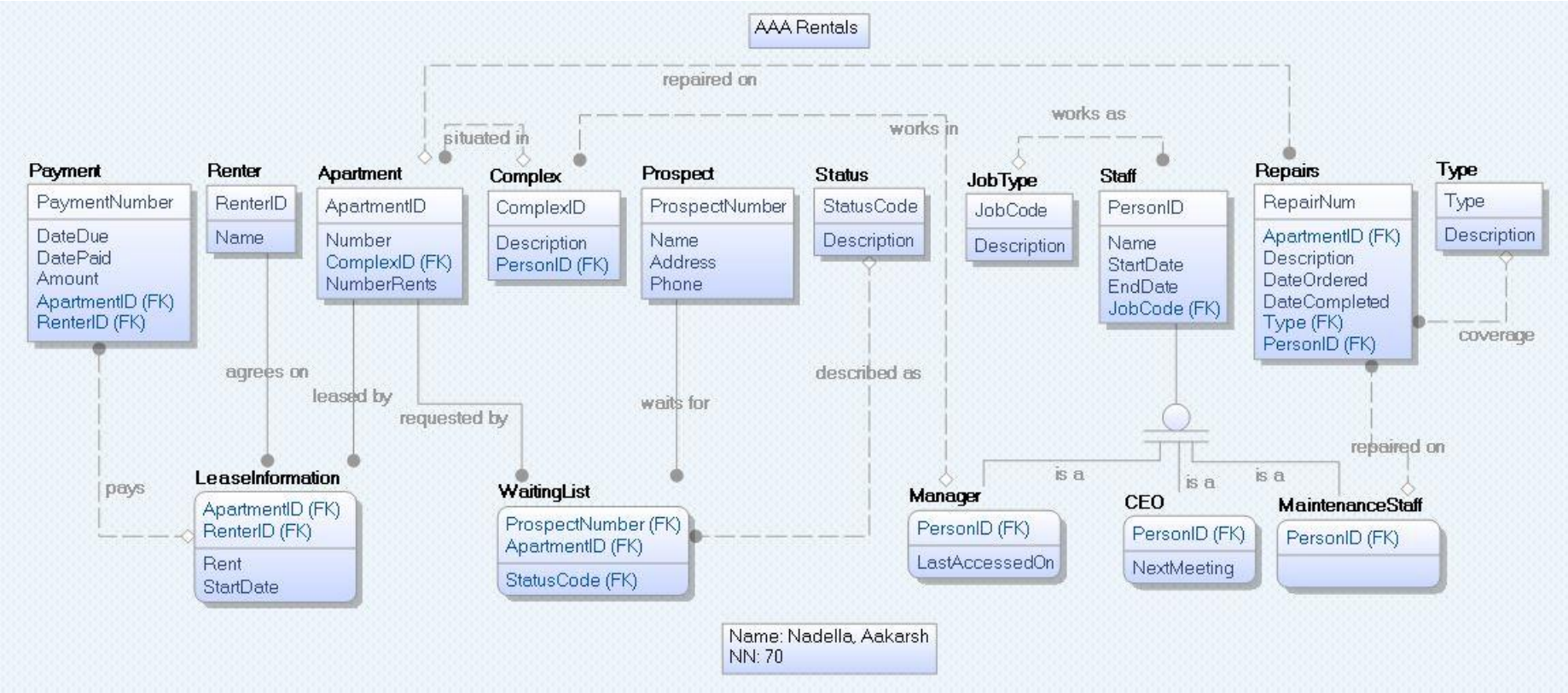

Figure 4: Final Logical Data Model

The corresponding final Physical Data Model is as follows:

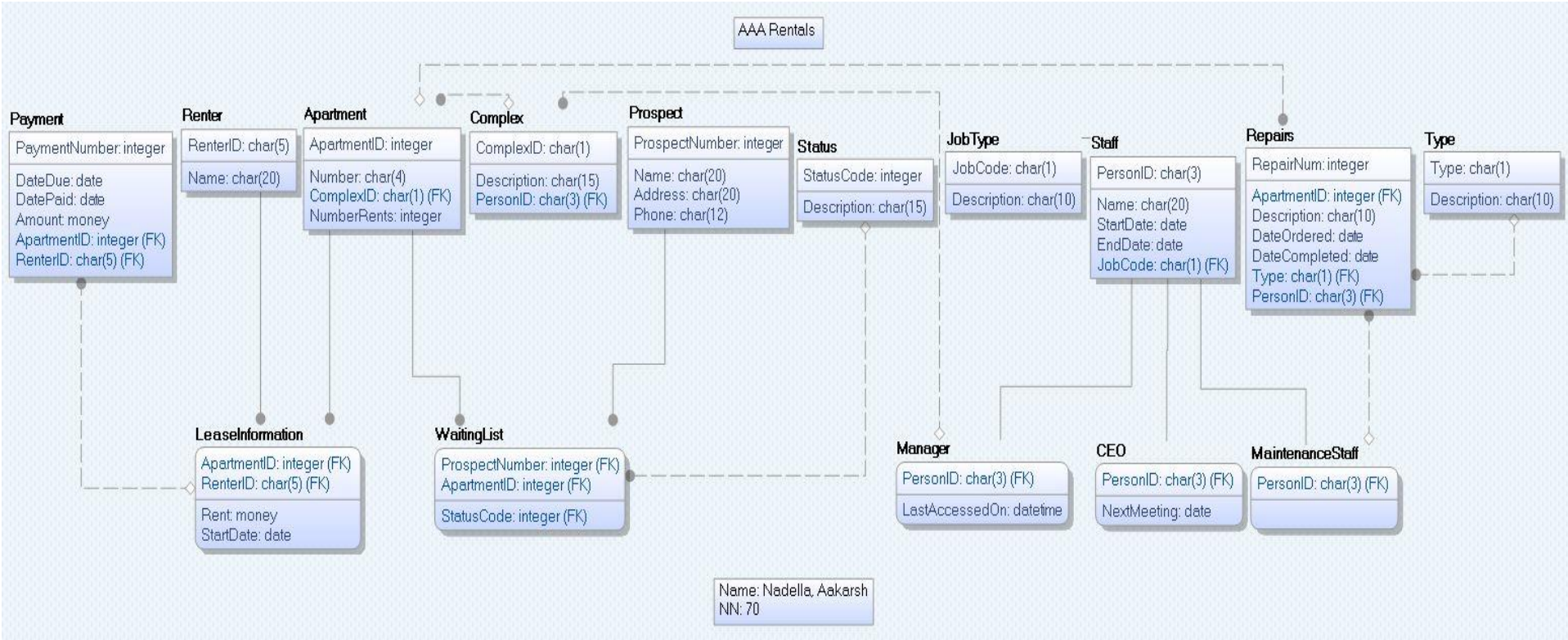

Figure 5: Final Physical Data Model

Note: The previous data models that are created earlier as part of designing the database and the steps involved in achieving the complete database design is as illustrated in appendix as a reference.

3. Prototype of Relational Database Design

Once the Logical and Physical data models are designed using ERwin, the prototype of Relational Database Design is generated using SQL Server 2016. Before generating database from Erwin database should be created in SQL Server. This is done using the following commands.

```
use master;

create database AAA70;
```

The Data Definition Language (DDL) commands or the script that is used to generate database from ERwin is as follows:

```
CREATE TABLE [Prospect]
(
        [ProspectNumber]     integer  NOT NULL ,
        [Name]               char(20)  NULL ,
        [Address]            char(20)  NULL ,
        [Phone]              char(12)  NULL ,
        PRIMARY KEY  CLUSTERED ([ProspectNumber] ASC)
)
go


CREATE TABLE [JobType]
(
        [JobCode]            char(1)  NOT NULL ,
        [Description]        char(10)  NULL ,
        PRIMARY KEY  CLUSTERED ([JobCode] ASC)
)
go


CREATE TABLE [Staff]
(
        [PersonID]           char(3)  NOT NULL ,
```

```
	[Name]		char(20)  NULL ,
	[StartDate]	date  NULL ,
	[EndDate]	date  NULL ,
	[JobCode]	char(1)  NULL ,
	PRIMARY KEY  CLUSTERED ([PersonID] ASC),
	 FOREIGN KEY ([JobCode]) REFERENCES [JobType]([JobCode])
)
go

CREATE TABLE [Manager]
(
	[PersonID]	char(3)  NOT NULL ,
	[LastAccessedOn]	datetime  NULL ,
	PRIMARY KEY  CLUSTERED ([PersonID] ASC),
	 FOREIGN KEY ([PersonID]) REFERENCES [Staff]([PersonID])
)
go

CREATE TABLE [Complex]
(
	[ComplexID]	char(1)  NOT NULL ,
	[Description]	char(15)  NULL ,
	[PersonID]	char(3)  NULL ,
	PRIMARY KEY  CLUSTERED ([ComplexID] ASC),
	 FOREIGN KEY ([PersonID]) REFERENCES [Manager]([PersonID])
)
go
```

```
CREATE TABLE [Apartment]
(
	[ApartmentID]	integer  NOT NULL ,
	[Number]	char(4)  NULL ,
	[ComplexID]	char(1)  NULL ,
	[NumberRents]	integer  NULL ,
	PRIMARY KEY  CLUSTERED ([ApartmentID] ASC),
	 FOREIGN KEY ([ComplexID]) REFERENCES [Complex]([ComplexID])
)
go

CREATE TABLE [Status]
(
	[StatusCode]	integer  NOT NULL ,
	[Description]	char(15)  NULL ,
	PRIMARY KEY  CLUSTERED ([StatusCode] ASC)
)
go

CREATE TABLE [WaitingList]
(
	[ProspectNumber]	integer  NOT NULL ,
	[ApartmentID]	integer  NOT NULL ,
	[StatusCode]	integer  NULL ,
	PRIMARY KEY  CLUSTERED ([ProspectNumber] ASC,[ApartmentID] ASC),
	 FOREIGN KEY ([ProspectNumber]) REFERENCES [Prospect]([ProspectNumber]),
	 FOREIGN KEY ([ApartmentID]) REFERENCES [Apartment]([ApartmentID]),
	 FOREIGN KEY ([StatusCode]) REFERENCES [Status]([StatusCode])
```

```
)
go


CREATE TABLE [Type]
(
	[Type]		char(1)  NOT NULL ,
	[Description]	char(10)  NULL ,
	PRIMARY KEY  CLUSTERED ([Type] ASC)
)
go

CREATE TABLE [MaintenanceStaff]
(
	[PersonID]		char(3)  NOT NULL ,
	PRIMARY KEY  CLUSTERED ([PersonID] ASC),
	 FOREIGN KEY ([PersonID]) REFERENCES [Staff]([PersonID])
)
go

CREATE TABLE [Repairs]
(
	[RepairNum]		integer  NOT NULL ,
	[ApartmentID]		integer  NULL ,
	[Description]		char(10)  NULL ,
	[DateOrdered]		date  NULL ,
	[DateCompleted]		date  NULL ,
	[Type]		char(1)  NULL ,
```

```
	[PersonID]	char(3)  NULL ,
	PRIMARY KEY  CLUSTERED ([RepairNum] ASC),
	 FOREIGN KEY ([ApartmentID]) REFERENCES [Apartment]([ApartmentID]),
	 FOREIGN KEY ([Type]) REFERENCES [Type]([Type]),
	 FOREIGN KEY ([PersonID]) REFERENCES [MaintenanceStaff]([PersonID])
)
go

CREATE TABLE [Renter]
(
	[RenterID]	char(5)  NOT NULL ,
	[Name]	char(20)  NULL ,
	PRIMARY KEY  CLUSTERED ([RenterID] ASC)
)
go

CREATE TABLE [LeaseInformation]
(
	[ApartmentID]	integer  NOT NULL ,
	[RenterID]	char(5)  NOT NULL ,
	[Rent]	money  NULL ,
	[StartDate]	date  NULL ,
	PRIMARY KEY  CLUSTERED ([ApartmentID] ASC,[RenterID] ASC),
	 FOREIGN KEY ([ApartmentID]) REFERENCES [Apartment]([ApartmentID]),
	 FOREIGN KEY ([RenterID]) REFERENCES [Renter]([RenterID])
)
go
```

CREATE TABLE [Payment]

(

[PaymentNumber] integer NOT NULL ,

[DateDue] date NULL ,

[DatePaid] date NULL ,

[Amount] money NULL ,

[ApartmentID] integer NULL ,

[RenterID] char(5) NULL ,

PRIMARY KEY CLUSTERED ([PaymentNumber] ASC),

FOREIGN KEY ([ApartmentID],[RenterID]) REFERENCES [LeaseInformation]([ApartmentID],[RenterID])

)

go

CREATE TABLE [CEO]

(

[PersonID] char(3) NOT NULL ,

[NextMeeting] date NULL ,

PRIMARY KEY CLUSTERED ([PersonID] ASC),

FOREIGN KEY ([PersonID]) REFERENCES [Staff]([PersonID])

)

go

Database is now populated using sample data given by using following Data Manipulation Language (DML) commands or insert statements,

```
use AAA70;

insert into Renter values('A021','Jack Black');
insert into Renter values('C222','Fred Jones');
insert into Renter values('A025','Mike Allen');
insert into Renter values('A023','Jane Black');
insert into Renter values('B444','John Dough');
insert into Renter values('B456','Bill Smith');
```

```
insert into Prospect values(55, 'Jack Black','73 Maple Ave','555-881-3232');
insert into Prospect values(70, 'Aakarsh Nadella','359 N West St','469-900-9835');
insert into Prospect values(11, 'Kevin White','234 Main St','555-222-1234');
insert into Prospect values(31, 'Gail Green','P.O. Box 22','555-234-2525');
insert into Prospect values(45, 'Ed Brown','12 N 1st St','555-234-8888');
insert into Prospect values(46, 'Ann Black','73 Maple Ave','555-881-3233');

insert into Status values(0, 'Incomplete');
insert into Status values(1, 'References');
insert into Status values(2, 'Waiting');
insert into Status values(3, 'Complete');

insert into JobType values('C', 'CEO');
insert into JobType values('M', 'Manager');
insert into JobType values('R', 'Repair');

insert into Type values('I', 'Insured');
insert into Type values('U', 'Uninsured');

insert into Staff values('P00', 'Bob Bureaucrat',NULL,NULL,'C');
insert into Staff values('P01', 'Sam Supervisor',NULL,NULL,'M');
insert into Staff values('P02', 'Fred Foreman',NULL,NULL,'M');
insert into Staff values('P03', 'Mary Manager',NULL,NULL,'M');
insert into Staff values('P04', 'Alex Johnson','2015-10-01',NULL,'R');
insert into Staff values('P06', 'Ben Jackson','2017-12-22',NULL,'R');
insert into Staff values('P05', 'Gail Steward','2015-10-01','2017-12-21','R');
insert into Staff values('P07', 'Beth Redding','2017-12-22',NULL,'R');

insert into Manager values('P01',NULL);
insert into Manager values('P02',NULL);
insert into Manager values('P03',NULL);

insert into CEO values('P00',NULL);

insert into MaintenanceStaff values('P04');
insert into MaintenanceStaff values('P05');
insert into MaintenanceStaff values('P07');
insert into MaintenanceStaff values('P06');

insert into Complex values('L','Lakeview','P01');
insert into Complex values('N','Northside','P02');
insert into Complex values('P','Princeton','P03');

insert into Apartment values(1,'101G','L',2);
insert into Apartment values(2,'201','L',1);
insert into Apartment values(5,'201','N',1);
insert into Apartment values(7,'209','L',1);
insert into Apartment values(8,'333','P',1);
insert into Apartment values(9,'431P','P',2);
insert into Apartment values(12,'310','L',NULL);
insert into Apartment values(14,'201','P',NULL);

insert into LeaseInformation values(1,'A021',1100,'2016-12-01');
insert into LeaseInformation values(1,'C222',1200,'2017-11-15');
insert into LeaseInformation values(2,'A025',1100,'2016-12-01');
insert into LeaseInformation values(5,'A023',1200,'2017-11-01');
insert into LeaseInformation values(7,'A023',1250,'2017-11-15');
insert into LeaseInformation values(8,'B444',700,'2017-12-01');
```

```
insert into LeaseInformation values(9,'B456',900,'2017-12-01');

insert into Payment values(211,'2017-01-01','2016-12-30',1100,1,'A021');
insert into Payment values(397,'2017-02-01','2017-01-29',1100,1,'A021');
insert into Payment values(402,'2017-12-15','2017-12-30',1200,1,'C222');
insert into Payment values(399,'2017-01-01','2017-01-01',1100,2,'A025');
insert into Payment values(400,'2017-12-01','2017-12-01',1200,5,'A023');
insert into Payment values(401,'2017-12-15','2017-12-15',1200,7,'A023');
insert into Payment values(488,'2018-01-01','2016-12-30',700,8,'B444');
insert into Payment values(511,'2018-01-01','2017-12-30',500,9,'B456');
insert into Payment values(512,'2018-01-01','2017-12-31',400,9,'B456');

insert into WaitingList values(55,1,3);
insert into WaitingList values(70,1,2);
insert into WaitingList values(70,2,2);
insert into WaitingList values(11,5,1);
insert into WaitingList values(31,5,2);
insert into WaitingList values(45,12,1);
insert into WaitingList values(46,14,0);

insert into Repairs values(23,1,'Faucet','2017-12-22','2017-12-23','U','P04');
insert into Repairs values(28,5,'Window','2017-12-28','2017-12-31','I','P06');
insert into Repairs values(31,5,'Carpet','2017-12-28','2017-12-31','I','P05');
insert into Repairs values(33,8,'Roof','2018-01-05','2018-02-23','I',NULL);
insert into Repairs values(35,2,'Lock','2018-01-10','2018-01-11','U','P04');
```

The data once inserted is then checked executing Select statements as follows:

```
Select * from Renter;
Select * from Prospect;
Select * from Status;
Select * from JobType;
Select * from Type;
Select * from Staff;
Select * from Manager;
Select * from CEO;
Select * from MaintenanceStaff;
Select * from Complex;
Select * from Apartment;
Select * from LeaseInformation;
Select * from Payment;
Select * from WaitingList;
Select * from Repairs;
```

4. Results from the Prototype

1) List the information for rent payments (and when and for what apartment) that are less than the lease specifies

```
use AAA70;

Select P.PaymentNumber, L.ApartmentID, L.RenterID, P.DatePaid,
               L.Rent as 'Actual Rent', P.Amount As 'Rent Paid'
from Payment P inner join LeaseInformation L on
              L.RenterID = P.RenterID and L.ApartmentID = P.ApartmentID
where P.Amount < L.Rent;
```

```
PaymentNumber ApartmentID RenterID DatePaid   Actual Rent           Rent Paid
------------- ----------- -------- ---------- --------------------- ---------------------
401           7           A023     2017-12-15 1250.00               1200.00
511           9           B456     2017-12-30 900.00                500.00
512           9           B456     2017-12-31 900.00                400.00

(3 rows affected)
```

2) For each apartment, list the rent payments in chronological(date) order.

```
use AAA70;

Select PaymentNumber, ApartmentID, RenterID, Amount,DatePaid
from Payment P
Order by ApartmentID, DatePaid;
```

```
PaymentNumber ApartmentID RenterID Amount                DatePaid
------------- ----------- -------- --------------------- ----------
211           1           A021     1100.00               2016-12-30
397           1           A021     1100.00               2017-01-29
402           1           C222     1200.00               2017-12-30
399           2           A025     1100.00               2017-01-01
400           5           A023     1200.00               2017-12-01
401           7           A023     1200.00               2017-12-15
488           8           B444     700.00                2016-12-30
511           9           B456     500.00                2017-12-30
512           9           B456     400.00                2017-12-31

(9 rows affected)
```

3) List the name and number for prospects who are waiting for an apartment.

```
use AAA70;

Select W.ApartmentID,P.ProspectNumber,P.Name,S.Description
from Prospect P inner join WaitingList W
        on P.ProspectNumber = W.ProspectNumber inner join Status S
        on W.StatusCode = S.StatusCode
where S.Description = 'waiting'
```

```
order by ApartmentID;
ApartmentID ProspectNumber Name                Description
----------- ------------- ------------------- --------------
1           70            Aakarsh Nadella     Waiting
2           70            Aakarsh Nadella     Waiting
5           31            Gail Green          Waiting

(3 rows affected)
```

4) List the top 2 leases based on rent.

```
use AAA70;

Select top 2 [ApartmentID], [RenterID], [Rent], [StartDate]
from LeaseInformation L
order by Rent desc;

ApartmentID RenterID Rent                 StartDate
----------- -------- -------------------- ----------
7           A023     1250.00              2017-11-15
1           C222     1200.00              2017-11-15

(2 rows affected)
```

5) List information about leases signed/started on November 1, 2017 and those signed/started on December 1, 2017

```
use AAA70;

Select [ApartmentID], [RenterID], [Rent], [StartDate]
from LeaseInformation
where StartDate = '2017-11-01' or StartDate = '2017-12-01'

ApartmentID RenterID Rent                 StartDate
----------- -------- -------------------- ----------
5           A023     1200.00              2017-11-01
8           B444     700.00               2017-12-01
9           B456     900.00               2017-12-01

(3 rows affected)
```

6) List information about late rent payments

```
use AAA70;

Select [PaymentNumber], [DateDue], [DatePaid], [Amount], [ApartmentID], [RenterID]
from Payment
where DatePaid > DateDue;

PaymentNumber DateDue    DatePaid   Amount               ApartmentID RenterID
------------- ---------- ---------- -------------------- ----------- --------
402           2017-12-15 2017-12-30 1200.00              1           C222
```

```
(1 row affected)
```

7) List apartment rental information by complex.

```
use AAA70;

Select A.ComplexID, C.Description, L.ApartmentID, L.RenterID,L.Rent,L.StartDate
from Apartment A inner join LeaseInformation L
       on A.ApartmentID = L.ApartmentID inner join Complex C
       on C.ComplexID = A.ComplexID
order by A.ComplexID;
```

```
ComplexID Description     ApartmentID RenterID Rent                 StartDate
--------- --------------- ----------- -------- -------------------- ----------
L         Lakeview        1           A021     1100.00              2016-12-01
L         Lakeview        1           C222     1200.00              2017-11-15
L         Lakeview        2           A025     1100.00              2016-12-01
L         Lakeview        7           A023     1250.00              2017-11-15
N         Northside       5           A023     1200.00              2017-11-01
P         Princeton       8           B444     700.00               2017-12-01
P         Princeton       9           B456     900.00               2017-12-01

(7 rows affected)
```

8) Who (Name and ID) manages the Princeton Complex?

```
use AAA70;

Select C.Description, S.PersonID, S.Name
from Complex C inner join Staff S on C.PersonID = S.PersonID
where C.Description = 'Princeton';
```

```
Description     PersonID Name
--------------- -------- --------------------
Princeton       P03      Mary Manager

(1 row affected)
```

9) Who (Name and ID) has access to repair data for the Northside complex?

```
use AAA70;

Select C.Description, S.PersonID, S.Name
from Complex C inner join Staff S on C.PersonID = S.PersonID
where C.Description = 'Northside';
```

```
Description     PersonID Name
--------------- -------- --------------------
Northside       P02      Fred Foreman

(1 row affected)
```

10) How many times each apartment has been leased?

```
use AAA70;

Select ApartmentID, NumberRents
from Apartment;

ApartmentID NumberRents
----------- -----------
1           2
2           1
5           1
7           1
8           1
9           2
12          NULL
14          NULL

(8 rows affected)
```

11) List all insured repairs. Include the name of maintenance person if known.

```
use AAA70;

Select R.RepairNum,R.Description,R.ApartmentID,T.Description,S.Name
from Repairs R inner join Type T on R.Type = T.Type
                    left outer join Staff S on R.PersonID = S.PersonID
where T.Description = 'Insured';

RepairNum   Description ApartmentID Description Name
----------- ----------- ----------- ----------- --------------------
28          Window      5           Insured     Ben Jackson
31          Carpet      5           Insured     Gail Steward
33          Roof        8           Insured     NULL

(3 rows affected)
```

Appendix

The first Logical Data Model designed is:

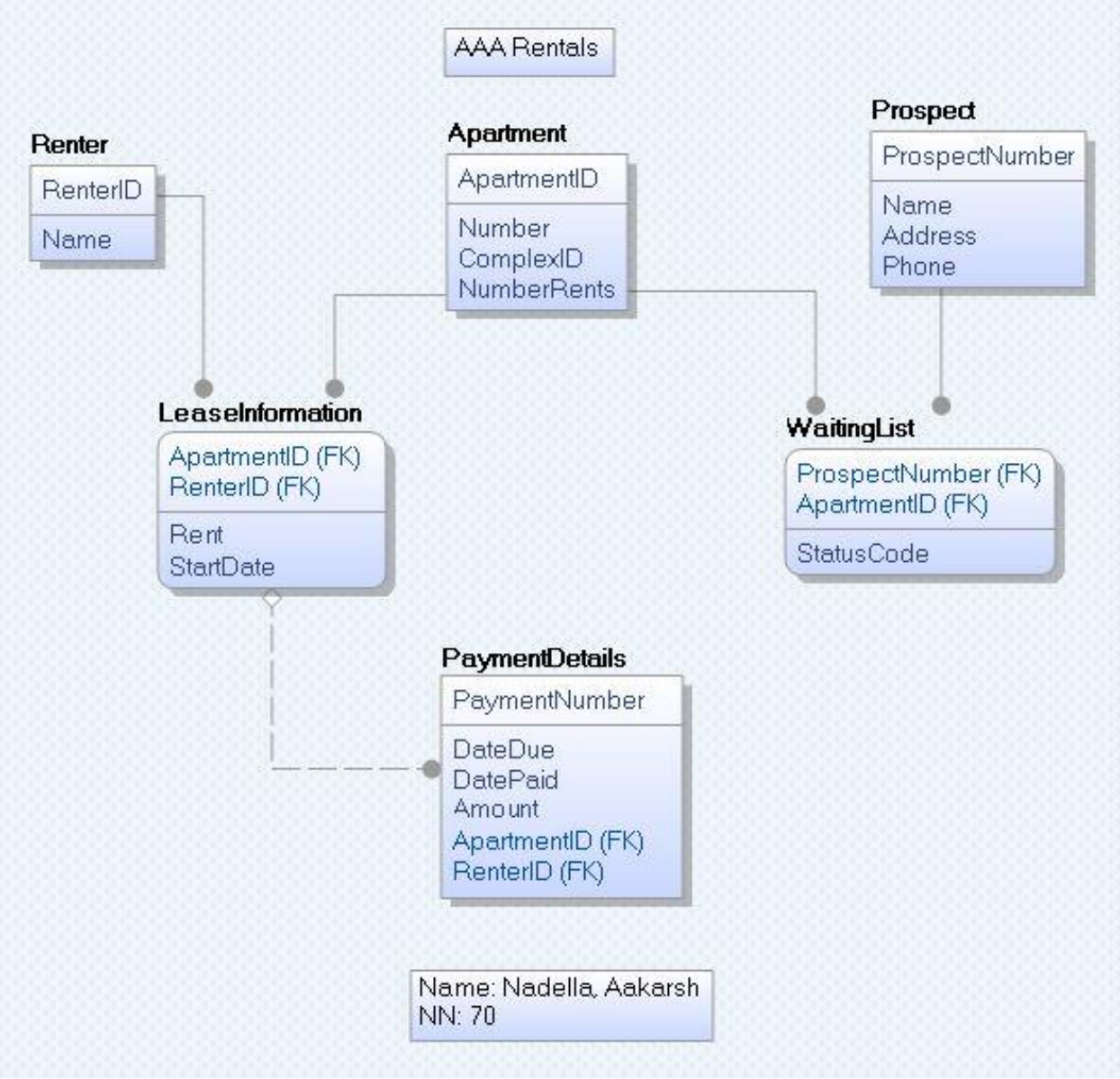

Figure 6: First Logical Data Model

Changes are made to the first logical model, considering adding new information requested by the user to include in the database and to make the model more reliable.

The second Logical Data Model designed is:

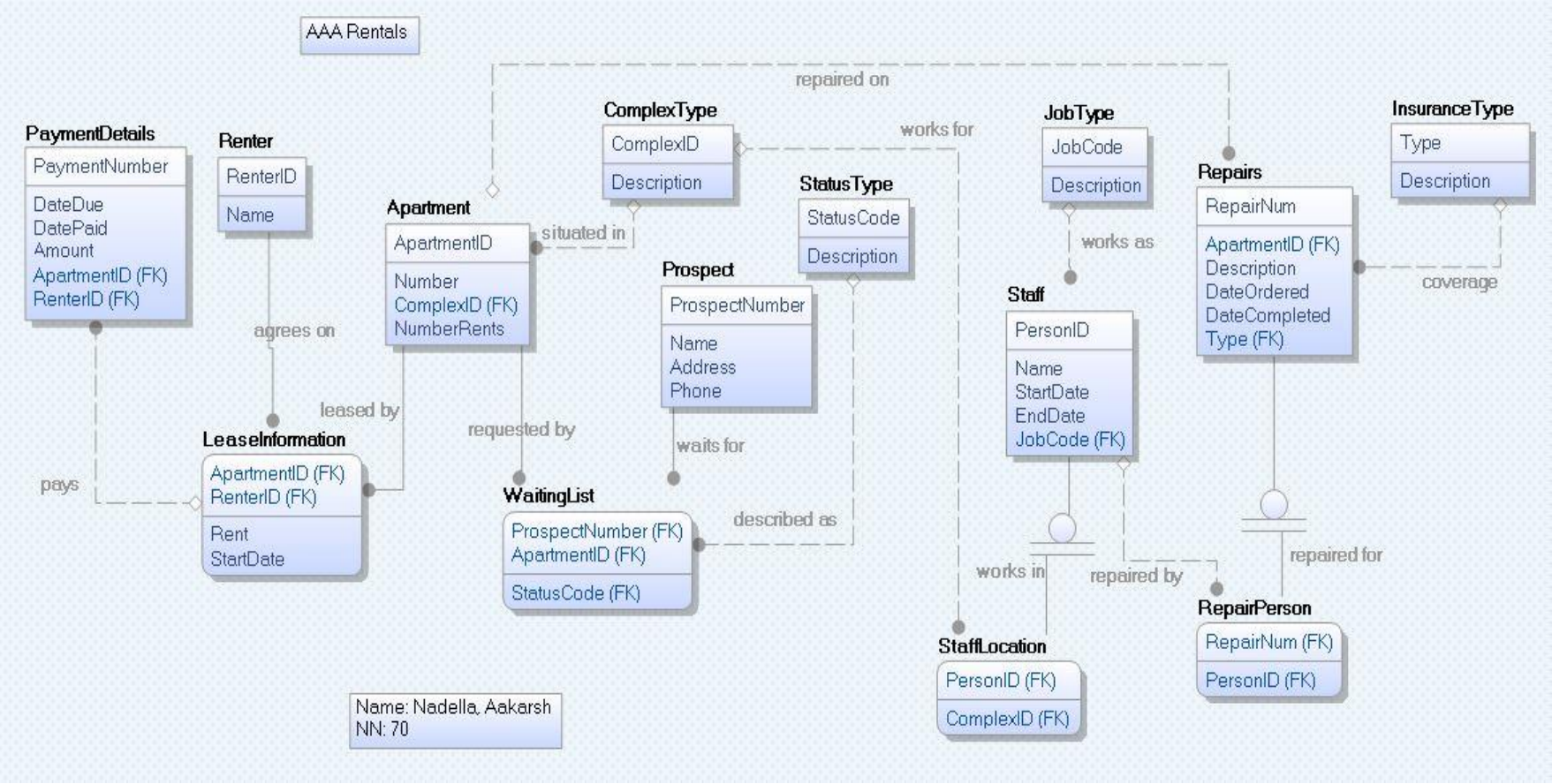

Figure 7: Second Logical Data Model

The corresponding Physical Data Model for the second Logical Data Model is as follows:

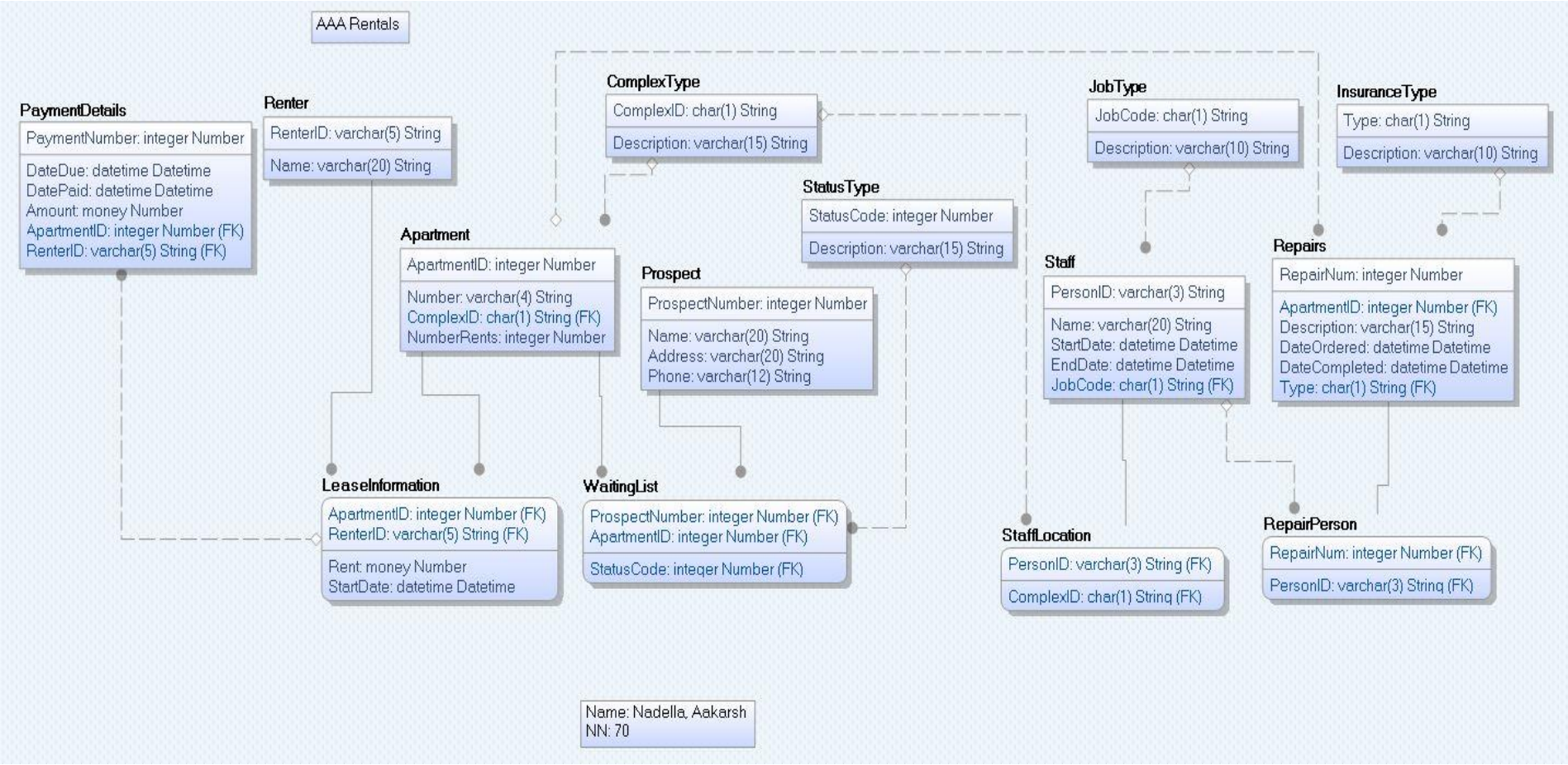

Figure 8: Old Physical Data Model

Steps carried out in migration from first logical data model to second logical data model are:

- Code tables were added to the previous LDM as Bob wanted them to be include in the database.
- Independent entities are moved to top to follow perfect design methodology.
- Verb Phrases associated with the relationships are included to determine the association between 2 entities.
- The design model is updated to add new information of repair data into the database.
- Functional Dependency analysis is carried out to make sure that the design satisfies BCNF.
- Staff table is created with PersonID, Name, StartDate, End Date and JobCode.
- Repairs table is created with RepairNum, Description, DateOrderd, DateCompleted, Type and a non-identifying relationship with Apartment table to get the AptId in which repair is carried out.
- As the database should also contain the maintenance staff associated with each repair if known, a sub-type relationship is created with new entity RepairPerson along with non-identifying relationship with Staff table to get PersonID.
- In the same way to track the working staff of each complex say Manager and maintenance staff, a sub-type relationship is created with new entity StaffLocation along with non-identifying relationship with ComplexType table to get ComplexID.
- *Note: These sub-types are created so that they can handle null values if any. i.e. Only the values which don't have null are been placed in the newly created table. For example, the CEO of AAA Rentals, Bob also has a PersonID. But to keep track of working staff of each complex say manager, we can't include ComplexID in Person table as the CEO Bob will have ambiguity to associate a ComplexID.*

Steps carried out in achieving complete design are:

- To limit user's access only to the data they need to rightfully access, sub-types are created for CEO, Manager and Maintenance Staff.
- In addition, as Bob needs to track manager's last access and his next meeting these 2 attributes are added in Manager and CEO sub-type respectively.
- Also, non-identifying relationship between Manager and Complex are altered to ensure better design.
- Non-identifying relationship between Person and Repair data is deleted and a new non-identifying relationship between MaintenanceStaff and Repair is placed. This ensure only the PersonID of MaintenanceStaff will be placed in Repair data.

Proper data types are assigned to each attribute, so that all the integrity constraints are satisfied.