SQL and Databases for Research

Eric Seymour, PhD

June 14, 2017

Workshop Overview

- Overview of databases
- Overview of SQL syntax
- Creating databases with real world data
- Using databases and SQL with property data
- Spatial databases and spatial queries

Overview & SQL Basics

Database Overview

- DBs best for storing and working with large datasets

Database Overview

- DBs best for storing and working with large datasets
- DBs keep related data files in a single place

Database Overview

- DBs best for storing and working with large datasets
- DBs keep related data files in a single place
- DBs enable let you link different files using common fields

Database Overview

- DBs best for storing and working with large datasets
- DBs keep related data files in a single place
- DBs enable let you link different files using common fields
- DBs enforce data integrity, e.g., only numbers in number column

Database Overview

- DBs best for storing and working with large datasets
- DBs keep related data files in a single place
- DBs enable let you link different files using common fields
- DBs enforce data integrity, e.g., only numbers in number column
- SQL helps with research reproducibility (reusable code)

Database Design

- DBs contain one or more tables
- Tables consist of columns (fields) and rows (observations)
- Columns have explicitly defined column types (text, numbers, etc.)
- Tables can be created from preexisting **text** files (usually CSV files)

Database Software

Name	Type
MS Access	File based
SQLite	File based
MS SQL Server	Server based
MySQL (Oracle)	Server based
PostgreSQL	Server based

SQLite

- Does not require a server process
- No configuration required
- self-contained, no external dependencies
- works on all platforms

SQLite Usage

- Command line interface (CLI)
- Firefox SQLite Manager plugin
- Plugin needs to be installed for workshop

Creating Tables

- SQLite Manager simplifies table creation with wizard
- Table creation requires a CREATE TABLE statement using SQL
- These statements define the table name, columns, and column data types

Simple Table Creation

- Open SQLite Manager
- Select Databse -> New In-Memory Database
- Enter code in dialog box in 'Execute SQL' tab, hit RUN SQL

```
CREATE TABLE cars (
id      INTEGER PRIMARY KEY,
make    TEXT,
model   TEXT,
price   INTEGER);
```

Add data

- Remove CREATE code
- Copy statement below into dialog box, hit RUN SQL

```
INSERT INTO cars VALUES (1, 'Acura', 'NSX', 47045);
INSERT INTO cars VALUES (2, 'Audi', 'A8', 63890);
INSERT INTO cars VALUES (3, 'BMW', 'X1', 108900);
```

View Table

This query returns all columns and rows

```
SELECT *    /* '*' means ALL columns */
FROM cars;
```

SQL Query Basics

SQL: Structured Query Language

```
SELECT field1, field2   /* specify columns */
FROM myTable;           /* specify tables   */
WHERE condition = 'met' /* filter (optional) */
```

- SELECT and FROM are mandatory elements
- Many optional clauses, including WHERE

Operators in WHERE Clause

Operator	Description
$=$	Equal
$<>$	Not equal
$>(=)$	Greater than (or equal)
$<(=)$	Less than (or equal)

WHERE Clause, Cont.

Operator	Description
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a colum

SQL Challenge!

Find make, model, price for cars that cost 50,000 or more

SQL Challenge!

Find make, model, price for cars that cost 50,000 or more

```
SELECT make, model, price
FROM cars
WHERE price >= 50000;
```

Order Results

- Use ORDER clause to sort results by field(s)
- Default is ASC values; can also specify DESC
- Let's ORDER results of entire table by price

```
SELECT *
FROM cars
ORDER BY price DESC;
```

SQL Functions

- SQL can returns summary information
- MIN, MAX, COUNT, AVG, SUM
- You need to wrap the field in the function

```
SELECT COUNT(*), AVG(price), MIN(price), MAX(price)
FROM cars;
```

SQL Functions, cont.

- Many string/text manipulation functions
- Let's find all cars whose manufacturer starts with 'A'
- Add SQL SUBSTR function to query
- Takes 2 arguments: position, length

```
SELECT *
FROM cars
WHERE SUBSTR(make,1,1) = 'A'
ORDER BY make;
```

LIKE Function

- We could use LIKE for same purpose
- '%' is the wildcard character

```
SELECT *
FROM cars
WHERE make LIKE 'A%'
ORDER BY make;
```

SQL Core Functions

- Many functions for operating on text and numbers
- Useful for cleaning data and customizing output
- Full list for SQLite core functions

Managing SQL output

- SQLite Manager allows you to copy results as CSV to clipboard
- Usually have to click on each row of data (or first and last)
- Can also copy data as SQL INSERT statement that may be used to create a new table

Create table from query

- In some cases you may want to make a new table from a query
- Rather than supply explicit VALUES, supply a SELECT statement
- This table has all cars whose make starts with 'A'

```
CREATE TABLE a_cars AS
SELECT *
FROM cars
WHERE make LIKE 'A%';
```

Deleting Tables

Not Reversible-keep backups!

```
DROP TABLE a_cars;
```

Tip of the SQL/ Database Iceberg

- You now know how to create and query a database table
- Everything else builds on these fundamental ideas
- Are there questions before we move on?

Table Join

Adding a second table

Create new table of safety ratings for cars

```
CREATE TABLE ratings (
id       INTEGER PRIMARY KEY,
make     TEXT,
model    TEXT,
rating   TEXT);
```

Add data

Cars have hypothetical A to F ratings

```
INSERT INTO ratings VALUES (1, 'Acura', 'NSX', 'C');
INSERT INTO ratings VALUES (2, 'Audi', 'A8', 'A');
INSERT INTO ratings VALUES (3, 'BMW', 'X1', 'B');
```

Basic Table Joins

- Join tables on common field
- We have several fields in common in this case
- Join on id field in both tables

```
SELECT A.make, A.model, B.rating
FROM cars AS A JOIN ratings AS B /* AS alias */
ON A.id = B.id; /* specify field to join on */
```

Basic Table Joins, Cont

- If tables lack common id number, find alternative
- Join on make AND model

```
SELECT A.make, A.model, B.rating
FROM cars AS A JOIN ratings AS B
ON A.make = B.make AND A.model = B.model;
```

Types of Joins in SQLite

- Inner Join (Default)

Returns all rows from multiple tables where the join condition is met

Types of Joins in SQLite

- Inner Join (Default)

Returns all rows from multiple tables where the join condition is met

- LEFT OUTER JOIN

Returns **all** rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal

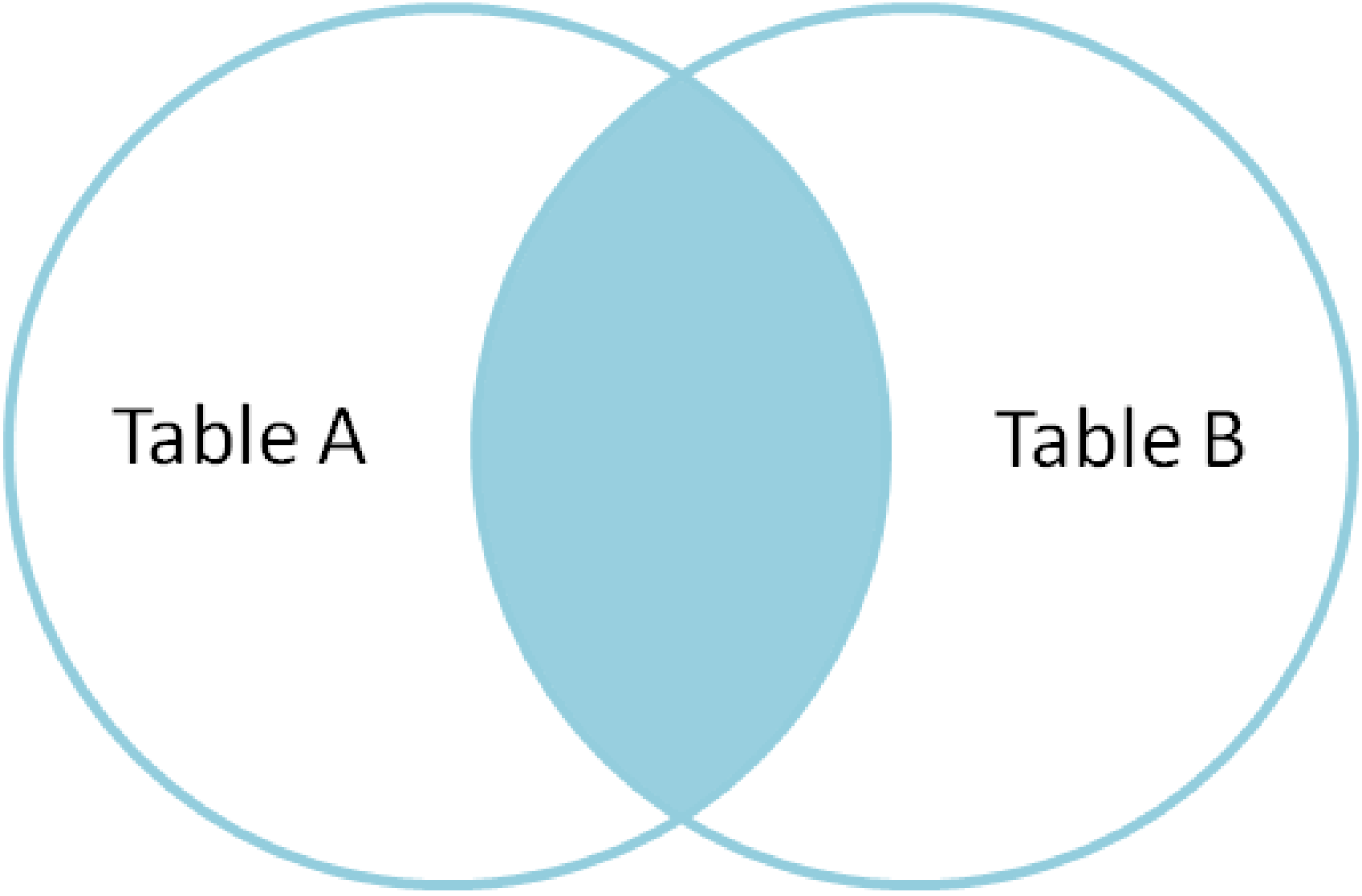

Figure 1: Inner Join

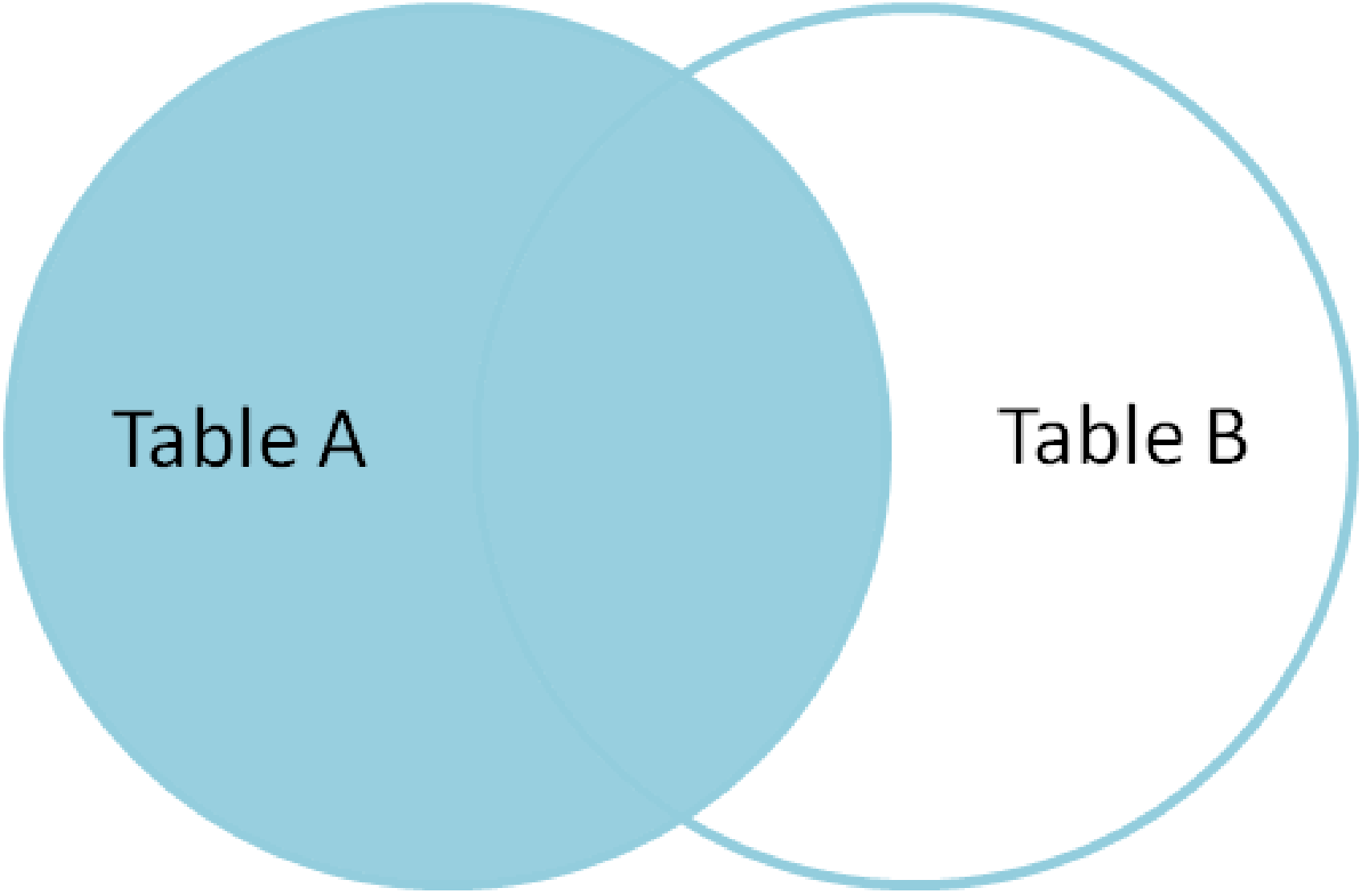

Figure 2: Left Outer Join

Join comparison

```
/* add another data */
INSERT INTO cars VALUES (4, 'Ford',  'Fusion', 72000);

/* inner join */
SELECT A.make, A.model, B.rating
FROM cars AS A JOIN ratings AS B
ON A.make = B.make AND A.model = B.model;

/* left outer join */
SELECT A.make, A.model, B.rating
FROM cars AS A LEFT OUTER JOIN ratings AS B
ON A.make = B.make AND A.model = B.model;
```

Working with property data

Overview

- We will now put our SQL skills to use querying Detroit property records
- We will create tables from publicly available data
- We will perform basic, but powerful queries on these data

Motor City Mapping

Motor City Mapping Survey

- Citywide parcel-by-parcel survey of property conditions 2013-2014
- Results viewable at motorcitymapping.org
- Data available at Data Driven Detroit
- Spreadsheet located in data folder: Motor_City_Mapping_Winter_201314_Certified_Results.csv
- Let's review fields using Metadata_MCM_CertifiedResults_Winter2014.xls

Inspecting and Loading Data

- What are the fields?
- What do the data elements look like?
- How many rows?
- Command line tools like *head* and *wc* and perfect for these tasks
- Mac OSX and Linux users have these tools installed, just open a terminal

Inspecting data

Display top 5 lines

```
$ head --lines=5 Motor_City_Mapping_Winter_201314_Certifie

OBJECTID,D3_SurveyParcelID,CityParcelID2013,AddCombo,Addres
3001,13016242.020,13016242.020,20560 KEYSTONE,20560, ,KEYS
3002,13016242.021L,13016242.021L,20570 KEYSTONE,20570, ,KE
3003,13022754.001,13022754.001,18027 REVERE,18027, ,REVERE
3004,13022754.002,13022754.002,18019 REVERE,18019, ,REVERE
```

Display line count

```
$ wc -l Motor_City_Mapping_Winter_201314_Certified_Results

380121 Motor_City_Mapping_Winter_201314_Certified_Results.
```

Connecting to Workshop DB

- SQLite Manger has a wizard for importing CSV
- Database -> Import
- Dropdown menus for selecting data type for each column
- We will use pre-made table to future-proof presentation
- Database -> Connect Database, navigate to sql_tutorial.sqlite

Explore Data

How many records in MCM table?

```
SELECT COUNT(*)
FROM MCM;
```

GROUP BY

- Aggregate records by each value of a given field
- Best for fields with few unique values
- Structure is a binary field

```
SELECT Structure, COUNT(*)
FROM MCM
GROUP BY Structure
ORDER BY Count(*) DESC;
```

Query Results

- Returns count for all values of structure, including ” and NULL values
- Empty string suggests records not surveyed

Structure	Count
“yes”	261158
“no”	112069
“”	6322

Filtering Results

- Data contains field for time surveyed
- Filter rows to exclude those where time surveyed is an empty string

```
SELECT Structure, COUNT(*)
FROM MCM
WHERE Time_Surveyed <> ''
GROUP BY Structure
ORDER BY Count(*) DESC;
```

Check data

Find addresses of properties where there is no property condition and time surveyed is an empty string

```
SELECT oid, AddCombo
FROM MCM
WHERE Structure = ''
AND Time_Surveyed <> ''
```

Updating Records

- We can use SQL to update records based on new information
- I did not perform these UPDATEs in the sample db

```
UPDATE MCM
SET Structure = 'yes' /* text must be in 'quotes' */
WHERE oid = 167214; /* specify record(s) to update */

UPDATE MCM
SET Structure = 'no'
WHERE oid = 168113;
```

Examining Property Conditions

- 'Condition' is short for 'structure condition'
- Do all records with 'condition' also have a structure?

```
SELECT Structure, Condition, COUNT(*)
FROM MCM
WHERE Time_Surveyed <> ''
AND Condition <> ''
AND Structure <> ''
GROUP BY Structure, Condition
ORDER BY Structure, COUNT(*) DESC;
```

Examining Property Conditions

- Some properties were likely miscoded as either having a structure or conditon rating
- One possibility is to update all records having a condition rating as also having a structure
- All public property datasets suffer from these types of inconsistencies and errors
- Use SQL to efficiently and rigorously explore data and diagnose issues

Examining Property Conditions

- Let us assume records with condition ratings are all structures (may be wrong)
- How many records have condition ratings?

```
SELECT COUNT(*) FROM MCM
WHERE Condition <> '' AND Use_='residential' ;
```

Examining Property Conditions

- Let us assume records with condition ratings are all structures (may be wrong)
- How many records have condition ratings?

```
SELECT COUNT(*) FROM MCM
WHERE Condition <> '' AND Use_='residential' ;
```

243,631

Calculating percentages

- Divide COUNT by result from last slide to get percentage for each condition rating
- Multiply COUNT by 1.0 to get floating point value

```
SELECT Condition,
( COUNT(*) * 1.0 / 243631 ) * 100 AS Share,
COUNT(*) AS Count
FROM MCM
WHERE Condition <> '' AND Use_='residential'
GROUP BY Condition
ORDER BY COUNT(*) DESC;
```

Rounding Numbers

- ROUND percentages for neater table
- Takes the value and the number of digits as arguments

```
SELECT Condition,
ROUND(( COUNT(*) * 1.0 / 243631 ) * 100, 2) AS Share,
COUNT(*) AS Count
FROM MCM
WHERE Condition <> '' AND Use_='residential'
GROUP BY Condition
ORDER BY COUNT(*) DESC;
```

Subqueries

Include the denominator as the result of a subquery

```
SELECT Condition,
( COUNT(*) * 1.0 / res_struc ) * 100 AS Share,
COUNT(*) AS Count
FROM MCM JOIN (
    SELECT COUNT(*) as res_struc FROM MCM
    WHERE Condition <> '' AND Use_='residential')
WHERE Condition <> '' AND Use_='residential'
GROUP BY Condition
ORDER BY COUNT(*) DESC;
```

Cross joins

- JOINs w/out ON result in a CROSS JOIN
- This produces the cartesian product of both tables
- That means every combination of table A and B
- This is only desirable in special cases
- In the code above, we wanted 'res_struc' to appear in the same row as the counts for each condition rating

Residential Properties with Structures

What share of residential properties have structures?

```
SELECT Structure, COUNT(*)
FROM MCM
WHERE Structure <> ''
AND Use_ = 'residential'
GROUP BY Structure;
```

Residential Properties with Structures

What share of residential properties have structures?

```
SELECT Structure, COUNT(*)
FROM MCM
WHERE Structure <> ''
AND Use_ = 'residential'
GROUP BY Structure;
```

This is inaccurate. There are far more vacant residential lots in Detroit. Survey did not correctly identify land use of vacant lots.

Residential Properties with Structures

What is the recorded use of vacant lots?

```
SELECT Use_, COUNT(*)
FROM MCM
WHERE Structure = 'no'
GROUP BY Use_
ORDER BY COUNT(*) DESC;
```

Residential Properties with Structures

What is the recorded use of vacant lots?

```
SELECT Use_, COUNT(*)
FROM MCM
WHERE Structure = 'no'
GROUP BY Use_
ORDER BY COUNT(*) DESC;
```

Most are unknown

Other Info in MCM Survey Results

- Information on dumping
- Whether properties need to be boarded
- Do you have questions of the data we can answer using SQL?

Changing the Unit of Analysis

From property to tract

- MCM data include tract number
- Find tract percentage of residential structures in good condition
- Need to GROUP BY tract

```
SELECT GEOID10_Tract, COUNT(*) AS good
FROM MCM
WHERE Condition='good'
GROUP BY GEOID10_Tract
ORDER BY good
LIMIT 10;
```

Take a deep breath

% of residential structures in good condition

```
SELECT A.trt AS tract, B.good, A.structures,
( B.good * 1.0 / A.structures )  * 100 as prct
FROM
    (SELECT COUNT(*) AS structures, GEOID10_Tract AS trt
    FROM MCM
    WHERE Condition!=''
    GROUP BY GEOID10_Tract) AS A
LEFT JOIN
    (SELECT COUNT(*) AS good, GEOID10_Tract AS trt
    FROM MCM
    WHERE Condition='good'
    GROUP BY GEOID10_Tract) AS B
ON A.trt = B.trt
ORDER BY ( B.good * 1.0 / A.structures )  * 100
LIMIT 10;
```

Explanation

- line 2: calculates percent of structures in good condition per tract
- lines 4-7: subquery returning the count of records where condition is not null per tract
- lines 9-12: subquery returning the count of records where condition is good per tract
- line 14: returns the results in ascending order
- line 15: limits the results to the 10 tracts w/ the lowest percentages

Views

- We can store results of query as a VIEW
- Veiws dynamically returns elements of SQL query behind VIEW
- Views change when underlying data changes
- Views function like a table and may be included in joins
- Views help clean up busy code

Create view

- View -> Create View
- Store code below as VIEW names 'prct_good'

```
SELECT A.trt AS tract, B.good, A.structures,
( B.good * 1.0 / A.structures )  * 100 as prct
FROM
    (SELECT COUNT(*) AS structures, GEOID10_Tract AS trt
    FROM MCM
    WHERE Condition!=''
    GROUP BY GEOID10_Tract) AS A
LEFT JOIN
    (SELECT COUNT(*) AS good, GEOID10_Tract AS trt
    FROM MCM
    WHERE Condition='good'
    GROUP BY GEOID10_Tract) AS B
ON A.trt = B.trt
ORDER BY ( B.good * 1.0 / A.structures )  * 100
LIMIT 10;
```

Tax foreclosure records

Detroit Tax Foreclosure Records

- Data downloaded from Data Driven Detroit
- Contain Detroit tax foreclosure records 2002-2013
- Field descriptions available Data Driven Detroit
- Data pre-loaded into workshop DB in 'ATF' table (Archived Tax Foreclosures)
- Take a moment to examine the data

Linking Tables

- MCM and ATF share parcel id

Linking Tables

- MCM and ATF share parcel id
- Locate the parcel field for each table and note the name

Linking Tables

- MCM and ATF share parcel id
- Locate the parcel field for each table and note the name
- MCM.CityParcelID2013 and ATF.ParcelNumb

Linking Tables

- MCM and ATF share parcel id
- Locate the parcel field for each table and note the name
- MCM.CityParcelID2013 and ATF.ParcelNumb
- Let's find the number of matching records between these two tables

Indexes

- Useful when joining large tables
- Speeds DB lookup of values in JOIN fields
- May be created for single or multiple fields

```
CREATE INDEX index_name ON myTable(field_to_index);
```

```
CREATE INDEX mcm_pid_idx ON MCM(CityParcelID2013);
CREATE INDEX atf_pid_idx ON ATF(ParcelNumb);
```

JOIN code

```
/* How many in ATF? */
SELECT COUNT(*) FROM ATF;

/* How many match a record in MCM? */
SELECT COUNT(*)
FROM ATF AS A JOIN MCM AS B
ON A.ParcelNumb = B.CityParcelID2013;
```

JOIN ATF to MCM

- Find condition of 2013 tax foreclosures
- Restrict to residential structures

```
SELECT B.Condition, COUNT(*)
FROM ATF AS A JOIN MCM AS B
ON A.ParcelNumb = B.CityParcelID2013
WHERE B.Use_='residential' AND B.Condition<>''
AND A.FC_2013 = 1
GROUP BY B.Condition
ORDER BY COUNT(*) DESC;
```

% of 2013 residential tax foreclosures by condition

```
SELECT B.Condition, COUNT(*),
( COUNT(*) * 1.0 / denom ) * 100 AS prct
FROM ATF AS A JOIN MCM AS B
ON A.ParcelNumb = B.CityParcelID2013
JOIN (SELECT COUNT(*) AS denom
    FROM ATF AS A JOIN MCM AS B
    ON A.ParcelNumb = B.CityParcelID2013
    WHERE B.Use_='residential' AND B.Condition<>''
    AND A.FC_2013 = 1)
WHERE B.Use_='residential' AND B.Condition<>''
AND A.FC_2013 = 1
GROUP BY B.Condition
ORDER BY COUNT(*) DESC;
```

- Find occupancy status of 2013 residential tax foreclosures
- Restrict to residential structures

```
1 SELECT B.Occupancy, COUNT(*)
2 FROM ATF AS A JOIN MCM AS B
3 ON A.ParcelNumb = B.CityParcelID2013
4 WHERE B.Use_='residential' AND A.FC_2013 = 1
5 GROUP BY B.Occupancy
6 ORDER BY COUNT(*) DESC;
```

Property Sale Data

Detroit Sales History

- Data found at Detroit Open Data
- Contains ‘’sales price, buyer, and seller of properties sold in the city of Detroit”
- Let’s download, inspect, clean, and load data into database

Data inspection

- Sale price has $
- Needs to be removed to add price as an INTEGER field
- SQLite stores dates in YYYY-MM-DD format. This is not how the records are stored in the file
- A good tool for cleaning these data is OpenRefine

OpenRefine

- Navigate to sales data from within OpenRefine
- Open data into new project
- From the dropdown menu for SALEPRICE, select EDIT CELLS -> TRANSFORM
- Enter the code below to remove $

```
value.replace('$','')
python
```

Fixing dates

- Change date to SQLite format: 'YYYY-MM-DD'
- Check results and export as new CSV file

```
value[6,10]+'-'+value[0,2]+'-'+value[3,5]
```

Import Data to Table

- Import the cleaned data as a new table
- Use SQLite Manager import wizard
- Specify data type for each field
- Rename table as 'detroit_sales'
- Be sure to specify that first row contains field names

Explore Sales Data

- Check the import worked as intended
- Select first 5 rows of data to check fields and data line up
- Select count to make sure all data imported properly

Annual Average Sale Price

- If your data have YEAR, you can GROUP BY YEAR
- If your data have DATE, you can extract the year from the date field by taking the first four characters

```
SELECT SUBSTR(SALEDATE,1,4),
ROUND(AVG(SALEPRICE), 0),
COUNT(*)
FROM detroit_sales
GROUP BY SUBSTR(SALEDATE,1,4);
```

Parsing Date Fields

- You can also use SQLite strftime() function
- It takes two arguments:
  - a special argument for format to return
  - the field containing the data info
- See here for more

```
SELECT strftime('%Y', SALEDATE) ,
ROUND(AVG(SALEPRICE), 0), COUNT(*)
FROM detroit_sales
GROUP BY strftime('%Y', SALEDATE);
```

Property Classification Codes

- Descriptions found here
- Residential properties have values between 400 and 499
- Restrict queries to residential only

```
SELECT COUNT(*)
FROM detroit_sales
WHERE SUBSTR(PROPCLASS,1,1) = '4';
```

Sales Types

- Some sales are market sales, some are transfers
- We can use SalesInstr and SALETERMS to filter for market sales
- What terms are used for residential sales?

```
SELECT SalesInstr, COUNT(*)
FROM detroit_sales
WHERE SUBSTR(PROPCLASS,1,1) = '4'
GROUP BY SalesInstr
ORDER BY COUNT(*) DESC;
```

Results

- 'PTA' is a Property Transfer Affidavit
- This is not a deed-it goes to the assessor
- Find the annual average price for residential WD (warranty deed) transactions and plot them
- Copy results and paste into spreadsheet software

```
SELECT strftime('%Y', SALEDATE),
ROUND(AVG(SALEPRICE), 0), COUNT(*)
FROM detroit_sales
WHERE SUBSTR(PROPCLASS,1,1) = '4'
AND SalesInstr = 'WD'
GROUP BY strftime('%Y', SALEDATE);
```

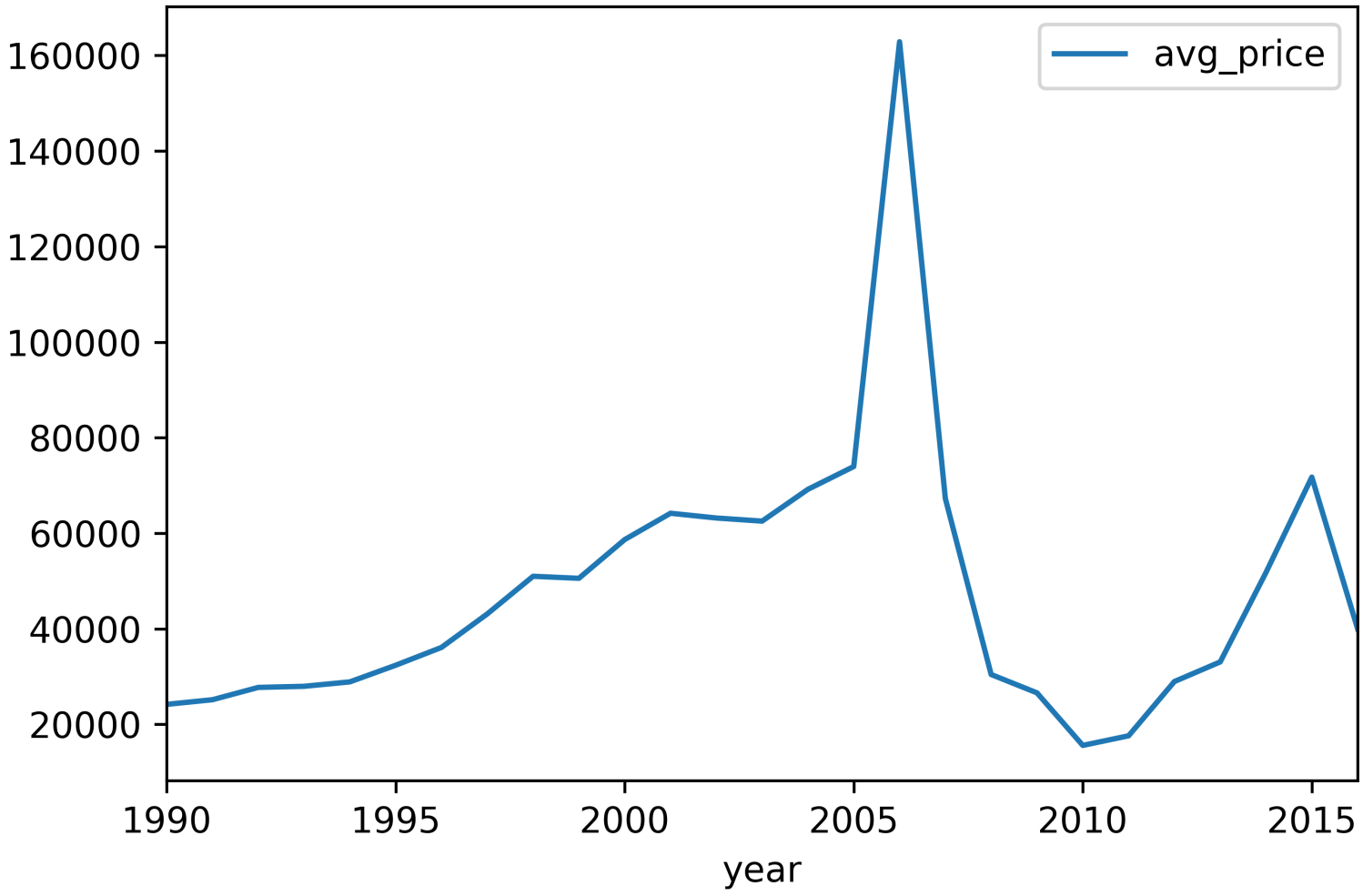
avg_price
160000
140000
120000
100000
80000
60000
40000
20000
1990
1995
2000
2005
2010
2015
year

Figure 3

Spatial Databases

Spatial Databases

- We can use coordinate/geometry data to make spatial tables
- SQLite -> Spatialite
- Special spatial SQL syntax for querying for relationships based on location
- QGIS makes it easy to create and work w/ Spatialite DBs

Spatial DB Fundamentals

- Spatial data have three types of geometry: points, lines, polygons
- Polygons may contains points
- Points may be within polygons
- Census geographies are polygons
- Properties with simple X, Y coordinates are points

Spatialite Tables from Shapefiles

- Open QGIS
- Navigate to Census Tracts 2010 folder
- Drag 'geo_export. . . .shp' into QGIS window
- Right-click layer name in table of contents in QGIS, select SAVE AS
- Under FORMAT, select SPATIALITE
- Change CRS (coordinate reference system) to 2898
- Rename table 'detroit_tracts10'
- Save to workshop folder

Coordinate Reference Systems

- Many different CRSs
- Each suited to different locations on the Earth's surface
- Each suited to different spatial scales, e.g., continents and states
- CRS 2898 is tailored to SE Michigan
- Distance in this CRS measured in **feet**

Connecting to Databases

- Select feather icon
- Find database you created in last step
- Select tracts table
- Tracts should now be displayed in QGIS

Import MCM into Spatialite

1. Navigate to Motor City Mapping folder
2. Drag 'Motor_City_Mapping. . . .shp' into QGIS window
3. Right-click layer name in table of contents in QGIS, select SAVE AS
4. Under FORMAT, select SPATIALITE
5. Change CRC (coordinate reference system) to 2898
6. Rename table 'MCM'
7. Save into BD created when saving tract table

QGIS DB Manager

- From the top menu, select Database, then DB Manager
- Find and connect to our new spatial DB
- Expand the DB and select any one table
- Press F2 to open the SQL Window

Spatial SQL

We can query Spatialite DBs w/ special spatial parameters. 'ST' means Space-Time

- ST_Contains(geom1, geom2)
- ST_Within(geom1, geom2)
- ST_Distance(geom1, geom2)
- ST_Intersects(geom1, geom2)

Spatial SQL, Cont.

Follow link for complete list of spatial functions

Find Properties in Tracts

```
SELECT A.tractce_10, COUNT(*), A.geometry
FROM detroit_tracts10 AS A, mcm AS B
WHERE st_contains(A.geometry, B.geometry)
AND B.ROWID IN (
    SELECT ROWID
    FROM SpatialIndex
    WHERE f_table_name="mcm"
    AND search_frame=A.geometry)
GROUP BY A.tractce_10;
```

- line 3 is spatial join syntax
- lines 4-8 invoke spatial index, not mandatory, but speeds processing time

Find Average Sale Price by Tract

```
SELECT A.geoid_10, AVG(C.SALEPRICE) AS avg_price, A.geometr
FROM detroit_tracts10 AS A JOIN mcm AS B
ON A.geoid_10 = B.geoid10_tr
JOIN detroit_sales AS C ON B.cityparcel = C.PARCELNO
WHERE SUBSTR(C.SALEDATE,1,4) IN ('2013','2014','2015')
AND SUBSTR(C.PROPCLASS,1,1)='4' /* residential only */
AND C.SalesInstr='WD' /* warranty deeds */
GROUP BY A.geoid_10
HAVING COUNT(*) >= 5 /* only tracts w/ 5+ sales */ ;
```

- line 5 IN allows selection of multiple values
- line 9 HAVING filters GROUP BY results

Spatial analysis using census data

Detroit tract level tenure

- Collect 5-year ACS data for all tracts in Wayne County from American Fact Finder
- Import into 'workshop_spatal.sqlite'
- Replace '.' in field names with '_'

```
CREATE TABLE "ACS_11_5YR_B25003" (
"GEO_id" INTEGER,
"GEO_id2" INTEGER,
"GEO_label" TEXT,
"HD01_VD01" INTEGER,
"HD02_VD01" INTEGER
"HD01_VD02" INTEGER
"HD02_VD02" INTEGER
"HD01_VD03" INTEGER
"HD02_VD03" INTEGER)
```

Join to census tract polygons

- Calculate homeownership rate
- Divide homeowner households by total households
- Metadata in 'ACS_11_5YR_B25003_metadata.csv'

```
SELECT A.geoid_10,
( B.HD01_VD02 * 1.0 / B.HD01_VD01 ) * 100 AS HO_rate,
A.geometry
FROM detroit_tracts10 AS A JOIN ACS_11_5YR_B25003 AS B
ON A.geoid_10 = B.GEO_id2
WHERE B.HD01_VD01 > 0; /* remove tracts w/ 0 households */
```

Make choropleth map of homeownership rate

- Layers created from DB Manager in QGIS return all values as *string*
- Need to use the expression dialog in QGIS to convert to *real*

```
to_real( "HO_rate" )
```

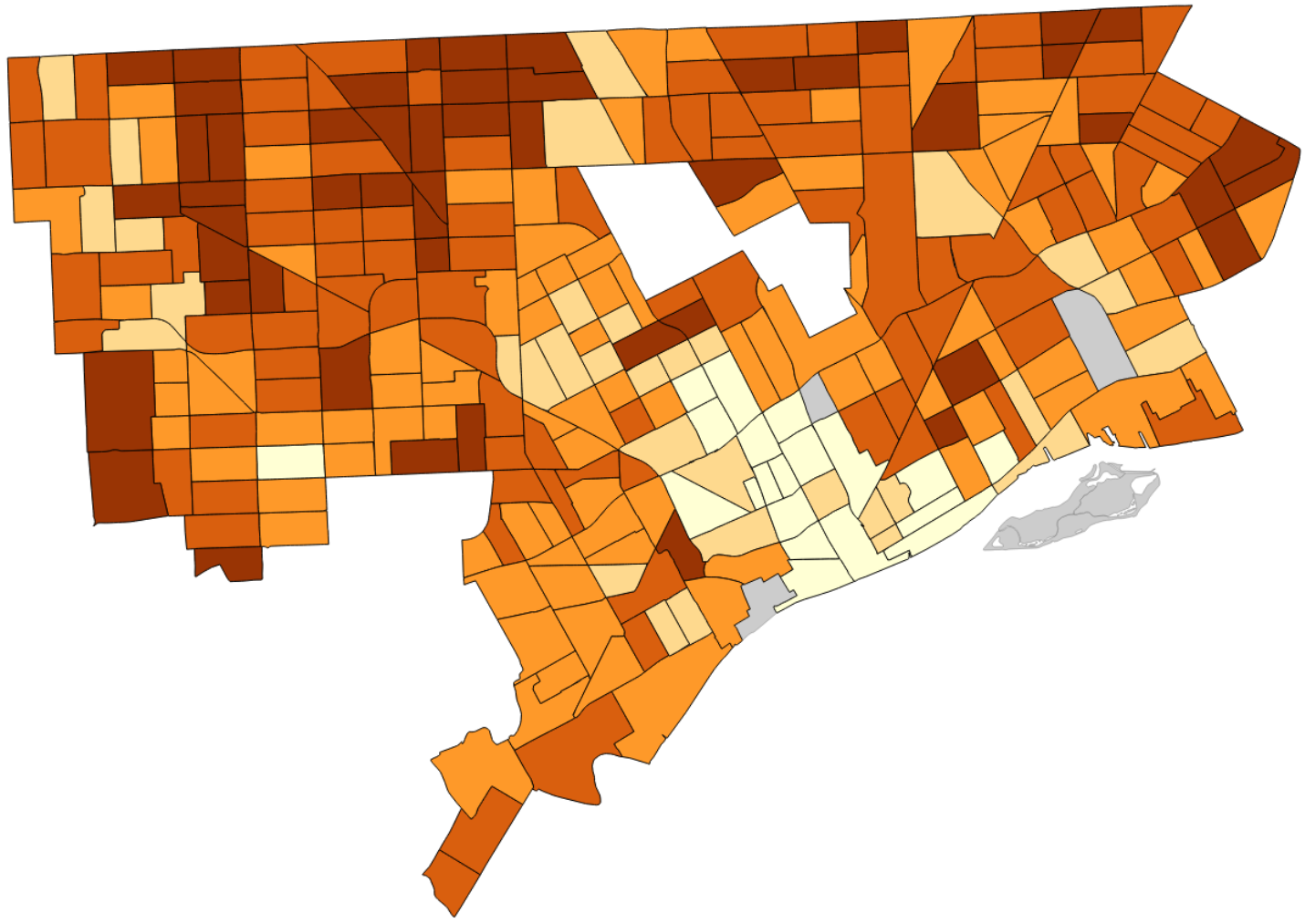

Figure 4: Homeownership Rate

Tax foreclosure choropleth map

Steps

- Download shapefile of dataset from Data Driven Detroit
- Save data into 'workshop_spatial.db'
- Set CRS 2898 like rest of spatial tables in db
- Note these are *point* data

Count foreclosures by tract

```
SELECT A.tractce_10, COUNT(*) AS foreclosure, A.geometry
FROM detroit_tracts10 AS A, tax_foreclosures AS B
WHERE st_contains(A.geometry, B.geometry)
AND B.ROWID IN (
    SELECT ROWID
    FROM SpatialIndex
    WHERE f_table_name="tax_foreclosures"
    AND search_frame=A.geometry)
GROUP BY A.tractce_10;
```

Tax foreclosure concentration

- Join count of parcels per tract to normalize count
- Multiply by 1,000 to get rate

```
SELECT A.geoid_10,
COUNT(*) * 1.0 / C.parcels * 1000 AS rate,
A.geometry
FROM detroit_tracts10 AS A, tax_foreclosures AS B
JOIN
    (SELECT geoid10_tr, COUNT(*) AS parcels /* subquery of
    FROM mcm
    GROUP BY geoid10_tr) AS C
ON A.geoid_10 = C.geoid10_tr
WHERE st_contains(A.geometry, B.geometry)
AND B.ROWID IN (
    SELECT ROWID
    FROM SpatialIndex
    WHERE f_table_name="tax_foreclosures"
    AND search_frame=A.geometry)
```

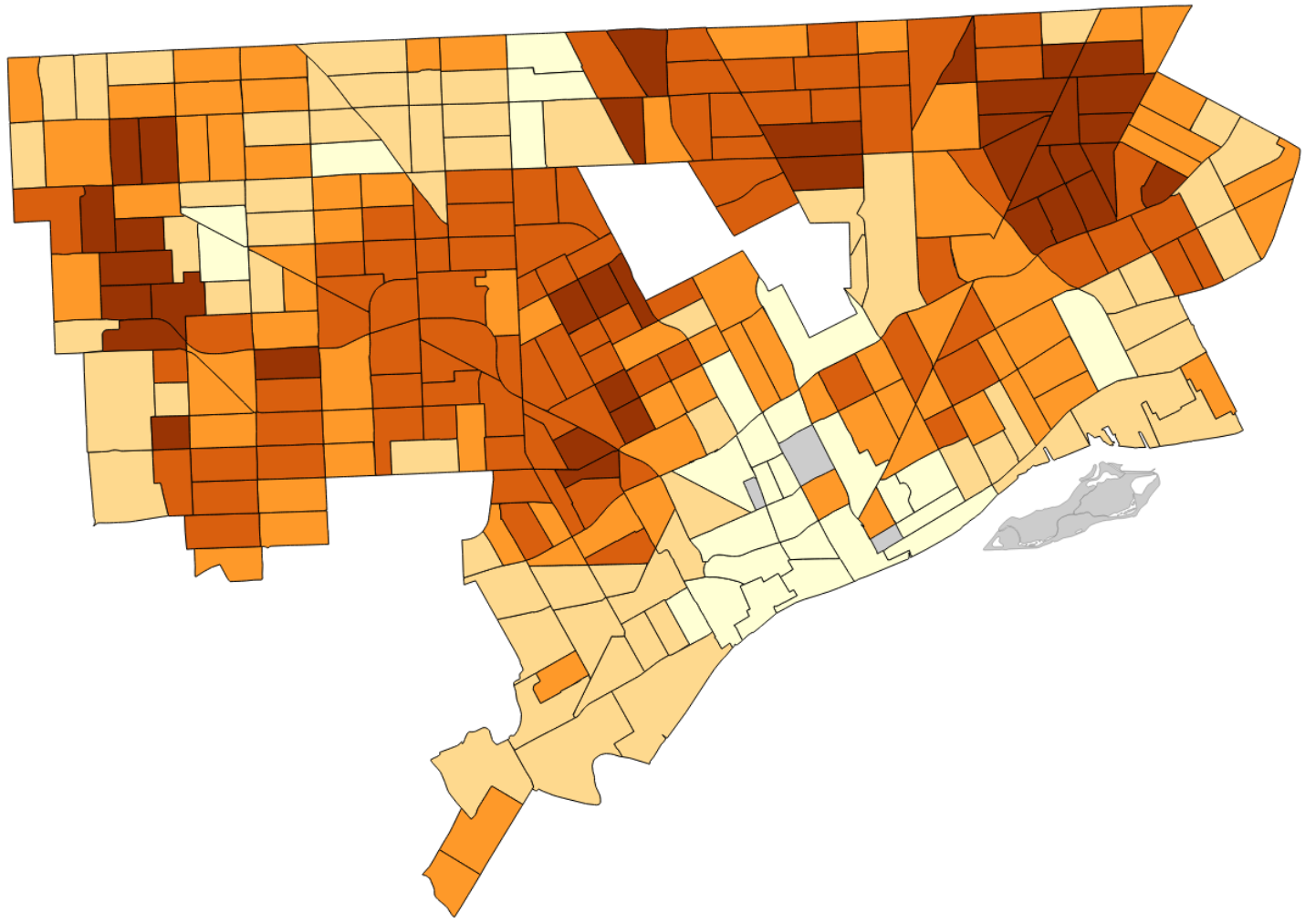

Figure 5: Tax Foreclosure Rate

Property Ownership Analysis

Detroit Property Ownership

- Shapefile from Detroit Open Data Portal
- Contains polygon for every parcel in Detroit
- Time period reported to be March 2016
- Drag .shp file into QGIS and save into spatialite db
- We will find largest property owners
- And location of Fannie Mae's properties

Largest Property Owners

```
SELECT taxpayer1, COUNT(*)
FROM ownership
GROUP BY taxpayer1
ORDER BY COUNT(*) DESC
LIMIT 7;
```

taxpayer	COUNT
DETROIT LAND BANK AUTHORITY	85479
CITY OF DETROIT-P&DD	5964
MI LAND BANK FAST TRACK AUTH	5878
HANTZ WOODLANDS LLC	1917
TAXPAYER	1235
CITY OF DETROIT - P&DD	1058
CITY OF DETROIT	950
HUD	782