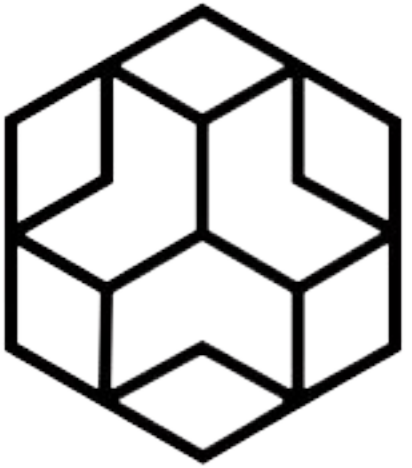
Database Design Proposal

Christopher Lee



BLUME

Table of Contents

Executive Summary

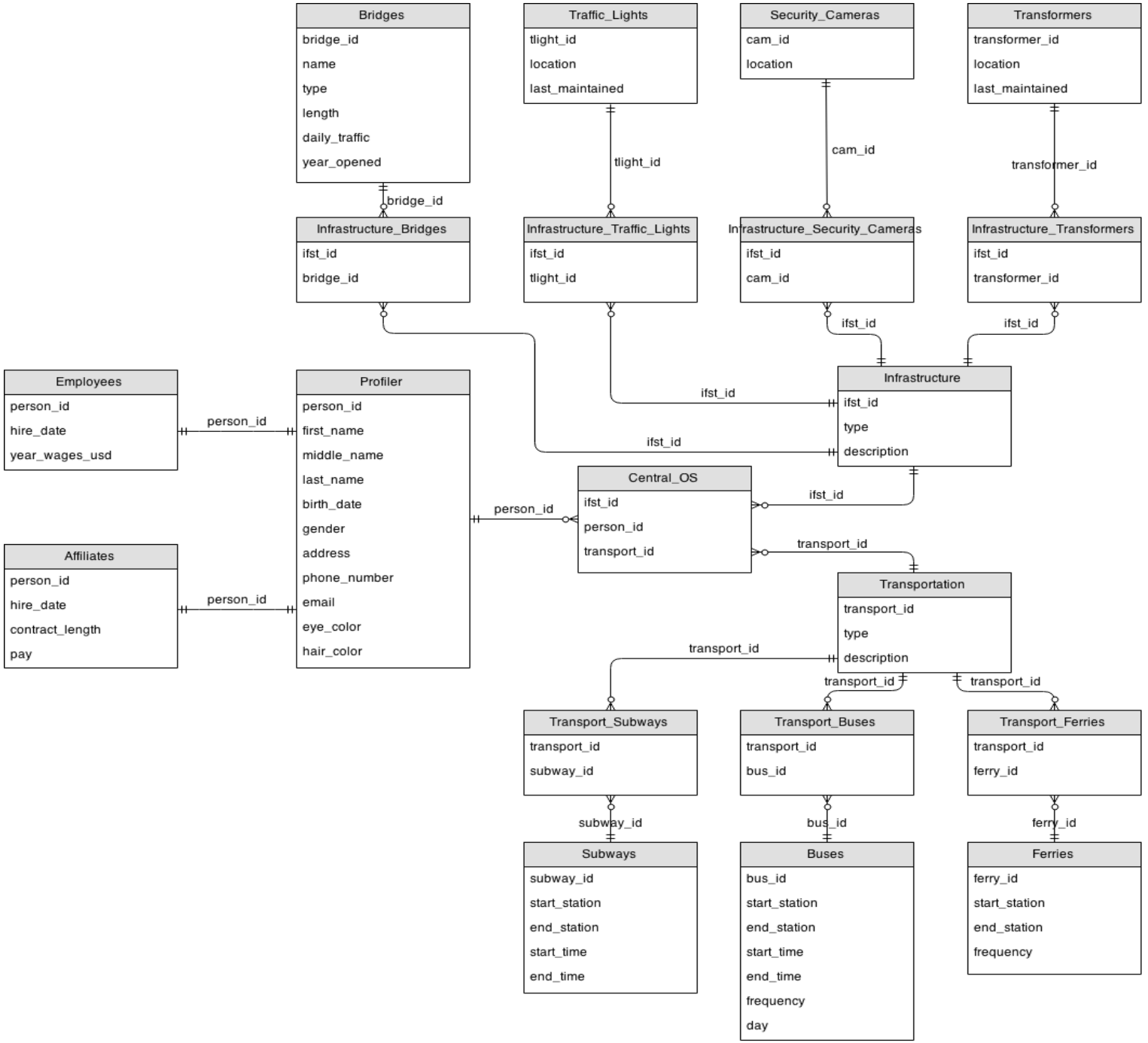
The Central Operating System (ctOS) is responsible for the management and facilitation of the city of Chicago and its over 2.7 million citizens. The city of Chicago requires a database to catalogue the various functionalities controlled by the operating system. Due to federal regulations, the data must be accurate and consistent.

This document outlines the structure and entities involved in the design and implementations of a database system for ctOS. The purpose of this database is to enable cataloging of the various functionalities of the operating system such as management of the roadways, the electric grid, the subway system, the security camera system, the citizen Profiler, and more.

This database will allow administration to create useful information from queries that provide valuable statistics and other facts from the catalogued data.

An overview of the database will be presented, followed by the details of every individual database table for each of the systems managed by ctOS. Purposes of each table will be suggested and triggers will be explained to reinforce the data integrity of the database. For each of the individual parts, sample reports will be shown.

This design was targetted for and tested on PostgreSQL 9.4.1, released on Feb 5, 2015.


Bridges
bridge_id
name
type
length
daily_traffic
year_opened
Traffic_Lights
tlight_id
location
last_maintained
Security_Cameras
cam_id
location
Transformers
transformer_id
location
last_maintained
bridge_id
tlight_id
cam_id
transformer_id
Infrastructure_Bridges
ifst_id
bridge_id
Infrastructure_Traffic_Lights
ifst_id
tlight_id
Infrastructure_Security_Cameras
ifst_id
cam_id
Infrastructure_Transformers
ifst_id
transformer_id
ifst_id
ifst_id
Employees
person_id
hire_date
year_wages_usd
Profiler
person_id
first_name
middle_name
last_name
birth_date
gender
address
phone_number
email
eye_color
hair_color
Infrastructure
ifst_id
type
description
ifst_id
ifst_id
person_id
Central_OS
ifst_id
person_id
transport_id
ifst_id
transport_id
person_id
Affiliates
person_id
hire_date
contract_length
pay
person_id
Transportation
transport_id
type
description
transport_id
transport_id
transport_id
Transport_Subways
transport_id
subway_id
Transport_Buses
transport_id
bus_id
Transport_Ferries
transport_id
ferry_id
subway_id
bus_id
ferry_id
Subways
subway_id
start_station
end_station
start_time
end_time
Buses
bus_id
start_station
end_station
start_time
end_time
frequency
day
Ferries
ferry_id
start_station
end_station
frequency

Infrastructure Table: Stores valid, unique identification numbers for the city's various infrastructure such as bridges, traffic lights, electric grid, etc.

```
CREATE TABLE IF NOT EXISTS infrastructure (
    ifst_id          SERIAL           NOT NULL UNIQUE,
    type             VARCHAR(25)      NOT NULL,
    description      VARCHAR(50)      NOT NULL,
    PRIMARY KEY (ifst_id)
);
```

Functional Dependencies

```
ifst_id -> type, description
```

ifst_id	type	description
1	Bridge	Washington Bridge
2	Traffic Light	Brown St and Park Ave
3	Transformer	152 Pensacola St
4	Security Camera	29 Myers Rd
5	Traffic Light	Atkins St and Bay Ave
6	Bridge	Bayview Bridge
7	Transformer	2 Blake Ct

Bridges Table: Contains the list of bridges crossing the Chicago River. Type refers to the bridge's structure or any other significant descriptions.

```
CREATE TABLE IF NOT EXISTS bridges (
    bridge_id        SERIAL           NOT NULL,
    name             VARCHAR(50)      NOT NULL,
    type             VARCHAR(25)      NOT NULL,
    length           VARCHAR(25)      NOT NULL,
    daily_traffic    INTEGER          NOT NULL,
    year_opened      INTEGER          NOT NULL,
    PRIMARY KEY (bridge_id)
);
```

Fuctional Dependencies

```
bridge_id -> name, type, length, daily_traffic, year_opened
```

bridge_id	name	type	length	daily_traffic	year_opened
1	Michigan Avenue Bridge	bascule	339 ft	49600	1920
2	La Salle Street Bridge	bascule	242 ft	12050	1928
3	Nichols Bridgeway	pedestrian	620 ft	8200	2009
4	Clark Street Bridge	bascule	346 ft	72830	1929
5	BP Pedestrian Bridge	pedestrian	935 ft	17890	2004
6	Outer Drive Bridge	bascule	480 ft	40000	1937
7	Sky Ride	ferry	3200 ft	65000	1933
8	Kinzie Street Bridge	bascule	196 ft	0	1908

Traffic_Lights Table

```
CREATE TABLE IF NOT EXISTS traffic_lights (
    tlight_id        SERIAL           NOT NULL,
    location         VARCHAR(50)      NOT NULL,
    last_maintained  DATE             NOT NULL,
    PRIMARY KEY (tlight_id)
);
```

Functional Dependencies

```
tlight_id -> location, last_maintained
```

tlight_id	location	last_maintained
1	Brown St and Park Ave	2007-04-27
2	N Kennedy St and Fairbanks Ct	2009-06-16
3	Meyer Ave and Damien Ave	2010-07-02
4	Atkins St and Bay Ave	2008-02-09
5	W 38 St and Kemper Pl	2013-05-18
6	N Emmett St and Felton Ave	2015-05-01
7	S Independence Blvd and 29 St	2014-12-12

Security_Cameras Table: The location of security cameras are usually in intersections, but can be located at the end of certain streets. In this situation, the precise address is recorded.

```
CREATE TABLE IF NOT EXISTS security_cameras (
    cam_id           SERIAL           NOT NULL,
    location         VARCHAR(50)      NOT NULL,
    last_maintained  DATE             NOT NULL,
    PRIMARY KEY (cam_id)
);
```

Functional Dependencies

```
cam_id -> location, last_maintained
```

cam_id	location	last_maintained
1	S Ingleside Ave and Raven Rd	2014-03-17
2	492 Bandle Pl	2007-05-11
3	Lawndale Ave and Princeton Ave	2010-07-02
4	Quinn St and S Prospect Ave	2009-01-02
5	Atkins St and Bay Ave	2011-05-16
6	12 W 89th St	2008-11-01
7	N Kennedy St and Fairbanks Ct	2013-10-12

Transformers Table

```
CREATE TABLE IF NOT EXISTS transformers (
    transformer_id   SERIAL           NOT NULL,
    location         VARCHAR(50)      NOT NULL,
    last_maintained  DATE             NOT NULL,
    PRIMARY KEY (transformer_id)
);
```

Functional Dependencies

```
transformer_id -> location, last_maintained
```

transformer_id	location	last_maintained
1	42 Riverside Rd	2015-02-14
2	132 Park Ave	2013-12-03
3	4 N Kennedy St	2008-09-12
4	92 Emmett St	2010-10-10
5	1439 Atkins St	2011-04-20
6	2 W 89th St	2009-09-11
7	50 Fairbanks Ct	2012-05-01

Infrastructure_Bridges Table

```
CREATE TABLE IF NOT EXISTS infrastructure_bridges (
    ifst_id          INTEGER          NOT NULL,
    bridge_id        INTEGER          NOT NULL,
    PRIMARY KEY (ifst_id, bridge_id),
    FOREIGN KEY (ifst_id)    REFERENCES infrastructure(ifst_id),
    FOREIGN KEY (bridge_id)  REFERENCES bridges(bridge_id)
);
```

Functional Dependencies

```
ifst_id -> bridge_id
```

ifst_id	bridge_id
1	1
6	2

Infrastructure_Traffic_Lights Table

```
CREATE TABLE IF NOT EXISTS infrastructure_traffic_lights (
    ifst_id          INTEGER          NOT NULL,
    tlight_id        INTEGER          NOT NULL,
    PRIMARY KEY (ifst_id, tlight_id),
    FOREIGN KEY (ifst_id)    REFERENCES infrastructure(ifst_id),
    FOREIGN KEY (tlight_id)  REFERENCES traffic_lights(tlight_id)
);
```

Functional Dependencies

```
ifst_id -> tlight_id
```

ifst_id	tlight_id
2	1
5	2

Infrastructure_Security_Cameras Table

```
CREATE TABLE IF NOT EXISTS infrastructure_security_cameras (
    ifst_id          INTEGER          NOT NULL,
    cam_id           INTEGER          NOT NULL,
    PRIMARY KEY (ifst_id, cam_id),
    FOREIGN KEY (ifst_id)    REFERENCES infrastructure(ifst_id),
    FOREIGN KEY (cam_id)     REFERENCES security_cameras(cam_id)
);
```

Functional Dependencies

```
ifst_id -> cam_id
```

ifst_id	cam_id
4	1

Infrastructure_Transformers Table

```
CREATE TABLE IF NOT EXISTS infrastructure_transformers (
    ifst_id          INTEGER          NOT NULL,
    transformer_id   INTEGER          NOT NULL,
    PRIMARY KEY (ifst_id, transformer_id),
    FOREIGN KEY (ifst_id)        REFERENCES infrastructure(ifst_id),
    FOREIGN KEY (transformer_id) REFERENCES transformers(transformer_id)
);
```

Functional Dependencies

```
ifst_id -> transformer_id
```

ifst_id	transformer_id
3	1
7	2

Transportation Table: Stores valid, unique identification numbers for the city's various transportation such as subway, ferry and buses.

```
CREATE TABLE IF NOT EXISTS transportation (
    transport_id    SERIAL          NOT NULL UNIQUE,
    type            VARCHAR(25)     NOT NULL,
    description     VARCHAR(50),
    PRIMARY KEY (transport_id)
);
```

Functional Dependencies

```
transport_id -> type, description
```

transport_id	type	description
1	Bus	
2	Subway	Irving Park - Belmont
3	Ferry	
4	Bus	
5	Bus	
6	Subway	Racine – Forest Park

Subways Table: Station determines where the subway starts and stops, route determines the time it takes for the subway to perform its route.

```
CREATE TABLE IF NOT EXISTS subways (
    subway_id        SERIAL           NOT NULL,
    start_station    VARCHAR(50)      NOT NULL,
    end_station      VARCHAR(50)      NOT NULL,
    start_time       TIME             NOT NULL,
    end_time         TIME             NOT NULL,
    frequency        VARCHAR(50)      NOT NULL,
    PRIMARY KEY (subway_id)
);
```

Functional Dependencies

```
subway_id -> start_station, end_station, start_time, end_time, frequency
```

subway_id	start_station	end_station	start_time	end_time	frequency
1	O'Hare	Logan Square	08:00:00	10:00:00	15 min
2	Irving Park	Belmont	09:00:00	10:00:00	12 min
3	Montrose	Jackson	07:00:00	11:00:00	18 min
4	Logan Square	Racine	11:00:00	14:00:00	13 min
5	Jackson	Harlem	12:30:00	15:00:00	16 min
6	Racine	Forest Park	16:00:00	19:00:00	17 min

Buses Table: In addition to the hours of operation, route also covers the frequency of buses.

```
CREATE TABLE IF NOT EXISTS buses (
    bus_id          SERIAL          NOT NULL,
    start_station   VARCHAR(50)     NOT NULL,
    end_station     VARCHAR(50)     NOT NULL,
    start_time      TIME            NOT NULL,
    end_time        TIME            NOT NULL,
    frequency       VARCHAR(50)     NOT NULL,
    day             VARCHAR(25)     NOT NULL,
    PRIMARY KEY (bus_id)
);
```

Functional Dependencies

```
bus_id -> start_station, end_station, start_time, end_time, frequency, day
```

bus_id	start_station	end_station	start_time	end_time	frequency	day
1	Indiana/35th	Union Station	05:40:00	21:00:00	27 min	Weekdays
2	St. Lawrence	Fairbanks	04:45:00	23:05:00	15 min	Weekdays
3	South Shore	Wacker	04:00:00	23:45:00	20 min	Weekdays
4	South Shore	Wacker	04:45:00	00:05:00	22 min	Saturday
5	Harrison	Michigan	06:10:00	22:05:00	10 min	Weekdays
6	Halstead	Broadway	04:05:00	00:30:00	12 min	Sunday

Ferries Table: Ferry stations are denominated by direction.

```
CREATE TABLE IF NOT EXISTS ferries (
    ferry_id         SERIAL           NOT NULL,
    start_station    VARCHAR(50)      NOT NULL,
    end_station      VARCHAR(50)      NOT NULL,
    frequency        VARCHAR(50)      NOT NULL,
    PRIMARY KEY (ferry_id)
);
```

Functional Dependencies

```
ferry_id -> start_station, end_station, frequency
```

ferry_id	start_station	end_station	frequency
1	Belfast	Harlem	20 min
2	Boruch	Radon	30 min
3	Harlem	Belfast	35 min
4	East Side	Grant	25 min
5	East Side	West Side	27 min
6	Radon	Boruch	32 min

Transport_Subways Table

```
CREATE TABLE IF NOT EXISTS transport_subways (
    transport_id     INTEGER          NOT NULL,
    subway_id        INTEGER          NOT NULL,
    PRIMARY KEY (transport_id, subway_id),
    FOREIGN KEY (transport_id)   REFERENCES transportation(transport_id),
    FOREIGN KEY (subway_id)      REFERENCES subways(subway_id)
);
```

Functional Dependencies

```
transport_id -> subway_id
```

transport_id	subway_id
2	1
6	2

Transport_Buses Table

```
CREATE TABLE IF NOT EXISTS transport_buses (
    transport_id     INTEGER          NOT NULL,
    bus_id           INTEGER          NOT NULL,
    PRIMARY KEY (transport_id, bus_id),
    FOREIGN KEY (transport_id)   REFERENCES transportation(transport_id),
    FOREIGN KEY (bus_id)         REFERENCES buses(bus_id)
);
```

Functional Dependencies

```
transport_id -> bus_id
```

transport_id	bus_id
1	1
4	2
5	3

Transport_Ferries Table

```
CREATE TABLE IF NOT EXISTS transport_ferries (
    transport_id     INTEGER          NOT NULL,
    ferry_id         INTEGER          NOT NULL,
    PRIMARY KEY (transport_id, ferry_id),
    FOREIGN KEY (transport_id)   REFERENCES transportation(transport_id),
    FOREIGN KEY (ferry_id)       REFERENCES ferries(ferry_id)
);
```

Functional Dependencies

```
transport_id -> ferry_id
```

transport_id	ferry_id
3	1

Central_OS Table: This table is, at its most basic purpose, meant to record all unique ids in one place.

```
CREATE TABLE IF NOT EXISTS central_os (
    ifst_id          INTEGER          NOT NULL,
    person_id        INTEGER          NOT NULL,
    transport_id     INTEGER          NOT NULL,
    PRIMARY KEY (ifst_id, person_id, transport_id),
    FOREIGN KEY (ifst_id)        REFERENCES infrastructure(ifst_id),
    FOREIGN KEY (person_id)      REFERENCES profiler(person_id),
    FOREIGN KEY (transport_id)   REFERENCES transportation(transport_id)
);
```

Functional Dependencies
None

ifst_id	person_id	transport_id
1	1	1
2	2	2

Profiler Table: The ctOS Profiler tracks people and keeps records of personal information.

```
CREATE TABLE IF NOT EXISTS profiler (
    person_id        SERIAL           NOT NULL,
    first_name       VARCHAR(50)      NOT NULL,
    middle_name      VARCHAR(50),
    last_name        VARCHAR(50)      NOT NULL,
    birth_date       DATE             NOT NULL,
    gender           CHAR(1)          NOT NULL,
    address          VARCHAR(50)      NOT NULL,
    phone_number     CHAR(15)         NOT NULL,
    email            CHAR(256)        NOT NULL,
    eye_color        VARCHAR(25)      NOT NULL,
    hair_color       VARCHAR(25)      NOT NULL,
    is_employee      BOOLEAN          NOT NULL,
    is_affiliate     BOOLEAN          NOT NULL,
    CONSTRAINT valid_gender      CHECK (gender = 'M' OR gender = 'F'),
    PRIMARY KEY (person_id)
);
```

Functional Dependencies

```
person_id -> first_name, middle_name, last_name, birth_date, gender, address,
phone_number, email, eye_color, hair_color, is_employee, is_affiliate
```

Sample table on next page →

person_id	first_name	middle_name	last_name	birth_date	gender	address	phone_number	email	eye_color	hair_color	is_employee	is_affiliate
1	Bob	Randal	Tarly	1989-04-20	M	123 Kenny Ln	312-483-2035	bob_tarly@icloud.com	Blue	Blonde	TRUE	FALSE
2	Frank		Underwood	1956-08-25	M	426 Rook St	312-928-3058	funderwood@gmail.com	Black	Brown	FALSE	TRUE
3	Jaime		Lannister	1967-09-11	M	1 Casterly Rock Rd	312-312-3120	kingslayer@hotmail.com	Green	Blonde	FALSE	FALSE
4	Grace	Rose	Kelly	1990-07-16	F	92 Flower Ln	312-213-9999	grace_kelly@gmail.com	Gray	White	FALSE	FALSE
5	Eileen	Calvin	Hobbes	1970-01-01	F	172 Brooks Rd	312-183-5720	calvin_hobbes@gmail.com	Black	Black	TRUE	FALSE
6	Rhodes		Rodney	1930-02-15	M	304 56th St	572-381-3957	rrodney12@hotmail.com	Brown	Brown	FALSE	TRUE
7	Susan	Dumont	Morgan	1948-03-10	F	95 Flower Ln	572-395-2934	susan_morgan@gmail.com	Pale	Red	TRUE	FALSE
8	Cercei		Baratheon	1975-12-25	F	1 King's Landing Rd	312-304-2950	stupid_queen@gmail.com	Green	Blonde	FALSE	FALSE
9	Jon		Snow	1982-09-09	M	1 Knows Nothing Rd	312-304-5820	clueless@hotmail.com	Black	Black	FALSE	TRUE

Employees Table

```
CREATE TABLE IF NOT EXISTS employees (
    person_id        INTEGER          NOT NULL,
    hire_date        DATE             NOT NULL DEFAULT CURRENT_TIMESTAMP,
    year_wages_usd   MONEY            NOT NULL,
    PRIMARY KEY (person_id),
    FOREIGN KEY (person_id) REFERENCES Profiler(person_id)
);
```

Functional Dependencies

```
person_id -> hire_date, year_wages_usd
```

person_id	hire_date	year_wages_usd
3	2002-02-05	42000
7	2009-04-20	92000
9	2000-01-17	50000

Affiliates Table: Not to be confused with Blume employees, affiliates are people such as temporary contractors or other such people with temporary connections to Blume.

```
CREATE TABLE IF NOT EXISTS affiliates (
    person_id        INTEGER          NOT NULL,
    hire_date        DATE             NOT NULL DEFAULT CURRENT_TIMESTAMP,
    contract_length  VARCHAR(25)      NOT NULL,
    pay              MONEY            NOT NULL,
    PRIMARY KEY (person_id),
    FOREIGN KEY (person_id) REFERENCES Profiler(person_id)
);
```

Functional Dependencies

```
person_id -> hire_date, contract_length, pay
```

person_id	hire_date	contract_length	pay
1	2001-07-08	16 months	42000
3	2008-02-16	18 months	50000
8	2010-10-01	6 months	20000

EmployeeInformation View: This view keeps track of every Blume employee's important contact information all in one view display, specifically: full name, phone number, e-mail, hire date, and salary.

```
CREATE OR REPLACE VIEW employeeInformation AS
    SELECT p.first_name,
           p.middle_name,
           p.last_name,
           p.phone_number,
           p.email,
           e.hire_date,
           e.year_wages_usd
    FROM   profiler p,
           employees e
    WHERE  p.person_id = e.person_id
    ORDER BY p.last_name DESC
```

Data Output | Explain | Messages | History

	first_name character varying(50)	middle_name character varying(50)	last_name character varying(50)	phone_number character(15)	email character(256)
1	Jon		Snow	312-304-5820	clueless@hotmail.com
2	Susan	Dumont	Morgan	572-395-2934	susan morgan@gmail.com
3	Jaime		Lannister	312-312-3120	kingslayer@hotmail.com

AffiliateInformation View: This view keeps track of every Blume affiliate's important contact information all in one view display, specifically: full name, phone number, e-mail, hire date, contract length and salary.

```
CREATE OR REPLACE VIEW affiliateInformation AS
    SELECT p.person_id AS Employee ID,
           p.first_name,
           p.middle_name,
           p.last_name,
           p.phone_number,
           p.email,
           a.hire_date,
           a.contract_length,
           a.pay
    FROM   profiler p,
           affiliates a
    WHERE  p.person_id = a.person_id
    ORDER BY p.last_name DESC
```

Data Output | Explain | Messages | History

	employeeid integer	first_name character varying(50)	middle_name character varying(50)	last_name character varying(50)	phone_number character(15)	email character(256)
1	1	Bob	Randal	Tarly	312-483-2035	bob tarly@icloud.com
2	3	Jaime		Lannister	312-312-3120	kingslayer@hotmail.com
3	8	Cercei		Baratheon	312-304-2950	stupid queen@gmail.com

TLight_Maintain View: Traffic lights, electricity transformers and security cameras all need regular maintenance. This view keeps track of the traffic light or transformer that needs the most attention (has the oldest last maintained date).

```
CREATE OR REPLACE VIEW tlight_maintain AS
    SELECT t.tlight_id,
           t.location,
           t.last_maintained
    FROM   traffic_lights t
    ORDER BY t.last_maintained ASC
```

Transformer_Maintain

```
CREATE OR REPLACE VIEW transformer_maintain AS
    SELECT t.transformer_id,
           t.location,
           t.last_maintained
    FROM   transformers t
    ORDER BY t.last_maintained ASC
```

Cam_Maintain

```
CREATE OR REPLACE VIEW cam_maintain AS
    SELECT s.cam_id,
           s.location,
           s.last_maintained
    FROM   security_cameras s
    ORDER BY s.last_maintained ASC
```

Reports and Queries

Average Bridge Daily Traffic: When the traffic load on bridges is especially high on a certain day, this query can be used to find the total average daily traffic in order to manipulate traffic into using certain bridges over others.

```
SELECT b.bridge_id AS BridgeID,
       b.name AS Name,
       avg(b.daily_traffic) AS Avg_Daily_Traffic
FROM   bridges b
WHERE  b.daily_traffic IS NOT NULL
GROUP BY b.bridge_id;
```

Affiliate Financial Planning: Blume Corporation hires many affiliates and independent contractors, and the finances to hire and train them (if necessary) must be kept track of at all times.

```
SELECT a.contract_length * a.pay AS Financial_Cost,
       p.first_name,
       p.middle_name,
       p.last_name
FROM   affiliates a,
       profiler p
WHERE  a.person_id = p.person_id
GROUP BY Financial_Cost;
```

Population Percentage: For managing censuses and keeping track of population percentages, this query returns the percentage of people under 21.

```
SELECT TRUNC (
          CAST (
                 (SELECT COUNT(person_id) as count
                   FROM Profiler
                   WHERE date_part('year', age( Profiler.birth_date )) < 21
                 ) as decimal(5, 2)
               ) / (SELECT COUNT(person_id) as total
                      FROM Profiler
                    ) * 100
             ) as Underage
```

Stored Procedures

Potential Criminal Search: Given a set of physical attributes such as eye color, hair color and gender, the Profiler can access all the citizens in Chicago as an initial step to find a potential criminal.

```
CREATE OR REPLACE FUNCTION potential_crime(eye_color text, hair_color text,
gender CHAR(1))
RETURNS TABLE(First_Name text, Middle_Name text, Last_Name text) AS
$BODY$
BEGIN
    SELECT DISTINCT p.first_name, p.middle_name, p.last_name
    FROM Profiler p
    WHERE eye_color = p.eye_color
          AND hair_color = p.hair_color
          AND gender = p.gender
END;
$BODY$
LANGUAGE plpgsql;
```

New Employee Hire: While all citizens are automatically tracked and registered by the ctOS Profiler, new employee and affiliate hires must be managed by the database separately.

```
CREATE OR REPLACE FUNCTION new_employee()
RETURNS trigger AS $$
BEGIN
    IF NEW.is_employee = true THEN
        INSERT INTO Employees VALUES(NEW.person_id, NEW.hire_date,
                                     NEW.year_wages_usd);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql
```

New Affiliate Hire

```
CREATE OR REPLACE FUNCTION new_affiliate()
RETURNS trigger AS $$
BEGIN
    IF NEW.is_affilaite = true THEN
        INSERT INTO Affiliates VALUES(NEW.person_id, NEW.hire_date,
                                      NEW.contract_length, NEW.pay);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql
```

Triggers

New Employee: This example triggers on a new entry being created for a Blume employee who recently moved to Chicago and a new entry must be created in the Profiler.

```
CREATE TRIGGER add_employee
AFTER INSERT OR UPDATE ON Profiler
FOR EACH ROW
EXECUTE PROCEDURE new_employee();
```

New Affiliate

```
CREATE TRIGGER add_affiliate
AFTER INSERT OR UPDATE ON Profiler
FOR EACH ROW
EXECUTE PROCEDURE new_affiliate();
```

Security

Admin: High ranking officials; access to the entire ctOS database.

```
CREATE ROLE admin;
GRANT ALL ON ALL TABLES
IN SCHEMA PUBLIC
TO admin;
```

Infrastructure Management Employee: Employees working in the infrastructure department have access to areas of infrastructure only: broadly speaking, this includes security cameras, electric transformers, bridges, and traffic lights.

```
CREATE ROLE ifst_employee;
GRANT SELECT, INSERT, UPDATE ON infrastructure, security_cameras, transformers,
bridges, traffic_lights, infrastructure_security_cameras,
infrastructure_transformers, infrastructure_bridges,
infrastructure_traffic_lights
TO ifst_employee;
```

Transportation Management Employee: Employees working in the transportation department have access to areas of transportation only: broadly speaking, this includes buses, ferries, and subways.

```
CREATE ROLE trans_employee;
GRANT SELECT, INSERT, UPDATE ON transportation, buses, ferries, subways,
transport_buses, transport_ferries, transport_subways
TO trans_employee;
```

Profiler Management Employee: Employees working in the ctOS Profiler department have access to the profiler and central OS databases for tracking and recording people.

```
CREATE ROLE profiler_employee;
GRANT SELECT, INSERT, UPDATE ON profiler, central_os
TO profiler_employee;
```

Affiliates are given similar access permissions, but without INSERT and UPDATE, as those actions are reserved for authorized Blume employees only. The example below shows the permissions for a bridge planner affiliate.

```
CREATE ROLE bridge_affiliate;
GRANT SELECT ON infrastructure, bridges, infrastructure_bridges
TO bridge_affiliate;
```

Implementation Notes, Known Problems & Future Enhancements

This database is meant to be an information storage system for an operating system that controls and manages an entire city of millions of people. As a result, this database is in its early, simplest form.

- In the game Watch_Dogs, ctOS is capable of tracking any device capable of connecting to the Internet or any local networks existing in the city of Chicago.
- This includes devices such as laptops, cellphones, smart watches, even satellites overlooking the city of Chicago.
- As a result, improvements to this database system would include coverage of all these devices with unique identification as well as supplementary information.

The Profiler is capable of keeping a “relationship network” between every citizen in the city, allowing for advanced uses such as searches for potential criminals or even predicting crime (by tracking conversations and text messages on phones and computers).

- Future implementations would possibly use Social Security Numbers as unique identification, or use it alongside the current person_id.
- Addition of GPS with the use of locational coordinates would allow ctOS to keep track of every person through the use of Internet-enabled devices.

Finally, this operating system and database system is limited to Chicago only (at the moment). The most significant future enhancement would be to apply this system to every major city and eventually every city in the world for a global system. Obviously, there will be ethical and moral considerations to take into account, but that is not the purpose of this proposal.