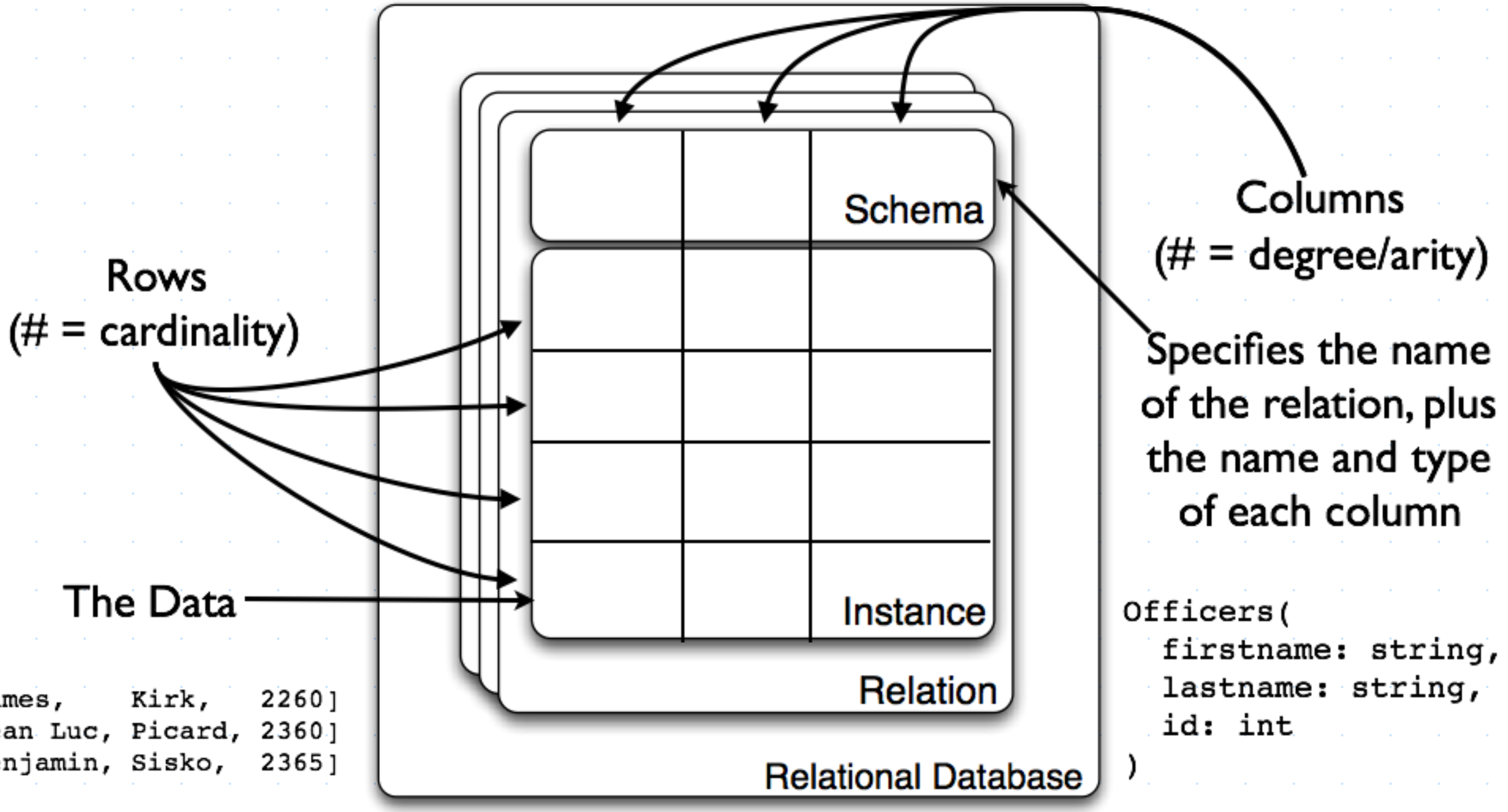
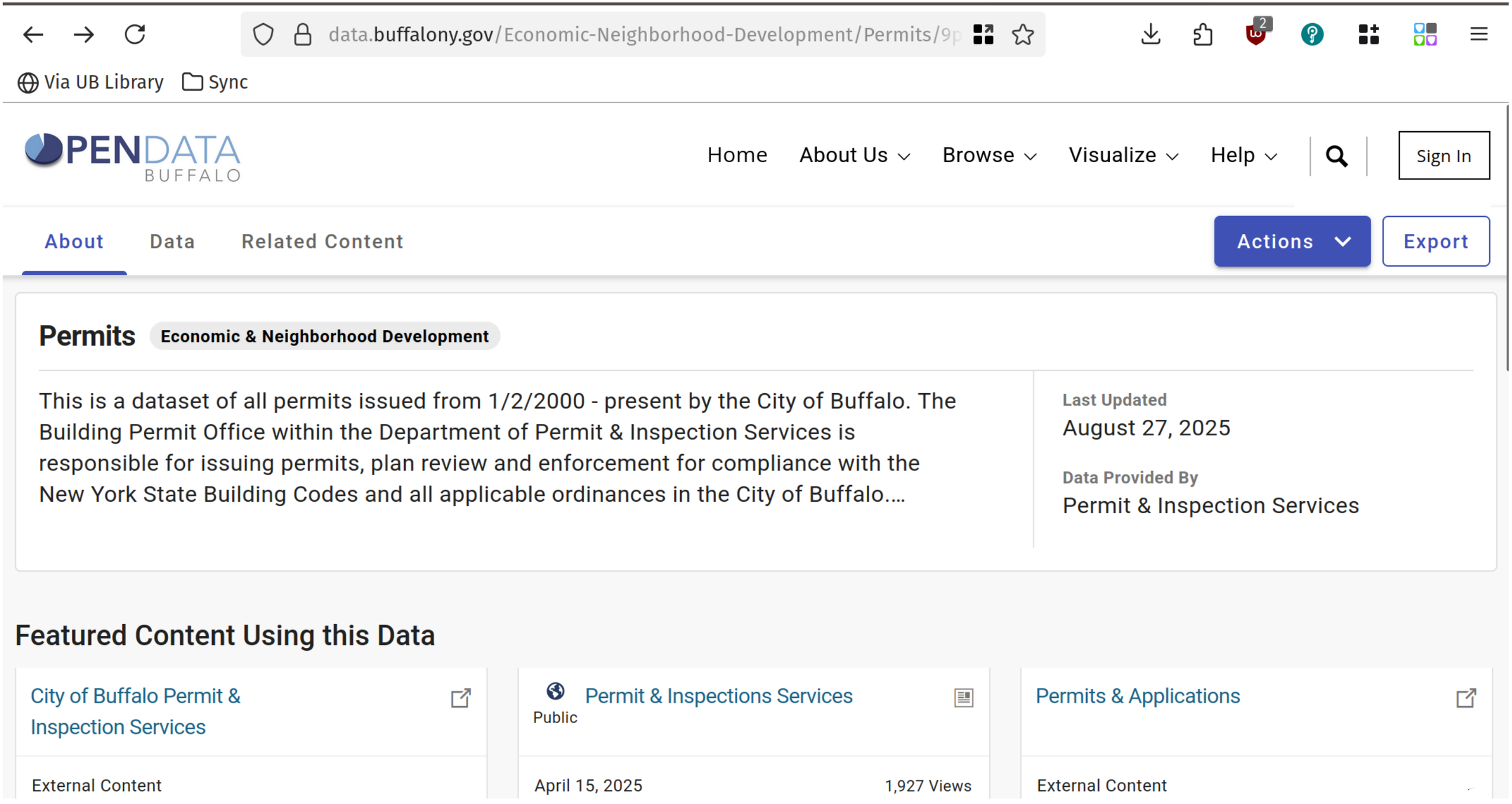
CSE 350

Advanced Data Structures

Lecture 2: Tidy Data


Rows
(# = cardinality)
The Data
[James, Kirk, 2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko, 2365]
Schema
Instance
Relation
Relational Database
Columns
(# = degree/arity)
Specifies the name of the relation, plus the name and type of each column
Officers(
firstname: string,
lastname: string,
id: int
)

Example Data



Explore: Examples/Permits.csv

1. What's the schema?

Permits, permit type, - - -

2. What's the arity?

37

3. What's the cardinality?

999

4. Are the records of the table a set or a bag? Why?

Bag - No guarantee of uniqueness in a csv file

Set - Conceptually unique way to reference records

4. If you're in charge of the CSV file, what could go wrong?

- Accidental Deletion
- Invalid Values
- Inconsistency

Goals for ...

Storage

No Redundancy

Uniquely Identifiable data

Usage

Asking Questions

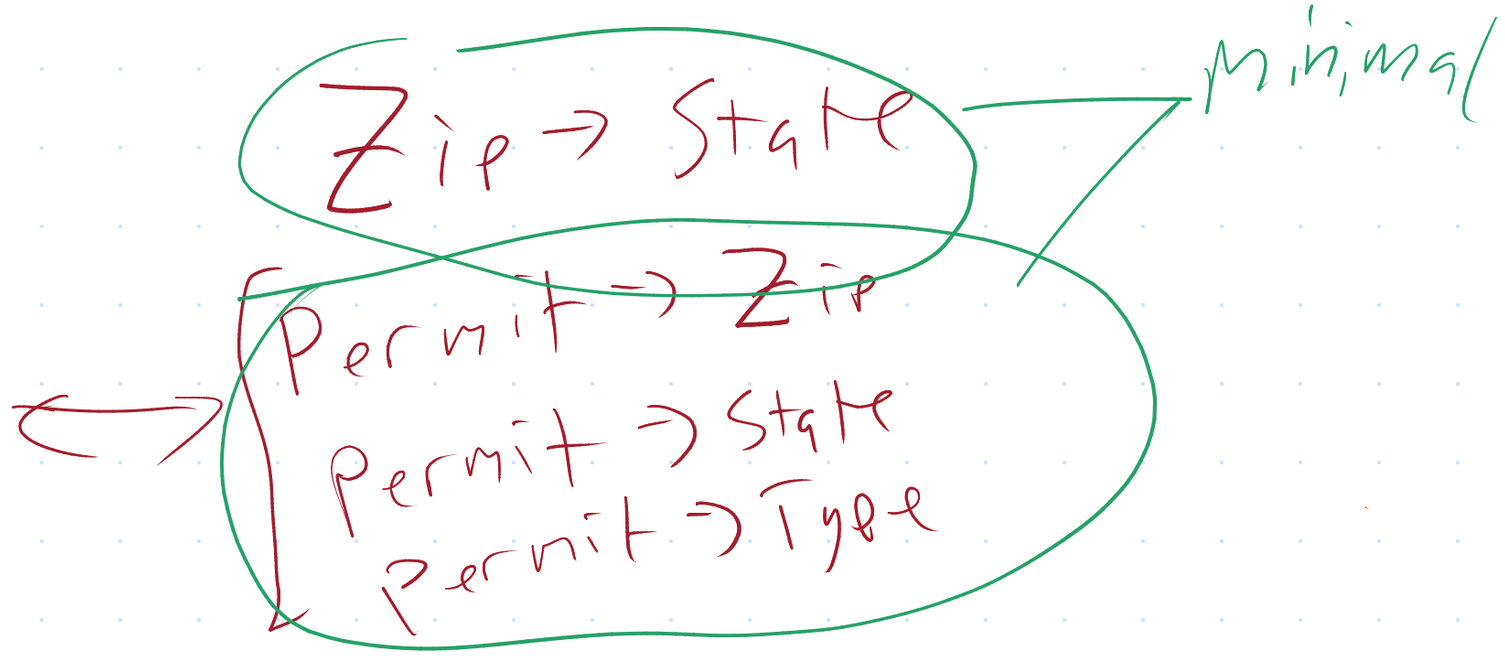
Functional Dependencies

Permit # Zip Code State Permit Type

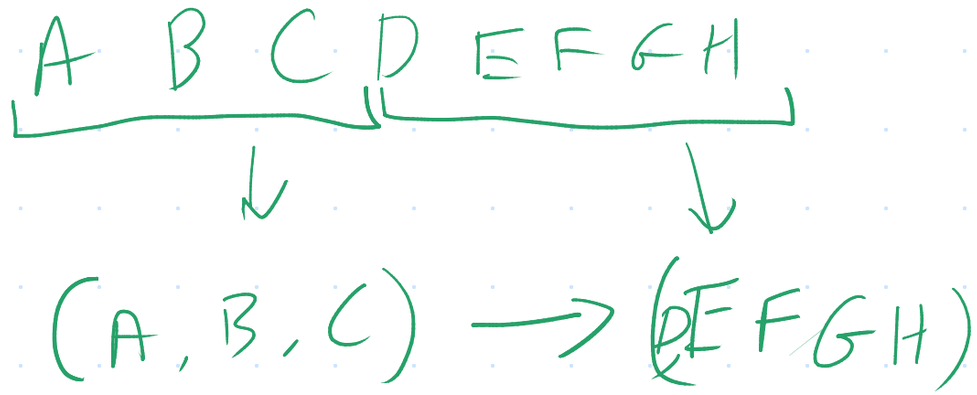
A → B

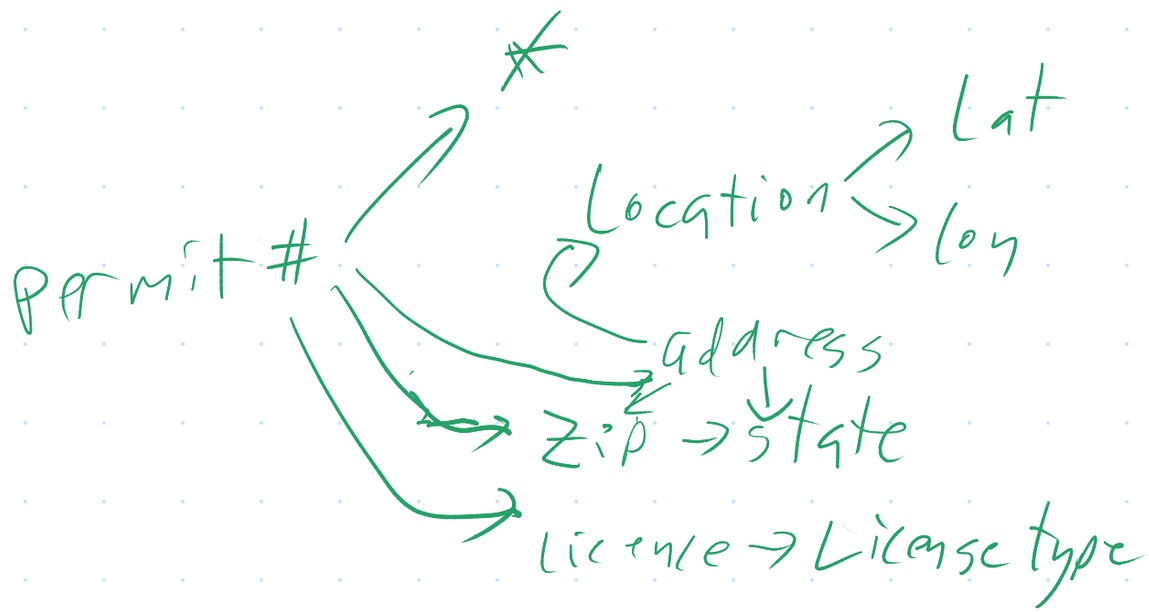
Permit #	Zip	State	Type
123	14228	NY	Electrical
125	14228	NY	Construction
127	10001	NY	Electrical

Permit #, Zip → State
Permit #, Zip → Type
Permit #, Zip → Zip



Superkeys

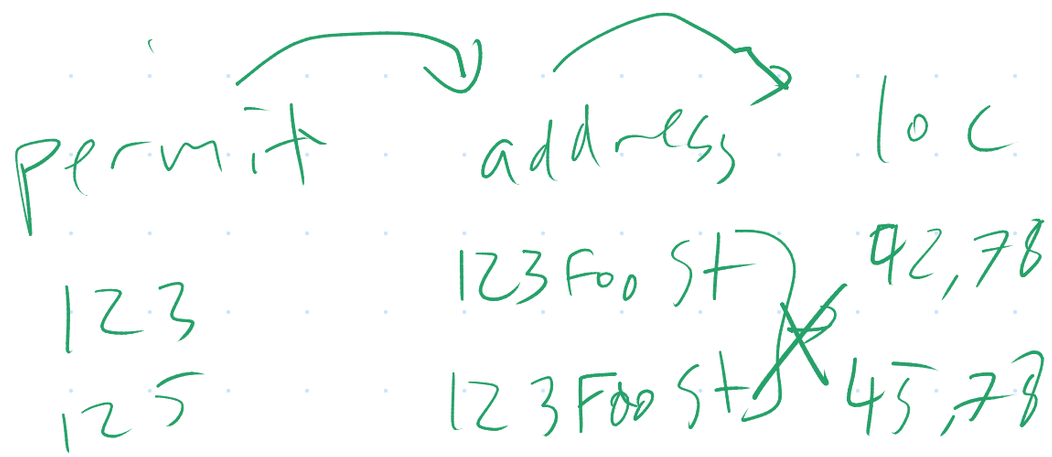




Candidate keys

permit #

Primary Keys



Rules of Tidy Data

1. One 'cell', one value
(is this broken anywhere?)

Location = Latitude & longitude

1 Normal Form

2. One key to rule them all
(is this broken anywhere?)

2 Normal Form

3. No delegation
(is this broken anywhere?)

3 Normal Form

permit → address → location

permit → address

address → loc

permits.csv

addresses.csv

Guidelines for Schema Design

1. What is the 'thing' that one record in the table is meant to model?

2. Can the 'thing' be broken down into smaller component things?

3. Does each 'thing' have a name by which it can be uniquely identified?

4. Is there one canonical record where you would expect to find the 'thing'?

Schema design for Permits.csv

Permits(PID, Type, Zip, State)

↳ Permits(PID, Type, Zip)

States(Zip, State)

The Raw Table ADT

Table

get_cursor()

insert(row)

Cursor

get() → row

next() → bool ← at end?

prev() → bool

update(row)

delete()

seek(position)

... but that's awkward. How about something easier?

Project (Map)

Π_{city} (Permits) → Table w/ 1 row per permit

```
output = []
for row in Permits:
    output.push(row["city"])
return output
```

Select (Filter)

$\sigma_{city = "Buffalo"}$ (Permits) → Table w/ only Buffalo permits

```
output = []
for row in permits:
    if row["city"] = "Buffalo"
        output.push(row)
return output
```

Union

A ∪ B

Join (Flatmap)

R	A	B
	1	2
	1	3
	2	2

S	B	C
	1	3
	2	4
	3	5

```
output = []
for r in R
  for s in S
    output.push(r+s)
```

R×S	A	R_B	S_B	C
	1	2	2	4
	1	2	3	5
	1	3	3	5
	1	3	2	4
	2	2	3	5

$$\sigma_{R.B=S.B}(R \times S) = R \bowtie_{R.B=S.B} S$$

"Equi join"

Implicit equality

$R \bowtie S$ "Natural Join"

PermitsOld(PermitID, —, Zip, State)

Permits(PID, —→ Zip)

States(<u>Zip</u>, State)

↑ Key

Permits ⋈ State

P

$$Q(PID, State) = \Pi_{\underline{PID}, \underline{State}}(Permits \bowtie States)$$

$$= \Pi_{PID, State}(\Pi_{PID, Zip}(Permits) \bowtie \Pi_{Zip, State}(States))$$

[Group-By] Aggregate (Fold)

{A, B, C, D, ... ____ }

init → x

accum → (value, x) → x

postprocess → x → result

```
x = init()
for r in R:
    x = accum(r, x)
output(postprocess(x))
```

SUM

init = 0

accum(r, x) = r + x

postprocess(x) = x

Average

init = <0, 0>

accum(r, <x_c, x_s>) = <x_c + 1, x_s + r>

postprocess(<x_c, x_s>) = $\frac{x_s}{x_c}$

$$\Sigma_{sum(price)}\ Permits$$

$$\Sigma_{type,\ \underline{sum(price)}}\ Permits$$

group by attribute

aggregate

Others: Unique, Sort, Limit, Window

Why stick to these simple operators?

- Composable: Simple but expressive
- Easy to implement!
- Easy to implement efficiently
- Clear rules for when they can be rewritten (see 462)

... Even more friendly!

```
SELECT Applicant, Zipcode
FROM Permits
WHERE `Permit Type` = 'REPAIR'
```

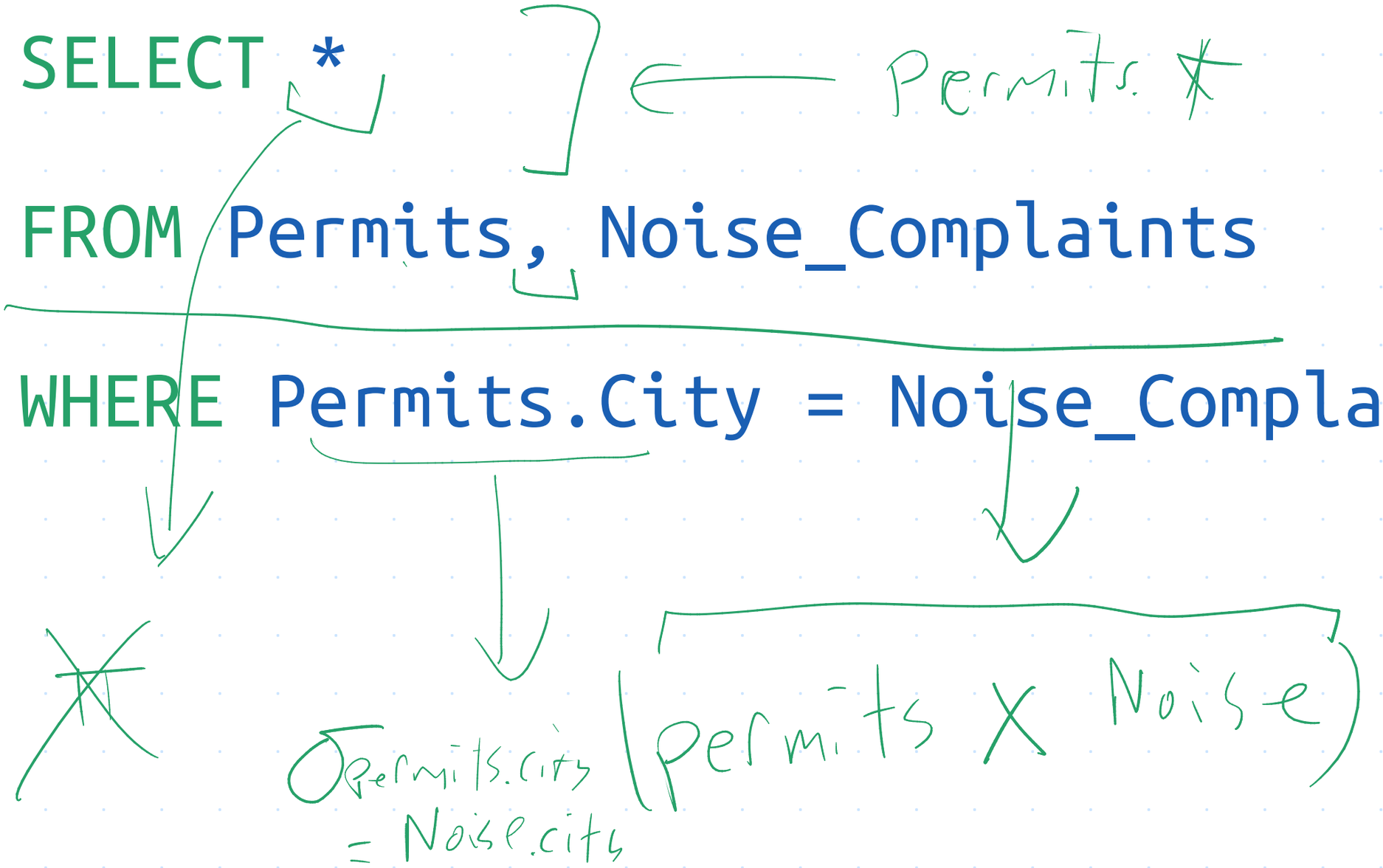
$\Pi_{App,Zip}\left(\sigma_{\substack{PermitType \\ =Repair}}(permits)\right)$

```
SELECT DISTINCT `Permit Type`

FROM Permits

ORDER BY
    'Permit Type'
```

output must be a set

output must be a list

```
SELECT *
FROM Permits, Noise_Complaints
WHERE Permits.City = Noise_Complaints.City
```



```
SELECT COUNT(*)
FROM Permits, Noise_Complaints
WHERE Permits.City = Noise_Complaints.City
```

$$\sum_{count(*)} \left(\sigma_{Permits.city = Noise.city} (permits \times Noise) \right)$$

```
SELECT `Permit Type`, COUNT(*)

FROM Permits, Noise_Complaints

WHERE Permits.City = Noise_Complaints.City
```

GROUP BY 'Permit Type', Zip

$\sum_{\text{permitT, count(*)}}$ ($\sigma_{\text{Permits.city} = \text{Noise.city}}$ (permits X Noise))

More in a a few lectures...

How does this tie to data structures?

σ[Permit Type = 'Repair'](Permits)

HashMap

USE

108416|USE|07/02/2007|||||||||$0.00|0|||NONE|||||||||||||||||||||

17761|ELECTRICAL|03/21/2000|||||||||$49.00|$0.00|0|||NONE||WIRE 5 TON A C UNIT (LAUNDRY MAT)|||||||||||||||||||

17172|ELECTRICAL|03/01/2000||||||MEL11-195040|MASTER ELECTRICIAN|$49.00|$0.00|0|||NONE||INSTAL ADELPHIA CABLE POWER SUPPLY ON TELEPHONE POLE|||||||||||||||||||

ELECTRICAL

3

Permits ⋈[City] Noise_Complaints

```
for p in Permits
    for n in Noise
        if p.city = n.city
            output (p, n)
```

$O(|Permits| \cdot |Noise|)$

```
index = {}
for p in Permits
    index[p.city] += p
for n in Noise
    for p in index[n.city]
        output(p, n)
```

$O(|Permits| + |Noise| + |output|)$

$$\Sigma[\text{Street}, \text{COUNT}(*)]\ (\ \text{Permits}\ )$$

Many of these

Big!