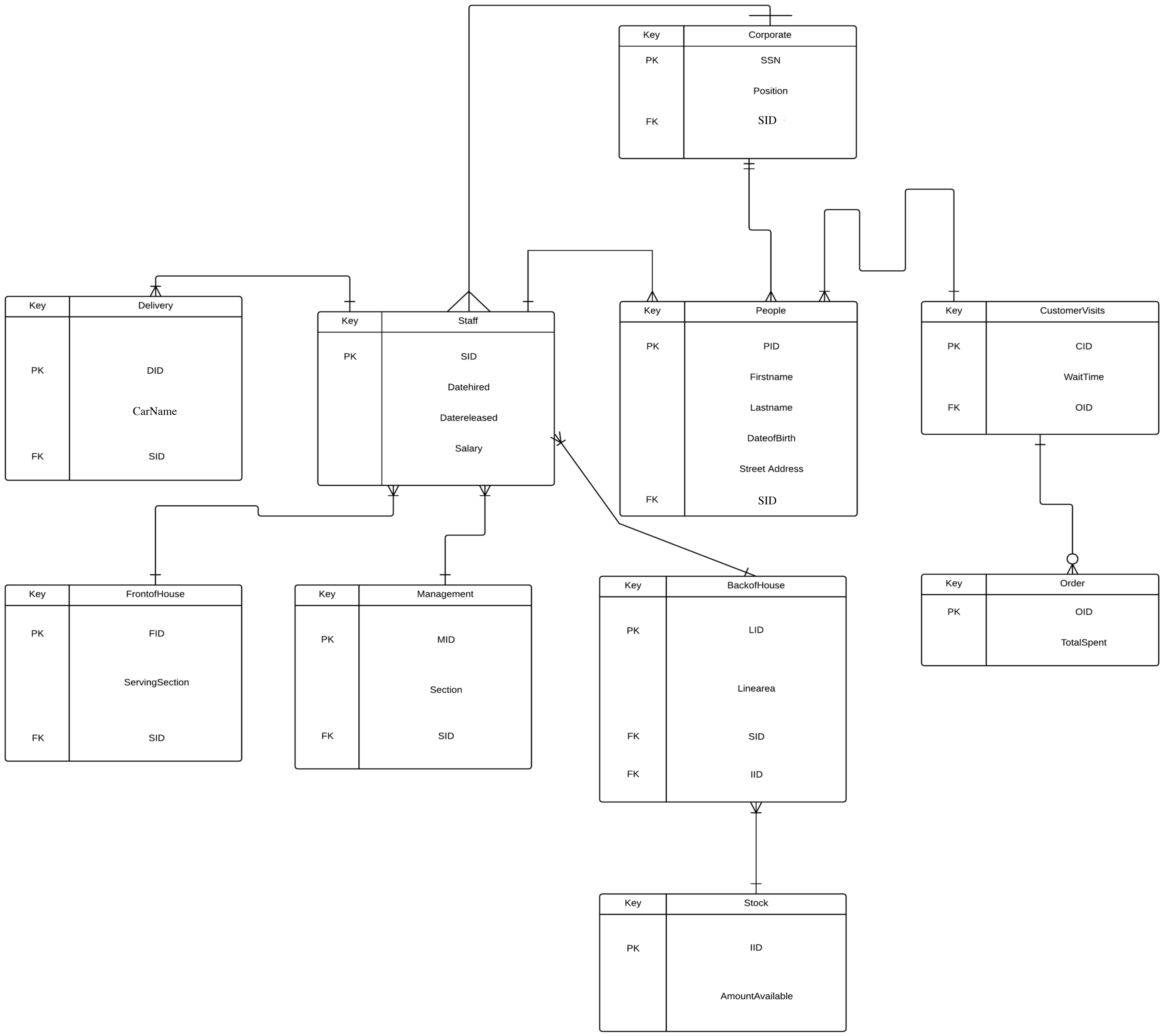
Database Design Proposal

By: Ryan Neumann

Table of Contents

Executive Summary

This document outlines the structure and entities involved in the design and implementation of owning a restaurant. The database organizes the restaurant into categories, which simplifies and assists employee's on what tasks need to be completed. Corporate will be able to see all new hires, both in the front and the back of the restaurant, as well as all their information, tasks, and the new employee's position.


Key
Corporate
PK
SSN
Position
FK
SID
Key
Delivery
PK
DID
CarName
FK
SID
Key
Staff
PK
SID
Datehired
Datereleased
Salary
Key
People
PK
PID
Firstname
Lastname
DateofBirth
Street Address
FK
SID
Key
CustomerVisits
PK
CID
WaitTime
FK
OID
Key
FrontofHouse
PK
FID
ServingSection
FK
SID
Key
Management
PK
MID
Section
FK
SID
Key
BackofHouse
PK
LID
Linearea
FK
SID
FK
IID
Key
Order
PK
OID
TotalSpent
Key
Stock
PK
IID
AmountAvailable

Table

Corporate

Purpose:
This table is created to store corporate positions along with the salary.

```
CREATE TABLE corporate (
      SSN int NOT NULL,
      DateHired date,
      DateReleased date,
      SID int NOT NULL,
  PRIMARY KEY(SSN),
  FOREIGN KEY(SID) references staff(SID)
);
```

Functional Dependencies
SSN -> DateHired, DateReleased, SID

Corporate Sample Data

	SSN integer	DateHired Timestamp without time zone	DateReleased Timestamp without time zone	SID integer
1	480466669	2013-01-12		1
2	501145494	2014-11-30		2
3	544900941	2014-12-08		3
4	449825815	2015-01-13		4
5	530719531	2015-10-03		5
6	127849320	2015-10-15		6
7	121309512	2015-11-03		7
8	901238410	2016-02-19		8

Table

Staff

Purpose:
This table is created to keep track of when employees are hired or let go.

```
CREATE TABLE staff (
      SID int NOT NULL,
      DateHired date,
      DateReleased date,
      Salary int NOT NULL,
  PRIMARY KEY(SID)
);
```

Functional Dependencies
SID -> DateHired, DateReleased, Salary

Staff Sample Data

	SID integer	DateHired Timestamp without time zone	DateReleased Timestamp without time zone	Salary integer
1	1	2013-01-12		195000
2	2	2014-11-30		127000
3	3	2014-12-08		101000
4	4	2015-01-13		94000
5	5	2015-10-03		142000
6	6	2015-10-15		42000
7	7	2015-11-03		57000
8	8	3		30000

Table

People

Purpose:
This table is created to list basic information about everyone who is employed at the company.

```
CREATE TABLE people (
      PID int NOT NULL,
      Firstname text,
      Lastname text,
      DateofBirth date,
      StreetAddress text,
      SID int NOT NULL,
  PRIMARY KEY(PID),
  FOREIGN KEY(SID) references staff(SID)
);
```

Functional Dependencies
PID -> Firstname, Lastname, DateofBirth, StreetAddress, SID

People Sample Data

	PID integer	Firstname text	Lastname text	DateofBirth date	StreetAddress text	SID integer
1	1001	Charles	Hersheberg	1964-11-21	291 Hunting Street	1
2	1002	Nick	Barnett	1977-12-08	31 Main Street	2
3	1003	Joe	Shmoe	1968-10-21	38 Brooklyn Blvd	3
4	1004	Amanda	Mctigue	1979-03-05	932 Fulton Street	4
5	1005	Sherla	Jermain	1981-12-06	54 Fuller Court	5
6	1006	Jerald	Bronze	1985-05-16	83 Bear Oak Lane	6
7	1007	Mason	Shaw	1992-06-18	339 Rock Oak Road	7
8	1008	Kayla	Marhefka	1987-03-12	1 Hampton Street	8
9	1009	Brian	Monahan	1990-04-20	301 Eisenhower Lane	9
10	1010	Carla	Sofia	1997-02-01	194 Red Oak Road	10
11	1011	Vlad	Donavan	1981-01-02	35 Pin Oak Road	11
12	1012	Ryan	Neumann	1995-11-30	38 Scarlet Drive	12

Table

CustomerVisits

Purpose:
This table was created to keep track of every customer, how long their estimated wait time is, as well as the order identification number.

```
CREATE TABLE customerVisits (
      CID int NOT NULL,
      WaitTime time NOT NULL,
      OID int NOT NULL,
  PRIMARY KEY(CID),
  FOREIGN KEY(OID) references order(OID)
);
```

Functional Dependencies
CID -> WaitTime, OID

CustomerVisits Sample Data

	CID **integer**	**WaitTime** **time without timezone**	**OID** **integer**
1	103213	11:15:32	101
2	103214	12:32:41	102
3	103215	12:47:51	103
4	103216	13:01:41	104
5	103217	13:16:18	105
6	103218	13:36:53	106
7	103219	14:09:10	107
8	103220	14:51:39	108

Table

Delivery

Purpose:
This table keeps track of all outgoing orders and which car is delivering.

```
CREATE TABLE delivery (
      DID int NOT NULL,
      CarName text,
      SID int NOT NULL,
  PRIMARY KEY(CID),
  FOREIGN KEY(SID) references staff(SID)
);
```

Functional Dependencies
DID -> CarName, SID

Delivery Sample Data

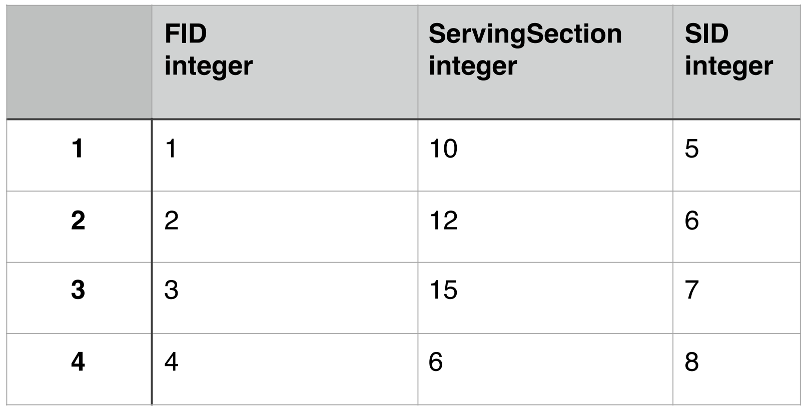
	DID integer	CarName text	SID integer
1	1	Nissan	12
2	2	Chevy	8
3	3	Chevy	10
4	4	Chevy	12
5	5	Nissan	10
6	6	Nissan	9
7	7	Nissan	8
8	8	Chevy	10

Table

FrontofHouse

Purpose:
This table is created for all waiters or waitresses, signaling which tables they will be serving and it what section.

```
CREATE TABLE frontofHouse (
      FID int NOT NULL,
      ServingSection int NOT NULL,
      SID int NOT NULL,
  PRIMARY KEY(FID),
  FOREIGN KEY(SID) references staff(SID)
);
```



	FID integer	ServingSection integer	SID integer
1	1	10	5
2	2	12	6
3	3	15	7
4	4	6	8

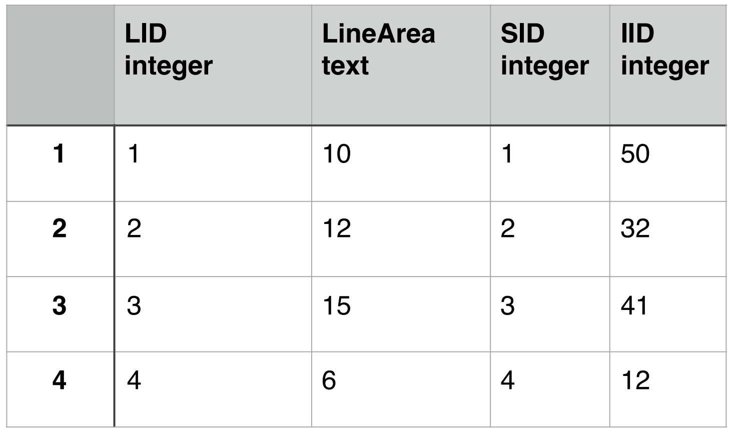
<u>Functional Dependencies</u>
FID -> ServingSection, SID

Table

BackofHouse

Purpose:
This table is created for everything that goes on behind the kitchen doors of restaurant(stock, chefs, and assembly).

```
CREATE TABLE backofHouse (
      LID int NOT NULL,
      LineArea text NOT NULL,
      SID int NOT NULL,
      IID int NOT NULL,
  PRIMARY KEY(LID),
  FOREIGN KEY(SID) references staff(SID),
  FOREIGN KEY(IID) references stock(IID)
);
```



	LID integer	LineArea text	SID integer	IID integer
1	1	10	1	50
2	2	12	2	32
3	3	15	3	41
4	4	6	4	12

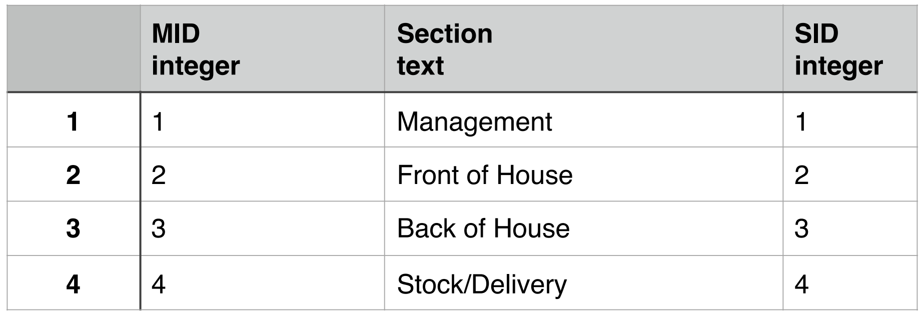
Functional Dependencies
LID -> LineArea, SID, IID

Table

Management

Purpose:
This table was created to keep track of all managers at the location. The section will designate who is in charge of what, along with what administrative rights come with that section.

```
CREATE TABLE management (
      MID int NOT NULL,
      section text,
      SID int NOT NULL,
  PRIMARY KEY(MID),
  FOREIGN KEY(SID) references staff(SID)
);
```



	MID integer	Section text	SID integer
1	1	Management	1
2	2	Front of House	2
3	3	Back of House	3
4	4	Stock/Delivery	4

<u>Functional Dependencies</u>
MID -> Section, SID

Table

Order

Purpose:
This table was created to keep track of orders and order costs. When a customer places an order, it will take the OID provided, match it with the OID in the order table, and get the corresponding order total(TotalSpent).

```
CREATE TABLE order (
      OID int NOT NULL,
      TotalSpent float(6,2),
  PRIMARY KEY(OID)
);
```

	OID integer	TotalSpent float(6,2)
1	101	121.42
2	102	75.56
3	103	24.98
4	104	309.61

<u>Functional Dependencies</u>
OID -> TotalSpent

Views

Order Total of the Current Customer

This view lists the complete current order, including wait time, along with the correct total for the customerID.

```
CREATE VIEW CurrentOrder AS
    SELECT *
    FROM customerVisits
    INNER JOIN order
    ON customerVisits.oid = order.oid
    ORDER BY oid DESC;
```

Views

Salary of Corporate Positions

This view lists the complete current order, including wait time, along with the correct total for the customerID.

```
CREATE VIEW CorporateSalary AS
    SELECT staff.salary, staff.sid, corporate.sid
    FROM staff
    INNER JOIN corporate
    ON staff.sid = corporate.sid
    ORDER BY sid DESC;
```

Reports

Average Wait Time per Orders

Purpose: Understanding the average wait time per order is extremely important when running a restaurant. Any improvements made will be directly reflected in the wait time.

```
SELECT ROUND(avg(WaitTime), 2) AS Avg_Wait_Time_Per_Order
FROM customerVisits;
```

Reports

Average Salary per Employee

Purpose: Salary is an extremely important statistic in a company, and an employee needs to be assured they will be paid properly. Over time, the average salary should increase, raising morale and confidence within the workplace.

```
SELECT ROUND(avg(staff.salary, 2) AS Avg_Salary_Per_Employee
FROM staff st,
        corporate co
WHERE co.sid = st.sid
ORDER BY Avg_Salary_Per_Employee DESC;
```

Stored Procedures

InsertNewOrder

Purpose: When an order is placed, a new entry is required in the database to get the cost and estimated wait time of an order.

Query:

```
CREATE OR REPLACE FUNCTION insertNewOrder()
Return trigger as
$$
    Begin
        IF NEW.cid IS NULL THEN
            RAISE EXCEPTION 'Invalid OrderID provided!'
        END IF;
        IF NEW.WaitTime IS NULL THEN
            RAISE EXCEPTION 'Can only update WaitTime if order exists'
        END IF;
        IF OLD.WaitTime IS NOT NULL THEN
            RAISE EXCEPTION 'The order has already been received!'
        END IF;
        INSERT INTO InsertNewOrder(cid, WaitTime)
                    VALUES(NEW.cid, ’30');
        RETURN NEW;
    End
$$ LANGUAGE plpgsql;
```

Stored Procedures

EmployeeReleased

Purpose: Whenever an employee is Fired/Let go, the staff table needs to be updated with the date they were released.

Query:

```
CREATE OR REPLACE FUNCTION employeeReleased()
RETURN trigger AS
$$
    BEGIN
        IF NEW.sid is NULL THEN
            RAISE EXCEPTION 'Invalid StaffID provided!'
        END IF;
        IF NEW.DateReleased IS NULL THEN
            RAISE EXCEPTION 'Can only update DateReleased on Staff table'
        END IF;
        IF OLD.DateReleased IS NOT NULL THEN
            RAISE EXCEPTION 'Employee already released.'
        END IF;
        INSERT INTO employeeReleased (sid, DateReleased)
                                    Values(NEW.sid, Date);
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

Triggers

newOrder

Purpose:

When a new order is placed, that means the order table needs to be updated. It will be updated with the ‘InsertNewOrder’ procedure.

Query:

```
CREATE TRIGGER newOrder
AFTER UPDATE ON order
     FOR EACH ROW EXECUTE PROCEDURE insertNewOrder();
```

Triggers

releaseEmployee

Purpose:

When a staff member is released, the staff table needs to be updated to add the correct DateReleased to the appropriate cell.

Query:

```
CREATE TRIGGER releaseEmployee
AFTER UPDATE ON staff
    FOR EACH ROW EXECUTE PROCEDURE employeeReleased();
```

Security

Customers:

Customers indirectly interact with the database whenever a new order is placed.

```
GRANT INSERT ON orders TO customerVisits;
```

Management:

Management will need to update any table, so all access is granted to them.

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO
management;
```

Implementation Notes

The following are suggestions and/or requirements for implementation:

- CustomerVisits table can be updated to include orderTotal. This would eliminate the order table, however, this is left at the moment in case of adjustments and/or additions.
- CustomerVisits could also include a delivery column, which would be a boolean, signaling whether the customer placed the order to eat in or take out.
- Stock table can be improved, however, as not many items have been added, it's usability will increase over time.
- FrontofHouse only has table sections that can be assigned. It would be ideal to number/name all tables, and assign a group to the server, which would easily allow adding/removing table(s) from the server's ServingSection.

Known Problems

It is known that this database design possesses the following undesirable traits:

- At the moment, only the owner has full administrative rights. This may not be a flaw, however if addition management members need permission, this may present a problem.
- At the moment, the customer's total displays the full amount. However, if a customer would like to split a bill between two people, a problem may occur.
- Only one server is designated to a unique area, however, what if servers are designated to a multiple sections or multiple tables? The database does not recognize this.

Future Enhancement

Some features and functionalities that might be desirable in the future:

- Section the WaitTime from the CustomerVisits table into two sections: OrderStarted and OrderReceived.
- Allow server's to serve more than one section at a time.
- Combine Order and CustomerVisits into one table.
- Organize areas of Front and Back of the house to be more specific, which in turn would drastically improve the database as a whole.
- Implement an OrderCancelled section into the Orders table, in the case that the customer cancelled their order.