Ethics, Normalization, and Python Integration in Databases

```
$ echo "Data Sciences Institute"
```

Beyond SQL:

Ethics

Normal Forms

Run SQLite in Python

Ethics

Data Ethics issues have seen a huge increase in papers on the subject and are more and more newsworthy.

Many of these issues involve SQL databases, but not explicitly. SQL is rarely the culprit, but instead part of a system affected by *other* issues: a database full of credit cards is leaked, personal data is sold to a third party without consent, records are destroyed either intentionally or by accident.

Ethics

But Data Ethics are still an important part of any well-rounded SQL course. We will consider:

- privacy and privacy policies
- security and appropriate access to data
- how database design affects our perception
- labour rights for data entry
- bias in data entry

A Personal Story

- In my first job, our most Senior Business Intelligence Analyst was involved in a motorcycle accident
- All of us were terribly concerned:
  - We had very little details of his status, other than he was in the ER and later ICU
  - We knew his 8-year old son had been involved
  - Motorcycle accidents are usually *bad*
- We were all working for the state of Colorado's largest healthcare provider, and he ended up at one of our hospitals
  - Being the organization's team of data analysts we had access to every patient's medical record

A Personal Story

- What are the ethical issues important to this story? 💭💬 Think, Pair, Share

A Personal Story

- Privacy: we were legally and morally required to respect his privacy
  - Even if our thoughts well-intentions, we did not have his consent to these details
- Policies: his data was protected under HIPAA (PHIPA in Canada)
- Security: the organization protected his data from as many people as possible
  - We still had our job functions, requiring some access
    - The organization had logging functions in place to ensure none of us looked
  - Systems were designed to rigourously protect all patient data, including his
- He lived, with a long road to recovery
- His son had very few injuries and recovered quickly
- None of us looked at any of his records

Pakistani Computerized National ID Card

- Read: Qadri, R. (2021, November 11). *When Databases Get to Define Family*. Wired.

  https://www.wired.com/story/pakistan-digital-database-family-design/

Pakistani Computerized National ID Card

- **What values systems are embedded in databases and data systems you encounter in your day-to-day life?**

Pakistani Computerized National ID Card

- Fairness: Our expectations of what is "normal" can inform software and database design
  - Many studies show that lack of diversity in tech creates a narrow viewpoint
    - Design choices can reinforce harmful stereotypes, be exclusionary, further marginalize people
    - Shifting norms...?
- Inequality: Data and data systems are representation of history, but influence the present and the future
  - This effect is compounded when we add automation to these data systems
    - These systems often claim to address bias and inequity, but are better at disguising it
    - Allow us to escape the difficulties of making decision about actual people
  - What gets counted? Who gets counted?

It's All People

- Read: Boykis, V. (2019, October 16). *Neural nets are just people all the way down*. Normcore Tech.

  https://vicki.substack.com/p/neural-nets-are-just-people-all-the

It's All People

- What are the ethical issues important to this story?

It's All People

- Labour: like Machine Learning, SQL databases are built on a foundation human labour
  - Much of this labour is invisible
    - I was aware that much of the health data I used was produced by "coders", but generally only thought about these people when there were mistakes
- Bias: Humans make choices, labels, determinations before a neural network ever exists
  - Humans can create bias and are subject to certain pre-conceptions (i.e. purposes for labelling) that will create different outcomes
    - Biases have to be addressed/moderated...more human labour!

Beyond SQL:

Ethics

Normal Forms

Run SQLite in Python

Purpose

- Databases often become more useful if they are *normalized*
    - When a table is normalized, it splits complex data stored in single columns and stores them instead in many smaller tables
    - Our farmersmarket.db is normalized
- When a collection of databases is normalized we call this a data warehouse
- There's a fine balance between the number of tables created and what is stored in any single table
    - The degree of normalization is based on criteria defined by "forms"
    - e.g. our product table could be further normalized:
        - we could place product_qty_type into its own table and reference it with an id.

First Normal Form

- First Normal Form (1NF) requires that each column contain one single value
- Many tables are in 1NF by default

Consider a non-normalized table:

name	OS	software	supervisor
A	win	VSCode, MSSQL, RStudio	Eric Yu
Thomas	mac	Spyder, SQLite, RStudio	Rohan Alexander

We can shift it into 1NF by unpivoting the software column:

name	OS	software	supervisor
A	win	VSCode	Eric Yu
A	win	MSSQL	Eric Yu
A	win	RStudio	Eric Yu
Thomas	mac	Spyder	Rohan Alexander
Thomas	mac	SQLite	Rohan Alexander
Thomas	mac	RStudio	Rohan Alexander

(SQLite doesn't support `UNPIVOT`, so we'll use `SUBSTR` and `UNION` to create this)

Second Normal Form

- Second Normal Form (2NF) requires that each non-key column is dependent on the primary key
    - Therefore, no row deletions can affect the integrity of another table
- Our farmersmarket.db is 2NF!

Our table is currently 1NF, which is required for 2NF:

name	OS	software	supervisor
A	win	VSCode	Eric Yu
A	win	MSSQL	Eric Yu
A	win	RStudio	Eric Yu
Thomas	mac	Spyder	Rohan Alexander
Thomas	mac	SQLite	Rohan Alexander
Thomas	mac	RStudio	Rohan Alexander

Second Normal Form

2NF requires that we prevent supervisors from being deleted, should A or myself leave the school.

Supervisors now becomes its own table:

id	name
1	Eric Yu
2	Rohan Alexander

Second Normal Form

We introduce a PK id and supervisor_id becomes our FK, replacing supervisor:

id	name	OS	supervisor_id
1	A	win	1
2	Thomas	mac	2

...and student_software now references that PK:

id	software
1	VSCode
1	MSSQL
1	RStudio
2	Spyder
2	SQLite
2	RStudio

Third Normal Form

- Third Normal Form (3NF) requires that we replace any non-key transitive functional dependency
    - This means if any non-key column's value changed, and therefore another non-key value would be invalidated, we must replace this dependency with a table relationship instead

Our table is currently 2NF, which is required for 3NF:

id	name	OS	supervisor_id
1	A	win	1
2	Thomas	mac	2

Third Normal Form

Because MSSQL is only available on Windows, any change in OS will change whether MSSQL can be installed

OS must become its own table:

OS_id	OS	win_only
1	win	TRUE
2	mac	FALSE

...and we must create a software table referencing it

software_id	software	win_only
1	MSSQL	TRUE
2	RStudio	FALSE
3	VSCode	FALSE
4	SQLite	FALSE
5	Spyder	FALSE

Third Normal Form

At last, we have 3NF, for both our student_software table:

id	OS_id	software_id
1	1	1
1	1	2
1	1	3
2	2	2
2	2	4
2	2	5

...and our original table:

id	name	OS_id	supervisor_id
1	A	1	1
2	Thomas	2	2

Normal Forms

(Normal Forms live coding, putting the previous example into action!)

What questions do you have about normal forms?

Beyond SQL:

Ethics

Normal Forms

Run SQLite in Python

Run SQLite in Python

- We have seen how to connect a SQLite database to Python and interact with tables
- **...but did you know you can also run SQLite queries on Python dataframe objects?**
- Both languages offer good support
    - Often there is corresponding Python syntax to achieve similar results
        - As always with data, there is no prescribed way of doing something!

Python Example

```
import pandas as pd
import pandasql as sql #this allows us to run SQLite queries!

p = "https://raw.githubusercontent.com/allisonhorst/palmerpenguins/master/inst/extdata/penguins.csv"
penguins = pd.read_csv(p) #create a dataframe
yrly_penguins = sql.sqldf('''SELECT DISTINCT year, COUNT(*) AS count,
                          SUM(COUNT(*)) OVER (ORDER BY year) AS running_total
                          FROM penguins
                          GROUP BY year''') #run a SQLite query with sqldf()
```

year	count	running_total
2007	110	110
2008	114	224
2009	120	344

What questions do you have about anything from today?