DATA 13600 Final Study Guide

Anthony Yoon

2024 Spring Quarter

DISCLAIMER

THIS IS NOT A BE ALL END ALL FINAL GUIDE. THIS IS JUST A FINAL STUDY GUIDE THAT A STUDENT MADE. PLEASE REFER TO APPROPRIATE NOTES THAT ARE PUBLISHED BY THE PROFESSOR.

Lecture 0 and 1: Basic File Processing

The UNIX command line is a powerful tool that can be used for many different types of data manipulation. It is a very efficient way of doing things, as it takes things from memory directly which yields results.

Unix Command: *Cat*

This command will quite literally print out the whole contents of the file.
For example:

$ cat data-eng.csv

Will literally print out each of the lines in the file.

However, there is the question of what happens when there exists files like the CSV (Comma Separated File) Format. This can be seen with the following file

field1,field2,field3
record1,1, 1905
record2,2, 1906

The command ***cut*** can take this file and return workable items

Unix Command: *cut*

This command will cut on a specific character and return a "list" of items. One thing to note is that this command will operate on a **Line by Line** basis.
For example the code

cut -d ',' -f5 data-eng.csv

It will **cut** on the delimiter (denoted by the '**,**') and return the **fifth** value in the **data-eng.csv** file.

Unix Command: *grep*

This command literally returns the lines that contain the certain parameter of interest. The command:

$ grep Economics data-eng.csv

This line will return all lines that contain the word of interest: ***Economics*** If you want to count all values, then you can use this command.

$ grep -c Economics data-eng.csv

Do note that there can be instances of double counting that can occur. For example, with the above command, *if there are more than one instance of economics within a line, the word will be counted twice.*

Unix Command: *Piping*

Connecting one input into the other. This is read from left to right, just for clarity. You can think of this as taking one input and **piping** it into another. This is usually denoted by the symbol —, and this is the main way of chaining multiple unix commands together. For example:

$ cut -d ',' -f5 data-eng.csv | grep -c Economics

Will first execute the cut command, which will return only the fifth entry within the CSV file where the comma is the main delimiter. Then this input is **piped using**

the — sign to the next input, where the grep command is executed. Do note that the order of operations does matter, as doing this:

$ grep Economics data-eng.csv | cut -d ',' -f5 data-eng.csv

Are completely different commands.

Unix Command: *join*

Think of a SQL merge. This will join on a common entry. You can look at actual data and see how this works, but the command is done like this when working with a txt file that has commas as the main delimiter

$ join -t, 1-2 -2 1 file1.txt file2.txt

Anything after the **-t** is the delimiter, which in this case is the comma. The following **-1 2** and **-2 1** mean that the command will join on the 2nd term on the 1st file *(Think of the* **-** *as the indicator of file)* and the 1st term on the 2nd file.

Unix Command: *sed*

This will look for certain strings and replace them. The main format of this command is expressed as follows:

$ sed 's/(Thing being replaced)/('Thing that *will replace* these values) name.txt

So if the command was written as

$ sed 's/sed/awk/' names.txt

where the file contains the following:

This is a test file.
It contains some sample text
we will use sed to do replacements

Running the above command will return the following

This is a test file.
It contains some sample text
we will use awk to do replacements

Why is any of this relevant?

[1] Storing data is a crucial part of life. We can learn from the past, and thus make predictions based on past behavior. There are some things happening in the past that we can use to make assumptions about the future. Hence, the key part of all of this:

Prediction

This is brought about by recent trends in technology

1. **Digitization**: Things are digital now. Enough said
2. **Price of storage**: The price of digital storage has gone down tremendously.
3. **Standardization**: The Internet allows for easy transfer of data, and also standard file names
4. **Machine Learning and AI**: This is very self explanatory. We can now use train AI and ML models to make good predictions in the long run.

Data Models

Data in its digital form is very artificial. Thus we must set some rules on it. This can be seen in spreadsheets and such. The following table is an representation of this.

Make	Model	Year	Color
Toyota	Corolla	2018	Red
Toyota	Camry	2015	Blue
Honda	Fit	2014	Black

The data could also make references to other pieces of data. This will be covered later in more depth, but for now there are 2 types of data models.

- **Logical Data Model**: A model that describes the semantics of how the data is stored.
- **Physical Data Model**: Describes how the data is physically stored within computer hardware.

[1] This is not really important, there is no way that we are going to be tested on any of this, but might as well put this on here for completion sake lmao

Types of Data

- **Tabular Data**: Think spreadsheet or table. Each Column is fixed to a domain, and each row can change.
- **Semi-Structured Data**: Data structure that does not have any solid structure. Rather, things within the data can change. I.E
  {'name': Car1, 'features': [BackupCamera, CruiseControl]}
  {'name': Car2, 'features': [BackupCamera, ParkAssist, CruiseControl]}
- **Named Attribute**: Properties of a given row can bve accesed by using a name.

Notably, there are two main perspectives on what data can be. There is the **database** and **statistical** view.

Database are tabular, and can use queries to access data. All of this is made under the closed world assumption, where every fact is read as true and everything else is read as false.[2] The statistical view of things sees things as samples of a broader population.

Lecture 2 and 3: SQL

Important note: If you have taken DATA 118/119, a lot of the following is trivial. For the sake of a global study guide, I will write in the bare minimum for this.

to create SQL Tables, the following syntax is generally used.

CREATE TABLE table name(
column1 datatype constraints
column2 datatype constraints
column3 datatype constraints)

and so forth. Datatypes include **INT, VARCHAR, DATE** and many many more. Constraints include **PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE**. Then from here, run the **CREATE TABLE** command to get the table created. You can also use the **ALTER TABLE** command to alter the structure of the table and you can also add data into the table using the command **INSERT**. The following is

[2]The following is not that important and honestly I have no clue what this means

used to insert data.

INSERT INTO tablename (column1, column2, column3, ...)
VALUES (value1, value2, value3

The following will be used to update values in the table.
UPDATE tablename
SET column1 = value1, column2 = value2, ...
WHERE condition;

Reading SQL Data: AKA SQL Queries

Again, if you are coming from 118/119, this has already been covered. So this will be a very small overview of this
The basic order of the SQL query:

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
An explanation of each thing is as follows:

1. **SELECT**: **SELECT** columns from the databases. Does not matter where they come from if they have unique column names. If they have different column names, then specify using *TableName.ColumnName*.

2. **FROM**: **FROM** which table are you **SELECTING** from.

3. **WHERE**: Pick data **WHERE** it meets this condition

4. **GROUP BY**: If you have used pandas, this is very similar to the group by there. Basically, you will aggregate common areas in one area. For example,

Job	Salary
Banker	150000
Banker	180000
Software Engineer	50000
Software Engineer	80000

if you were to group by the Job using the **SUM**, you will be returned.

Job	Salary
Banker	150000 + 180000 = 330000
Software Engineer	50000 + 80000 = 130000

5. **HAVING**: Summary function for aggregate data, usually done after group by
6. **ORDER BY**: This is an order by function. Basic syntax of this is

   ORDER BY COLUMN1 ASEC | DESC

Entity - Relationship (ER Diagrams)

Basically, these are relationships you can draw in between 2 or more tables. We can represent these things with relationships, like we can in any situation. First, there are 3 types of relationships, **One to One, Many to Many, Many to One**[3]

One to One

Figure 1: One-to-One Relationship

A one to one relationship is a relationship where one thing correlates to another.

Think of ID determining Name

Many to Many

This is a relationship where multiple entries can be related to each other and vice versa.

Think of books in a library.

Books in a library that can be checked out multiple times

[3]A lot of the stuff is what it literally sounds like, for formality, I will be covering it

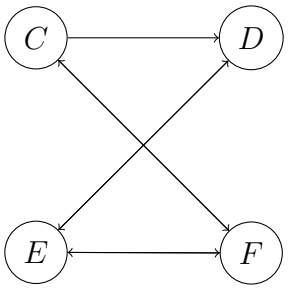

Figure 2: Many-to-Many Relationship

Many to One

This is a relationship is where one entry in a table will have multiple recurrences in other tables.

Think of ID and name. ID will be everywhere, and you can refer name from it

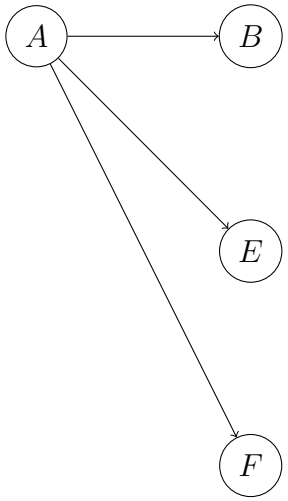

Figure 3: Many-to-One Relationship

THESE ARE INTERCHANGEABLE THINGS.MEANING THAT YOU CAN GO IN ANY ORDER OF OPERATIONS. MANY TO ONE AND ONE TO MANY ARE THE SAME THING DEPENDING ON THE ORDER OF READING

1 Lecture 4: Wrapping up Relational Data Modeling

TL;DR, this is just a join review. Skip if you already know.

Inner Join

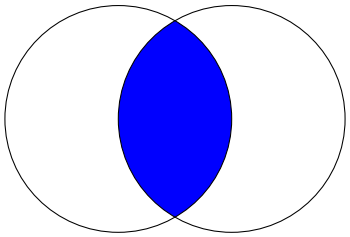

Figure 4: Inner Join

This will join an entry on the common values. I.E values that are present within both tables. The above diagram depresents this.

Right / Left Outer Join

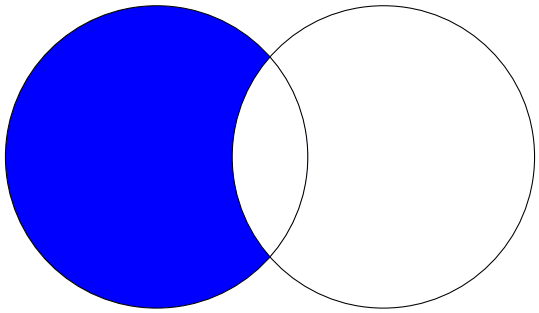

Figure 5: Left Outer Join

Similarly, this above diagram represents what a **LEFT** join does. It will join the values based on the left values. If there are any values that are not present in the left that are present in the right, there will be **NaN** values present.

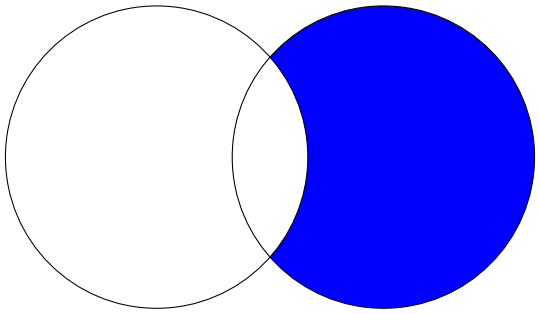

Figure 6: Right Outer Join

Outer Join

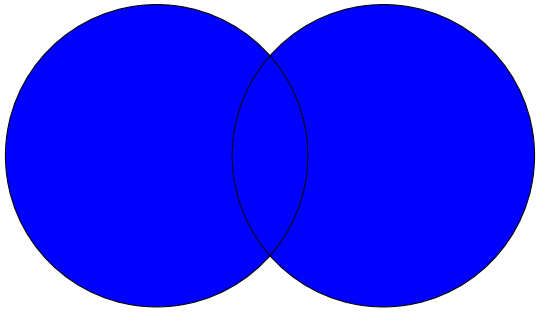

Figure 7: Venn diagram representing an outer join of Sets A and B, including the intersection, without labels.

Basically takes into consideration both tables. If there are missing values in the overlap, return NaN in those respective values.

The same logic applies to the **RIGHT** outer join. The basic syntax for the join in SQL is

(INSERT JOIN HERE) Column1 *ON* Column1 on Column2

Functional Dependencies

This is basically a one to one relationship. One thing will uniquely define the other, where you can think of it as **f(key/unique constraint) = output**. An example of this is in a school system, student ID would be a key constraint as it would help determine name, grade, major, etc. Basically, a key will uniquely determine all other entries in the table.

Lecture 5: ORM Frameworks

We use Django (pronounced jango) in this class. We use python objects to represent entities in a server. So here are the important parts of this framework.

1. **Models**: These serve as databases, where each attribute refers to a column
2. **ModelFields**: These define data types (I.E Integers, Booleans, etc)
3. **Migration**: Files that help serve the database migrations and sync up any changes
4. **QuerySets**: Ways of interacting with DataBases Objects. Think of SQL Querys.
5. **QuerySet Methods**: Ways of going through and manipulating data. These include: filter(), exclude(), get(), annotate(), aggregate()
6. **Lazy Evaluation**: Queries are not executed until everything evaluated.
7. **Database Operations**: Querysets generate SQL and execute.

Basics of setting up Django of Modeling Building

Im skipping this lmao

Django Field Types

If you can read, then this is very, very easy.

Basically, the django model field name is what it is storing

Lecture 6: More ORM and Data Loading

1. **Missing Values**: Literally missing values
2. **Extra Commas or fields**: Literally extra comma
3. **Inconsistent data format**: Literally inconsistent data format.
4. **Out-of-range values**: Literally out of range values.

Service Oriented Architecture (SOA)

This is the idea of splitting up services for different functions.

1. **Modularity**: Services are designed as modular components, each components, each responsible for a specific business function. Think of it as like puzzle pieces.
2. **Loose Coupling**: Operate independently, changes to one service doesn't need others.
3. **Interoperability**: Communication across multiple languages.
4. **Discoverability**: [4] Locate and invoked dynamically at runtime.
5. **Scalability**: Easy to scale up.
6. **Reuse**: This service is very reusable and can be used in a multitude.
7. **Security**: Easy implementation and very safe to implement.

Building Simple Django Services

So to make this very simple there are 3 main files that are used to make a database.

- **urls.py**: These all contain the URLs that are mapped to unique view functions that will execute certain files in the views.py
- **views.py**: These contain python logic that are associated with each URL
- **models.py**: This file contains all the models and respective functions that are used to create the server.

All of these handle many types of things, including types of HTTP responses. These include

- **Request Processing**: Match URL provided by client to an **views.py**
- **View Processing**: Execute logic associated with appropriate logic in **views.py**
- **Response Rendering**: Handles the things that will render in the browser.

[4] I have no idea wtf this is.

What is HTTP processing

HTTP stands for Hypertext Transfer Protocol) and is the underlying protocol used for communication between web browsers and web servers. There are 2 types of communications.

GET

The GET method requests data from a specific point or server. These are done using URL arguments, in where the URL contains the query itself. These will be notated in the form.

$$url.com/seach?q = django\&page = 5$$

The = sign represents a key value pair, and & represents chaining multiple parameters together. These are used to **Pass data to a server, filter and search data in a server, and bookmark or share state in URL form**. In Django, these are usually noted by the .GET function.

POST

The post method is used to submit data to be processed to a specific resource. Often used in submitting forms to a website. When you are using anything that uses a POST request, think of a google form. You first have to create a HTML file that contains fields for people to enter in data, and then create logic in views.py that can take this data and store it. (usually all of this is handled within the views.py file)

Lecture 8: JSON and Serialization

JSON stands for JavaScript Object Notation. It is used because it is easy to write and for machines to parse. It is written in key-value pairs. (Think Dictionary in Python). It is often used to

- **Data Exchange**: Easy to exchange
- **Configuration Files**: Easy to read and write for humans and computers
- **APIs**: Application Programming Interface use this format as ways of transferring data between cilient and server.

Loading and Storing JSON Data

Usually, you would open a JSON file using the *open* command. And you can also *write* a dictionary to a JSON file using the *json.dumps* file. [5]

Lecture 9: Data Retrieval

One thing I would like to mention that if you have taken DATA 119, this is very similar to what we have learned. Basically in a query, this measures which data was relevant to the query. We only use 2 metrics here.

- **Precision**: This is mathematically

$$\frac{TruePositive}{TruePositive + FalsePositive}$$

  In English, this means **Of all positive predictions, how many of them were actually true** or **Of all the items retrieved, how many are relevant?**

  For example, if a search engine retrieves 10 documents in response to a query, and 8 of them are relevant, the precision would be 8/10, or 80%.

- **Recall**: This is mathematically.

$$\frac{TruePositive}{TruePositive + FalseNegative}$$

  In English, this means **Of all the true positive predictions, how many of them were actually true.** or **Of all the relevant items in the database, how many were actually retrieved.**

  For example, if there are 100 relevant documents in the collection, and a search engine retrieves 70 of them in response to a query, the recall would be 70/100, or 70%

Precision and Recall are often inversely related, meaning increasing one will decrease the other.

[5] Note, you must import the JSON package in python to work with JSON files.

Indexing

The idea of attaching an **Index** to data to make storage and stuff easy. Here is the process of this.

1. **Document Collection**: First collect the data.
2. **Tokenization**: Break down the text into smaller units called tokens.
3. **Normaliation**: Convert the tokens into a standard language. This might include converting all letters to lowercase, removing accents or diacritics, and stemming (reducing words to their base or root form)
4. **Index Construction**: Assign an index to each token and store in a data structure. Store in a data structure (The one we are most familiar with are dictionaries)
5. **Query Processing**: When someone submits a query, the program will look up entries based on index and then return that. This is very, very similar to how ChatGPT works.

Example python code is as follows (Credit to Professor Krishan)

```
def build_index(documents):
    index = {}
    for doc_id, doc_text in enumerate(documents):
        terms = tokenize(doc_text)
        for term in terms:
            if term not in index:
                index[term] = [doc_id]
            else:
                if doc_id not in index[term]:
                    index[term].append(doc_id)
    return index

def tokenize(text):
    # Simple tokenization: split text into words (terms)
    return text.lower().split()

# Example documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

```

```python
# Build the inverted index
index = build_index(documents)

# Print the inverted index
for term, doc_ids in index.items():
    print(f"{term}: {doc_ids}")
```

Will Output

```
this: [0, 1, 2, 3]
is: [0, 1, 2, 3]
the: [0, 1, 2, 3]
first: [0, 3]
document.: [0, 3]
second: [1]
and: [2]
third: [2]
one.: [2]
```

Lecture 11

String Similarity

[6] There are varying types of string similarities.

1. **Edit Distance**: The minimum number of edits required to get one string to another.
2. **Jaccard Similarity**: The Intersection ($\cap$) divided by the Union ($\cup$). Or $\frac{A \cap B}{A \cup B}$ if you want to approach things mathematically. The way that this is applied is the number of overlapping words in a string divided by the total number of words (You can become more granular if you want to become really specific.
3. **N-grams**: Breaks strings into continuous sequences of n items and compute similarity based on the similarity on the overlap of these sequences.
4. **Vectorization**: Assign a vector to each word and use cosine similarity to calculate how close things are through angles.

Strengths of Jaccard Similarity

Strenghths:

[6] Lecture 10 was the midterm

1. **Set based Comparison**: Effective at set comparisons where order does not matter. Good at DNA sequence matching, Document analysis, and recommendation systems.

2. **Insensitive to Magnitutde**: Jaccard Similarity only considers the presence or absence of elements in sets. Thus, makes it robust to sets of big sizes.

3. **Simple and Efficient**: Very easy to compute.

Weaknesses of Jaccard Similarity

1. **Limited to Set Comparison**: Jaccard disregards order and frequency within sets. In some cases, not the most ideal.

2. **Sensitive to Set Size**: While Jaccard Similarity is insensitive to magnitude, if the two sets are significantly different in size, then it is heavily affected.

3. **Doesn't Capture Semantic Meaning**: It treats all elements with the same meaning, and does not distinguish between different uses of the word.

Different types of Tokenization

When performing Jaccard Similarity, there are many different types of tokenization we can perform. They go as follows.

1. **Word Tokenization**: Tokenize the words

2. **Sentence Tokenization**: Divide a text into sentences and tokenize on sentences.

3. **Regular Expression**: Splits a text defined by regular expressions, (E.G if divided by spaces, get every word + punctuation)

4. **Whitespace Tokenization**: Splits a text based on "whitespace" markers. Meaning splitting on spaces, lines, or line breaks.

5. **Treebank Tokenization**: Splits based on conventions in the Penn Treebank Corpus, which includes splitting on correct contractions and punctuation.

6. **Punkt Tokenization**: Rule based tokenizer using pre-trained models.

Implementing in Python

Refer to the notes posted. Most of these are recursive algorithms where you take the output and recalculate stuff. One concept that may come in handy is the concept of the **inverse index**. This is basically the key in the dictionary is the word itself and the values are the sentences it appears in.

Lecture 12: Geospatial analysis and others

Geospatial Analysis

Study of data through the eyes of geographic data. This can include longitude, latitude where the python library **Geopy** and **Geopandas** aids in this very well. You can actual start plotting points within the outlines of a map given the featured data points. There are specific examples of codes that you can refer to within the given files, but if I have enough time I will add that later. [7]

Python Search Engine

There are different types of retrieval that are used within the search engine process.

- **Naive Search**: An example of this is going through every single data point and try to manually match strings within strings.
- **Indexed Search**: Uses indexes to search for stuff. If you account for the time it takes to create a index and search, this is significantly worse than naive search. But searching itself is very quick. However, if you are working in big databases, you can save a lot of time by simply indexing.

When search engines are trying correct your query, people can use the *Levenshtein* distance to correct it, which is considered the "naive" approach.

A general approach to optimize these algorithms is to create a **Fast algorithm to find good candidates** then use a **slow algorithm to refine these candidates.** This can be seen within the correction query. One example of this is n-grams, where you can divide the words into n-grams with different amounts of n-grams and then search based on the index. From there, do some sort of naive search based on the values.

[7] I probably will never add it, but refer to the geospatial anlaysis .ipynb files

Lecture 13: Cloud Computing

Cloud computing is the idea that an outside organization handles the organization. Fun story, Amazon had empty servers that were not being used during regular season, so they decided to sell that space to companies who needed that space. Thus this is the origin of AWS.

Benefits and differences of Cloud Computing

Cloud Computing was revolutionary businesses. Here's what matters.

1. **Scalability and Elasticity**: Traditional web infrastructure could not handle spikes in user traffic. However, cloud computing provides scalable and elastic infrastructure, allowing web services to automatically adjust resources based on demand which ensures consistent performance and reliability, even during peak usage periods
2. **Cost Efficiency**: Pay as you go, which is very cost efficient
3. **Global Reach**: Servers and computing servers are everywhere, thus allowing a diverse audience.
4. **Agility and Innovation**: There are many developer tools people can refer and use to help scale up their businesses.
5. **Reliability and Resilience**: The servers are robust and resilience with built in fail safe mechanisms.
6. **Security and Compliance**: Good cybersecurity measures that are inherently built into Cloud Computing.

The basics of virtualization

According to Professor Krishan, "Virtualization in computing is the process of creating a virtual (rather than actual) version of something, such as a server, storage device, network, or even an operating system".

I define it as "virtualization is the process of creating a virtual computers within physical computers". So that would mean allocating physical resources to make mini machines in a big computer. This can be seen with

1. **Resource Pooling**: You can pool physical resources to machines who may need it more.

2. **Multi-Tenancy**: Multiple people can use the same computer with privacy and isolation

3. **Elasticity and Scalability**: Virtualization enables cloud platforms to scale resources up or down dynamically based on workload demands.

4. **Fault Tolerance and High Availability**: In case of failures, you can move around the virtual systems to other places in times of failure.

Lecture 14: Cloud Computing Continued

In class, we were shown how to use AWS and other things to create databases.[8]. Here are the basic steps.

1. **Creating EC2 Instances and Availability zone**:
    - Choose the appropriate instance type based on your application's requirements such as compute optimized, memory optimized, etc.
    - Choose a region to run your stuff.
    - Launch EC2 instances and configure with necessary settings
2. **Deploy the code**: Literally deploy the code
3. **Scaling and Load Balancing**: You can actually have multiple instances of code running. Because of this you need to monitor your usage and balance accordingly.
4. **Monitoring and Logging**: Set up logging to track application errors, user activities, and system events.
5. **Billing and Cost Management**: Manage costs and everything before you do anything.

OLTP vs OLAP Databases

OLTP stands for Online Transaction Processing and **OLAP stands for Online Analytical Processing**. These are 2 different types of database designs that can handle different types of workloads. The differences can be treated as follows:

[8]let's be real, no one really payed attention

- **Purpose**: Like the name suggests, OLTP are designed for transactional workloads, where data is rapidly updated. OLAPs, on the other hand, are designed for analyzing workloads.
- **Data Model**: OLTPs have a normalized data model, which minimizes redundancy and ensures data integrity. This is done by creating relationships between multiple tables. OLAPs are less structured, which allows the easy addition of new data. [9]
- **Workload Characteristics**: OLTP: High volume of short transactions. OLAP: Lower volume with focus on complex aggregation, grouping, and filtering operations.
- **Performance Requirements**: OLTP: These transactions must be as fast as possible. OLAP: Must be scalable to higher datasets.

CAP Theorem

Consists of **C A P**, which are follows. The idea is that **You can have only 2 of these at times**. Examples of each combination is follows. [10]

Consistency	**Availability**	**Partition Tolerance**
Any read operation on the system returns the most recent write operation's value	Every request comes back in a certain time, regardless of correctness or state of system	The system can tolerate environments when some parts of the system fail

- **Consistency and Partition Tolerance**: You are going to sacrifice availability. Your service is going to be able to survive partial shutdowns and be able to consistently give back data, but your data retrieval will not be as quick.
- **Partition Tolerance and Availability**: You sacrifice Consistency. In an event of a partial failure, you won't be able to grab the correct data as well as you can.

And this trend tends to continue through time.

[9] This feels wrong, but I'll check

[10] I am not sure about these examples

GeoSpatial Analysis

I'm skipping this or coming back to this, just know that to refer to the actual lecture notes to see what to do.

Hashing

This is a very important concept. Essentially, what hashing is that you have an input that can be anything, but the output will be an numeric (or alphanumeric) string of fixed length. Here are some uses of hash functions.

1. **Data Retrieval**: Hash functions are used in data structures like hash tables to quickly locate a data record given its search key
2. **Data Integrity Verification**: Let's say you download a file off of the internet and you wanted to see if what was tampered with. A website will provide a "default" hash value and you can hash the file yourself to see if it was tampered with.
3. **Cryptographic Applications**: They are used in cryography all the time.

Quite simply, hash functions can be anything. However, that does mean that there such things as a *good* or *bad* hash function. Thus, we can think of something like the **Simple Uniform Hashing Assumption** is a theoretical concept in the analysis of hash functions and hash tables. It assumes that **each key is equally likely to hash to any of the slots in the hash table, independently of where any other key has hashed to**. In more simpler terms, **every key has an equal probability of being placed in any available slot in the hash table.** With this, there are some properties that come with his that may help in understanding good hash functions.

1. **Deterministic**: Given the same input, a hash function should always produce the same output. This property is crucial for consistency and reliability.
2. **Uniform Distribution**: A good hash function should evenly distribute the hash values across the entire range of possible hash values. This minimizes collisions and ensures efficient use of the hash tables.
3. **Collision Resistance**: A collision is defined as when *2 different inputs produce the same value.* A good hash function should minimize the likelihood of collisions, especially for different inputs. Something to note: *collisions are inevitable, due to the pigeonhole principle*[11]

[11] Pigeonhole principle is the idea that if you have m boxes with n items and $n > m$, then there will be a box with more than 1 item.

4. **Avalanche Effect**: A small change in the input should produce a significant different hash value, hence the avalanche effect.

5. **Preimage Resistance**: It is hard to determine the input given the hash value.

6. **Second Preimage Resistance**: Given an input, it should basically impossible to find another input that produces the same hash value.

Applications of hashing

Hash Indexing

This is a data structure used for efficient searching in databases or other data storage systems. Think of this as a python dictionary. This is how it works in more detail:

1. **Index Storage**: The index structure is often a simple data array, where each entry is referenced a numerical value (in this case a hash) and contains a reference/pointer to the actual data

2. **Hash Function**: This takes a key and turns it into a hash code.

3. **Indexing**: Think of this hash code as an index for the index.

4. **Lookup Operation**: When we want to retrieve data associated with a key, we first compute the hash code for the key using the hash index and them uses that hash code to locate the index in the index structure. Then retrieves the value using the key

5. **Insertion and Deletion**: Hash index also support insertion and deletion operations. When inserting data, the hash index calculates the hash code for the key and stores the data in the appropriate index location. Similarly, when deleting data, it locates the index entry using the hash code and removes it.

6. **Collision Handling**: Collisions can occur when different keys produce the same hash code. This is prevented by using recursion techniques such as chaining (using linked lists or other structures to store multiple items with the same hash codes in the same index bucket)

Hash Data Tampering

We can use hashing to prevent data tampering or corruption. Here are some examples of this:

1. **Checksums**: These are used in file integrity checks, where it is a fixed-size digest generated from the data. You can store this data and compare it with other checksums to see if the file has a different value, which indicates tampering.

2. **Cryptographic Hash Functions**: Crypotgraphic hash functions, such as SHA-256 or MD5, are designed to be one-way functions. This means that it is computationally infeasible to reverse-engineer the original data from the hash value. Remember the avalanche effect? If the original input is changed even a little bit, the resulting in a massive change in hash value.

Hash Partitioning

: Hash partitioning is a technique used in distributed computing and database systems to divide data across multiple nodes or partitions based on a hash function applied to a specific attribute or key.

1. **Partitioning Process**: In hash partitioning, each record in the dataset is processed by a hash function, which generates a hash values based on a chosen attribute or key of the record. This hash value determines which "block" the record belongs to.

2. **Equal Distribution**: Hash partitioning is that it aims to distribute data equally throughout the partitions. Since an ideal hash function equally distributes the data across its range, each partition should receive approximately the same number of records. This prevents any issues wen there are uneven distribution in the data.

3. **Efficient Query Processing**: Hash partitioning can also improve query performance. The query will only need to access only a certain part of the database, rather than the either one, which leads to faster execution times.

4. **Scalability**: Hash partitioning can allow for easy scalability by adding partitions or nodes to the system as the data grows or the workload increases. Since the data is evenly distrusted across partitions, adding more nodes does not affect the workload of any single node.

5. **Fault Tolerance**: If 1 node fails, then the data in that mode can be redistributed among different nodes and the system can operate without interruption.

Proof of Work

A simple proof of work (PoW) mechanism with hashing is often used in blockchain systems to secure the network against malicious actors and ensure the integrity of transactions. Here's a basic explanation of how it works.

1. **Problem Definition**: In a PoW system, participants (miners) compete to find a solution to a computationally difficult problem. This problem in particular is usually find a hash function that meets the criteria.

2. **Hash Function**: Each miner uses a cryptographic hash function to generate hash values from input data. The hash function takes an input (often a block of data including transaction data and a nonce) and produces a fixed-size output (the hash value)

3. **Target Value**: The network sets a target value that the hash of a block must be less than or equal to in order for the block to be considered valid. This target value determines the difficulty of the PoW problem

4. **Finding the Nonce**: Miners start with a block of data and a nonce (a number only used once). They repeatedly with different nonce values until they find a hash value that meets the target criteria (i.e less than or equal to the target value). Computationally intensive because you must do trial by error.

5. **Validation**: Once a miner finds the good block, they broadcast the block and make the other miners make sure that the block meets the target criteria.

6. **Block Addition**: Once a block is validated by the network, it is added to the network.

Encryption and Digital Signatures

Data encryption is the process's of converting plaintext (readable data) into ciphertext (unreadable data) using an encryption algorithm and a secret key as:

$$\text{Plaintext} \underset{\text{Encryption Algorithm and Secret Key}}{\rightleftharpoons} \text{Ciphertext}$$

Symmetric Encryption

This is the process where the same key is used for both the encryption and decryption process. Both sender and reciever share the same secret key, which they both use to both encrypt and decrypt messages. This is how it works.

- **Key Generation**: Both senders and reciever agree on a secret key.
- **Encryption**: The sender uses the secret key to encrypt the plaintext into ciphertext.
- **Decryption**: The reciever then uses the secret key to decrypt the message.

Symetric Encryption algorithms are typically faster and more efficient than asymmetric encryption algorithms, making them more suited for handling large amounts of data. However, one key flaw in symmetric encryption is the secure distribution of keys over insecure systems or in large scale distributed systems.

Boolean Arthimetic Concepts

- **XOR (Exclusive OR)**: It returns true (1) if the inputs are different and false (0) if the inputs are the same.
- **AND**: It returns true (1) if both inputs are true, otherwise false (0)
- **OR**: It returns true (1) if at least one input is true, otherwise false (0) if both inputs are false.
- **NOT**: It returns the opposite of the input (true becomes false and vice versa).

The stream cipher

An example of an symmetric encryption algorithm, where it encrypts plaintext bit by bit. Here is how it works.

1. **Key Setup**: The encryption key is used to initialize a pseudorandom number generator (PRNG). This PRNG generates the keystream, which is as long as the plaintext.
2. **Keystream Generation**: The PRNG generates a sequence of pseudorando bits based on the key. This sequence is the keystream, and must be unpredictable and must not be repeat for a given key.

3. **Encryption**: Each bit of the plaintext is combined (Usually XORed) with the corresponding bit of the keystream. For example, if plaintext bit is 1 and the corresponding keystream bit is also 1, the resulting ciphertext bit is 0. If the plaintext is 0 and the corresponding keystream bit is 1, the resulting ciphrtext bit is 1.

4. **Decryption**: The ciphertext is XORed with the keystream to retrieve the original plaintext.

In stream ciphers, XOR is the primary operation as XORing twice will just give you the original output.

1.0.1 Asymmetric Encryption

This is also known as public-key encryption. This is when a *a pair of keys* are used to encrypt and decrypt. Here is how it works:

1. **Key Generation**: Each user uses a pair of keys: A public key and a private key. The public key is shared openly and can be accessed by anyone.

2. **Encryption**: To send an encrypted message to a recipient, the sender obtains the public key to encrypt the message. Only the recipient, who has the private key, can use the private key to decrpt the message

3. **Decryption**: The recipient then uses their own private key to decrypt the ciphertext message, recovering the original plaintext. The private key in theory should be only kept by the owner.

Here are the advantages of using asymmetric encryption

- **Security**: Even if the public key is known to everyone, it is basically impossible to derive the private key. This ensures that messages with the public key can only be decrypted by the individuals that posses the corresponding private key.

- **Key Distribution**: Asymmetric encryption eliminates the need for secure key exchange between parties, as each user has their own key pair. Users can freely distribute their public keys without compromising the security of their private keys.

- **Digital Signatures**: One can sign a message with their private key, a sender can prove their identity and thus demonstrate that the message has not been tampered with.

A common algorithm used is RSA (Rivest-Shamir-Adleman). Here is a very basic overview of the RSA algorithm.

1. **Key Generation**
    - Choose 2 large prime numbers, (p) and (q).
    - Compute their product, (n = p x q), which serves as the modulus for both public and private keys.
    - Compute Euler's totient function, (phi(n) = (p -1) x (q -1)).
    - Choose an integer (e) (the public exponent) that is coprime[12] and less that (phi(n))
2. **Encryption**: To encrypt a message (M), the sender uses the recipient's public key ((e,n)) and computes a ciphertext (C) is then sent to the recipient.
3. **Decryption**: The recipient uses their private key ((d,n)) to compute the original text, recovering the original message.

The secuirty of RSA relies on the difficulty of factoring the modulus (n) into its prime factors (p) and (q). As long as (n) is sufficiently large and (p) and (q) are kept secret, RSA encryption remains secure.

Dask and Lazy Computation

I already covered what lazy computation is before. Essentially, lazy computation is just the idea that you first compile the calculations needed before executing any calculations. We do this using the python package **Dask**[13]. This inherently has many advantages.

- **Efficient Memory Usage**: Dask operates on a large dataset that may exceed the available memory. By deferring computation until necessary, Dask minimizes memory usage by only loading and processing data as needed, avoiding unnecessary overload.
- **Optimized task Scheduling**: Dask will consturct a task graph representing the computation, where each task represents a single operation on data. Using this graph, tasks are optimized and parallelized based on dependencies and available resources. Lazy Eval allows Dask to optimize task scheduling dynamically, distrubuting computations across multiple cores or machines for efficient parallel execution

[12] pair of integers who do not have a common factor other than 1

[13] When using Dask, make sure you pip install it using *pip install dask*

- **Avoid Wasted Work**: Lazy Eval allows Dask to fail early if there is a syntax error. For example a data processing task consisting of loading data and then filtering it might fun for hours before failing if there is a syntax or usage error in the filter.

The .compute method() in Dask is used to trigger the execution of the deffered computations and retrieve the final results. The significance of .compute() lies in its role as the point where lazy computation becomes eager evaluation.

AWS S3

Amazon Simple Storage Service (Amazon S3) is a highly scalable, secure and durable object storage service offered by AWS. It is designed to store and retrieve any amount of data from anywhere on the web, providing developers and businesses a cost effective solution for managing data.

1. **Overview of Amazon S3**:
    - Amazon S3 provides a simple web services interface that allows users to store and retrieve data from anywhere on the web.
    - It is built to be durable, have high scalability, and high availability.
    - Data is stored in buckets, which act as containers for objects. Each object can range in size from a few bytes to multiple terabytes.
2. **How Amazon S3 Works**:
    - Upload data into Amazon S3 using the AWS management console, AWS Command Linear Interface, or through Software Development Kits.
    - Once uploaded, the data is spread evenly throughout multiple available zones within a selected AWS region, allowing for high durability and durability.
    - Each bucket can have its own access permissions, allowing for secure access
    - Users can retrieve their data from Amazon S3 using HTTP/HTTPS requests, either directly through the AWS Management Console or progammatically via APIs.
3. **Use Cases and Benefits**
    - Data Backup and Archiving: Amazon S3 is cost effective and reliable at backing up and archiving data, providing long-term durability and availability

- Static Website Hosting: Users can host static websites directly from Amazon S3, allowing for scalability and low-latency performance to deliver content to users globally.
- Data Lakes and Analytics: Amazon S3 serves as a foundational component for building data lakes, allowing organizations to centralize and analyze large volumes of structured and unstructured data.
- Content Distribution: Amazon S3 integrates seamlessly with the other Amazon services, which allow the distribution of content globally with low latency and high data transfer speeds.
- Application Data Storage: Developers can use Amazon S3 to store data, and thus allowing these tasks like storage management to be offloaded, which ensure scalability as larger applications grow.

4. **Security and Compliance**:
    - Amazon S3 offers a wide range of security features, including encryption at rest and in transit, access controls, audit logs, and integration with AWS Identity and Access Management
    - It is compliant with industry standards and regulations [14]

Modern Data Architectures

A **data lake** is a centralized repository that allows for the storage of vast amounts of structured, semi-structured, and unstructured data in its native format at scale. These systems allow companies to store data without rigid schema. Then, raw data is then processed, transformed, and analyzed using various tools and tech for a wider range of purposes.

Different types of Tables

1. Normalised tables are tables that are broken up into multiple tables. This ensures data integrity and reduces the risk of anomalies like update anomalies, insertion anomalies, and deletion anomalies.

2. Denormalized tables are tables that are just combined into one giant table, where information is often duplicated, to optimize query performance and simplify data retrieval. This, however, can lead to increased amounts of storage

[14] Yet companies can *lose our data somehow*

space and may introduce data redundancy, which could potentially lead to data inconsistency if not managed properly.

Example of a Data Lake

Google's BigQuery is one such data lake solution. It enables users to analyze vast amounts of data quickly and cost-effectively using SQL-like queries without the need for maintenance and such.

1. **Data Ingress**
    - Batch Data Ingestion is the process when users load data in BigQuery from various sources using bq command-line tool and other APIs
    - Streaming data ingestion: BigQuery supports real-time data ingestion through its streaming data capabilities. This allows users to stream data directly into BigQuery tables using the BigQuery API or other libaries. This allows real-time analytis and processing of continously generated data streams
2. **Querying**
    - BigQuery supports standard SQL for querying structured and semi-structured data stored in its tables. Users can run ad-hoc queries, complex analytics, join multiple data sets, and aggregate using familiar SQL syntax.
    - Federated Querying allows users to analyze data across external data sources such as Google Cloud etc. to query and analyze data without having to first load it into BigQuery.
3. **Pricing Model**: Pay as you go.

What the law says: expectation of privacy

In the US legal system, the concept of "expectation of privacy" refers to an individual's reasonable anticipation of privacy in certain places, communications, or activities. This stems from the Fourth Amendment to the United States Constitution, which protects against unreasonable search and seizures from the government. Courts have historically interpreted this to mean that individuals have a legitimate expectation of privacy in locations such as their homes, as well as in their personal belongings, communications, and activities conducted in private spaces. However, this can vary on circumstances. If in public spaces, which includes social media, there is

an diminished expectation of privacy. The determination of whether an expectation of privacy exists often plays a crucial role in legal proceedings involving issues such as search warrants, surveillance, and the admissibility of evidence. To make things more concerte, we look for **4 canonical torts/legal standards** that define a privacy violation.

1. **Intrusion on Seclusion**: his tort involves the intentional intrusion, physically or otherwise, into the private affairs or seclusion of another person in a way that would be highly offensive to a reasonable person.

2. **Appropriation of likeness or identity**: This tort occurs when someone uses another person's name, likeness, or identity for commercial purposes without permission typically for advertising or commercial gain.

3. **Public Disclosure of private facts**: This tort involves the public disclosure of private information about an individuals that would be highly offensive to a reasonable person

4. **False Light**: This tort occurs when information is published about an individuals that portrays them in a false or misleading light which would be highly offensive to a reasonable person even if the information is technically true.

So what does all this have to do with data? When you a sign a ToS, you sign away your rights to data, and establish terms of how your data is used.

The Future

The General Data Protection Regulartion (GDPR) is a comprehensive data protection law enacted by the European Union (EU) in 2018. It aims to harmonize data privacy laws across the EU and provide individals with greater control over their personal data. Here are the core principles of the GDPR:

1. **Lawfulness, fairness, and transparency**: Personal Data Processing must be based on legal grounds, conducted fairly, and transparently disclosed to data subjects.

2. **Purpose Limitation**: Personal data should be collected for specified, explicit, and legitimate purpose and not be processed in a a manner that is incompatible with those purposes.

3. **Data Minimization**: Only the minimum amount of personal data necessary for the intended purposes should be collected and processed.

4. **Accuracy**: Personal data should be accurate and where necessary kept up to date. Inaccurate data should be corrected or erased without delay.

5. **Storage Limitation**: Personal data should be kept in a form that permits identification of data subjects for no longer than necessary for the purposes for which the data is processed.

6. **Integrity and confidentiality**: Personal data should be processed in a manner that ensures appropriate security.

7. **Accountability**: Data controllers are held responsible in complience to protocal.

LLMs

[15]We can break down how these LLMs work into 3 main components.

1. **Transformer Models and Attention Mechanism**

   - Transformer models are a type of deep learning architecture designed to handle sequential data, such as text
   - Attention mechanisms allow the model to focus on different parts of the input sequence when processing each token. This is important because this allows you to to understand relationships between tokens.
   - The Transformer architecture consists of multiple layers of attention and feedforawrd networks. This means that each layer processes each input separately and pass its output to the next layer.
   - By stacking many of these layers, the model can effectively capture complex patterns in data and generate coherent outputs.

2. **Tokenization**:

   - First, tokenize the text.
   - This allows the text to be formatted into a way that the model can understand, and formulate patterns.

3. **Autoregressive Token Prediction**:

   - Autoregressive Token Prediction is the idea that the next token will be predicted based on the sequence of previous tokens

[15]This is probably the most relevant in this class because of how much we used ChatGPT

- During inference, the model generates one token at a time by sampling from the predicted probability distribution over the vocabulary. Basically, the next token will be the one with highest odds of appearing.

And that's it.