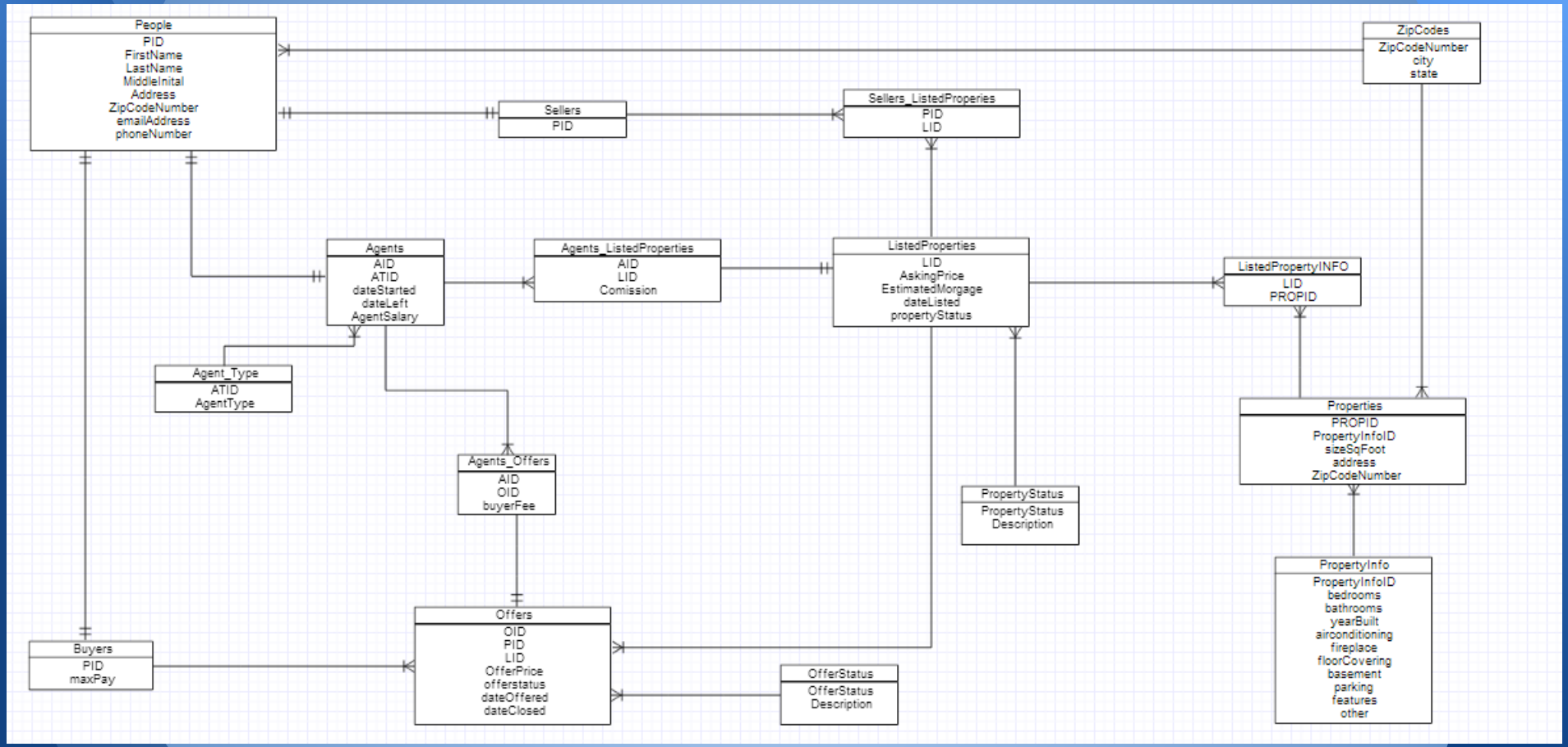
Free Lots Real Estate

By:Steven O'Meara

Table of Contents

Executive Summary

The goal of this database is to improve the operation of Free Lots Real Estate, a local Californian real estate agency. This would be an inhouse database used to track customers as well as active properties in the local area. Since a real estate company usually deals with a large amount of data it is easy to lose data or take a while to find it. This presentation will show each table in the database as well as many examples of views, queries, triggers and other functions that will display how this database will help you in the future.


People
PID
FirstName
LastName
MiddleInital
Address
ZipCodeNumber
emailAddress
phoneNumber
ZipCodes
ZipCodeNumber
city
state
Sellers
PID
Sellers_ListedProperies
PID
LID
Agents
AID
ATID
dateStarted
dateLeft
AgentSalary
Agents_ListedProperties
AID
LID
Comission
ListedProperties
LID
AskingPrice
EstimatedMorgage
dateListed
propertyStatus
ListedPropertyINFO
LID
PROPID
Agent_Type
ATID
AgentType
Agents_Offers
AID
OID
buyerFee
PropertyStatus
PropertyStatus
Description
Properties
PROPID
PropertyInfoID
sizeSqFoot
address
ZipCodeNumber
PropertyInfo
PropertyInfoID
bedrooms
bathrooms
yearBuilt
airconditioning
fireplace
floorCovering
basement
parking
features
other
Buyers
PID
maxPay
Offers
OID
PID
LID
OfferPrice
offerstatus
dateOffered
dateClosed
OfferStatus
OfferStatus
Description

ZipCode Table

This table is used to give many different zip code a corresponding city and state.

```
CREATE TABLE ZipCodes
(
  ZipCodeNumber char(5) NOT NULL,
  City varchar(50) NOT NULL,
  State char(2) NOT NULL,
  PRIMARY KEY (ZipCodeNumber)
);
```

Functional Dependencies
ZipCodeNumber→ City, State

	zipcodenumber character(5)	city character varying(50)	state character(2)
1	94101	San Francisco	CA
2	90001	Los Angeles	CA
3	90209	Beverly Hills	CA
4	90401	Santa Monica	CA
5	92101	San Diego	CA

People Table

This table is used because a real estate agent can also be a buyer or a seller of a property. This table includes all of the information that buyers, sellers and real estate agents have in common.

```
CREATE TABLE people
(
  PID serial NOT NULL,
  FirstName varchar(30) NOT NULL,
  LastName varchar(30) NOT NULL,
  MiddleInitial char(1),
  Address varchar(100) NOT NULL,
  ZipCodeNumber varchar(5) NOT NULL REFERENCES ZipCodes (ZipCodeNumber),
  EmailAddress varchar(100) NOT NULL,
  PhoneNumber varchar(10) NOT NULL,
  PRIMARY KEY (PID)
);
```

Functional Dependencies
PID → FirstName, LastName, MiddleInitial, Address, ZipCodeNumber, EmailAddress, PhoneNumber

	pid integer	firstname character varying(30)	lastname character varying(30)	middleinitial character(1)	address character varying(100)	zipcodenumber character varying(5)	emailaddress character varying(100)	phonenumber character varying(10)
1	1	John	Brown	R	30 Main Street	94101	John.Brown@yahoo.co	6197895216
2	2	Bob	Kelly	J	222 Acorn Street	90001	Bob.Kelly@gmail.com	3234123789
3	3	Jim	Dandy	B	465 Annette Street	94101	Jim.Dandy@gmail.com	6197994897
4	4	Steven	OMeara		745 Castle Lane	90209	Steven.Omeara@yahoo	3104931562
5	5	Joe	Lyktey		412 Barry Court	90001	Joe.Lyktey@yahoo.co	3233201049
6	6	Richard	Maguire	E	41 Ocean Avenue	94101	Richard.Maguire@aim	6194963157
7	7	Chris	Smith		68 Jarvis Street	94101	Chris.Smith@yahoo.c	6191067430
8	8	Annie	Kempf		712 Sky Drive	90001	Annie.Kempf@gmail.c	3235964031
9	9	Debra	Schafer		167 Mountain Place	90401	Debra.Schafer@gmail	3181034976
10	10	Kelly	Zapitella	J	256 Park Avenue	92101	Kelly.Zapitella@gma	7934136890
11	11	Meghan	Gonzalez		924 Rush Street	90209	Meghan.Gonzalez@aim	3104890314

Buyers Table

The buyers table contains the maxPrice column which represents the approximate max amount of money a buyer has to put towards a property

```
CREATE TABLE Buyers
(
  PID INTEGER NOT NULL REFERENCES people(PID),
  maxPay INTEGER NOT NULL,
  PRIMARY KEY(PID)
);
```

Functional Dependencies
PID → maxPay

	pid integer	maxpay integer
1	3	320000
2	6	400000
3	10	220000

Sellers Table

```
CREATE TABLE Sellers
(
  PID INTEGER NOT NULL REFERENCES people(PID),
  PRIMARY KEY(PID)
);
```

Functional Dependencies
PID →

	pid integer
1	2
2	4
3	7
4	9
5	11

Agents Table

This table is used to store information specific for agents. Keeping the start date and the end date of their employment will allow to see who is working of the company, a salary is also included.

```
CREATE TABLE Agents
(
  AID int NOT NULL REFERENCES people (PID),
  ATID int NOT NULL REFERENCES Agent_Type (ATID),
  AgentSalary INT NOT NULL,
  dateStarted date NOT NULL,
  dateAgentLeft date,
  PRIMARY KEY (AID)
);
```

Functional Dependencies
AID → ATID, agentSalary, dateStarted, dateAgentLeft

	aid integer	atid integer	agentsalary integer	datestarted date	dateagentleft date
1	1	1	60000	2010-06-12	
2	5	1	50000	2011-03-08	
3	8	1	45000	2012-03-12	2013-01-01

Agent_Type table

This table contains the classification of the real estate agent

```
CREATE TABLE Agent_Type
(
  ATID serial NOT NULL,
  AgentType varchar(100),
  PRIMARY KEY (ATID)
);
```

Functional Dependencies
ATID → agentType

	atid integer	agenttype character varying(100)
1	1	Realtor
2	2	Sellers Agent
3	3	Buyers Agent

ListedProperties Table

This table contains all of the properties that have ever been listed.

```
CREATE TABLE ListedProperties
(
  LID serial NOT NULL,
  askingPrice INTEGER NOT NULL,
  estimatedMorgage INTEGER NOT NULL,
  dateListed date NOT NULL,
  propertyStatus char(1) NOT NULL REFERENCES PropertyStatus
(PropertyStatus),
  PRIMARY KEY (LID)
);
```

Functional Dependencies
LID → askingPrice, estimatedMorgage, dateListed, propertyStatus

	lid integer	askingprice integer	estimatedmorgage integer	datelisted date	propertystatus character(1)
1	1	320000	1500	2013-08-13	C
2	2	450000	1000	2011-11-12	C
3	3	220000	2000	2011-03-25	O
4	4	300000	1750	2013-07-22	O
5	5	338000	1650	2012-05-09	O

Offers Table

This table shows all rejected and accepted offers on a listed property.

```
CREATE TABLE Offers
(
  OID serial NOT NULL,
  PID INTEGER NOT NULL REFERENCES Buyers (PID),
  LID INTEGER NOT NULL REFERENCES ListedProperties (LID),
  OfferPrice integer NOT NULL,
  OfferStatus char(1) NOT NULL REFERENCES Offerstatus (Offerstatus),
  dateOffered date NOT NULL,
  dateClosed date,
  PRIMARY KEY (OID)
);
```

Functional Dependencies
OID → PID, LID, offerPrice, dateOffered, dateClosed, offerStatus

	oid integer	pid integer	lid integer	offerprice integer	offerstatus character(1)	dateoffered date	dateclosed date
1	1	3	1	300000	R	2013-11-15	2013-12-12
2	4	3	1	310000	A	2013-11-20	2013-12-05
3	2	6	2	400000	A	2012-10-18	2013-11-12
4	3	10	3	200000	R	2012-01-20	2013-08-15

PropertyStatus Table

This table contains the status of a property.

```
CREATE TABLE PropertyStatus
(
  PropertyStatus char(1) NOT NULL,
  Description varchar(50),
  PRIMARY KEY (PropertyStatus)
);
```

Functional Dependencies
PropertyStatus → Description

	propertystatus character(1)	description character varying(50)
1	O	Open
2	C	Closed

OfferStatus Table

This table contains the status of an offer.

```
CREATE TABLE OfferStatus
(
  OfferStatus char(1) NOT NULL,
  Description varchar(50),
  PRIMARY KEY (OfferStatus)
);
```

Functional Dependencies
OfferStatus → Description

	offerstatus character(1)	description character varying(50)
1	A	Accepted
2	R	Rejected

Agents_Offers Table

This table contains the agent with the listing they worked on and their fee from the buyer.

```
CREATE TABLE Agents_Offers
(
  AID INTEGER references Agents(AID) NOT NULL,
  OID INTEGER references Offers(OID) NOT NULL,
  BuyerFee INTEGER,
  PRIMARY KEY (AID, OID)
);
```

Functional Dependencies
AID, OID → BuyerFee

	aid integer	oid integer	buyerfee integer
1	1	1	800
2	5	2	600
3	1	3	1000
4	8	4	1600

Agents_ListedProperties Table

This table contains the agent with the listing they worked on and their comission on this listing if there is any.

```
CREATE TABLE Agents_ListedProperties
(
  AID INTEGER references Agents(AID) NOT NULL,
  LID INTEGER references ListedProperties(LID) NOT NULL,
  Comission INTEGER NOT NULL,
  PRIMARY KEY (AID, LID)
);
```

Functional Dependencies
AID, LID → Comission

	aid integer	lid integer	comission integer
1	1	1	800
2	5	2	1200
3	8	3	1000
4	5	4	600
5	1	5	700

Sellers_ListedProperties Table

```
CREATE TABLE Sellers_ListedProperties
(
  PID INTEGER references Sellers(PID) NOT NULL,
  LID INTEGER references ListedProperties(LID) NOT NULL,
  PRIMARY KEY (PID, LID)
);
```

Functional Dependencies
PID, LID →

	pid integer	lid integer
1	7	1
2	4	2
3	11	3
4	2	4
5	9	5

Properties Table

This table contains different information about a property.

```
CREATE TABLE Properties
(
  PROPID serial NOT NULL,
  PropertyInfoID int NOT NULL REFERENCES PropertyInfo(PropertyInfoID),
  address varchar (255) NOT NULL,
  ZipCodeNumber varchar(5) NOT NULL REFERENCES ZipCodes
(ZipCodeNumber),
  sizeSqFoot int NOT NULL,
  PRIMARY KEY (PROPID)
);
```

Functional Dependencies
PROPID → PropertyInfoID, address, ZipCodeNumber, sizeSqFoot

	propid integer	propertyinfoid integer	address character varying(255)	zipcodenumber character varying(5)	sizesqfoot integer
1	1	1	14 Main Street	94101	2000
2	2	2	222 Acorn Street	90001	1759
3	3	3	68 Jarvis Street	94101	1593
4	4	4	496 Macon Road	90209	5000
5	5	5	167 Mountain Place	90401	1700
6	6	6	426 Dell Drive	92101	2350
7	7	7	745 Castle Lane	90209	2200

PropertyInfo Table

This table lists information about a property. It puts it all under an ID so that it can be used multiple times across many properties.

```
CREATE TABLE propertyInfo
(
  PropertyInfoID serial NOT NULL,
  bedrooms FLOAT(1),
  bathrooms FLOAT(1),
  yearbuilt char(4),
  airconditioning boolean,
  fireplace char(2),
  floorCovering varchar(50),
  basement boolean,
  parking varchar(100),
  features varchar(255),
  other varchar(255),
  PRIMARY KEY (PropertyInfoID)
);
```

Functional Dependencies
PropertyInfoID → bedrooms, bathrooms, yearbuilt, airconditioning, fireplace, floorCovering, basement, parking, features, other

	propertyinfoid integer	bedrooms real	bathrooms real	yearbuilt character(4)	airconditioning boolean	fireplace character(2)	floorcovering character varying(50)	basement boolean	parking character varying(100)	features character varying(255)	other character varying(255)
1	1	3	1	1956	t	1	Wood	t	Yes, one spot	Pool	None
2	2	2	2.5	1987	t	0	Carpet	t	No Parking	Waterfall in Entrance	None
3	3	1.5	3	1956	f	0	Wood	f	No Parking	No Special Features	None
4	4	0	1	1923	f	0	Grass	f	City Street Parking	Old Bestbuy	None
5	5	1.5	2	1958	t	1	Tile	f	Yes, garage	Fully Furnished	None
6	6	1	2	1920	t	1	Carpet	t	Yes Four Car Garage	Pool	Busy Street
7	7	2.5	1.5	2000	t	1	Tile	t	No Parking	No Special Features	Far from school

ListedPropertyInfo Table

```
CREATE TABLE ListedPropertyInfo
(
  LID int NOT NULL REFERENCES ListedProperties(LID),
  PROPID int NOT NULL REFERENCES Properties(PROPID),
  PRIMARY KEY (LID, PROPID)
);
```

Functional Dependencies
PROPID, LID→

	lid integer	propid integer
1	1	1
2	2	2
3	3	5
4	4	7
5	5	5

OpenProperties report

This report is to show what listings are currently open.

```
SELECT *
 FROM ListedProperties
 WHERE propertystatus = 'O'
```

	lid integer	askingprice integer	estimatedmorgage integer	datelisted date	propertystatus character(1)
1	3	220000	2000	2011-03-25	O
2	4	300000	1750	2013-07-22	O
3	5	338000	1650	2012-05-09	O

current employees report

This report is to show what agents still employed.

```
SELECT DISTINCT p.firstName, p.lastName, a.
dateStarted
FROM people p, agents a
WHERE a.dateAgentLeft is NULL
AND
p.pid = a.aid;
```

	firstname character varying(30)	lastname character varying(30)	datestarted date
1	John	Brown	2010-06-12
2	Joe	Lyktey	2011-03-08

Buyerpricerange view

This view is to see what listings are 30,000 dollars above and below the users preferred price.

```
DROP VIEW IF EXISTS buyerpricerange;
CREATE VIEW buyerpricerange as
SELECT  DISTINCT
        p.firstname,
        p.lastname,
        p.middleInitial,
        b.pid,
        b.maxPay,
        lp.LID,
        lp.askingPrice
FROM    buyers b LEFT OUTER JOIN offers o ON b.pid = o.pid,
        ListedProperties lp LEFT OUTER JOIN offers o2 ON lp.lid = o2.lid,
        people p LEFT OUTER JOIN buyers b2 ON b2.pid = p.pid,
        buyers b3
WHERE   (lp.askingPrice BETWEEN (b.maxPay - 30000) AND (b.maxPay + 30000))
        AND
        b.pid = o.pid
        AND
        p.pid = b.pid
```

OfferHistory view

This view is to show the history of all previous offers, this includes the price offered and who offered it.

```
DROP VIEW IF EXISTS offerhistory;
CREATE VIEW offerhistory as
SELECT o.pid,
       p.firstName,
       p.lastName,
       p.middleInitial,
       o.oid,
       o.lid,
       ao.aid,
       o.offerPrice,
       o.offerstatus,
       o.dateOffered,
       o.dateclosed
FROM   offers o, people p, buyers b, agents_Offers ao
WHERE  p.pid = b.pid
       and
       b.pid = o.pid
       and
       ao.oid = o.oid;
```

Housing General Info Stored Procedure

This procidure will return a table worth of general information about a certain listing

```
CREATE OR REPLACE FUNCTION listingInfo (lid int)
returns table (askingPrice int, propertystatus char, sizeSqFoot int, address varchar, zipCodeNumber char) as $$
        SELECT
                askingPrice, propertystatus, sizeSQFoot, address, zipCodeNumber
        FROM
                listedProperties, properties, ListedPropertyInfo
        WHERE
                ListedProperties.lid = listedpropertyinfo.lid
                AND
                listedpropertyinfo.propid = properties.propid
        $$ language 'sql';
```

updateofferdate Trigger

This Trigger gives an inserted accepted order that has a closedDate thats null will change the date from null to the current date.

```
CREATE OR REPLACE FUNCTION updateofferdate()
RETURNS TRIGGER AS $updateofferdate$
  BEGIN
    IF (new.offerstatus = 'A' and new.dateClosed is NULL)
    THEN
          new.dateClosed = CURRENT_DATE;
   END IF;
   RETURN NEW ;
   END ;
   $updateofferdate$ LANGUAGE plpgsql;

CREATE TRIGGER updatelistings
    BEFORE INSERT ON offers
    FOR EACH ROW
    EXECUTE PROCEDURE updateofferdate();
```

updatelistingstatus Trigger

This Trigger changes the status of an active listing to closed when an inserted order is accepted on that listing.

```
CREATE OR REPLACE FUNCTION updatelistingstatus()
RETURNS TRIGGER AS $updatelistingstatus$
 BEGIN
   IF (new.offerstatus = 'A')
   THEN
          UPDATE listedproperties
          SET propertystatus = 'C'
          WHERE new.lid = listedproperties.lid;
  END IF;
  RETURN NEW ;
  END ;
  $updatelistingstatus$ LANGUAGE plpgsql;

CREATE TRIGGER updatelistingsstatus
   BEFORE INSERT ON offers
   FOR EACH ROW
   EXECUTE PROCEDURE updatelistingstatus();
```

Security ~ Admin Level

These Privileges will be for the admin, which would be the head of the company. They will have access to every table. They would also be able to use the current employees report to see who still works for them.

```
REVOKE ALL PRIVIEGES ON ZipCodes FROM admin;
REVOKE ALL PRIVIEGES ON propertyInfo FROM admin;
REVOKE ALL PRIVIEGES ON Properties FROM admin;
REVOKE ALL PRIVIEGES ON people FROM admin;
REVOKE ALL PRIVIEGES ON Buyers FROM admin;
REVOKE ALL PRIVIEGES ON PropertyStatus FROM admin;
REVOKE ALL PRIVIEGES ON ListedProperties FROM admin;
REVOKE ALL PRIVIEGES ON ListedPropertyInfo FROM admin;
REVOKE ALL PRIVIEGES ON OfferStatus FROM admin;
REVOKE ALL PRIVIEGES ON Sellers FROM admin;
REVOKE ALL PRIVIEGES ON Agent_Type FROM admin;
REVOKE ALL PRIVIEGES ON Agents FROM admin;
REVOKE ALL PRIVIEGES ON Offers FROM admin;
REVOKE ALL PRIVIEGES ON Sellers_ListedProperties FROM admin;
REVOKE ALL PRIVIEGES ON Agents_ListedProperties FROM admin;
REVOKE ALL PRIVIEGES ON Agents_Offers FROM admin;

GRANT INSERT, SELECT, UPDATE , DELETE on ZipCodes to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on propertyInfo to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Properties to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on people to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Buyers to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on PropertyStatus to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on ListedProperties to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on ListedPropertyInfo to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on OfferStatus to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Sellers to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Agent_Type to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Agents to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on offers to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Sellers_ListedProperties to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Agents_ListedProperties to Admin
GRANT INSERT, SELECT, UPDATE , DELETE on Agents_Offers to Admin
```

Security ~Agents Level

Agents should have good control over the database but not complete. They should not be able to delete most things. That should be for the admin only. They also have access to the order history view. They cannot have access to update the agents table as they could increase their salary.

```
REVOKE ALL PRIVIEGES ON ZipCodes FROM agents;
REVOKE ALL PRIVIEGES ON propertyInfo FROM agents;
REVOKE ALL PRIVIEGES ON Properties FROM agents;
REVOKE ALL PRIVIEGES ON people FROM agents;
REVOKE ALL PRIVIEGES ON Buyers FROM agents;
REVOKE ALL PRIVIEGES ON PropertyStatus FROM agents;
REVOKE ALL PRIVIEGES ON ListedProperties FROM agents;
REVOKE ALL PRIVIEGES ON ListedPropertyInfo FROM agents;
REVOKE ALL PRIVIEGES ON OfferStatus FROM agents;
REVOKE ALL PRIVIEGES ON Sellers FROM agents;
REVOKE ALL PRIVIEGES ON Agent_Type FROM agents;
REVOKE ALL PRIVIEGES ON Agents FROM agents;
REVOKE ALL PRIVIEGES ON Offers FROM agents;
REVOKE ALL PRIVIEGES ON Sellers_ListedProperties FROM agents;
REVOKE ALL PRIVIEGES ON Agents_ListedProperties FROM agents;
REVOKE ALL PRIVIEGES ON Agents_Offers FROM agents;

GRANT INSERT, SELECT, UPDATE on ZipCodes to agents
GRANT INSERT, SELECT, UPDATE on propertyInfo to agents
GRANT INSERT, SELECT, UPDATE on Properties to agents
GRANT INSERT, SELECT, UPDATE on people to agents
GRANT INSERT, SELECT, UPDATE on Buyers to agents
GRANT INSERT, SELECT UPDATE on PropertyStatus to agents
GRANT INSERT, SELECT, UPDATE on ListedProperties to agents
GRANT INSERT, SELECT, UPDATE , DELETE on ListedPropertyInfo to agents
GRANT INSERT, SELECT on OfferStatus to agents
GRANT INSERT, SELECT, UPDATE on Sellers to agents
GRANT INSERT, SELECT on Agent_Type to agents
GRANT INSERT, SELECT on Agents to agents
GRANT INSERT, SELECT, UPDATE  on offers to agents
GRANT INSERT, SELECT, UPDATE on Sellers_ListedProperties to agents
GRANT INSERT, SELECT, UPDATE on Agents_ListedProperties to agents
GRANT INSERT, SELECT, UPDATE on Agents_Offers to agents
```

Security ~Buyers/Sellers Level

Buyers/Sellers should only be able to select a limited amount of tables. They should not be able to look at tables like people as that is information that should not be available to them. Buyers can also see the buyerpricerange view.

```
REVOKE ALL PRIVIEGES ON ZipCodes FROM buyers;
REVOKE ALL PRIVIEGES ON propertyInfo FROM buyers;
REVOKE ALL PRIVIEGES ON Properties FROM buyers;
REVOKE ALL PRIVIEGES ON people FROM buyers;
REVOKE ALL PRIVIEGES ON Buyers FROM buyers;
REVOKE ALL PRIVIEGES ON PropertyStatus FROM buyers;
REVOKE ALL PRIVIEGES ON ListedProperties FROM buyers;
REVOKE ALL PRIVIEGES ON ListedPropertyInfo FROM buyers;
REVOKE ALL PRIVIEGES ON OfferStatus FROM buyers;
REVOKE ALL PRIVIEGES ON Sellers FROM buyers;
REVOKE ALL PRIVIEGES ON Agent_Type FROM buyers;
REVOKE ALL PRIVIEGES ON Agents FROM buyers;
REVOKE ALL PRIVIEGES ON Offers FROM buyers;
REVOKE ALL PRIVIEGES ON Sellers_ListedProperties FROM buyers;
REVOKE ALL PRIVIEGES ON Agents_ListedProperties FROM buyers;
REVOKE ALL PRIVIEGES ON Agents_Offers FROM buyers;

GRANT SELECT on ZipCodes to buyers
GRANT SELECT on propertyInfo to buyers
GRANT SELECT on Properties to buyers
GRANT SELECT on Buyers to buyers
GRANT SELECT on ListedProperties to buyers
GRANT SELECT on offers to buyers
GRANT SELECT on listedpropertyinfo to buyers
GRANT SELECT on Agents_Offers to buyers
```

```
REVOKE ALL PRIVIEGES ON ZipCodes FROM Sellers;
REVOKE ALL PRIVIEGES ON propertyInfo FROM Sellers;
REVOKE ALL PRIVIEGES ON Properties FROM Sellers;
REVOKE ALL PRIVIEGES ON people FROM Sellers;
REVOKE ALL PRIVIEGES ON Buyers FROM Sellers;
REVOKE ALL PRIVIEGES ON PropertyStatus FROM Sellers;
REVOKE ALL PRIVIEGES ON ListedProperties FROM Sellers;
REVOKE ALL PRIVIEGES ON ListedPropertyInfo FROM Sellers;
REVOKE ALL PRIVIEGES ON OfferStatus FROM Sellers;
REVOKE ALL PRIVIEGES ON Sellers FROM Sellers;
REVOKE ALL PRIVIEGES ON Agent_Type FROM Sellers;
REVOKE ALL PRIVIEGES ON Agents FROM Sellers;
REVOKE ALL PRIVIEGES ON Offers FROM Sellers;
REVOKE ALL PRIVIEGES ON Sellers_ListedProperties FROM
Sellers;
REVOKE ALL PRIVIEGES ON Agents_ListedProperties FROM
Sellers;
REVOKE ALL PRIVIEGES ON Agents_Offers FROM Sellers;

GRANT  SELECT on ZipCodes to Sellers
GRANT SELECT on propertyInfo to Sellers
GRANT SELECT on Properties to Sellers
GRANT SELECT on ListedProperties to Sellers
GRANT SELECT on offers to Sellers
GRANT SELECT on listedpropertyinfo to Sellers
GRANT SELECT on Agents_ListedProperties to Sellers
GRANT SELECT on Sellers_ListedProperties to Sellers
```

Implementation Notes

The implementation of this database would not only take time but also a decent amount of money. There would be a large amount of time that should go into inserting all of the previous offers on a house as well as all the active listings at the time. This database could very quickly get very large so it could cost some money for some more space.

Known Problems

The commission of an accepted offer will remain stationary until it just updated through a query, unlike buyerFee the commission does not come immediately with a new offer, it only comes with an accepted offer. There needs to be a trigger that does this calculation when an accepted offer is inputted.

Future Enhancements

-The database will definitely expand so there will be a need for more space for tables likes offers, propertylistings, and propertyinfo.
- Later it would be nice to include a table that gets the average price for a certain area or zip code so the buyer can use their price point to see what neighbourhoods best fit them.