

---

# Goldi-Locks Self-Storage Relational Database

Built by Kyle Wright

Powered by PostgreSQL

---

## Overview

Goldi-Locks Self-Storage is a company I work for in my hometown of Colchester, CT. The facility is comprised of 5 buildings containing nearly 600 units and these provide most of the profitability of the business. The sample data I use in this database will not be accurate to the actual facility, but rather representative of situations likely to occur in the normal operation of the company.

A well-managed database would be highly beneficial to Goldi-Locks, because as it stands now several hours per week are devoted to keeping proper and accurate records for the current standing of every unit. While no database can replace the need for physical "walk-throughs", the total time investment of record keeping can be slashed down by a substantial margin.

Finally, it is worth noting that this database will not be used by Goldi-Locks in its current state. There are many existing programs which the database would need to communicate with that I do not have the specifications for, and although I suspect that there is little to no data security currently in place, I would not trust this database to keep credit card information safe for example.

---

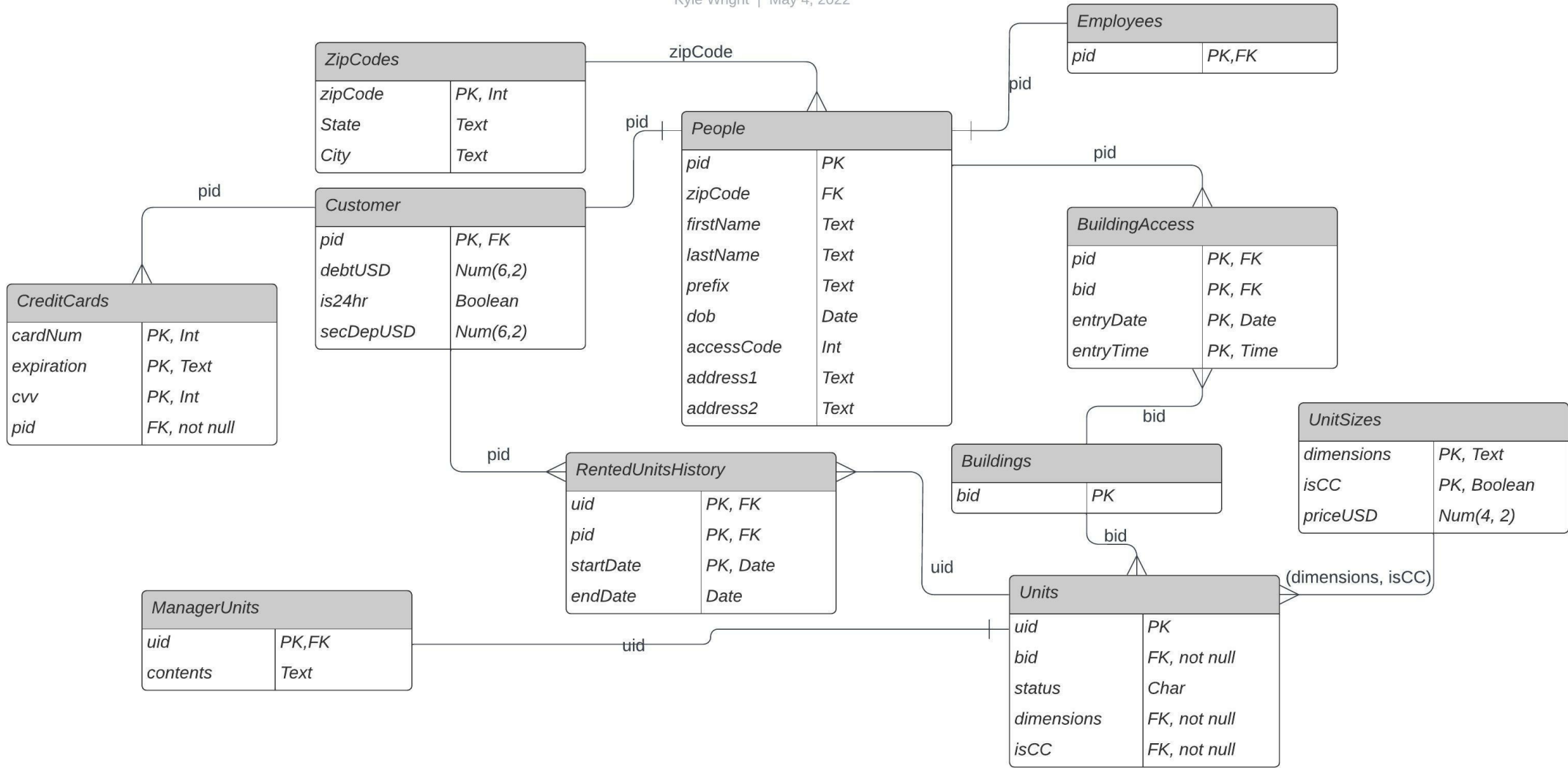
---

## Objectives

- Establish a database capable of handling the relationships between units and customers
- Leave room for expansion upon the database
- Organize pertinent information with proper views
  - For this objective, I have investigated the most common questions asked through the daily operation of the storage facility

# Goldi-Locks Self-Storage


Kyle Wright | May 4, 2022



## Buildings (table)

- Basic table with only a primary key
- Used to enforce referential integrity with units (units must be in a building)
- Also used to track traffic through buildings as all doors are keypad entry
- Helpful for formatting of certain programs, such as a GUI of all units sorted by building

```
--Buildings--  
CREATE TABLE Buildings (  
    bid          int not null,  
    primary key(bid)  
);
```

	bid [PK] integer 
1	1
2	2

## UnitSizes (table)

- All units at a storage facility conform to specific dimensions
- Climate controlled units are temperature and humidity regulated, thus being more expensive
- Not accurate prices
- Dimensions is not technically atomic, as it can be separated, but the information is used only when both are together, thus they represent one column

```
--UnitSizes--  
CREATE TABLE UnitSizes (  
    dimensions  text not null, --not strictly atomic  
    isCC        boolean not null, --is climate controlled  
    priceUSD    numeric(6,2), --monthly rate  
    primary key(dimensions, isCC)  
);
```

	dimensions [PK] text	iscc [PK] boolean	priceusd numeric (6,2)
1	5x5	true	65.00
2	10x10	true	110.00
3	10x10	false	80.00
4	10x20	false	130.00




---

ZipCodes (table)

- Included mostly to maintain conformity to 3NF
- Currently used only for the People table, but could be used for a MovingCompanies table for example

--ZipCodes--

```
CREATE TABLE ZipCodes (  
    zipCode      text not null,  
    state        text,  
    city         text,  
    primary key(zipCode)  
);
```

	zipcode 	state 	city 
	[PK] text	text	text
1	06415	Connecticut	Colchester
2	06249	Connecticut	Lebanon

## Units (table)

- Each row represents one unit
- The status is rather self-explanatory, but "restricted" refers to units which are sold, but that have not been paid in full
- Status can be null, as there are times when unit status is unknown
- Foreign keys to Buildings and UnitSizes

--Units--

```
CREATE TABLE Units (  
  uid          int not null,  
  bid          int not null references Buildings(bid),  
  status       char, --V=vacant,0=occupied,R=restricted  
  dimensions   text not null,  
  isCC         boolean not null,  
  foreign key(dimensions, isCC) references UnitSizes(dimensions, isCC),  
  primary key(uid)  
);
```

	uid [PK] integer	bid integer	status character (1)	dimensions text	iscc boolean
1	1	1	O	10x10	false
2	2	1	O	10x20	false
3	3	1	O	10x10	true
4	4	2	V	5x5	true
5	5	2	R	5x5	true
6	6	2	O	10x10	false
7	7	2	O	10x20	false



## ManagerUnits (table)

- ManagerUnits are units which have been claimed by the manager
  - This could be storage for operations of the business such as tools or inventory
  - Can also be units which have been reclaimed after a customer accrues enough debt
- Contents is another column which is not necessarily atomic, but the business value of knowing what is in each unit is too high not to include it
  - Contents could be its own table
  - The text field is used as a description – thus the description need not be divided

--ManagerUnits--

```
CREATE TABLE ManagerUnits (  
  uid          int not null references Units(uid),  
  contents     text --not strictly atomic  
);
```

	uid integer	contents text
1	1	tools, light bulbs, spare boxes
2	7	Oldsmobile Cutlass 442

## People (table)

- Rather standard people table, used as a base for Customers and Employees
- AccessCode is unique as it is used to identify people who enter buildings
- Worthy addition to this table would be phone numbers, though this could also be accomplished with another table so that people can have multiple phone numbers

--People--

```
CREATE TABLE People (  
    pid            int not null,  
    zipCode        text references ZipCodes(zipCode),  
    address1       text, --line 1  
    address2       text, --line 2  
    prefix         text,  
    firstName      text not null,  
    lastName       text,  
    dob            date,  
    accessCode     int UNIQUE,  
    primary key(pid)  
);
```

	pid [PK] integer	zipcode text	address1 text	address2 text	prefix text	firstname text	lastname text	dob date	accesscode integer
1	10	06415	30 Main St.	[null]	Dr.	Alan	Labouseur	1972-02-13	2633
2	20	06415	786 Park Blvd.	[null]	Dr.	Donald	Schwartz	2061-07-05	5714
3	30	06249	13 Smith St.	Apt. A	[null]	Rick	Treu	1984-05-16	[null]
4	40	06415	89 Lebanon Ave.	[null]	Mr.	Greg	Vilardo	2001-04-26	1198
5	50	06249	2 Cragin Dr.	[null]	Ms.	Linda	Baker	1972-02-13	[null]

---

## Employees (table)

- This table is mostly included as a preparation for future expansion of the database
- Still useful in distinguishing between people and permissions
- Foreign key to People

```
--Employees--  
CREATE TABLE Employees (  
    pid          int not null references People(pid),  
    primary key(pid)  
);
```

	pid [PK] integer
1	40

## Customers (table)

- Foreign key to People
- DebtUSD and secDepUSD are self-explanatory
  - SecDepUSD is generally one month of unit's rent
- Is24hr refers to Customers who pay extra to be able to access their unit at any time
  - Others can access only when the facility is open – typically 9-5

--Customers--

```
CREATE TABLE Customers (  
  pid          int not null references People(pid),  
  debtUSD      numeric(8,2) not null,  
  is24hr       boolean not null,  
  secDepUSD    numeric(8,2),  
  primary key(pid)  
);
```

	pid [PK] integer	debtusd numeric (8,2)	is24hr boolean	secdepusd numeric (8,2)
1	10	0.00	true	245.00
2	20	100.00	false	65.00
3	40	20.00	false	110.00
4	30	0.00	false	0.00

## BuildingAccess (table)

- Provides a record of People who entered Buildings and the date and time of access
  - Data can be imported from the security system theoretically
- Foreign keys to Buildings and People

```
--BuildingAccess--  
CREATE TABLE BuildingAccess (  
    pid          int not null references People(pid),  
    bid          int not null references Buildings(bid),  
    entryDate    date not null,  
    entryTime    time not null,  
    primary key(pid, bid, entryDate, entryTime)  
);
```

	pid [PK] integer	bid [PK] integer	entrydate [PK] date	entrytime [PK] time without time zone
1	40	1	2022-03-02	13:17:09
2	10	1	2022-03-02	22:31:43
3	40	2	2022-03-03	09:03:54
4	20	1	2022-03-04	11:42:43
5	20	2	2022-03-04	10:14:27

---

## CreditCards (table)

- Contains credit card information
  - Billing address should be present in People table
- A credit card cannot exist without a person
  - All credit cards must be removed from the database before a person is removed
  - Cannot have a card that is not assigned to a person

```
--CreditCards--  
CREATE TABLE CreditCards (  
    cardNum    text not null,  
    expiration  text not null,  
    cvv        text not null,  
    pid        int not null references Customers(pid),  
    primary key(cardNum, expiration, cvv)  
);
```

	cardnum [PK] text	expiration [PK] text	cvv [PK] text	pid integer
1	5927384994129411	10/25	905	10

## RentedUnitsHistory (table)

- This table seems confusing at first, but it is a complete record of all rentals on all units
  - Current and past
- A currently rented unit is signified by a "null" in the endDate column
  - This seems like data duplication with the Units "status" column, but the truth is that units can be occupied even if there is no renter
    - This must be reconciled by clearing out the unit

--RentedUnitsHistory--

```
CREATE TABLE RentedUnitsHistory (  
  uid          int not null references Units(uid),  
  pid          int not null references Customers(pid),  
  startDate    date not null,  
  endDate      date CHECK (endDate is null or endDate > startDate),  
  primary key(uid, pid, startDate)  
);
```

	uid [PK] integer	pid [PK] integer	startdate [PK] date	enddate date
1	2	10	2022-03-01	[null]
2	6	10	2022-03-01	[null]
3	5	20	2021-11-15	[null]
4	3	40	2021-05-12	[null]
5	4	30	2021-12-03	2022-02-28
6	2	20	2021-11-15	2022-02-17

## VacantUnits (view)

- Very commonly asked question in normal operation
- Shows all available units

```
--VacantUnits--
```

```
CREATE VIEW VacantUnits (uid, bid, status, dimensions, isCC)
```

```
AS
```

```
select u.uid, u.bid, u.status, u.dimensions, u.isCC
```

```
from Units u
```

```
where status = 'V'
```

```
;
```

	uid integer	bid integer	status character (1)	dimensions text	iscc boolean
1	4	2	V	5x5	true



## SoldUnits (view)

- Another question often asked
- Removes ManagerUnits, as these units are not owned by Customers and thus are not being paid for

```
--SoldUnits--  
CREATE VIEW SoldUnits (uid, bid, status, dimensions, isCC)  
AS  
    select u.uid, u.bid, u.status, u.dimensions, u.isCC  
    from Units u  
    where status = '0' and u.uid not in(select uid  
                                         from ManagerUnits)  
    ;
```

	uid integer	bid integer	status character (1)	dimensions text	iscc boolean
1	2	1	0	10x20	false
2	3	1	0	10x10	true
3	6	2	0	10x10	false

## RestrictedUnits (view)

- A restricted unit is the unit of someone who is behind on payments, and is locked by an additional lock which can only be unlocked by the manager key
- After a certain period, unit will be reclaimed by management and the contents dealt with
  - The period is technically 30 days, though Goldi-Locks often gives people more leeway

```
--RestrictedUnits--  
CREATE VIEW RestrictedUnits (uid, bid, status, dimensions, isCC)  
AS  
    select u.uid, u.bid, u.status, u.dimensions, u.isCC  
    from Units u  
    where status = 'R' and u.uid not in(select uid  
                                         from ManagerUnits)  
    ;
```

	uid integer	bid integer	status character (1)	dimensions text	iscc boolean
1	5	2	R	5x5	true

---

## ManagerUnits (view)

- This view is used simply to show unit data on ManagerUnits

--ManagerUnits--

**CREATE VIEW** ManagerUnitsView (uid, bid, status, dimensions, isCC, contents)

**AS**

```
select u.uid, u.bid, u.status, u.dimensions, u.isCC, mu.contents
from Units u inner join ManagerUnits mu on(u.uid = mu.uid)
;
```

	uid integer	bid integer	status character (1)	dimensions text	iscc boolean	contents text
1	1	1	0	10x10	false	tools, light bulbs, spare boxes
2	7	2	0	10x20	false	Oldsmobile Cutlass 442

## CustomersInDebt (view)

- This is not a specific question often asked, but a useful view nonetheless, especially when deciding where to attempt to collect debt from

```
--CustomersInDebt--  
CREATE VIEW CustomersInDebt (pid, firstName, lastName, debtUSD, address1, address2, city, state, zipCode)  
AS  
    select c.pid, p.firstName, p.lastName, c.debtUSD, p.address1, p.address2, zip.city, zip.state, zip.zipCode  
    from Customers c inner join People p on(c.pid = p.pid)  
        inner join ZipCodes zip on(p.zipCode = zip.ZipCode)  
    where c.debtUSD > 0  
;
```

	pid integer	firstname text	lastname text	debtusd numeric (8,2)	address1 text	address2 text	city text	state text	zipcode text
1	20	Donald	Schwartz	100.00	786 Park Blvd.	[null]	Colchester	Connecticut	06415
2	40	Greg	Vilardo	20.00	89 Lebanon Ave.	[null]	Colchester	Connecticut	06415

---

## TotalDebtAndSecDep (view)

- These totals are useful for the accounting aspect of the business – must be able to pay back security deposits, and should attempt to keep debt low

```
--TotalDebtAndSecDep--  
CREATE VIEW TotalDebtAndSecDep (totalDebtUSD, totalSecDepUSD)  
AS  
    select sum(c.debtUSD), sum(c.secDepUSD)  
    from Customers c  
    ;
```

	totaldebtusd. numeric	totalsecdepusd. numeric
1	120.00	420.00

## EmployeeAccess (view)

- Simple view which shows all employee access history

```
--EmployeeAccess--
```

```
CREATE VIEW EmployeeAccess (pid, building, entryDate, entryTime)  
AS
```

```
select ba.pid, ba.bid, ba.entryDate, ba.entryTime  
from BuildingAccess ba  
where pid in(select pid  
              from Employees)  
order by entryDate desc  
;
```

	pid integer	building integer	entrydate date	entrytime time without time zone
1	40	2	2022-03-03	09:03:54
2	40	1	2022-03-02	13:17:09

## CustomerAccess (view)

- Another simple view which shows Customer access minus the access by Customers who are also Employees

```
--CustomerAccess--  
CREATE VIEW CustomerAccess (pid, building, entryDate, entryTime)  
AS  
    select ba.pid, ba.bid, ba.entryDate, ba.entryTime  
    from BuildingAccess ba  
    where pid in(select pid  
                  from Customers  
                  where pid not in(select pid  
                                     from Employees))  
    order by entryDate desc  
;
```

	pid integer	building integer	entrydate date	entrytime time without time zone
1	20	1	2022-03-04	11:42:43
2	20	2	2022-03-04	10:14:27
3	10	1	2022-03-02	22:31:43

## UnitsQuickReport (view)

- Uses multiple views and tables to give a quick report on the status of the who facility
- In a UI, each total could be used as a hyperlink which pulls up a window of the corresponding view or table referenced from

```
--UnitsQuickReport--
CREATE VIEW UnitsQuickReport (total, manager, vacant, sold, restricted)
AS
    select count(u.uid),
           count(m.uid),
           count(v.uid),
           count(s.uid),
           count(r.uid)
    from Units u left outer join ManagerUnits m on(u.uid = m.uid)
                left outer join VacantUnits v on(u.uid = v.uid)
                left outer join SoldUnits s on(u.uid = s.uid)
                left outer join RestrictedUnits r on(u.uid = r.uid)
    ;
```

	total bigint	manager bigint	vacant bigint	sold bigint	restricted bigint
1	7	2	1	3	1



## VacantSearch (stored procedure)

- Provides a more focused view of vacant units by allowing the user to specify the size and whether it is climate controlled
- Not particularly useful on the current scale of the database, but when dealing with a larger data set this will be essential
- Customers often ask if we have units available using these specifications

```
--vacantSearch--
CREATE OR REPLACE FUNCTION vacantSearch(text, boolean, refcursor) RETURNS refcursor AS
$$
DECLARE
    dim text := $1;
    isClimate boolean := $2;
    resultset refcursor := $3;
BEGIN
    open resultset FOR
        select *
        from VacantUnits v
        where dim = v.dimensions and isClimate = v.isCC
    ;
    RETURN resultset;
END
$$
language plpgsql;

SELECT vacantSearch('5x5' , '1', 'results');
FETCH ALL FROM results;
```

	uid integer	bid integer	status character (1)	dimensions text	iscc boolean
1	4	2	V	5x5	true

CustHistory (stored procedure)

- Shows all rentals, current and past from RentedUnitsHistory base table for a specific pid
- Answers a common question - "What units does this person own?"
- Could be joined with people data, customer data, unit data

```
--custHistory--
CREATE OR REPLACE FUNCTION custHistory(int, refcursor) RETURNS refcursor AS
$$
DECLARE
  cid int := $1;
  resultset refcursor := $2;
BEGIN
  open resultset FOR
    select *
    from RentedUnitsHistory h
    where cid = h.pid
  ;
  RETURN resultset;
END
$$
language plpgsql;

SELECT custHistory(10, 'results');
FETCH ALL FROM results;
```

	uid [PK] integer	pid [PK] integer	startdate [PK] date	enddate date
1	2	10	2022-03-01	[null]
2	6	10	2022-03-01	[null]

## UnitHistory (stored procedure)

- Shows all rentals, current and past from RentedUnitsHistory base table for a specific uid
- Answers a common question - "Who owned this unit?"
- Could be used to identify who to charge for damage to a unit, for example
- Could be joined with people data, customer data, unit data

```
--unitHistory--
CREATE OR REPLACE FUNCTION unitHistory(int, refcursor) RETURNS refcursor AS
$$
DECLARE
    unit int := $1;
    resultset refcursor := $2;
BEGIN
    open resultset FOR
        select *
        from RentedUnitsHistory h
        where unit = h.uid
        ;
    RETURN resultset;
END
$$
language plpgsql;

SELECT unitHistory(2, 'results');
FETCH ALL FROM results;
```

	uid [PK] integer	pid [PK] integer	startdate [PK] date	enddate date
1	2	10	2022-03-01	[null]
2	2	20	2021-11-15	2022-02-17