Used Car Dealership Database

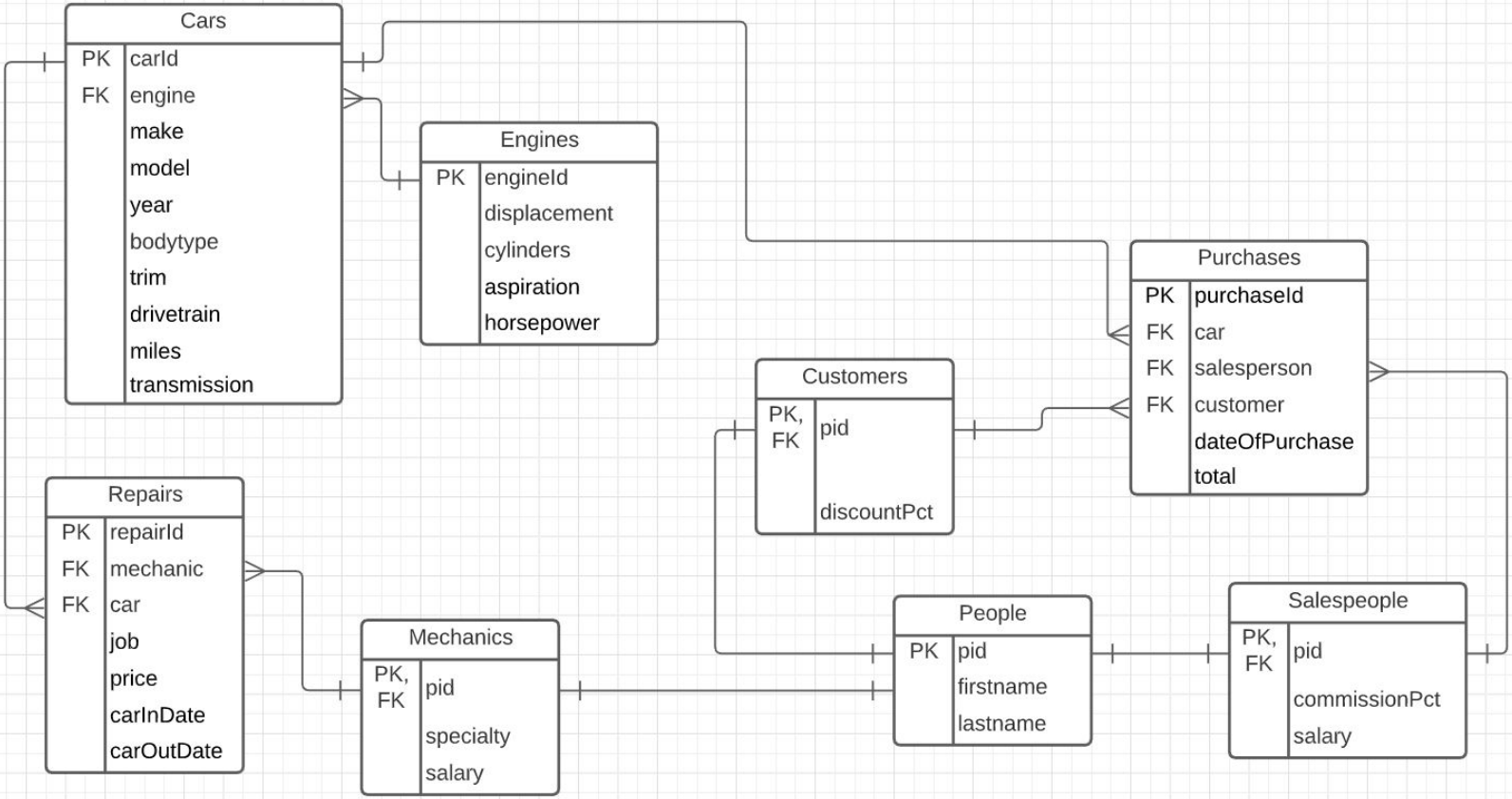
Matthew Medina

Table of Contents

Executive Summary

The goal of this design project is to create a database that accurately represents and functions as a database for a used car dealership. The database will allow the owner of the dealership to view, update, or delete cars, purchases, employees, customers and other information that pertains to the company. The database will be normalized to ensure future expandability and reliable long-term use.

Entity Relationship Diagram


Cars
PK carId
FK engine
make
model
year
bodytype
trim
drivetrain
miles
transmission
Engines
PK engineId
displacement
cylinders
aspiration
horsepower
Purchases
PK purchaseId
FK car
FK salesperson
FK customer
dateOfPurchase
total
Customers
PK, FK pid
discountPct
Repairs
PK repairId
FK mechanic
FK car
job
price
carInDate
carOutDate
Mechanics
PK, FK pid
specialty
salary
People
PK pid
firstname
lastname
Salespeople
PK, FK pid
commissionPct
salary

Tables

Engines

The engines table holds information about engines which will then be referenced by individual cars in the cars table.

Dependencies:

engineId -> liters, cylinders, aspiration, horsepower

```
create table Engines (
    engineId    integer not null,
    liters      float(2) not null,
    cylinders   integer not null,
    aspiration  text not null,
    horsepower  integer not null,
  primary key (engineId)
);
```

	engineid [PK] integer	liters real	cylinders integer	aspiration text	horsepower integer
1	1	3.2	6	NA	260
2	2	6.2	8	SC	707
3	3	2	4	Turbo	306

Cars

The cars table holds information about cars in the dealership. It has a primary key carId and a foreign key engine which references engineId.

Dependencies: carId-> engine, make, model, year, color, bodytype, trim, drivetrain, miles, transmission

```
create table Cars (
    carId       integer not null,
    engine      integer not null references Engines(engineId),
    make        text    not null,
    model       text    not null,
    year        integer not null,
    color       text    not null,
    bodytype    text    not null,
    trim        text    not null,
    drivetrain  text    not null,
    miles       integer not null,
    transmission text not null,
  primary key (carId)
);
```

	carid [PK] integer	engine integer	make text	model text	year integer	color text	bodytype text	trim text	drivetrain text	miles integer	transmission text
1	1	1	Acura	TL	2003	silver	sedan	Type-S	FWD	106000	Automatic
2	2	2	Dodge	Challenger	2015	black	coupe	Hellcat	RWD	5000	Manual
3	3	2	Dodge	Charger	2015	red	sedan	Hellcat	RWD	11000	Automatic
4	4	3	Honda	Civic	2020	white	hatchback	Type-R	FWD	9000	Manual

People

The people table simply holds the primary key pid and the names of the people in the database. Pid here has a one-to-one relationship with pid in Salespeople, Customers, and Mechanics

Dependencies:
firstname, lastname

pid ->

```
create table People (
    pid         integer not null,
    firstname   text not null,
    lastname    text not null,
  primary key (pid)
);
```

	pid [PK] integer	firstname text	lastname text
1	1	Matthew	Medina
2	2	Alan	Labouseur
3	3	Maggie	Hurst
4	4	Robert	Beringer
5	5	Enzo	Ferarri
6	6	Charles	Monette

Salespeople

The salespeople table has a primary key of pid which references pid in People. It includes extra information for people who are salespeople such as commission percentage and salary

Dependencies: pid -> commissionPct, salary

```
create table Salespeople (
    pid             integer not null references People(pid),
    commissionPct   integer not null,
    salary          integer not null,
  primary key (pid)
);
```

	pid [PK] integer	commissionpct integer	salary integer
1	1	10	60000
2	6	12	50000

Customers

The customers table takes pid from People as a foreign key and uses it as a primary key. The only column currently exclusive to customers is discountPct.

Dependencies: pid -> discountPct

```
create table Customers (
    pid         integer not null references People(pid),
    discountPct integer not null,
  primary key (pid)
);
```

	pid [PK] integer	discountpct integer
1	2	50
2	3	5

Mechanics

The mechanics table uses foreign key pid from the People table as it's primary key. The mechanics table shows us which people are mechanics as well as their specialties and salaries.

Dependencies:
specialty, salary

pid ->

```
create table Mechanics (
    pid         integer not null references People(pid),
    specialty   text not null,
    salary      integer not null,
  primary key (pid)
);
```

	pid [PK] integer	specialty text	salary integer
1	4	General	70000
2	5	Engines	100000

Purchases

The purchases table has an artificial primary key of purchaseId as well as foreign keys of car, salesperson, and customer. Additional information given is the date of the purchase and the total.

```
create table Purchases (
    purchaseId      integer not null,
    car             integer not null references Cars(carId),
    salesperson     integer not null references Salespeople(pid),
    customer        integer not null references Customers(pid),
    dateOfPurchase  date not null,
    total           integer not null,
  primary key (purchaseId)
);
```

Dependencies: purchaseId
-> car, salesperson, customer, dateOfPurchase, total

	purchaseid [PK] integer	car integer	salesperson integer	customer integer	dateofpurchase date	total integer
1	1	2	1	3	2017-09-23	53999
2	2	4	6	2	2020-11-02	32999

Repairs

The repairs table is a log of repairs done on cars in the dealership. The artificial primary key is repairId. Mechanic and car are foreign keys and the rest is information about the repair.

Dependencies: repairId -> mechanic, car, job, price, carInDate, carOutDate

```
create table Repairs (
    repairId    integer not null,
    mechanic    integer not null references Mechanics(pid),
    car         integer not null references Cars(carId),
    job         text not null,
    price       integer not null,
    carInDate   date not null,
    carOutDate  date not null,
  primary key (repairId)
);
```

	repairid [PK] integer	mechanic integer	car integer	job text	price integer	carindate date	caroutdate date
1	1	4	1	brakes	200	2019-02-13	2019-02-13
2	2	5	3	oil change	50	2020-04-14	2020-04-15

Views

Available Cars

This view allows available cars to be quickly selected by showing all cars which do not appear in the purchases table.

```
create or replace view available_cars as
select *
from cars
where carId not in ( select car
                     from Purchases );


select *
from available_cars;
```

	carid integer	engine integer	make text	model text	year integer	color text	bodytype text	trim text	drivetrain text	miles integer	transmission text
1	1	1	Acura	TL	2003	silver	sedan	Type-S	FWD	106000	Automatic
2	3	2	Dodge	Charger	2015	red	sedan	Hellcat	RWD	11000	Automatic

Employees

The view for employees allows you to easily query for people who are either mechanics or salespeople and therefore work for the dealership.

```
create or replace view employees as
select *
from people
where pid in ( select pid
               from mechanics )
or pid in ( select pid
            from salespeople );

select *
from employees;
```

	pid integer	firstname text	lastname text
1	1	Matthew	Medina
2	4	Robert	Beringer
3	5	Enzo	Ferarri
4	6	Charles	Monette

Stored Procedure

getCommission

This stored procedure getCommission calculates the amount of commission a salesperson earned in a purchase. The function parameters are an integer which will be the purchaseId and a refcursor. The function uses the given purchaseId to find the total and the commissionPct from the salespeople table to return the commission earned.

```
create or replace function getCommission(int, refcursor) returns refcursor as
$$
declare
    purchase    int         := $1;
    resultset   refcursor   := $2;
begin
    open resultset for
        select p.total * (.01 * s.commissionPct) as commissionEarned
        from Purchases p, Salespeople s
        where p.salesperson = s.pid and purchaseId = purchase;
    return resultset;
end;
$$
language plpgsql;

select * from getCommission(1, 'results');
fetch all from results;
```

Roles

Manager

The manager role is generally for higher ups who work for the dealership and need to interact with the database on a basic level. Managers can add cars, engines, purchases, and repairs to the database as well as update and delete them.

```
create role manager;

grant insert, select, update, delete
on cars, engines, purchases, repairs
to manager;
```

Public

The publicview role was made for the general public. This would be integrated with the dealership's website and allow people to view the cars, engines, and details on the page.

```
create role publicview;

grant select
on cars, engines
to publicview;
```

Implementation

Creating and working with this database was a challenge. The final result is a database that is normalized, fully functional, and ready for expansion if it's needed. There isn't currently much data in the database yet, but the tables will fill over time as the dealership does business.

What can be done better?

One section that could be done better in the database is the engines table. The solution right now has one engine in multiple cars which is meant to reflect how two cars can have the same model of engine. The main issue with this is even if two cars have the same engine, one could theoretically have performance mods that increase the horsepower, which would force us to create a new engine entry in the database. Other future improvements could see are adding more data to the people table for real world functionality and adding more tables like engines for more car parts.